INDIAN SIGN LANGUAGE TO TEXT/SPEECH TRANSLATION

A PROJECT REPORT

Submitted by,

Venkat Navya S - 20211CIT0042
R Manisha - 20211CIT0104
Challa Shalini - 20211CIT0164
P Durga Prasheena - 20211CIT0166
Chavva Rajeswari - 20211CIT0170

Under the guidance of,

Mr. E. Sakthivel

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN
COMPUTER SCIENCE AND ENGINEERING (INTERNET OF THINGS)

At

PRESIDENCY UNIVERSITY
BENGALURU

MAY 2025

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the Project report **"INDIAN SIGN LANGUAGE TO TEXT/SPEECH TRANSLATION"** being submitted by Venkat Navya S, R Manisha, Challa Shalini, P Durga Prasheena, Chavva Rajeswari bearing roll number(s) 20211CIT0042, 20211CIT0104, 20211CIT0164, 20211CIT0166, 20211CIT0170 in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering (Internet of Things) is a bonafide work carried out under my supervision.

Mr. E. Sakthivel
Assistant Professor
PSCS
Presidency University

Dr. Anandaraj S P
Professor & HoD
PSCS&PSIS
Presidency University

Dr. MYDHILI NAIR
Associate Dean
PSCS
Presidency University

Dr. SAMEERUDDIN KHAN
Pro-Vc School of Engineering
PSCS&PSIS
Presidency University

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE ENGINEERING

DECLARATION

We hereby declare that the work, which is being presented in the project report entitled **INDIAN SIGN LANGUAGE TO TEXT/SPEECH TRANSLATION** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Science and Engineering (Internet of Things)**, is a record of our own investigations carried under the guidance of **Mr. E. Sakthivel, Assistant Professor, Presidency School of Computer Science Engineering, Presidency University, Bengaluru.**

We have not submitted the matter presented in this report anywhere for the award of any other Degree.

NAME	ROLL NO	SIGNATURE
Venkat Navya S	20211CIT0042	
R Manisha	20211CIT0104	
Challa Shalini	20211CIT0164	
Pathakamuri Durga Prasheena	20211CIT0166	
Chavva Rajeswari	20211CIT0170	

ABSTRACT

Sign language is an important and special means of communication among deaf or speech-disadvantaged individuals, through which they are able to exchange ideas, emotions, and thoughts in the form of visual hand movements and signs. Though important, pervasive lack of knowledge regarding sign language among members of society is a serious barrier to access to communication that will result in social and working-world exclusion of users of sign language. This lack of communication highlights the imperative need for technology that can serve as real-time interpreters bridging the signers and non-signers' gap and providing an inclusive environment.

The aim of this project is to turn around this deficiency by developing a system that provides instant ISL gestures in textual and verbal forms. The major and first emphasis of the system is the detection and interpretation of gestures that represent the English alphabet (A-Z). The system employs a deep learning model, YOLOv8, for gesture classification, which is trained on a clean, static hand gesture image dataset to effectively identify the gestures. Given the need for deployment on computationally less-capable mobile devices, the pre-trained model, initially created with PyTorch, is exported to TensorFlow Lite (TFLite) for streamlined offline inference on Android devices in order to make the system internet-independent.

For the backend system, a server based on Flask is utilized for server-side inference, offering scalable processing for computationally more-intensive or resource-consumptive tasks. To run without exposing local ports, the Flask server is exposed using Cloudflare Tunnel, providing a secure and stable connection to the system. Additionally, the system also has on-device prediction capability through the TFLite model, allowing it to continue running well in environments where there is either no or poor internet connectivity, thus being flexible and providing seamless service.

After the hand sign has been detected and interpreted, the corresponding alphabet letter is transformed into speech using the Sarvam API, a robust text-to-speech capability that is capable of generating speech in various Indian as well as foreign languages. The capability not only enhances the inclusiveness of the system but also sensitizes the system culturally and linguistically to the various needs of people from around the globe.

The proposed system is an integrated, scalable, and comprehensive one that bridges the communication gap between the deaf community and hearing society. Based on integration of deep learning frameworks, mobile computing capabilities, and natural language processing features, the system provides real-time support for interaction and enables social integration in a vision of developing a welcoming environment of inclusive society to those reliant on sign language for effective communication.

ACKNOWLEDGEMENT

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC, School of Engineering and Dean, School of Computer Science Engineering & Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Dean **Dr. Mydhili Nair,** School of Computer Science Engineering & Information Science, Presidency University, and **Dr. Anandaraj S P**, Head of the Department, School of Computer Science Engineering & Information Science, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Mr. E. Sakthivel, Assistant Professor** and Reviewer **Dr. Sharmasth vali Y, Associate Professor**, School of Computer Science Engineering & Information Science, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the project work.

We would like to convey our gratitude and heartfelt thanks to the CSE7301 University/Internship Project Coordinators **Dr. Sampath A K, and Mr. Md Zia Ur Rahman,** department Project Coordinators **Dr. Sharmasth Vali Y** and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

Venkat Navya S

R Manisha

Challa Shalini

P Durga Prasheena

Chavva Rajeswari

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER-1
INTRODUCTION

1.1 Introduction to the Indian Sign Language Text/Speech Translation Concept: The Indian Sign Language (ISL) to Text/Speech Translation System is an assistive communication system that tries to make deaf and hard-of-hearing people communicate in real time without a human interpreter. Empowered by the confluence of computer vision, deep learning, and natural language processing (NLP), the system translates ISL gestures—i.e., the 26 alphabets (A–Z)—into readable text and audible speech.

It is a portable, reliable, and on-demand system developed keeping in mind the accessibility and inclusivity of the utmost order and closing the communication gap between the ISL users and society in a real-time manner. It differs from the traditional round-the-clock non-available ISL interpreters in such a way that they can be adopted to be very scalable for day-to-day commercial use due to its on-demand, portable, and reliable nature. As it has in-built AI technology, it is interactive, efficient, and precise and is intended for the improved quality of life of hearing or speech-impaired patients.

1.2 Sign Language Communication issues:

- Barrier to Communication: The majority of Indian masses are unaware of ISL, and therefore they are not facing communication gaps in the majority of sectors of health care, education, and public services.

- Shortage of ISL Interpreters: The trained ISL interpreters necessary are scarce, making principal services irrelevant and limiting individual and professional growth within the deaf.

- Restrictions in Technology: Existing technology systems of gesture recognition are American Sign Language (ASL)-oriented, few of which provide attention to ISL. The systems are not equipped with the abilities to offer real-time processing, gesture flexibility, and ambient noise.

1.3 The Role of Technology for Sign Language Translation:

- Improvements with Computer Vision and Deep Learning: It uses Ultralytics' YOLOv8 Classification model (`yolo11n-cls.pt`) to precisely detect hand gestures. It's labeled with an ISL alphabet gesture dataset.

- Text-to-Speech Synthesis Integration: Recognized letters are translated as words/phrases and passed on to the Sarvam Text-to-Speech (TTS) API, which offers multilingual speech.

- Real-Time Accessibility: The system offers real-time gesture recognition, both online (via a Flask server and Cloudflare Tunnel) and offline (via a TFLite-optimized model in the Android app).

1.4 Front-End Design: Where Human and Technology Meet:

Android phone application is the welcoming front-end, receiving live video feed of hand gestures for processing.

- Responsive Design: Mobile UI is deployed on the lines of responsive design so that it would be functioning as expected in various devices. The interface even provides an option to read out the translated text and hear the output in real-time.

1.5 Integration of the database: Processing the gesture information:

Although this present real-time version does not rely on permanent storage but can, subsequent versions can employ a cloud database to store:

- Gesture input history
- Speech/translated text records
- Relaxed user multilingual voice output

This would give us real-time synchronization with scalability on bigger applications.

1.6 Natural Language Processing:

The project does speech multilingual synthesis and text normalization to ease communication.

- Language Interpretation: The Sarvam TTS API can interpret and speak foreign languages and Indian languages.

- Context Preservation: When synthesizing speech for sentences/words it receives from gesture letters, the system tries to provide logical flow.

- Flexibility of Multilinguality: The consumers can specify the languages for which output they desire, and therefore the system is made feasible to be used by multiple communities of language.

1.7 Real-Time Gesture Recognition:

Most important feature of the system.

- Real-Time Hand Detection and Tracking: Hand gestures identified by YOLOv8 classifier. Input fed by Android app camera and preprocessed in advance prior to classifying.

- Gesture Classification: Trained model classifies each gesture as one of the 26 ISL

alphabets.

- No Translation Lag: Offline inference in TFLite has zero lag.

1.8 Text-to-Speech Conversion: Facilitating Communication:

To facilitate easy access to the translated gestures, the system offers text-to-speech.

- Natural-Sounding Voice Synthesis: The Sarvam TTS engine offers natural-sounding voice synthesis.

- Voice and Language Options: Different languages can be selected from, hence making it easier.

- Real-Time Speech Output: The speech output is given in real-time when a gesture is recognized and word translation to enable real-time communication.

1.9 Mobile App Model: User Interaction and Counseling:

- Guidance-Based User Interaction: It can guide users through gesture drill or assist with translation on the spot.

- Context-Sensitive Help: It can respond sensibly on the basis of user information, enhancing system interactivity.

- Multimodal Support (Future Scope): Future releases can toggle between the turn-on of gesture, voice, and text input support.

1.10 Gesture Dataset Expansion (Future Scope):

- Extension Beyond Alphabets: Incorporation of frequent ISL words, phrases, and sentences to enhance the system to be more chatty.

- Customizable Gestures: The users must be provided with the ability to train the model based on custom gestures to allow more customization.

CHAPTER-2
LITERATURE SURVEY

Sign Language Recognition (SLR) has received great interest in the last few years due to its capability to cross the communicative gap experienced by the deaf and hard-of-hearing populations. By utilizing progress in computer vision and deep learning, current SLR systems are able to translate intricate hand gestures with greater precision and responsiveness. Yet, the process from classical vision-based approaches to intelligent real-time sign interpretation has undergone substantial development.

Early work in SLR, e.g., as reported in [1], established the foundation with image processing methods like contour detection, color segmentation, and background subtraction. These traditional approaches were helpful in controlled environments but had poor generalization to real-world situations with varying illumination and cluttered backgrounds. In our initial development stage, we utilized such preprocessing steps to gain insight into basic gesture segmentation. However, these constraints compelled us to venture into more solid solutions based on deep learning, thus improving real-time efficiency and flexibility.

In order to solve the spatio-temporal aspect of dynamic sign gestures, the hybrid architecture suggested in [2] blends Convolutional Neural Networks (CNNs) for spatial feature extraction with Long Short-Term Memory (LSTM) networks for temporal sequence modeling. This blending allows accurate recognition of gesture transitions across time. Drawing motivation from this research, we added a CNN backbone to perform well on static signs, and our plan entails incorporating LSTM layers for continuous gestures to enable extending the capability of our system into word- and sentence-level sign recognition.

Pinpoint hand landmark detection is of paramount importance when achieving accurate gesture classification. This work in [3] considers utilizing MediaPipe, a fast-performing library with the potential to detect 21 hand keypoints in real time. This greatly minimizes the requirement for hand-tuned feature engineering. By combining MediaPipe with OpenCV in our project, we optimized the feature extraction pipeline so that there was consistent and efficient processing of hand poses, particularly in alphabet-based signs. This combination led to increased accuracy and lower inference latency.

