Sign Language Recognition Using Deep Learning

Sayali Vasantrao Gund, Shridevi Amol Nandi

Department of Computer Science & Engineering, V.V.P. Institute of Engineering & Technology, Solapur

DOI: https://doi.org/10.51244/IJRSI.2026.1306000452

Received: 04 July 2026; Accepted: 09 July 2026; Published: 17 July 2026

INTRODUCTION

Sign language is an essential means of communication for individuals who are speech- or hearing-impaired. It uses structured hand gestures, movements, and facial expressions to convey meaning. However, most people are not familiar with sign language, which creates a significant communication barrier between speech-impaired individuals and the rest of society.

With advancements in Artificial Intelligence and Deep Learning, it has become possible to automatically recognize hand gestures and translate them into understandable text or speech. Sign language recognition systems aim to bridge this communication gap by converting hand gestures into natural language in real time.

This project focuses on developing a real-time sign language recognition system using deep learning techniques. The system captures live hand gestures using a smartphone or webcam and converts them into corresponding text and speech outputs. By enabling real-time processing, the system supports faster and more intuitive communication.

Traditional gesture recognition systems rely on static images or offline video datasets, which are ineffective for real-time gesture interpretation. These systems often fail to capture the temporal dynamics of gestures. To overcome these challenges, the proposed system integrates Media Pipe for real-time hand landmark detection and a LSTM hybrid deep learning model for gesture classification. The LSTM models temporal dependencies across gesture sequences.

Background of the Study

Human communication is not limited to spoken language. For individuals with hearing or speech impairments, sign language plays a vital role in expressing thoughts, emotions, and intentions. Despite its importance, sign language remains inaccessible to most people, limiting social interaction and inclusion.

Earlier sign language recognition systems depended on wearable sensors or rule-based image processing techniques. These approaches were intrusive, expensive, and inaccurate. With the emergence of computer vision and deep learning, camera-based gesture recognition systems have gained popularity.

Recent advancements such as Media Pipe enable efficient, real-time hand landmark extraction even on mobile devices. Deep learning architectures such as LSTMs have proven effective in modeling complex spatial and temporal patterns. Combining these technologies enables the development of accurate and lightweight sign language recognition systems suitable for real-world deployment.

2.2 Motivation:

The motivation for this project arises from the following factors:

- Growing need for inclusive communication technologies
- Communication challenges faced by speech- and hearing-impaired individuals
- Limitations of traditional offline gesture recognition systems

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

- Advancements in real-time hand tracking using MediaPipe
- Availability of deep learning models for sequence learning
- Demand for mobile-friendly AI applications

These factors emphasize the need for a real-time, accurate, and mobile-deployable sign language recognition system.

Problem Statement

Existing Systems

Most existing sign language recognition systems operate on offline datasets and rely on pre-recorded images or videos. Many systems analyze individual frames without considering gesture motion, resulting in poor recognition of dynamic gestures. Additionally, several deep learning models are computationally expensive and unsuitable for mobile deployment.

Problem Statement

To design and implement a real-time sign language recognition system that accurately identifies hand gestures using MediaPipe hand landmarks and a LSTM based deep learning model, and converts the recognized gestures into text and speech on mobile devices.

Objectives of the Project

The main objectives of this project are:

- To capture hand gestures in real time using a camera
- To extract hand landmarks using MediaPipe
- To develop a LSTM based deep learning model
- To classify dynamic hand gestures accurately
- To convert recognized gestures into text and speech
- To deploy the system on mobile devices using TensorFlow Lite

Scope of the Project

The scope of the project includes:

- Real-time hand gesture recognition
- Offline processing on mobile devices
- Recognition of predefined sign gestures
- Text and speech output generation
- Academic and experimental usage

The system does not support full sentence-level sign language interpretation in its current version but provides a strong foundation for future enhancements.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

LITERATURE REVIEW

Overview of Sign Language Recognition Systems:

Sign language recognition systems aim to automatically interpret hand gestures and translate them into meaningful text or speech. These systems generally follow a computer vision–based pipeline that includes gesture acquisition, preprocessing, feature extraction, and classification. With the evolution of Artificial Intelligence and Deep Learning, modern systems increasingly rely on neural network architectures to improve recognition accuracy and robustness.

Early sign language recognition approaches used wearable sensor-based gloves to capture hand movements. Although these systems provided precise motion data, they were expensive, intrusive, and unsuitable for real-world usage. Later, vision-based systems using cameras were introduced, but they relied heavily on handcrafted features and traditional image processing techniques, resulting in limited performance under varying conditions.

Recent advancements have shifted focus toward deep learning–based methods using Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks. LSTMs are highly effective in modeling temporal dependencies across gesture sequences, making them well-suited for dynamic sign language recognition. By analyzing sequences of hand landmarks over time, LSTM networks can learn motion patterns and improve classification accuracy. Frameworks such as MediaPipe have further enhanced real-time gesture recognition by providing efficient hand landmark extraction optimized for mobile and embedded platforms.