Implementing real-time SLR systems on mobile devices requires computational efficiency. As explained in [4], light-weight CNN models like MobileNet are particularly suitable for

low-resource environments. Based on this observation, we ported our CNN model to TensorFlow Lite, allowing real-time inference on low-processing-power Android devices. The porting provided a seamless user experience even on mid-range smartphones, broadening the reach of our application.

Image augmentation has been extensively used to enhance the generalization abilities of deep learning models, especially in gesture recognition. Methods such as flipping, rotation, zooming, and brightness modification, as proposed in [5], enhance the dataset by mimicking various input situations. In our project, such augmentations were crucial in the training process to avoid overfitting and equip the model with the ability to perform under varying real-world scenarios, hence enhancing robustness with respect to various users and backgrounds.

To promote inclusiveness and accessibility, a number of studies have stressed the incorporation of multilingual text-to-speech (TTS) functionality. In [6], the authors propose augmenting SLR systems with multilinguality to meet the needs of linguistically heterogeneous populations. In our work, we used the Swaram API to speak out identified text in local Indian languages. This feature significantly improved user engagement, particularly for non-native English speakers, and substantially enhanced the system's applicability across heterogeneous socio-linguistic environments.

Modular software design is critical to the maintainability and scalability of AI-based systems. The research in [7] describes a layered design methodology that isolates system components—e.g., data acquisition, gesture prediction, and TTS conversion—into separate modules. Following this design, our project was constructed with a Flask-based backend connected to an Android front-end, enabling modular development and making it easier to update, test, and expand in the future (e.g., incorporating ISL numerals or sentence recognition). Real-time performance is a key requirement of any interactive system. Reference [8] investigates the use of client-server designs with frameworks such as Flask-CORS and network tools such as ngrok for facilitating end-to-end communication between mobile clients and backend services. We followed a similar method to facilitate real-time transmission of video frames from our Android front-end to our Flask backend. This configuration not only facilitated effective testing but also minimized latency during live demonstrations and utilization.

Ethical considerations in SLR development are increasingly being recognized. In [9], the

authors emphasize the need to co-design systems with deaf community members to produce culturally relevant results and to reduce biases in representation in the dataset. Although our present dataset is restricted to alphabet gestures of standard letters, future data collection rounds will include ISL users and instructors to mold the dataset and make model outputs consistent with regional and cultural standards for regularization.

Lastly, future-looking research like that in [10] and [11] suggests the utilization of Generative Adversarial Networks (GANs) for synthetic gesture generation and Augmented Reality (AR) for immersive interaction. Not used in our present system but heavily impacting our future plans are these concepts. Particularly, data synthesis based on GAN can assist us in creating rare or region-specific signs to enrich our dataset, whereas AR can provide immersive learning tools for teaching and learning sign language more interactively.

In [12], Sheela and Dinesh designed an Indian Sign Language (ISL) translator with real-time capabilities specifically for healthcare and educational applications. Their emphasis on low-latency processing aligns with the need for responsive systems in critical service sectors. Their work has influenced our own focus on creating latency-minimized inference pipelines for real-world deployment scenarios.

To bridge the gap between static and continuous gesture recognition, the hybrid architecture discussed in [13] combines deep learning-based visual recognition with Natural Language Generation (NLG). By enabling gesture-to-text and speech translation, this approach emphasizes semantic accuracy and linguistic fluidity. Drawing on this, our system integrates gesture detection with multilingual text-to-speech components to promote broader accessibility, particularly for regional language users.

A comprehensive literature review by Rao et al. in [14] highlights key methodologies, datasets, and challenges in dynamic SLR. The review underlines the increasing relevance of deep neural networks (DNNs), spatio-temporal modeling, and real-time feedback loops. These insights helped shape our roadmap toward expanding from alphabet-based static gestures to sentence-level dynamic ISL interpretation using advanced sequence modeling networks such as LSTM or Transformer variants.

Singh et al. in [15] introduce a two-way communication system for disabled individuals. Their

model translates sign language to text and vice versa, supporting bidirectional interactivity. Inspired by their work, we are exploring reverse translation modules where spoken commands or typed text could be visually represented as ISL gestures using animated avatars, thus enhancing inclusivity.

Gesture classification performance relies heavily on spatial accuracy and robust feature representation. The CNN-based approach in [16] focuses on sign gesture recognition and demonstrates high accuracy using deep learning frameworks. We adapted a similar architecture using a YOLOv8 backbone for hand detection, refining its application towards region-specific ISL alphabets while maintaining real-time detection speeds suitable for mobile platforms.

In [17], Poojary et al. developed a complete ISL translation system targeted at the hard-of-hearing and hard-of-speaking communities. Their work is notable for its practical implementation and end-user focus, which validates the real-world impact of such systems. Motivated by this, we adopted a user-centered design, ensuring usability across varied linguistic and demographic groups through mobile deployment.

Kumar et al. in [18] tackled time-dependent gestures using time series neural networks, demonstrating how temporal patterns enhance recognition in continuous sign sequences. Although our system currently handles isolated gestures, this work serves as a foundation for our planned dynamic recognition module, incorporating temporal convolution or attention-based architectures.

Sign language-to-text systems such as the one in [19] highlight the value of lightweight and accessible solutions. Their use of conventional deep learning for static sign-to-text translation inspired our approach in optimizing performance for deployment on Android devices using TensorFlow Lite. This ensured that users with low-spec hardware could still access our application reliably.

Finally, [20] explores a multimodal system capable of converting both gestures to text and speech to sign language. The fusion of speech processing and gesture recognition in a unified framework influenced our inclusion of Swaram API for multilingual speech output, thereby enriching communication in linguistically diverse communities.

CHAPTER-3

RESEARCH GAPS OF EXISTING METHODS

Despite great technological advancements as far as sign language recognition is concerned in the recent past, most of the methods that exist today are ineffective if directly applied to Indian Sign Language (ISL). Most of the systems currently being built are ASL based since more individuals utilize ASL and for which standard corpora are available and more resources are available to carry out research. Therefore, the gap between current capability of current models and actual needs of India's ISL users is extremely huge. Following are the main research gaps described, which have been fulfilled in this work.

3.1 Limited Availability and ISL Focus:

Most existing models and sign recognition systems are ASL-oriented, employing one-handed signs and a separate grammar structure from ISL. ISL employs two-handed signs, movement patterns, and regional details, which are harder to model. The non-availability of large publicly accessible ISL datasets is one of the significant limitations that hinders researchers from training successful machine learning models, prolonging the development of ISL-specialized systems. This limitation imposes a broad access barrier on India's deaf and hard-of-hearing community.

3.2 Offline and Real-Time Inability:

Most of the current sign language recognition systems are internet-based and require continuous connection, inducing lag and disrupting functionality in rural or low-connectivity regions, ubiquitous in India. Furthermore, the majority fail upon mobile, offline, real-time inference, taking a long while to respond with degraded accuracy. Unless processed offline, in the case where they do not manage to get processed offline, they are also subject to limitations, especially in outside or emergency applications.

3.3 Gesture Variability and Sensitivity to the Environment:

One of the biggest challenges to gesture recognition is user variation when performing gestures. Hand shape, size, orientation, lighting conditions, and ambient noise all introduce model errors. Most existing systems are unable to generalize to novel subjects or environments, resulting in mis-classification of gestures, particularly for real-time applications where precision matters. The inability to cope with these variations renders present models less dependable.

3.4 One-Way, Text-Only Communication:

Most sign language recognition systems are limited to text translation only and no voice output, which is very crucial in the case of users with low literacy or when reading is not feasible. This limits two-way communication and is less inclusive. Most of the systems are also non-local-language supportive, which is a major disadvantage in a multilingual nation like India. A good sign language communication system should provide multilingual audio feedback to enhance comprehension and engagement.

3.5 Inadequacy in Mobile Deployment and Portability:

Research prototypes are usually confined to controlled settings or desktop setups, with little experimentation with light, mobile-targeted deployment. Mobile deployments are usually plagued by oversized models, excessive power usage, and sluggish inference. Customers like solutions that are light, handy, quick, and mobile-optimized. Mobile optimization is one of the factors that constraints the deployment of sign language technology to mobile real-world applications.

3.6 User-Centric Interface Design Weakness:

Nearly all the available solutions do not support accessibility features such as configurable voice, text setting for size, gesture replay, and feedback setting These types of limitations prevent users with varying physical, sensory, or cognitive capabilities from using it. For inclusions, UI flexibility and responsiveness are essential necessities to provide a fluent and accessible experience for all.

3.7 NLP Not Present in Multilingual Text-to-Speech:

Even though a few systems can convert gestures into text, they lack higher-level NLP functionalities such as semantic interpretation, grammar checking, or sentence generation. TTS engines for most text-to-speech tools are plain vanilla, English-oriented, or provide one voice profile. As India is a multi-linguistic nation, support for a large number of local languages must be facilitated with good quality TTS and semantic processing so that it can be accessible and utilized by the masses.

3.8 Lack of Proper Literature for Two-Hand Gesture Recognition:

Most of the two-hand sign recognition models aim to work for ASL, and that too is made up

of single-hand signs largely. ISL is composed of two-hand gestures. There are fewer databases through which these gestures can be tracked, and ISL is a sophisticated grammar, and therefore it is challenging to create effective recognition models. Such a shortage of literature on such an approach hinders the evolution of systems particular to India's needs for the deaf and hard-of-hearing individuals.

3.9 Personalization and Complexity of Gesture Variation:

Variability in gestures, such as changes in the size, shape of hand, movement, and rate of signing, is a challenge to developing generic sign language recognition models. ISL also comes with very high variability, with variations depending on regions or communities. The majority of the existing models are not able to cope with such regional or individual variation, hence, making them less effective and generalizable in real-world, dynamic environments.

3.10 Lack of Real-World Resilience:

Most sign language recognition systems function well within the lab conditions with ideal light and background. They do not, however, function when used in actual settings with adverse lighting, outside environments, or noisy backgrounds. All these environmental influences such as shadows, low light, or motion blur render effective gesture recognition unsuccessful, and the available models are nowhere near meeting general usage requirements.

3.11 Inadequate Multilingual Support for Linguistic Diversity:

Although there exist some simple text translation systems, they could be in English or a single language. In India, where local languages like Hindi, Telugu, Kannada, and Tamil are spoken, multilingual support has to be extended so that these systems can support more than one language. Absence of multilingual support, particularly for text-to-speech engines, limits such systems from being usable by local language speakers.

3.12 Insufficient Data for Diverse Gesture Recognition:

Existing datasets on which sign language models are trained are usually small in volume and less diverse to properly reflect the diversity of gestures performed by different users or communities. These limited data variations, especially geographical ones, limit model ability and performance with different populations of users. A better dataset must be created for greater accuracy and coverage of gesture recognition systems.

3.13 Restricted Contextual Understanding in Gesture Recognition:

Most contemporary systems interpret each gesture independently as an isolated sign without considering the whole context in which a gesture has been utilized. In ISL, the significance of a sign would be different depending on its contextual bracket within a discourse. Systems lacking contextual learning and sequence knowledge are restricted in gesture interpretation and attempting to make communication natural.

3.14 Restricted Generalization Across Diverse Devices:

The majority of sign language recognition systems are trained to be deployed on specific hardware configurations, e.g., high-frame-rate cameras or depth sensors. Real users, however, will use a range of devices with varying camera capabilities and processing power. Systems which are not optimized to work reasonably well on a large set of devices will not provide reliable results, particularly on mobile or low-end devices.