Despite these advancements, many existing systems remain limited to offline processing or require high computational resources, restricting their applicability in real-time and mobile environments.

Summarized Research Papers

1. Kumar & Rao (2022) – CNN-based gesture recognition on videos; accurate but not real-time on mobile.
2. Sharma et al. (2021) – LSTM for sequential gesture recognition; improves temporal modeling but heavy for deployment.
3. MediaPipe Developers (2020) – Hand landmark detection; real-time and mobile-friendly.
4. Li & Deng (2020) – Gesture recognition using hand keypoints; highlights importance of preprocessing landmarks.
5. Zhang & Liu (2019) – LSTM-based for action recognition; captures spatial-temporal features effectively.
6. Yoon & Chung (2018) – Sequence modeling in gesture/video classification; highlights temporal importance.
7. Tripathi & Beigi (2018) – Sequence modeling with LSTM; better gesture tracking over time.
8. Vaswani et al. (2017) – Transformers in sequence learning; high accuracy but computationally intensive.

Limitations of Previous Work

- Most systems are offline or heavy for mobile deployment.
- Many do not combine spatial and temporal features (LSTM).
- Real-time gesture prediction is rarely addressed.
- Lack of standardized dataset integration for mobile use.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

Research Gap

The review of existing literature reveals several research gaps that motivate the proposed sign language recognition system:

- Limited availability of **real-time sign language recognition systems** suitable for mobile devices
- Inadequate integration of **spatial and temporal feature modeling** using LSTM architectures
- Heavy computational models that are **unsuitable for lightweight mobile deployment**
- Minimal use of **hand landmark–based approaches** for efficient gesture representation
- Lack of systems that provide **instant text and speech output** for practical communication
- Insufficient focus on **offline, on-device processing** without reliance on cloud services

These gaps highlight the need for a **real-time, lightweight, and mobile-deployable sign language recognition system**. The proposed project addresses these challenges by integrating MediaPipe for real-time hand landmark extraction with a LSTM based deep learning model and deploying the system using TensorFlow Lite for efficient mobile execution.

System Analysis and Design

Existing System

Sign language and gesture recognition has been an active research domain for several years. Various systems have been proposed using traditional image processing, machine learning, and deep learning techniques. However, most existing approaches are developed as research prototypes and lack practical deployment architecture suitable for real-world communication environments.

Earlier systems primarily relied on **static image-based classification**, where a single captured frame is processed independently to identify a gesture. While these systems are simpler in design, they fail to capture the dynamic characteristics of sign language, where gesture meaning depends on motion continuity and temporal transitions.

Subsequent research introduced **offline video-based gesture recognition**, where pre-recorded gesture sequences are processed to improve temporal modeling. Although these approaches enhance recognition accuracy, they operate only on stored datasets and do not support real-time interaction.

In addition, many recent deep learning-based systems utilize computationally intensive architectures such as 3D-CNNs or transformer-based models. While these models achieve high accuracy, they require substantial hardware resources and are generally unsuitable for deployment on mobile devices or low-power systems.

Most existing implementations also lack:

- Proper client–server deployment architecture
- Real-time API-based communication
- Mobile-integrated inference mechanisms
- Scalable backend processing systems

Thus, despite significant academic progress, practical, scalable, and mobile-friendly real-time sign language recognition systems remain limited.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

Limitations of Existing System

Despite advancements in gesture recognition research, existing systems suffer from several significant limitations:

1. Unimodal and Frame-Based Gesture Analysis

Many systems analyze isolated frames without modeling temporal continuity. This unimodal processing approach ignores motion dynamics, which are essential in interpreting dynamic sign language gestures.

2. Lack of Real-Time Interaction

Most systems operate on pre-recorded image or video datasets. They do not support live camera input or continuous gesture recognition, limiting their use in interactive communication scenarios.

3. Absence of Client–Server Architecture

Traditional implementations often integrate camera processing and model inference in a single local system. This architecture reduces scalability and does not support centralized model management or API-based deployment.

4. High Computational Complexity

Deep learning models such as 3D-CNNs and transformer networks require high GPU computation and memory resources. Such models are unsuitable for mobile environments or lightweight devices.

5. Poor Mobile Compatibility

Many existing systems lack mobile application integration or require cloud-based inference, resulting in:

- Increased latency
- Internet dependency
- Reduced reliability

6. No Structured API Communication Layer

Existing research prototypes often do not implement RESTful APIs (such as FastAPI) for handling prediction requests. As a result, they are not scalable for real-world deployment or integration into multi-platform applications.

Proposed System

To overcome the limitations of existing approaches, the proposed system introduces a **real-time client–server architecture** integrating mobile-based gesture capture with backend deep learning inference through a FastAPI server.

The system consists of two primary components:

1. Mobile Application (Client)

- Captures live camera frames
- Sends captured frames to backend server via HTTP POST request
- Receives prediction results in JSON format

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

- Displays recognized gesture as text
- Optionally generates speech output

Python FastAPI Backend (Server)

- Receives image frames
- Performs preprocessing
- Extracts hand landmarks using MediaPipe
- Maintains sliding window sequence buffer
- Performs LSTM inference
- Returns predicted gesture and confidence score

Working Principle

The mobile application continuously captures frames and transmits them to the FastAPI backend. The backend processes the received image, extracts 21 hand landmarks using MediaPipe, normalizes the coordinates, and constructs a temporal sequence buffer.

Once sufficient frames are accumulated, the landmark sequence is passed to a Long Short-Term Memory (LSTM) based deep learning model. LSTM is a type of recurrent neural network designed to learn temporal patterns in sequential data.

The model processes the sequence of hand landmark coordinates and learns how the hand position changes over time to recognize gestures.

• LSTM layer captures temporal dependencies across consecutive frames
• Dense layer with Softmax activation produces the gesture probability distribution

The server returns the predicted gesture label along with the confidence score in JSON format. The mobile client displays the prediction and can optionally convert the recognized gesture into speech output.

This architecture provides:

- Real-time interaction
- Modular client–server deployment
- Centralized model management
- Reduced mobile computational load
- Scalable API-based design

System Architecture

The system architecture for the Sign Language Recognition Using Deep Learning project is designed to facilitate real-time recognition of hand gestures through a streamlined pipeline combining video capture, hand landmark extraction, deep learning-based classification, and output generation. The architecture enables efficient processing on mobile devices, ensuring accessibility and ease of use.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

Camera Input (Mobile Client Layer)

The process begins with live video capture using the smartphone camera.

- Frames are captured at predefined frame rate.
- Each frame is encoded (JPEG/PNG).
- The image is converted into Base64 or binary format.
- The frame is sent to FastAPI endpoint via HTTP POST request.

This marks the transition from frontend to backend processing.

Fast API Server Layer (Request Handling Module)

Upon receiving the API request:

1. The server validates request payload.
2. Extracts image data.
3. Converts image into NumPy array.
4. Prepares image for processing.

The server acts as an inference engine and manages model execution.

MediaPipe Hands Module (Backend Processing Layer)

The image is passed to MediaPipe Hands, which performs:

(a) Hand Detection

Identifies the hand region and removes background interference.

(b) Landmark Extraction

Extracts 21 three-dimensional (x, y, z) landmarks representing:

- Wrist
- Finger joints
- Fingertips

These landmarks provide compact spatial representation of the hand.

If no hand is detected, the server returns a response indicating absence of gesture.

Preprocessing and Sequence Buffer Module

Extracted landmarks undergo:

- Normalization (relative to wrist)
- Scaling

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

- Noise reduction
- Conversion to structured arrays

A sliding window buffer stores landmark vectors across consecutive frames.

Once the buffer reaches predefined length (e.g., 30 frames), it forms a valid temporal sequence for classification.

LSTM based Deep Learning Model

The structured landmark sequence is passed to the deep learning model.

LSTM Component:

- Processes sequential spatial features
- Captures motion flow
- Learns temporal transitions

Softmax Output Layer

Produces probability distribution across gesture classes.

Prediction Response Generation

The backend generates a JSON response:

- Predicted gesture label
- Confidence score

Text and Speech Output (Client Layer)

Upon receiving the response:

- The mobile application extracts the predicted label.
- Displays it on the user interface.
- Optionally triggers Text-to-Speech module.

This completes one recognition cycle.

Continuous System Operation

The system operates in a continuous request–response loop:

WHILE application is running:

- Capture frame
- Send API request
- Backend performs inference
- Receive prediction

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

- Display output

This loop ensures real-time, uninterrupted gesture recognition.

The sliding window mechanism on the server side ensures continuous temporal modeling without restarting the system.

System Termination

Termination occurs under the following conditions:

Client Side

- User exits application
- Camera capture stops
- API requests cease

Server Side

- FastAPI server continues running unless manually stopped
- Upon shutdown:
    - Model session closed
    - Memory released
    - Resources freed

This ensures graceful system termination and prevents resource leakage.