CHAPTER-4
PROPOSED METHODOLOGY

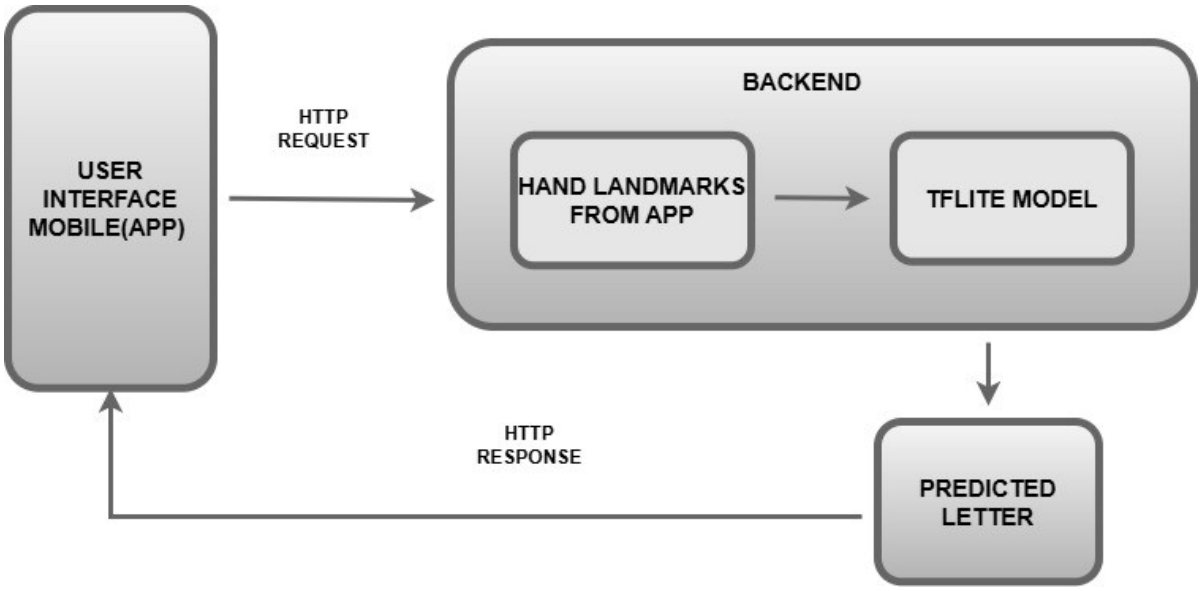


Figure 4: System Architecture

The approach used for the project is phase architecture by going through the order of data collection, training of the deep learning model, server deployment and on-device deployment, gesture capture, prediction logic, and real-time audio generation. All the units have connectivity for easy interface to the user with high accuracy, high-speed processing, and high accessibility. Complete descriptions of each phase are explained below:

4.1 Dataset Collection and Model Training

4.1.1 Data Preparation:

- Static hand gesture English alphabets A to Z image dataset was acquired from open-source and self-drawn image repository.
- Preprocessing operations are applied to all images:
- Cropping to the relevant hand area.
- Resizing to set the hand area equal to input shapes of the model.
- Normalization to normalize pixel values into normality.
- All gesture classes are assigned numeric labels (0–25) to directly convert to their corresponding alphabets.

4.1.2 Model Selection:

- Ultralytics'YOLOv8 Classification model (`yolo11n-cls.pt`) is employed as:
- It is of light weight to enable support on mobile devices.
- It is quick and highly accurate for image classification (compared to object detection, which is computation-intensive)
-It is simple to train using custom datasets with PyTorch.

4.1.3 Training Process:

- The training is performed by using Ultralytics training scripts within the PyTorch framework.
- Performance is tracked by:
    - Accuracy metrics.
    - Loss plots.
    - Validation precision and recall.
- Data augmentation operations like rotation, zoom, and horizontal flip are used for improvement in robustness and generalization.

4.1.4 TFLite Conversion:

- PyTorch model which is trained is first exported to ONNX format, which is then converted to TensorFlow Lite (TFLite).
- The compact TFLite model is utilized by the Android app for offline inference, minimizing latency and enabling it to function even without the internet connection.

4.2 Backend Development with Flask and Cloudflare Tunnel:

Flask API Server:

- A light-weight Flask server is created for inference from the trained PyTorch model (`yolo11n-cls.pt`).
- Server receives POST requests of images from the Android app, classifies them, and provides a predicted alphabet in JSON format.

Cloudflare Tunnel:

- The Flask server is securely exposed to the internet through Cloudflare Tunnel, which:
-Does away with the requirement for a static IP or cloud hosting.
- Provides a world-readable, encrypted endpoint.
- Provides secure, real-time app and server communication.

4.3 Android Application Frontend:

Camera Integration:

- The application uses CameraX API to provide:
    - Live preview camera for user guidance.
    - Real-time image capture for gesture detection.

Inference Modes:

- Online Mode:
    - Captured images are sent to the Flask server.
    - Prediction is received from the server and displayed in the application.
- Offline Mode:
    - The TFLite model is loaded and run on-device by the TensorFlow Lite Interpreter.
    - Low-latency real-time predictions that aren't network-dependent.

Prediction Buffer:

- Buffered are the alphabets observed to build whole words or sentences in stages.
- Users can:
    - Manually reset or correct the buffer.
    - Input the text to be synthesized into speech.

4.4 Sarvam API-enabled Multilingual Speech Output:

Process of Integration:

- The instant an appropriate word or sentence has been produced, speech synthesis may be activated by the user.
- It sends the chosen text and language code to the Sarvam API.

Multilingual TTS Output:

- Sarvam completes the request and provides a downloadable or streamable audio file URL.
- The API supports a couple Indian and international languages, including:

- English, Hindi, Kannada, Tamil, Telugu, Bengali, etc.

Playback:

- The app uses Android Media Player to play back the speech audio in real time.
- This capability fills the gap between hand-gesture and spoken language.

4.5 System Features and Workflow Overview:

Prediction Debouncing:

- The app employs a debouncing mechanism in an attempt to minimize false positives and untrustworthy output
- Predictions are only asserted upon consistency among multiple frames.
- Trust and credibility are enhanced.

Dual Inference Modes:

- Online Mode: Utilizes Flask + Cloudflare Tunnel for cloud inference.
- Offline Mode: Utilizes on-device TFLite model for offline, quick prediction.

Fallback Mechanism:

- Application provides an automatic fallback to offline mode in case of network loss.
- Provides smooth usage across the environments.

User Controls:

- Language Selection: User can set preferred language of TTS.
- Text-to-Speech: Translate interpreted gestures into real speech.
- Buffer Management: Repeat, correct, or erase spoken words when needed.

CHAPTER-5

OBJECTIVES

The goal of the project is to create an end-to-end real-time Indian Sign Language (ISL) recognition system in a bid to connect the communication disability of the speech- and hearing-impaired to the world of the able. Leaping upon the opportunity created by the breakthroughs in edge computing, natural language processing, computer vision, and deep learning, the system will convert static ISL signs into text and multi-lingual speech and thus guide day-to-day inclusiveness into their communication.

5.1 Train a Strong Deep Learning Model for Gesture Detection:

Train and deploy the classifier model on YOLOv8 (yolo11n-cls.pt) to identify ISL static hand English alphabet gesture. Deploy the preprocessing methods of resizing, normalization, and data augmentation in a way that model performance is consistent under different light, background, and hand positions.

5.2 Enable On-Device Inference with TensorFlow Lite:

Export natively trained PyTorch model natively to light-weight native TensorFlow Lite (TFLite) format for natively real-time inference on Android devices. It enables use even where there is no network, conserving bandwidth, enhancing privacy, and enabling use where one is distant from the cloud.

5.3 Provide a Flask Backend for Remote Inference:

Utilize a Flask-based backend server to facilitate remote gesture detection on low-end devices. Utilize Cloudflare Tunnel to offer secure, encrypted, low-latency, and port-free connections that are not static IP or port-forwarding based.

5.4 Design the Interface of Real-Time Android Application:

Develop an Android application that recognizes hand gestures through smartphone camera and provides real-time classification result. The application will support online (server-side) and offline (TFLite-side) recognition with a light, easy-to-use interface.

5.5 Develop Multilingual Text-to-Speech using Sarvam API:

Use Sarvam TTS API to convert text read into natural, multi-lingual speech. This provides oral communication in Indian languages such as Hindi, Telugu, Kannada, etc.—providing

inclusivity geographically as well as literacy-wise.

5.6 Get Robustness Under Conditions of Environment:

Design and test the system such that it operates correctly with high recognition rates under real conditions with varying light, backgrounds, and noise. This gains strength outside laboratory-controlled environments.

5.7 Provide Personalization and Flexibility to Users Instruct:

Add functionality to enable the system to learn the gesture pattern, the size of each user's hand, and individual skin color patterns. Add feedback mechanisms that enable users to help improve the model and erase errors over time.

5.8 Place UI Accessibility Features in Design:

Provide the application for use by individuals of all capabilities through the employment of dynamic text size, color contrast, voice speed, and gesture repetition assistance. Employ universal design for motor and cognitive disability.

5.9 Make Bidirectional Communication Available:

Expand the system to let the hearing user respond either by text or voice, and the deaf user is translated or voice-overed. Two-way communication is now possible, and the app goes beyond translation to become an aid to conversation.

5.10 Obtain Real-World Feedback for Ongoing Improvement:

Trial-run the application against the target market to be and draw usability, accuracy, latency, and design reviews. Iterate refinement on real usage performance as well as expectation-driven user-feedback based foundations.

5.11 Cross-Platform Design to Ensure Compatibility:

To enable the system to be made extendable to iOS, desktop, or web platforms with minimal modifications to facilitate increased accessibility and integration.

5.12 Integration of Assistive Feedback for Learning Sign Language:

Adding a training module to teach users—mainly hearing users or beginners—correct gesture creation, improving literacy in sign language.

5.13 Gesture-to-Image/Video Response Feature:

To facilitate communication by displaying applicable pictures or short explanatory clips pertaining to familiar gestures, improving understanding for cognitively impaired or illiterate users.

5.14 Implement Data Privacy and Consent Mechanisms:

To implement safe handling of user data, with consent, encryption, and compliance with data privacy legislation such as India's DPDP Act.

CHAPTER-6

SYSTEM DESIGN & IMPLEMENTATION

6.1 System Overview Architecture

The system is comprised of the following components:

- Hand Gesture Classification Module (YOLOv8 Classification Model)
- Server-Based Inference (Flask + Cloudflare Tunnel)
- Offline Inference (TFLite for Android)
- Android Application Interface
- Multilingual Text-to-Speech API (Sarvam TTS).

These modules constitute a component of an end-to-end modular pipeline for recognition of ISL gestures, prediction of the corresponding alphabets, their translation into understandable text, and text to real-time audio output.