Summary of System Design

The proposed system design introduces:

- Client–Server architecture
- RESTful API-based communication
- MediaPipe landmark extraction
- LSTM spatial–temporal modeling
- Real-time mobile integration
- Sliding window inference mechanism
- JSON-based response handling
- Graceful termination strategy

This structured architecture makes the system scalable, efficient, and suitable for real-world deployment in assistive communication applications.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

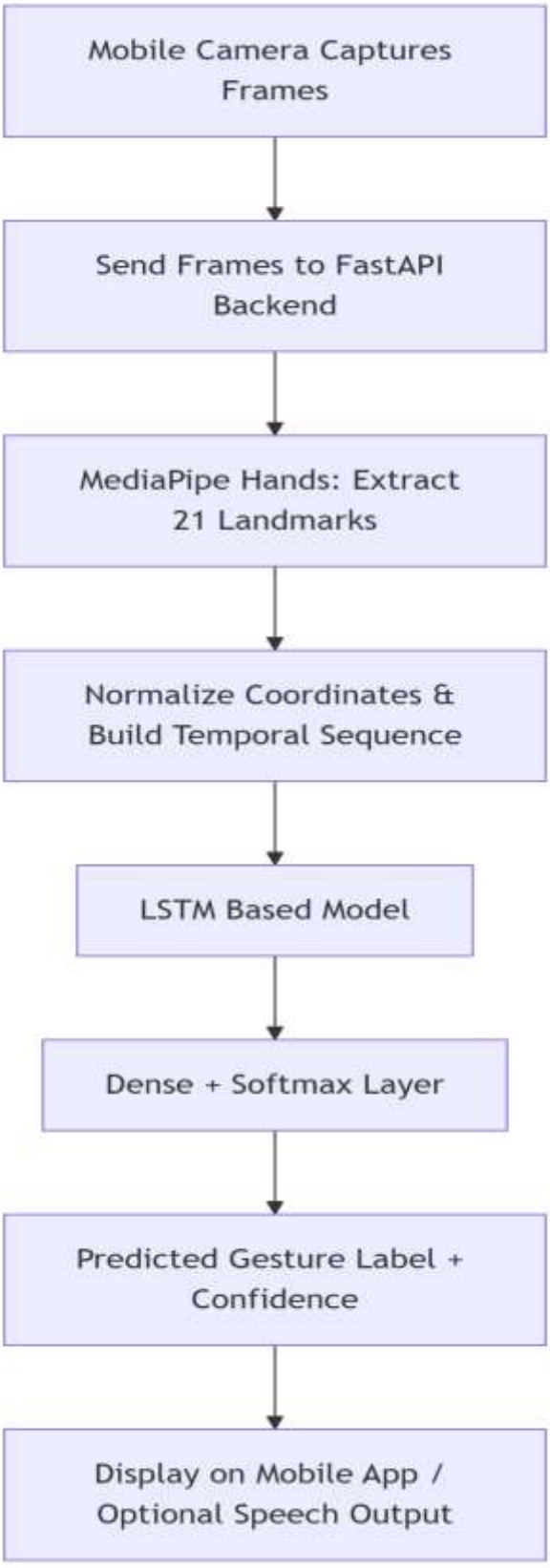

Figure 1: Block diagram of the Real-Time Gesture Recognition System Pipeline.

Feasibility Study

Technical Feasibility

The proposed sign language recognition system is technically feasible due to the choice of well-established, open-source technologies and widely available hardware. The core software components include **Python**,

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

TensorFlow, **MediaPipe**, and **TensorFlow Lite**—all of which have proven track records in computer vision and deep learning applications.

- **Python** provides a versatile programming environment with extensive libraries for machine learning and computer vision.
- **TensorFlow** is a powerful deep learning framework that supports the development and training of complex models such as the LSTM used in this project.
- **MediaPipe** offers efficient, real-time hand landmark detection optimized for mobile and edge devices, enabling smooth landmark extraction without requiring expensive hardware.
- **TensorFlow Lite** facilitates the deployment of trained models on mobile devices, allowing lightweight and fast inference suitable for smartphones.

The hardware requirements are minimal and affordable. Since modern smartphones and laptops typically possess multi-core processors, sufficient RAM (4GB+), and built-in cameras, no specialized hardware is needed. This ensures the system can run efficiently on commonly available consumer devices.

Economic Feasibility

From an economic perspective, the project is highly feasible due to the reliance on free and open-source software components. The use of:

- Open-source libraries (MediaPipe, TensorFlow, NumPy, OpenCV)
- Free development environments (Python, Android/iOS SDKs)
- No need for paid APIs or expensive hardware peripherals

Minimizes costs associated with software licensing or hardware procurement. This significantly lowers the barrier to development and deployment.

Furthermore, the project's focus on mobile deployment leverages existing smartphones, avoiding additional costs for hardware investment by the end-users. This economic accessibility makes the system suitable for widespread adoption, especially in resource-constrained environments.

Operational Feasibility

The operational feasibility of the system is grounded in its user-centric design and ease of use. The application provides a simple and intuitive mobile interface designed to facilitate real-time interaction without requiring specialized technical knowledge from the user.