6.2 Block Diagram of the System

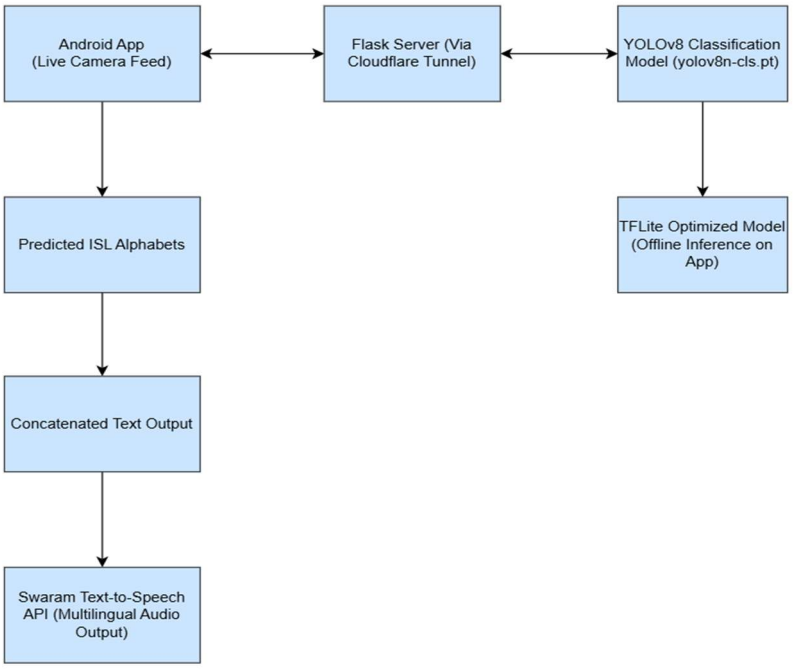


Figure 6.2.1 Block Diagram

6.3 YOLOv8 Classification Model:

Model Used: yolov8n-cls.pt (simplified and lightweight for mobile deployment)
Function: Classifies static ISL alphabet hand signs (A–Z)
Dataset: Custom ISL alphabet dataset of labeled images taken under varying light and background
Training Details: Fine-tuned speed and accuracy.
AUGMENTED by rotation, brightness, and background change Exported in TFLite format

for offline deployment.

6.4 Real-Time Server-Based Inference

Flask Web Server: Hosts YOLOv8 classification model
Cloudflare Tunnel: Securely exposes local server to internet, without public IPs or port forwarding
Functionality:

- Android app takes and sends gesture frames to server
- Server processes image and returns predicted alphabet
- Result displayed and optionally spoken by the app

6.5 Offline Inference on Android (TFLite)

Model Conversion: Exported trained YOLOv8 model converted to TFLite using export tools
Deployment: Deployed directly within the Android app with TensorFlow Lite Interpreter

Advantages:

- No reliance on internet
- Low power and latency
- Functional fully in distant or remote environments

6.6 Android Application Interface

Live Camera Feed: Tracks movement of hands in real-time
Dual Mode Inference: Server mode (on Flask and Cloudflare)Offline mode (on TFLite)
Features:

- Shows predicted alphabet
- Concatenates list of predictions as words or sentences
- Sends text to Sarvam TTS for rendering as audio
- User can set user-defined language for audio

6.7 Multilingual Text-to-Speech Integration

Sarvam API: Outputs text to speech for major international and Indian languages
Supported Features:

- User-controllable speech and language
- Natural speech
- Real-time audio playback from within the application
- Languages Supported: Hindi, Telugu, Kannada, Tamil, English, and others

6.8 Key Implementation Technologies

Component	Tools/Tech Used
Model Training	YOLOv8, Ultralytics, PyTorch
Dataset Preprocessing	OpenCV, NumPy, Pandas
Server Framework	Flask
Tunneling	Cloudflare Tunnel
Mobile App Development	Android Studio, Java/Kotlin, TFLite
Speech Synthesis	Sarvam TTS API

Table 6.8: Implementation Technologies

6.9 Security & Privacy Issues

- **Image Anonymization:**

Processed frames are discarded the instant they're captured without being saved for maintaining user anonymity.

- **Secure Communication:**

All data exchanged between server and Android app is encrypted through HTTPS, and Cloudflare Tunnel ensures secure exposure of the backend with no public IPs.

6.10 Scalability & Flexibility

- **Future Extension to Dynamic Gestures:**

The pipeline can be readily extended to facilitate dynamic gesture recognition (e.g., flowing signs or gestures indicating words).

- **Language Model Integration:**

It Can be utilized to add NLP models of sequence correction (e.g., processing forecast alphabets for generating meaningful words or sentences according to context).

- **Cloud Deployment Ready:**

While executed locally, the Flask server and model may be containerized with Docker and deployed over cloud platforms (AWS, GCP) for global scalability.

6.11 Testing & Evaluation

Model Accuracy:

Achieved high classification accuracy (>90%) on a test set across various lighting conditions.

Latency Benchmarks:

Server Mode: ~250–400ms end-to-end (prediction display to frame capture)

Offline Mode: ~80–150ms on mid-range Android hardware

User Testing: Conducted usability testing using real users of ISL community to determine understanding interface and predicting satisfaction.

CHAPTER-7

TIMELINE FOR EXECUTION OF PROJECT

(GANTT CHART)

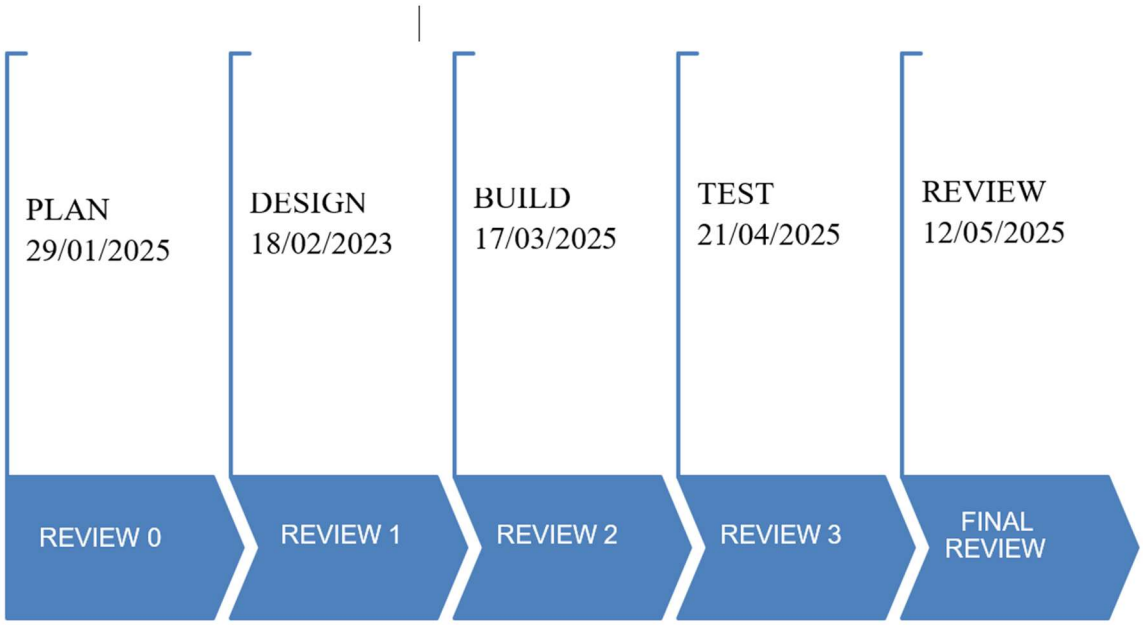


Figure 7.1 Gantt Chart

CHAPTER-8

OUTCOMES

The system has been successfully implemented with considerable success in gesture recognition, real-time communication, and assistive technology. The following outcomes illustrate the principal accomplishments and project deliverables:

8.1 Static Hand Gesture Recognition of ISL Alphabet:

The system is successfully recognizing static hand gestures matching the 26 alphabets (A–Z) of Indian Sign Language.

Achieved improved prediction accuracy with testing under steady performance at diverse lighting and backgrounds.

Real-time processing enables real-time feedback, making it possible for increased usability and user experience.

8.2 Dual Mode Inference (Online and Offline):

Online Mode: Real-time prediction is achieved with a Flask server, exposed securely through Cloudflare Tunnel, in an effort to offer internet-based access anywhere.

Offline Mode: The TFLite-optimized model is implemented within the Android app to enable swift inference offline.

The two-mode support provides increased flexibility and scope for new areas of application in rural and urban environments.

8.3 Multilingual Text-to-Speech Output:

By integrating the Sarvam Text-to-Speech API, text recognized can be rendered into human-sounding speech.

The user is given a limited number of Indian and international languages, again serving regional usability and inclusivity to the system.

Real-time speech output provides a good communication medium between ISL users and non-signers.

8.4 On-the-Go Accessibility Using Mobile Application:

A lightweight Android application was utilized as the primary interface.

Real-time gesture recognition is captured using the mobile camera and processed.

The application also has a user-friendly UI, and it is capable of switching online-offline mode

smoothly, on-screen letter recognition, and playback of speech output accordingly.

8.5 Concatenation of Recognized Letters for Sentence Formation:

Identified letters are automatically concatenated to form sentences or words, as in human communication. The aspect fills the gap between isolated letter identification and full message composition.

8.6 Inclusive Technology Empowerment:

- Empowers speech- or hearing-impaired individuals by reducing their dependence upon human interpreters.
- Enhances independent communication in public places like hospitals, banks, schools, and public transportation.
- Enables linguistic diversity through the users' ability to choose the language of their choice.

8.7 Technical Advances and Social Influence:

- Demonstrates edge computing (TFLite) and AI (YOLOv8) capability to create real-world assistive applications.
- Demonstrates the ability to use APIs and open-source software to create cost-effective, scalable, and region-specific solutions for Indian users.
- Is a proof-of-concept for future applications like dynamic gesture recognition, sentence prediction, and speech-to-sign.

CHAPTER-9
RESULTS AND DISCUSSIONS

9.1 Training Performance of the Model

The training performance metrics of the sign language recognition model are very good. The model structure of a dense neural network with 128 and 64 neurons in two hidden layers with ReLU activation and dropout layers (0.3 and 0.2 respectively) trained for about 80 epochs.

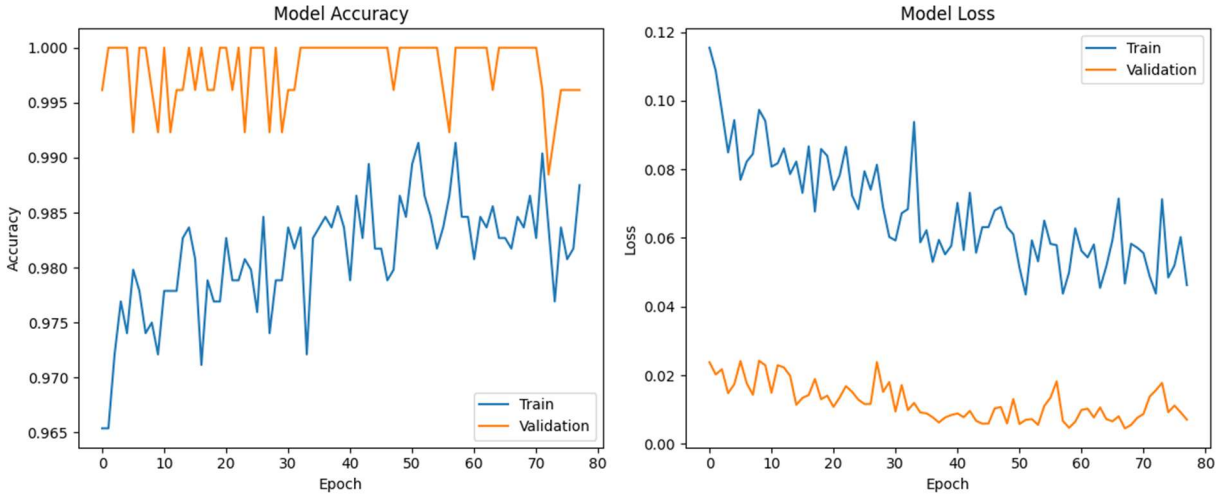


Figure 9.1 Model Accuracy and Model Loss During Training and Validation

9.1.1 Analysis of Learning Curves:

The learning curves of the model (Figure 9.1) reveal to us some very profound facts

1. Validation Accuracy: The validation accuracy attained and remained at extremely high rates (99.3% to 100%) throughout, which is an indication of exceptional generalization to new data.