Key operational benefits include:

- **User-friendly UI**: Clear text and speech outputs ensure users receive immediate, understandable feedback.
- **Real-time performance**: Low latency recognition allows smooth communication, essential for practical usage.
- **Minimal setup**: The system operates directly with the smartphone camera and requires no external sensors or calibration steps.
- **Accessibility**: Designed for speech- and hearing-impaired users, the system supports inclusive communication.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

By addressing user needs effectively and reducing complexity in operation, the system can be easily integrated into everyday communication, demonstrating strong operational feasibility.

METHODOLOGY

The methodology describes the systematic process followed to develop the real-time sign language recognition system. It outlines the key stages, from gathering raw data to delivering the final output in text and speech, ensuring accurate and efficient gesture recognition. The main components of the methodology include:

Model Development and Training Phase:

This phase focuses on dataset preparation, preprocessing, model design, and training.

Data Collection

The first stage involves collecting gesture data for training the LSTM model.

Instead of relying on static image datasets, the system collects **dynamic gesture sequences** using live camera input.

Data Acquisition Process

- A custom data collection script is implemented in Python.
- The camera captures continuous video frames.
- MediaPipe Hands extracts 21 three-dimensional hand landmarks per frame.
- Each landmark contains (x, y, z) coordinates.

These landmarks represent:

- Wrist position
- Finger joints
- Fingertips

Since sign language is dynamic in nature, sequences of frames are recorded for each gesture class.

Each gesture sample consists of:

- A fixed-length sequence (e.g., 30 consecutive frames)
- Labeled with corresponding gesture name

This structured collection ensures that the model learns motion-based features rather than static posture alone.

Data Preprocessing

The raw landmark data extracted during collection undergoes preprocessing before training.

(a) Normalization

Landmark coordinates are normalized relative to a reference point (usually the wrist). This removes variation due to:

- Hand size differences

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

- Camera distance
- Orientation changes

Normalization ensures translation and scale invariance.

(b) Sequence Formation

Since gestures depend on motion over time, the normalized landmark vectors are grouped into fixed-length temporal sequences.

Each sequence:

- Contains N consecutive frames
- Preserves temporal order
- Represents one complete gesture instance

This sequence formation is critical for LSTM-based learning.

(c) Noise Reduction

Minor detection noise and jitter are reduced through smoothing techniques to improve robustness and prevent overfitting.

6.1.3 Feature Extraction and Model Architecture:

The preprocessed sequences are fed into a LSTM based architecture.

LSTM (Temporal Modeling)

The extracted spatial features are passed sequentially into the LSTM network.

The LSTM:

- Models temporal dependencies
- Captures motion flow
- Learns gesture transition patterns

Unlike simple RNNs, LSTM uses memory gates to retain long-term dependencies and avoid vanishing gradient issues.

Fully Connected and Softmax Layer

The LSTM output is passed through:

- Dense layer
- Softmax activation

Softmax produces a probability distribution over predefined gesture classes.

Model Training

The LSTM model is trained using labeled gesture sequences.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

Training Procedure:

- Loss function: Categorical Cross-Entropy
- Optimizer: Adam
- Evaluation metric: Accuracy

Regularization techniques applied:

- Dropout layers to prevent overfitting
- Early stopping to avoid excessive training
- Data shuffling to improve generalization

The dataset is divided into:

- Training set
- Validation set
- Test set

Model performance is evaluated on unseen test data to ensure robustness.

After training, the model is saved and prepared for backend deployment.

Real-Time Client–Server Inference Phase

After model training, the system enters real-time operational mode using a client–server architecture.

Mobile Client – Frame Capture and API Request

The mobile application performs the following:

1. Captures live camera frames.
2. Converts frame into compressed format (JPEG/PNG).
3. Encodes image into binary or Base64 format.
4. Sends HTTP POST request to FastAPI server.

Request details:

- Endpoint: /predict
- Method: POST
- Payload: Captured image
- Content-Type: multipart/form-data or JSON

This marks the transition from frontend to backend.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

FastAPI Backend – Request Handling

Upon receiving request:

1. FastAPI validates requests.
2. Extracts image data.
3. Converts image into NumPy array.
4. Prepares frame for processing.

The backend acts as a centralized inference engine.

Hand Landmark Extraction (Backend Processing)

The backend applies MediaPipe Hands to:

- Detect hand region
- Extract 21 (x, y, z) landmarks

If no hand is detected:

- Server returns response indicating absence of gesture.

If hand detected:

- Landmarks proceed to the preprocessing stage.

Server-Side Preprocessing and Sliding Window Buffer

Extracted landmarks undergo:

- Normalization
- Scaling
- Structuring into numerical arrays

The backend maintains a sliding window buffer.

Each new frame:

- Appends a landmark vector to the buffer.
- If buffer length < required sequence size → wait for more frames.
- If buffer full → perform inference.

After prediction:

- Oldest frame removed.
- A new frame appended.
- Enables continuous temporal modeling.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

LSTM Model Inference

When valid sequence is formed:

1. The sequence of normalized hand landmark coordinates is passed to the **LSTM model**.
2. The **LSTM layer analyzes temporal patterns** in the sequence of frames.
3. The learned temporal features are processed through a **Dense layer**.
4. The **Softmax layer outputs probability scores** for each gesture class.

Highest probability class selected as predicted gesture.

Confidence threshold applied to avoid unstable predictions.

JSON Response Generation

The backend generates response:

```
{
 "prediction":"HELLO",
  "confidence":0.93
}
```

The JSON response is sent back to mobile client.

Output Generation (Client Side)

The mobile application:

- Receives JSON response.
- Extracts predicted label.
- Displays text on screen.
- Optionally activates Text-to-Speech engine.

This completes one recognition cycle.

Continuous Processing Loop:

The system operates in a continuous request–response loop.

Client-Side Loop:

WHILE application is running:

- Capture frame
- Send API request
- Receive response
- Display output

Server-Side Loop:

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

For each request:

- Process frame
- Update buffer
- Perform inference (if ready)
- Return response

This ensures real-time interaction without restarting application.

Exit and Termination Handling

Termination occurs under following conditions:

Client Side:

- User exits application
- Camera stops
- API calls cease

Server Side:

- FastAPI continues running unless manually stopped
- On shutdown:
    - Model session closed
    - Memory released
    - Resources deallocated

This guarantees graceful shutdown and prevents memory leaks.

Model Training

The processed sequences of landmarks are used to train the LSTM model.

- The training dataset contains labeled sequences for a set of predefined gestures.
- The model learns to map input landmark sequences to their corresponding gesture classes by minimizing a classification loss function through backpropagation.
- Techniques such as data augmentation, dropout, and early stopping are applied to enhance generalization and prevent overfitting.
- Model performance is validated on unseen test data to ensure robust accuracy and real-time responsiveness.

Training iteratively refines the model's ability to distinguish between different signs with high confidence.

Summary of Methodology

The proposed methodology integrates:

- Landmark-based gesture representation
- Spatial–temporal deep learning modeling
- RESTful API-based client–server communication
- Sliding window inference mechanism
- Real-time mobile interaction
- Structured request–response architecture
- Robust termination handling

This systematic design ensures accurate, scalable, and efficient real-time sign language recognition suitable for assistive communication applications.

Algorithms and Software Libraries

The proposed Sign Language Recognition system employs a combination of computer vision, deep learning, and sequence modeling algorithms to achieve real-time gesture recognition. The system integrates landmark-based feature extraction with spatial–temporal deep learning modeling.

The primary algorithms used in this project include:

1. MediaPipe Hand Landmark Detection Algorithm
2. Long Short-Term Memory (LSTM) Network
3. Sliding Window Sequence Modeling Algorithm
4. Softmax Classification Algorithm

Each of these is explained in detail below.

MediaPipe Hand Landmark Detection Algorithm

MediaPipe Hands is a real-time hand tracking framework that uses a two-stage pipeline:

Stage 1: Palm Detection:

A Single Shot Detector (SSD)-based deep neural network detects the palm region in the input image. This reduces the search space and improves computational efficiency.

Stage 2: Landmark Regression:

A regression-based neural network predicts 21 three-dimensional hand landmarks from the detected palm region.

Each landmark contains

- x-coordinate (horizontal position)
- y-coordinate (vertical position)
- z-coordinate (depth relative to wrist)

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

These landmarks represent anatomical keypoints including:

- Wrist
- Metacarpophalangeal joints
- Interphalangeal joints
- Fingertips

Why Landmark-Based Representation?

Instead of processing full image pixels:

- Reduces dimensionality
- Removes background noise
- Improves computational efficiency
- Makes model invariant to lighting and background conditions

This significantly enhances real-time performance in mobile environments

Long Short-Term Memory (LSTM) Network:

Since sign language gestures are dynamic, temporal modeling is required.

LSTM is a specialized Recurrent Neural Network (RNN) designed to capture long-term dependencies.

LSTM Internal Mechanism

LSTM contains:

- Forget Gate
- Input Gate
- Output Gate
- Cell State

Mathematical representation

Forget Gate: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

Input Gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

Cell State Update: $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$

Output Gate: $h_t = o_t \cdot tanh(C_t)$

Where:

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

- x_t= input at time t
- h_t= hidden state
- C_t= cell memory

Role in This Project

LSTM processes sequential CNN features and learns:

- Motion patterns
- Gesture transitions
- Temporal dependencies

This allows recognition of gestures involving movement rather than static poses.

Sliding Window Sequence Modeling Algorithm

To enable continuous real-time recognition, a sliding window approach is implemented on the backend server.