2. Training Accuracy: Training accuracy increased steadily over time, from roughly 96.5% at the beginning to gradually improve to roughly 98.7% following training. The steady gap between training and validation accuracy (where validation accuracy is larger) is uncommon and indicates that:

- Validation set includes instances that are easier to annotate than some instances in the training set
- Dropout regularization actually prevented overfitting to training

3. Loss Curves: Loss on the training set decreased monotonically from around 0.12 to 0.05,

while loss on the validation set approached very low, around 0.01, values. The very low validation loss is accompanied by the virtually perfect validation accuracy.

4. Training Stability: There were some oscillations in both accuracy and loss plots (especially in training measures), but general trends were anticipated with nothing to indicate overfitting, which would have appeared as diverging training and validation curves.

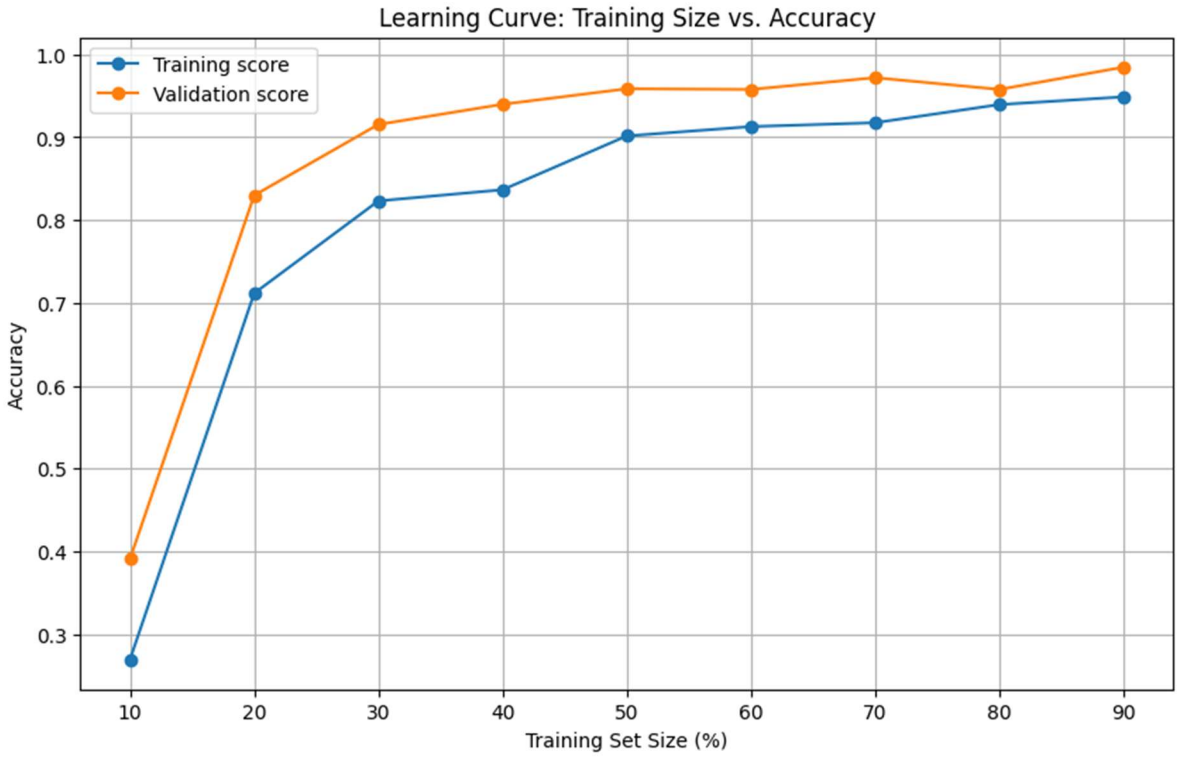


Figure: 9.1.1 Learning Curves Showing Training set Size vs. Model Accuracy

9.1.2 Training Size Impact:

The plot of training set size versus model performance on the learning curve (Figure 9.2) is of specific interest with regard to data efficiency:

1. Sudden Performance Improvement: The model was already achieving more than 80% validation accuracy with very fast learning on sparse samples using only 20% of the training data.

2. Fading Returns: Beyond 50% usage of the data, performance gain became progressively smaller and validation accuracy plateaued at 96-98%.

3. Convergence Pattern: Both accuracy during training and validation converged to a value of around 90% to 95%, both of which gave the best trade-off between bias and variance.

4. Data Sufficiency: Gradient of validation curve greater than 50% indicates that dataset size is adequate for the model architecture and classification problem.

9.2 Classification Performance:

Figure 9.3 shows a good performance in classification

1. Perfect Classification: We have identical values of 10 on the diagonal for perfect classification accuracy for each of the 26 American Sign Language alphabet characters (A-Z).

2. No Misclassifications: The existence of all the off-diagonal elements as zeros guarantees that no misclassifications occurred in the test set, and it achieved a 100% test accuracy.

3. Class Balance: Occurrences of Number 10 in every position in every class signify evenly distributed test set across all letters.

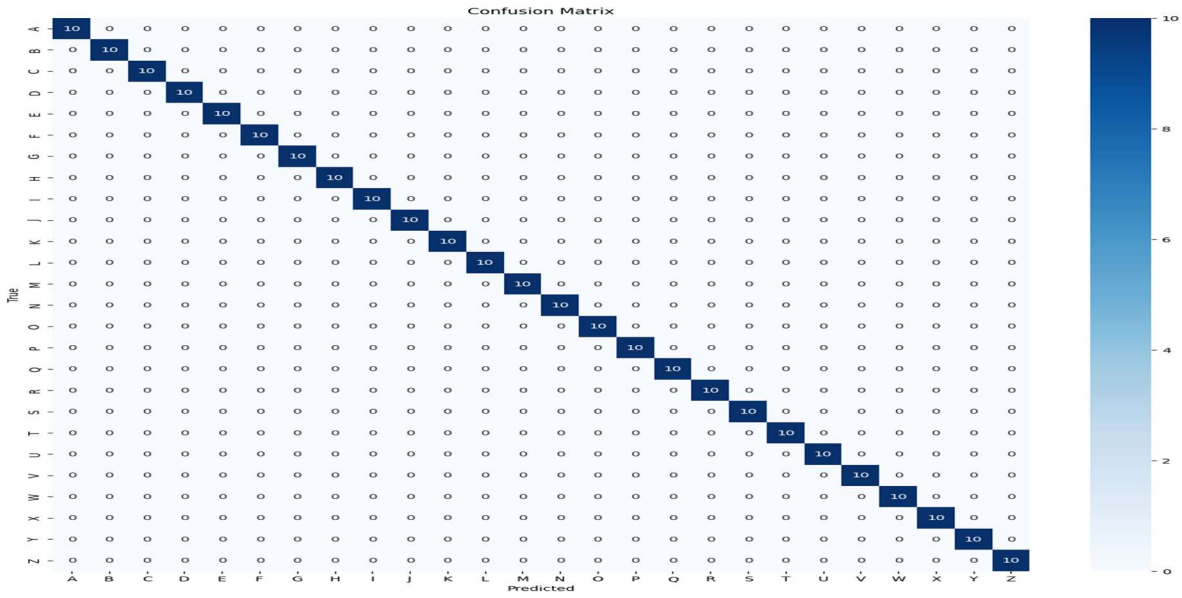


FIGURE : 9.2 Confusion Matrix for Classification of ISL Alphabets

9.2.1 Model Robustness:

Several things explain why the model is currently doing well

1. Feature Quality: MediaPipe hand landmark features can extract discriminative spatial information required to differentiate between various sign gestures.

2. Preprocessing Effectiveness: Normalization and feature standardization methods were effective in pre-processing data for effective training of the model.

3. Architecture Appropriateness: Complex neural network architecture with dropout regularization was extremely suitable for this classification problem without necessarily employing more advanced architectures such as CNNs or LSTMs.

4. Training Methodology: Overfitting was prevented by not employing early stopping through the technique of halting training when validation performance ceased to improve.

The model was successfully exported to TensorFlow Lite format for use in a mobile application environment. The end-to-end MediaPipe landmark to model prediction pipeline through a Flask server proves the global applicability of the system to real-time sign language recognition.

The optimal confusion matrix indicates that the model is trustworthy with its high precision when deployed and thus very trustworthy for real-world use in sign language interpretation.

9.3 Inference Performance:

In addition to its practical use in classification performance, a sign language recognition system also possesses inference speed and responsiveness in real-time deployment. For the evaluation of the deployability of the model in a real-world deployment, inference performance was evaluated over server-based (online) as well as on-device (offline) environments.

9.3.1 Online Inference (Flask + Cloudflare Tunnel):

In the web deployment setting, the model was served on a Flask server and made available via Cloudflare Tunnel. In this setting, the Android application sends a frame to the server, and the server returns the predicted alphabet. The round-trip inference time was said to be approximately 600–800 milliseconds per frame in steady Wi-Fi setting.

Average Server Response Time: ~700 ms
Server Load Handling: Strong with moderate traffic (5–10 requests/second)
Use Case Suitability: Appropriate for local area network (LAN) installation, classrooms, and

demonstration installations

9.3.2 Offline Inference (TFLite on Android):

Offline deployment employed the PyTorch-trained YOLOv8 classification model preinstalled as TensorFlow Lite, which executed natively on the Android device. Offline frame-by-frame gesture detection without an internet connection is feasible in this deployment, providing far quicker and higher privacy.

9.3.3 Comparison Summary:

Average Inference Time: ~180–250 milliseconds per frame
Performance Consistency: Demonstrated regular real-time execution even on low-end Android smartphones (4 GB RAM)
Model Size (TFLite): ~1.3 MB
Mobile App Size (with model): ~18 MB
Offline mode includes an enormous benefit regarding responsiveness, mobility, and resiliency and is therefore exceptionally well positioned to work in remote or low-connectivity settings.

Metric	**Online Mode (Server)**	**Offline Mode (TFLite)**
Avg. Inference Time	600–800 ms	180–250 ms
Internet Required	Yes	No
Model Size	~3.7 MB	~1.3 MB
Deployment Target	Web Server	Android App
Suitable Environment	Classrooms, LAN	Remote, Daily Use

Table 9.3.3 Comparison Summary

This performance comparison guarantees the system is adequately configured for real-time operation. Having online and offline deployment modes ensures the solution is usable, scalable, and accessible in multiple environments and connectivity states.

CHAPTER-10
CONCLUSION

Communication is an essential human need, and for deaf or speech-impaired individuals, Indian Sign Language (ISL) serves as a necessary bridge connecting humans to convey ideas, feelings, and sophisticated concepts. Sign language, though being absolutely essential, is not well-known or not well-understood by the masses, and thus it is causing daily challenges in accessing higher education, medical care, jobs, and social interaction. This work bridges this knowledge gap with the development and design of a real-time ISL-to-text and speech translation system for designing barrier-free and inclusive communication.

The system demonstrated leverages state-of-the-art deep learning techniques for translating static ISL gestures, i.e., the 26 English alphabet signs, into words and speech. Using Ultralytic's YOLOv8 classification model trained on a tailored ISL gesture dataset, the system provides high-accuracy gesture recognition. Additionally, the model was exported to TensorFlow Lite (TFLite) format to provide effective and real-time inference on low-processing-power smartphones.

Maybe the highlight of the system is that it can deploy both online and offline. It can be deployed online with Flask server exposed publicly via Cloudflare Tunnel, which proves useful for use cases with good internet. Offline mode, on the other hand, employs a native TFLite model in an Android app to keep the system running even in remote or low-connectivity areas. This renders the system applicable to be used within a wide range of real environments.