Working Mechanism

1. Initialize empty buffer
2. Append new landmark vector for each frame
3. When buffer size == N (e.g., 30 frames):
    - Perform prediction
4. Remove oldest frame
5. Continue appending new frames

This enables overlapping sequence prediction without restarting the process.

Benefits

- Continuous recognition
- Real-time operation
- Stable predictions
- Reduced computational delay

Softmax Classification Algorithm

The final dense layer uses Softmax activation to convert raw model outputs into probability distribution.

Softmax equation:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

Where:

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

- z_i= output score for class i
- k= total number of classes

The class with highest probability is selected as predicted gesture.

Libraries and Frameworks Used

The proposed system utilizes several open-source libraries for computer vision, deep learning, API handling, and mobile integration.

Following are the libraries and framework are used in project:

1. MediaPipe
2. TensorFlow
3. FastAPI
4. NumPy
5. OpenCV
6. Pydantic
7. Text-to-Speech

MediaPipe

Purpose:

- Real-time hand detection
- 21 landmark extraction

Advantages:

- Lightweight
- Optimized for mobile devices
- High detection accuracy
- **BACKGROUND** independent

Used in:

- Backend FastAPI inference stage
- Data collection stage

TensorFlow

Purpose:

- Building CNN–LSTM model
- Training and evaluation

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

- Gradient optimization

Advantages:

- Efficient computational graph execution
- GPU acceleration support
- Wide research adoption

Used in:

- Model training phase

FastAPI

Purpose:

- Backend API server
- Handling HTTP POST requests
- Returning JSON predictions

Advantages:

- High performance (ASGI-based)
- Automatic API documentation
- Asynchronous request handling

Used in:

- Real-time inference server
- Client–server communication

NumPy

Purpose:

- Numerical computations
- Matrix operations
- Landmark array manipulation

Used in:

- Preprocessing
- Model input formatting

5 OpenCV:

Purpose:

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

- Camera capture (training phase)
- Image processing
- Frame conversion

Used in:

- Data collection
- Backend image decoding

Pydantic (FastAPI Dependency)

Purpose:

- Request validation
- Data modeling
- JSON schema definition

Ensures structured API request handling.

Text-to-Speech (TTS) Engine

Purpose:

- Convert predicted text into speech

Options:

- gTTS
- Platform-specific TTS API

Enhances accessibility and usability.

Implementation

The implementation phase converts the conceptual design and methodology into a working system. The system is developed to capture live hand gestures, classify them using a trained deep learning model, and present the output in both text and speech on mobile devices.

Tools and Frameworks

- **Python**: Used for model development, training, and preprocessing.
- **MediaPipe**: Provides real-time hand landmark detection, extracting 21 key points from each hand frame.
- **TensorFlow**: Implements the LSTM based model for gesture classification.
- **TensorFlow Lite**: Converts the trained model into a lightweight format suitable for mobile deployment.
- **OpenCV**: Captures live video input from smartphone cameras and processes the frames for landmark extraction.
- **Text-to-Speech (TTS) API**: Converts recognized gestures into speech output.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

Implementation Steps

1. **Hand Landmark Extraction**
    - MediaPipe continuously detects hands in the camera feed.
    - 21 landmarks per hand are extracted with 3D coordinates (x, y, z) for each frame.
2. **Preprocessing**
    - Landmarks are normalized to remove variation due to hand size, position, and orientation.
    - Sequences of frames are created to capture the temporal aspect of gestures.
3. **Model Training**

- The sequence of **hand landmark coordinates** extracted from each frame is used as input data.
- The **LSTM model learns temporal patterns** across consecutive frames of hand movement.
- The model is trained on a **dataset containing predefined sign language gestures**.

4. **Model Conversion**
    - The trained model is converted to **TensorFlow Lite** for mobile compatibility.
    - Optimizations reduce model size and improve inference speed.
5. **Mobile Application Integration**
    - The app accesses the device camera for live input.
    - Landmark sequences are passed to the LSTM based model for inference.
    - Recognized gestures are displayed as text and optionally converted to speech using TTS.

Mobile Deployment

The system is designed to run efficiently on standard smartphones. Lightweight model architecture and TensorFlow Lite optimizations ensure real-time performance without high computational cost.