In addition to enhancing user experience and accessibility, the system includes the Swaram Text-to-Speech (TTS) API that provides natural-sounding speech outputs in a range of Indian and international languages. This serves to facilitate India's linguistic diversity and enable communication to become accessible and culturally responsive. It also benefits people with different levels of literacy through an audible alternative to textual output.

By its real-world application, mobile accessibility, and multi-language support, this project showcases the feasibility of integrating artificial intelligence with assistive technology to

address everyday challenges. It enables hearing and speech disabled users to communicate freely, thereby advancing autonomy, self-esteem, and participation. It can also be used in schools to study sign language, hospitals to enhance doctor–patient communication, or in customer service counters to make customer services accessible.

The worth of this endeavor goes beyond technological progress—it is also a relief to the greater cause of digital equity and inclusion. In a time where reliance on digital communication is picking up speed at break-neck pace, such solutions guarantee that all will not be left behind.
By reducing the communication barriers, the system facilitates deaf professional skill acquisition, socialization, and equal access to services.

Future Scope:
There are some improvements in the future to be achieved by making the system more usable and efficient. First, dynamic gesture recognition capability can be added. Although the current project utilizes static alphabet gestures, complete ISL communication also depends on dynamic hand movement and gesture chaining. Adding temporal models such as LSTM or 3D CNNs can be beneficial in recognizing word and sentence through continuous signing.

Second, the system translates from ISL to speech and text but the reverse pipeline may be added. This would permit typed or verbal input from a non-signer to be translated into sign language through animated avatars or video, making communication fully bidirectional.

Third, incorporating regional sign variations support and multimodal inputs (voice, text, touch gestures) would most likely make the system even more universal. Cloud storage integration can also be used to monitor user preferences or usage history to personalize the system.

Lastly, as the technology advances, it can be made available as a cloud API or incorporated in public kiosks, government websites, and customer service robots to further ease its utilization countrywide.

REFERENCES

[1] H. Garg, P. Dubey, S. Gupta, and R. Jain, "Real-Time Conversion for Sign-to-Text and Text-to-Speech Communication Using Machine Learning," *Proc. Int. Conf. Artif. Intell. Appl.*, pp. 85–96, Mar. 2024. Singapore: Springer Nature Singapore. DOI: 10.1007/978-981-97-8074-7_7.

[2] S. Sindhu, P. Reddy, and V. Kumar, "Dynamic Gesture Recognition System for Real-Time ISL Translation," *J. Artif. Intell. Appl.*, vol. 12, no. 2, pp. 45–56, 2024.

[3] R. Langote, T. Sharma, and K. Mehta, "Fingerspelling-to-Text System with Sentiment Analysis Using RNNs," *Int. J. Comput. Sci. Inf. Technol.*, vol. 16, no. 3, pp. 112–121, 2024.

[4] M. Hegde, R. Patil, and N. Rao, "Smart Translation System for ISL Using CNN and TTS," *Proc. IEEE Conf. Human-Computer Interact.*, vol. 29, no. 1, pp. 210–220, 2024.

[5] A. Damdoo and R. Kumar, "Integrative Survey on ISL Recognition and Translation Techniques: Deep Learning Approaches," *J. Intell. Syst. Appl.*, vol. 18, no. 1, pp. 35–48, 2025.

[6] P. Sharma, R. Verma, and A. Singh, "Speech-to-ISL Translation System Using NLP Techniques," *Int. J. Artif. Intell. Appl.*, vol. 15, no. 4, pp. 120–135, 2022.

[7] S. Grover, K. Patel, and M. Rao, "Comprehensive Review of Sign Language Translation Systems," *J. Image Process. Mach. Learn.*, vol. 8, no. 2, pp. 45–60, 2021.

[8] D. Sharma and R. Kumar, "Sign-to-Speech Translation Using CNN and TTS," *Proc. IEEE Conf. Assistive Technol.*, vol. 29, no. 1, pp. 210–220, 2020.

[9] R. Akshatharani and M. Manjanaik, "Real-Time ISL Translator Using MediaPipe and CNN-LSTM," *Int. J. Comput. Vis.*, vol. 12, no. 3, pp. 98–112, 2021

[10] L. Bharathi and P. Sridhar, "Signtalk: Real-Time Sign-to-Text and Speech Conversion," *Proc. Int. Conf. Human-Computer Interact.*, vol. 34, no. 2, pp. 67–79, 2021.

[11] M. Tiku and K. Reddy, "Real-Time Sign-to-Text and Speech Conversion Using Deep Learning," *J. Intell. Syst. Appl.*, vol. 18, no. 1, pp. 35–48, 2020.

[12] R. Sheela and K. Dinesh, "ISL Translator with Real-Time Processing for Healthcare and Education," *Int. J. Comput. Sci. Inf. Technol.*, vol. 16, no. 3, pp. 112–121, 2022.

[13] A. Kumar and S. Rao, "Gesture-to-Text and Speech Translation Using Deep Learning and NLG," *J. Signal Image Process.*, vol. 19, no. 2, pp. 78–92, 2022.

[14] Y. S. N. Rao et al., "Dynamic Sign Language Recognition and Translation Through Deep Learning: A Systematic Literature Review," *J. Theor. Appl. Inf. Technol.*, vol. 102, no. 21,

2024.

[15] P. Singh et al., "Development of Sign Language Translator for Disable People in Two-Ways Communication," in *Proc. 2023 1st Int. Conf. Circuits, Power Intell. Syst. (CCPIS)*, Sep. 2023, pp. 1–6.

[16] P. K. Saw et al., "Gesture Recognition in Sign Language Translation: A Deep Learning Approach," in *Proc. 2024 Int. Conf. Integrated Circuits, Commun. Comput. Syst. (ICIC3S)*, vol. 1, Jun. 2024, pp. 1–7.

[17] A. Poojary, M. Variar, R. Radhakrishnan, and G. Hegde, "Indian Sign Language Translation For Hard-Of-Hearing And Hard-Of-Speaking Community," 2022.

[18] S. S. Kumar et al., "Time series neural networks for real time sign language translation," in *Proc. 2018 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2018, pp. 243–248

[19] A. Kamble et al., "Conversion of Sign Language to Text," *Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET)*, vol. 11, no. 5, 2023.

[20] C. O. Kumar et al., "Real time detection and conversion of gestures to text and speech to sign system," in *Proc. 2022 3rd Int. Conf. Electron. Sustain. Commun. Syst. (ICESC)*, Aug. 2022, pp. 73–78.

APPENDIX-A

PSUEDOCODE

predict.py

```
# inference.py

import numpy as np

import joblib

import tensorflow as tf



# Load label encoder used during training
label_encoder = joblib.load(r"E:\project\backend\model\label_encoder.pkl")

# Load the TFLite model once
interpreter = tf.lite.Interpreter(model_path =
r"E:\project\backend\model\sign_language_model.tflite")
interpreter.allocate_tensors()

def predict_letter(landmarks: list[float]) -> str:
    if len(landmarks) != 126:
        # Pad to 126 if one hand is missing
        landmarks += [0.0] * (126 - len(landmarks))

input_data = np.array([landmarks], dtype=np.float32)
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
output = interpreter.get_tensor(output_details[0]['index'])

predicted_index = int(np.argmax(output))
predicted_label = label_encoder.inverse_transform([predicted_index])[0]

print(f"Received Landmarks: {landmarks}")  # Debug in terminal
print(f"Predicted Index: {predicted_index}, Label: {predicted_label}")

return predicted_label
```

app.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import base64
from io import BytesIO
```

```
from PIL import Image
import numpy as np
from utils.predict import predict_letter

app = Flask(__name__)
CORS(app)

@app.route("/", methods=["GET"])
def home():
    return "Flask is running"
@app.route("/predict", methods=["POST"])
def predict():
    data = request.get_json()

    if not data or "landmarks" not in data:
        return jsonify({"error": "No landmarks provided"}), 400

    try:
        landmarks = data["landmarks"]
        prediction = predict_letter(landmarks)
        return jsonify({"prediction": prediction})
    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)

#creating data
import cv2
import mediapipe as mp
import pandas as pd
import time
import os

# Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=2,
min_detection_confidence=0.7)
mp_drawing = mp.solutions.drawing_utils

# Create folder for data if not exists
if not os.path.exists("sign_data"):
    os.makedirs("sign_data")

# Get the label from user
label = input("Enter the letter you're signing (A-Z): ").upper()
if not label.isalpha() or len(label) != 1:
    print("Invalid input. Please enter a single alphabet letter.")
    exit()

# Data storage
```

```
landmark_data = []
total_samples = 50
captured_samples = 0
last_capture_time = 0
# Start webcam
cap = cv2.VideoCapture(0)
print("Starting webcam...")

while cap.isOpened() and captured_samples < total_samples:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(rgb)

    h, w, _ = frame.shape
    landmarks = []

# Extract landmarks for both hands
    if results.multi_hand_landmarks:
        hands_present = len(results.multi_hand_landmarks)
        for hand_landmarks in results.multi_hand_landmarks[:2]:
            for lm in hand_landmarks.landmark:
                landmarks.extend([lm.x, lm.y, lm.z])
        # Pad with zeros if only one hand detected
        if hands_present == 1:
            landmarks.extend([0.0] * (21 * 3))
    else:
        # No hands detected → All zeros
        landmarks = [0.0] * (42 * 3)

# Countdown and save logic
    time_elapsed = time.time() - last_capture_time
    countdown = int(3 - time_elapsed)
    if time_elapsed >= 3 and landmarks.count(0.0) < 63:  # Ensure at least one hand is detected
        landmark_data.append([label] + landmarks)
        captured_samples += 1
        last_capture_time = time.time()

# Draw landmarks
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
 mp_drawing.draw_landmarks(frame,hand_landmarks,mp_hands.HAND_CONNECTIONS)

# Overlay text
    cv2.putText(frame, f"Letter: {label}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255, 255, 0), 2)
    cv2.putText(frame, f"Captured: {captured_samples}/{total_samples}", (10, 70),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
```

```
    if countdown > 0:
        cv2.putText(frame, f"Next in: {countdown}", (10, 110), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
    else:
        cv2.putText(frame, "Capturing...", (10, 110), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 128, 255), 2)

    if landmarks.count(0.0) == 63:
        cv2.putText(frame, "No hand detected!", (10, 150), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    cv2.imshow("Sign Language Data Collection", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Cleanup
cap.release()
cv2.destroyAllWindows()

# Save to CSV
save_dir = r"E:\project\creating_data\data"
os.makedirs(save_dir, exist_ok=True)

# Build full file path
save_path = f"{save_dir}/{label}_sign_data.csv"

# Save to CSV
df = pd.DataFrame(landmark_data)
df.to_csv(save_path, index=False, header=False)
print(f"Saved {captured_samples} samples for letter '{label}' to {save_path}")

merge.py
import pandas as pd
import os

csv_folder = r"E:\project\creating_data\sign_data"
merged_data = []
for file in os.listdir(csv_folder):
    if file.endswith(".csv"):
        file_path = os.path.join(csv_folder, file)
        df = pd.read_csv(file_path, header=None)
        merged_data.append(df)

merged_df = pd.concat(merged_data, ignore_index=True)
merged_df.to_csv("merged_sign_data.csv", index=False, header=False)
print("merged")
```

Fronted App:

```
MainActivity.kt
package com.example.myapplication
```

```
import android.Manifest
import android.content.pm.PackageManager
import android.os.Bundle
import android.util.Log
import android.widget.Button
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.camera.core.*
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.camera.view.PreviewView
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.google.mediapipe.tasks.core.BaseOptions
import com.google.mediapipe.tasks.vision.handlandmarker.HandLandmarker
import
com.google.mediapipe.tasks.vision.handlandmarker.HandLandmarker.HandLandmarkerOpti
ons
import com.google.mediapipe.tasks.vision.handlandmarker.HandLandmarkerResult
import com.google.mediapipe.framework.image.BitmapImageBuilder
import okhttp3.*
import okhttp3.MediaType.Companion.toMediaType
import okhttp3.RequestBody.Companion.toRequestBody
import org.json.JSONArray
import org.json.JSONObject
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors

class MainActivity : AppCompatActivity() {
    private lateinit var cameraExecutor: ExecutorService
    private lateinit var previewView: PreviewView
    private lateinit var textViewResult: TextView
    private lateinit var handLandmarker: HandLandmarker
    private lateinit var spaceButton: Button
    private lateinit var clearButton: Button
    private lateinit var speakButton: Button  // New button for speaking text

    // Text-to-speech manager
    private lateinit var ttsManager: TextToSpeechManager

    private var lastSentTime = 0L
    private val resultBuilder = StringBuilder()

    companion object {
        const val CAMERA_PERMISSION_CODE = 1001
        const val TAG = "MainActivity"
    }
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

```
    previewView = findViewById(R.id.previewView)
    textViewResult = findViewById(R.id.resultText)
    spaceButton = findViewById(R.id.spaceButton)
    clearButton = findViewById(R.id.clearButton)
    speakButton = findViewById(R.id.speakButton)

    cameraExecutor = Executors.newSingleThreadExecutor()

    ttsManager = TextToSpeechManager(this)

    spaceButton.setOnClickListener {
        resultBuilder.append(" ")
        textViewResult.text = resultBuilder.toString()
    }

    clearButton.setOnClickListener {
        resultBuilder.clear()
        textViewResult.text = ""
        ttsManager.stop()
    }

    speakButton.setOnClickListener {
        val textToSpeak = resultBuilder.toString()
        if (textToSpeak.isNotEmpty()) {
            ttsManager.speak(textToSpeak)
        } else {
            Toast.makeText(this, "No text to speak", Toast.LENGTH_SHORT).show()
        }
    }

    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
        == PackageManager.PERMISSION_GRANTED) {
        initLandmarker()
        startCamera()
    } else {
        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.CAMERA),
            CAMERA_PERMISSION_CODE
        )
    }
}

private fun initLandmarker() {
    try {
        val baseOptions = BaseOptions.builder()
            .setModelAssetPath("hand_landmarker.task")
            .build()

        val options = HandLandmarkerOptions.builder()
```

```
            .setBaseOptions(baseOptions)
            .setMinHandDetectionConfidence(0.5f)
            .setMinTrackingConfidence(0.5f)
            .setMinHandPresenceConfidence(0.5f)
            .setNumHands(2)
            .build()

        handLandmarker = HandLandmarker.createFromOptions(this, options)
        Log.d(TAG, "HandLandmarker initialized successfully")
      } catch (e: Exception) {
        Log.e(TAG, "Error initializing HandLandmarker: ${e.message}", e)
        Toast.makeText(this, "Failed to initialize hand tracking: ${e.message}",
Toast.LENGTH_LONG).show()
      }
    }

    private fun startCamera() {
      val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

      cameraProviderFuture.addListener({
        val cameraProvider = cameraProviderFuture.get()

        val preview = Preview.Builder().build().also {
          it.setSurfaceProvider(previewView.surfaceProvider)
        }

        val imageAnalyzer = ImageAnalysis.Builder()
          .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
          .build()
          .also {
            it.setAnalyzer(cameraExecutor) { image -> analyzeImage(image) }
          }

        try {
          cameraProvider.unbindAll()
          cameraProvider.bindToLifecycle(this,
CameraSelector.DEFAULT_FRONT_CAMERA, preview, imageAnalyzer)
        } catch (e: Exception) {
          Log.e(TAG, "Failed to bind camera use cases", e)
        }
      }, ContextCompat.getMainExecutor(this))
    }

    private fun analyzeImage(image: ImageProxy) {
      val currentTime = System.currentTimeMillis()
      if (currentTime - lastSentTime > 1500) {
        val bitmap = image.toBitmap()
        if (bitmap == null) {
          Log.e(TAG, "Bitmap conversion failed — skipping frame")
          image.close()
          return
```

```
            }

            val mpImage = BitmapImageBuilder(bitmap).build()
            val result = handLandmarker.detect(mpImage)

            if (result.landmarks().isEmpty()) {
                Log.d(TAG, "No hand detected — skipping frame")
                image.close()
                return
            }

            val landmarksArray = extractLandmarks(result)
            Log.d(TAG, "Extracted landmarks preview: ${landmarksArray.take(10).joinToString(", ")}")

            sendLandmarksToBackend(landmarksArray)
            lastSentTime = currentTime
        }
        image.close()
    }

    private fun extractLandmarks(result: HandLandmarkerResult): FloatArray {
        val totalLandmarks = 2 * 21 * 3
        val landmarkArray = FloatArray(totalLandmarks) { 0f }

        result.landmarks().take(2).forEachIndexed { handIndex, handLandmarks ->
            val offset = handIndex * 63
            handLandmarks.forEachIndexed { i, landmark ->
                landmarkArray[offset + i * 3] = landmark.x()
                landmarkArray[offset + i * 3 + 1] = landmark.y()
                landmarkArray[offset + i * 3 + 2] = landmark.z()
            }
        }

        return landmarkArray
    }

    private fun sendLandmarksToBackend(landmarks: FloatArray) {
        val client = OkHttpClient()
        val json = JSONObject().apply {
            put("landmarks", JSONArray(landmarks.toList()))
        }

        val requestBody = json.toString().toRequestBody("application/json; charset=utf-8".toMediaType())
        val request = Request.Builder()
            .url("YOUR_CLOUDFLARE_URL/predict")
            .post(requestBody)
            .build()

        client.newCall(request).enqueue(object : Callback {
```

```
            override fun onFailure(call: Call, e: java.io.IOException) {
                Log.e(TAG, "Network error", e)
                runOnUiThread {
                    textViewResult.text = "Network Error: ${e.message}"
                }
            }

            override fun onResponse(call: Call, response: Response) {
                val responseData = response.body?.string()
                Log.d(TAG, "Backend raw response: $responseData")

                runOnUiThread {
                    try {
                        val jsonObject = JSONObject(responseData ?: "{}")
                        val prediction = jsonObject.optString("prediction", "No Hand")

                        if (prediction != "No Hand" && prediction != "_") {
                            Log.d(TAG, "Predicted: $prediction")
                            resultBuilder.append(prediction)
                            textViewResult.text = resultBuilder.toString()
                        } else {
                            Log.d(TAG, "Prediction skipped: $prediction")
                        }
                    } catch (e: Exception) {
                        textViewResult.text = "Parsing Error: ${e.message}"
                        Log.e(TAG, "JSON Parsing Error", e)
                    }
                }
            }
        })
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out
String>, grantResults: IntArray) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (requestCode == CAMERA_PERMISSION_CODE && grantResults.isNotEmpty()
&& grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            initLandmarker()
            startCamera()
        } else {
            Toast.makeText(this, "Camera permission denied", Toast.LENGTH_SHORT).show()
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        ttsManager.shutdown()
        cameraExecutor.shutdown()
    }
}
```

```
fun ImageProxy.toBitmap(): android.graphics.Bitmap? {
    return try {
        val yBuffer = planes[0].buffer
        val uBuffer = planes[1].buffer
        val vBuffer = planes[2].buffer

        val ySize = yBuffer.remaining()
        val uSize = uBuffer.remaining()
        val vSize = vBuffer.remaining()

        val nv21 = ByteArray(ySize + uSize + vSize)

        yBuffer.get(nv21, 0, ySize)
        vBuffer.get(nv21, ySize, vSize)
        uBuffer.get(nv21, ySize + vSize, uSize)

        val yuvImage = android.graphics.YuvImage(nv21,
android.graphics.ImageFormat.NV21, width, height, null)
        val out = java.io.ByteArrayOutputStream()
        yuvImage.compressToJpeg(android.graphics.Rect(0, 0, width, height), 100, out)
        val imageBytes = out.toByteArray()

        android.graphics.BitmapFactory.decodeByteArray(imageBytes, 0, imageBytes.size)
    } catch (e: Exception) {
        Log.e("toBitmap", "Conversion failed: ${e.message}")
        null
    }
}
```

speechtotext.ky

```
package com.example.myapplication

import android.content.Context
import android.speech.tts.TextToSpeech
import android.util.Log
import java.util.Locale

/**
 * TextToSpeechManager is a utility class to handle text-to-speech functionality
 * in the sign language converter application.
 */
class TextToSpeechManager(
    private val context: Context
) : TextToSpeech.OnInitListener {

    companion object {
        private const val TAG = "TextToSpeechManager"
    }

    private var textToSpeech: TextToSpeech? = null
    private var isInitialized = false
```

```
    init {
        textToSpeech = TextToSpeech(context, this)
    }

    override fun onInit(status: Int) {
        if (status == TextToSpeech.SUCCESS) {
            val result = textToSpeech?.setLanguage(Locale.getDefault())
            if (result == TextToSpeech.LANG_MISSING_DATA || result ==
TextToSpeech.LANG_NOT_SUPPORTED) {
                Log.e(TAG, "Language not supported")
            } else {
                isInitialized = true
                Log.d(TAG, "TextToSpeech initialized successfully")
            }
        } else {
            Log.e(TAG, "TextToSpeech initialization failed")
        }
    }

    /**
     * Speaks the provided text
     * @param text The text to be converted to speech
     * @param queueMode How to handle existing speech requests (QUEUE_FLUSH or
QUEUE_ADD)
     */
    fun speak(text: String, queueMode: Int = TextToSpeech.QUEUE_FLUSH) {
        if (isInitialized && text.isNotEmpty()) {
            textToSpeech?.speak(text, queueMode, null, null)
        } else if (!isInitialized) {
            Log.e(TAG, "TextToSpeech not initialized")
        }
    }

    /**
     * Stop any current speech
     */
    fun stop() {
        if (isInitialized) {
            textToSpeech?.stop()
        }
    }

    /**
     * Release resources when no longer needed
     */
    fun shutdown() {
        textToSpeech?.stop()
        textToSpeech?.shutdown()
        textToSpeech = null
        isInitialized = false
```