Software Requirements

1. **Programming Language**
    - **Python 3.10.8** – Used for model development, training, and preprocessing.
2. **Deep Learning Frameworks**
    - **TensorFlow / TensorFlow Lite** – For LSTM based model development and mobile deployment.
3. **Hand Landmark Detection**
    - **MediaPipe** – Real-time hand tracking and landmark extraction.
4. **Mobile Development**
    - **Android Studio / Xcode** – For building Android/iOS mobile applications.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

5. **Computer Vision & Preprocessing**
    - **OpenCV** – Capturing live video input and preprocessing frames for landmark extraction.
    - **NumPy, Pandas** – Handling arrays, landmark sequences, and datasets.
6. **Visualization**
    - **Matplotlib / Seaborn** – Visualizing training progress, gesture sequences, and accuracy metrics.
7. **Text-to-Speech (TTS)**
    - **gTTS (Google Text-to-Speech) or platform-specific TTS API** – Converts recognized gestures to speech output in real-time.
8. **Operating System**
    - Windows 10/11, macOS, or Linux (for model development and testing)
    - Android 8.0+ / iOS 12+ (for mobile deployment)

Hardware Requirements

1. **Smartphone / Mobile Device**
    - Android or iOS smartphone with camera support.
    - Recommended: Quad-core processor or higher.
    - RAM: 4 GB or more.
    - Camera resolution: 720p or higher.
2. **Development Machine**
    - Processor: Intel i5/i7 or AMD Ryzen 5/7 or higher.
    - RAM: 8 GB or more (16 GB recommended for faster training).
    - GPU (optional but recommended): NVIDIA GTX 1050/1650 or higher for faster LSTM training.
    - Storage: Minimum 500 GB (for datasets and models).

RESULTS AND DISCUSSION

The results section analyzes the performance of the proposed system and highlights its advantages and limitations.

Gesture Recognition Accuracy

- The system achieves high accuracy for predefined gestures under controlled conditions (good lighting and clear backgrounds).
- Accuracy is measured by comparing predicted gestures with the actual gesture performed.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

Performance Analysis:

- **LSTM Model:** Significantly outperforms single-frame models because it captures temporal motion along with spatial features.
- **Real-time Performance:** Achieves high frame-per-second (FPS) performance suitable for mobile deployment.
- **Output Generation:** Text and speech outputs are correctly synchronized with live gestures.

Observed Challenges

- Complex Gestures: Some gestures with subtle differences are occasionally misclassified.
- Environmental Conditions: Low lighting, occlusions, and background clutter reduce landmark detection accuracy.
- Limited Vocabulary: Initial prototype supports a limited number of gestures; expansion is required for real-world applications.

Discussion

The implementation validates that integrating MediaPipe with LSTM is effective for real-time sign language recognition. The hybrid model captures spatial and temporal patterns, making it superior to conventional single-frame approaches. Real-time mobile deployment proves feasibility for practical communication aid applications.

Graphical User Interface Results:

Fig. 1 User Registration Page

Fig. 2 User Login Page

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue VI June 2026

Fig. 3 Home Page

Fig. 4 Camera Interface and Gesture Prediction output

Fig. 5 Camera Interface and Gesture Prediction Output

CONCLUSION AND FUTURE SCOPE

Conclusion

- The project successfully develops a real-time sign language recognition system using deep learning.
- Integration of MediaPipe for hand landmark extraction and LSTM for gesture classification ensures high accuracy and temporal awareness.

- The mobile application delivers real-time text and speech outputs, enabling communication between speech-impaired users and others.
- The system demonstrates that lightweight deep learning models can operate efficiently on mobile devices without relying on cloud services.

Future Scope

- Expansion of Gesture Vocabulary: Support for more signs to cover a broader communication spectrum.
- Sentence-Level Sign Interpretation: Recognize sequences of gestures to form meaningful sentences.
- Multilingual Speech Output: Convert gestures into speech in multiple languages to increase accessibility.

REFERENCES

1. TensorFlow, "TensorFlow Lite Documentation," *Google AI Platform*, 2024. [Online]. Available: https://www.tensorflow.org/lite
2. M. U. Rehman, F. Ahmed, M. A. Khan, et al., "Dynamic hand gesture recognition using 3D-CNN and LSTM networks," *Computers, Materials & Continua*, vol. 72, no. 3, pp. 5901–5918, 2022 Available: https://doi.org/10.32604/cmc.2022.019586
3. A. Khan, M. Sayed, et al., "Hand gesture recognition using keypoint landmarks with machine learning," in *Proc. IEEE Int. Conf. Computing and Comm. Systems*, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9562108
4. F. Zhang, V. Bazarevsky, A. Vakunov, et al., "MediaPipe Hands: On-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, 2020. [Online]. Available: https://arxiv.org/abs/2006.10214
5. A. Gunduz, N. Kose, and G. Rigoll, "Real-time hand gesture detection and classification using CNNs," *arXiv preprint arXiv:1901.10323*, 2019. [Online]. Available: https://arxiv.org/abs/1901.10323
6. P. Molchanov, X. Yang, S. Gupta, et al., "Online detection and classification of dynamic hand gestures with recurrent 3D-CNN," in *Proc. IEEE CVPR*, 2016 Available: https://openaccess.thecvf.com/content_cvpr_2016/papers/Molchanov_Online_Detection_and_CVPR_2016_paper.pdf