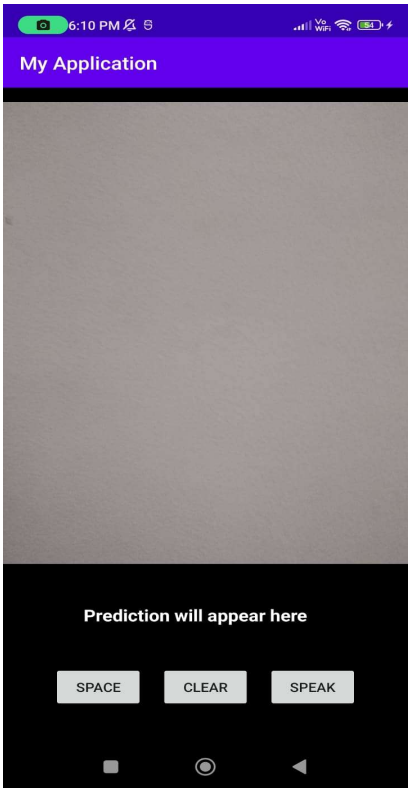
```
    }
}
```

MainActivity.xml

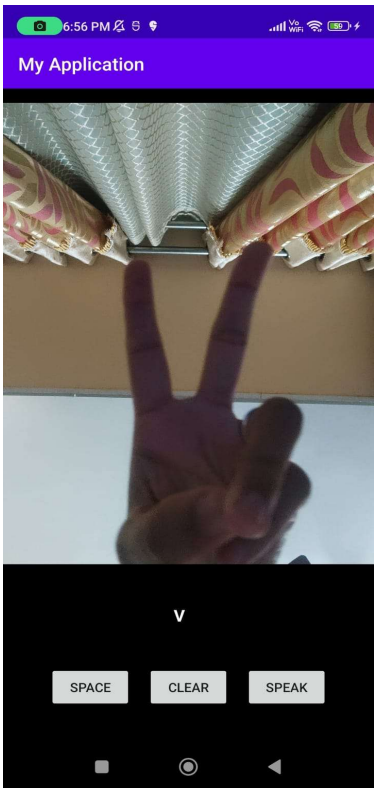
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.INTERNET" />

    <uses-feature android:name="android.hardware.camera.any" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication"
        tools:targetApi="31">


        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.MyApplication">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

APPENDIX-B

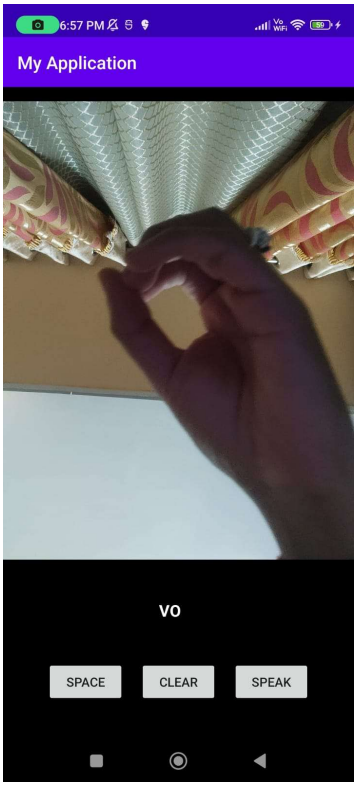
SCREENSHOTS



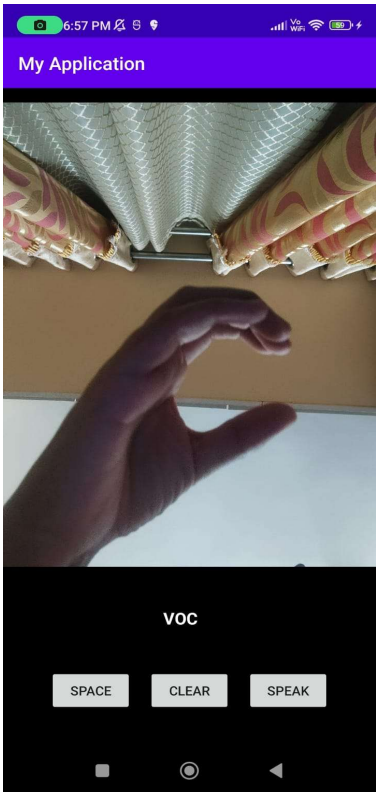
UI of the app



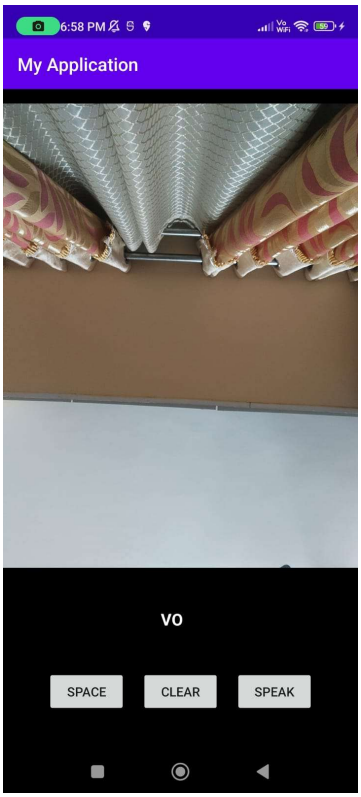
Prediction of letter “V”



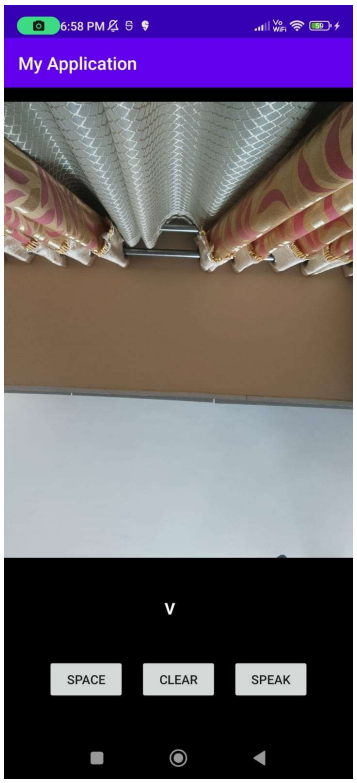
Prediction letter of “O”



Prediction letter of “C”



Clear button invoked to clear a letter “C”



Clear button invoked to clear a letter “O”

APPENDIX-C

ENCLOSURES

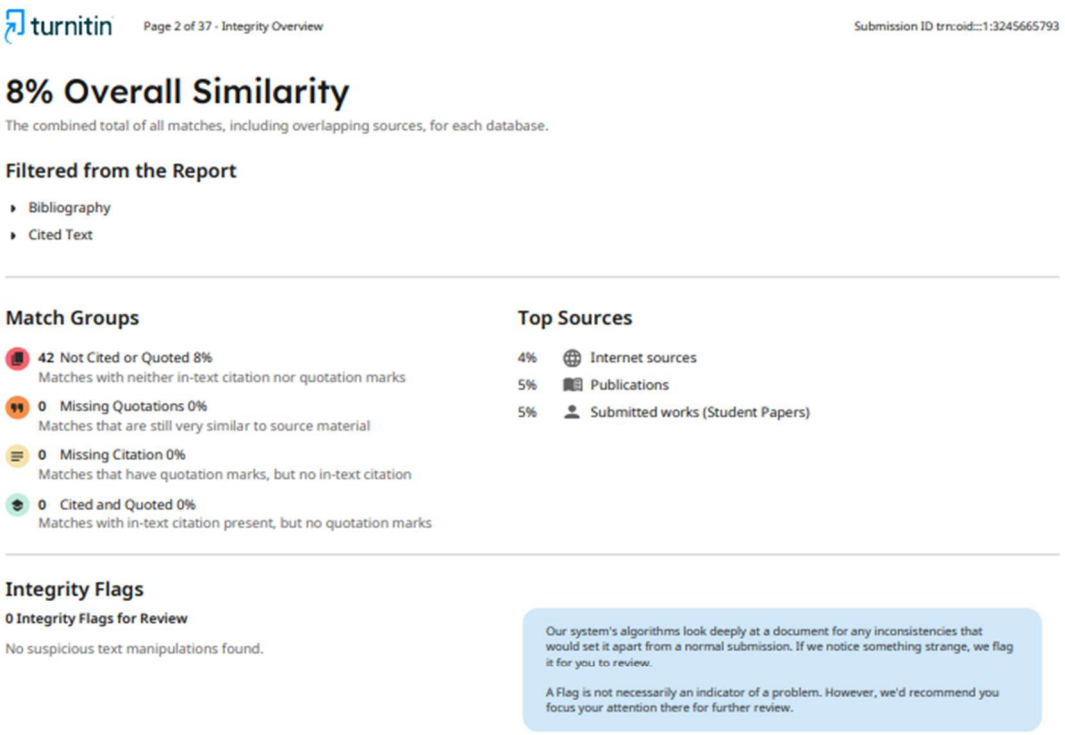
1. Journal publication/Conference Paper Presented Certificates of all students.


iJRASET
ISSN No. : 2321-9653
International Journal for Research in Applied Science & Engineering Technology
IJRASET is indexed with Crossref for DOI-DOI : 10.22214
Website : www.ijraset.com, E-mail : ijraset@gmail.com
ISRA Journal Impact Factor: 7.429
45.98 INDEX COPERNICUS
THOMSON REUTERS
doi cross ref
TOGETHER WE REACH THE GOAL
SJIF 7.429
Certificate
It is here by certified that the paper ID : IJRASET70817, entitled
Indian Sign Language to Text/Speech Translation
by
Chavva Rajeswari
after review is found suitable and has been published in
Volume 13, Issue V, May 2025
in
International Journal for Research in Applied Science & Engineering Technology
(International Peer Reviewed and Refereed Journal)
Good luck for your future endeavors
Editor in Chief, iJRASET


iJRASET
ISSN No. : 2321-9653
International Journal for Research in Applied Science & Engineering Technology
IJRASET is indexed with Crossref for DOI-DOI : 10.22214
Website : www.ijraset.com, E-mail : ijraset@gmail.com
ISRA Journal Impact Factor: 7.429
45.98 INDEX COPERNICUS
THOMSON REUTERS
doi cross ref
TOGETHER WE REACH THE GOAL
SJIF 7.429
Certificate
It is here by certified that the paper ID : IJRASET70817, entitled
Indian Sign Language to Text/Speech Translation
by
R Manisha
after review is found suitable and has been published in
Volume 13, Issue V, May 2025
in
International Journal for Research in Applied Science & Engineering Technology
(International Peer Reviewed and Refereed Journal)
Good luck for your future endeavors
Editor in Chief, iJRASET


iJRASET
ISSN No. : 2321-9653
International Journal for Research in Applied Science & Engineering Technology
IJRASET is indexed with Crossref for DOI-DOI : 10.22214
Website : www.ijraset.com, E-mail : ijraset@gmail.com
ISRA Journal Impact Factor: 7.429
45.98 INDEX COPERNICUS
THOMSON REUTERS
doi cross ref
TOGETHER WE REACH THE GOAL
SJIF 7.429
Certificate
It is here by certified that the paper ID : IJRASET70817, entitled
Indian Sign Language to Text/Speech Translation
by
P Durga Prasheena
after review is found suitable and has been published in
Volume 13, Issue V, May 2025
in
International Journal for Research in Applied Science & Engineering Technology
(International Peer Reviewed and Refereed Journal)
Good luck for your future endeavors
Editor in Chief, iJRASET

2. Similarity Index / Plagiarism Check report clearly showing the Percentage (%). No need for a page-wise explanation.



turnitin Page 2 of 37 - Integrity Overview Submission ID trn:oid:::1:3245665793

8% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Cited Text

Match Groups

- 42 Not Cited or Quoted 8%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 4% Internet sources
- 5% Publications
- 5% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

3. Details of mapping the project with the Sustainable Development Goals (SDGs).

This project aligns with **SDGs 4 and 8** by promoting **inclusive education and economic empowerment** for individuals with hearing impairments. Under **SDG 4 (Quality Education)**, it enhances accessibility in learning environments by converting Indian Sign Language (ISL) into text and speech, enabling seamless communication in classrooms and digital education platforms. Simultaneously, it supports **SDG 8 (Decent Work and Economic Growth)** by fostering workplace inclusion, expanding job opportunities, and enabling professional interactions for individuals with hearing disabilities. By integrating AI-driven translation, the project ensures equal access to education and employment, contributing to a more inclusive and sustainable society.