**Institute of Technology of Cambodia**

**Department of Electrical and Energy Engineering**

# Action Recognition for Khmer Sign Language with Text and Speech Translation

**Group** : I5-GEE-EA

**Lecture by** : Mr. SUM Rithea

**Prepared by** : LANG Bandithvipho

**Academic**

**Year 5, Semester I**

**2024-2025**

## ABSTRACT

This project introduces an open-source web application Interface for Khmer Sign Language Recognition, aimed to serve as a steppingstone towards more advanced technologies for Khmer Sign Language Translation Systems. Utilizing a combination of deep learning to work with sequential model and pose estimation models, the interface provides two modular components: a sign recognition module for translating Khmer Sign to text and speech generation module for converting from text to Khmer spoken.

This project addresses the communication challenges faced by the deaf and mute community in Cambodia over 100,000 individuals, due to the lack of tools for translating Khmer sign language into text and speech. Existing technologies primarily support other languages, leaving Khmer speakers underserved. To bridge this gap, we propose a real-time system that recognizes Khmer sign language gestures, converts them into text, generates to Khmer spoken.

The system is designed to be highly accessible, user-friendly, and capable of functioning in real-time under varying environments conditions like background, lighting, and hand sizes. We discuss the technical details of the model architecture, application, as well as the potential future enhancements for the real-world Khmer sign in daily life.

## ABBREVIATIONS

| | |
|---|---|
| **RNN** | Recurrent Neural Network |
| **LSTM** | Long-Short Term Memory |
| **SLP** | Sign Language Processing |

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# 1. INTRODUCTION

## 1.1. Project Statement

Sign Language Processing (SLP) is a field of research that focuses on the development of technologies to understand and generate sign language. With the rise of Artificial Intelligence, deep learning, and computer vision technologies, the field of SLP has seen significant advancements in recent years. Key techniques such as skeleton pose extraction, using 2D human pose estimation [1] and 2D-to-3D mapping [2], have played a crucial role in improving the accuracy and efficiency of sign language recognition and generation systems.

In Cambodia, an estimated 100,000 people live with deafness and muteness, facing many challenges such as struggle in communication and obstacles in seeking medical treatment, due to the lack of tools that translates Khmer Sign Language into text or speech, impeding their ability to integrate into society and access essential services. Existing technologies are not available for Khmer Sign Language, leaving a significant gap.

## 1.2. Objective

The primary objective of this project is to develop an open-source web application for Khmer Sign Language Translation as shown in **Figure 1.1**. This system aims to accurately interpret Khmer Sign language by using action and express it as Khmer Text and generate from that text into Khmer spoken. By providing a real-time and accessible solution, the project seeks to bridge communication gaps for the deaf and mute communities in Cambodia.

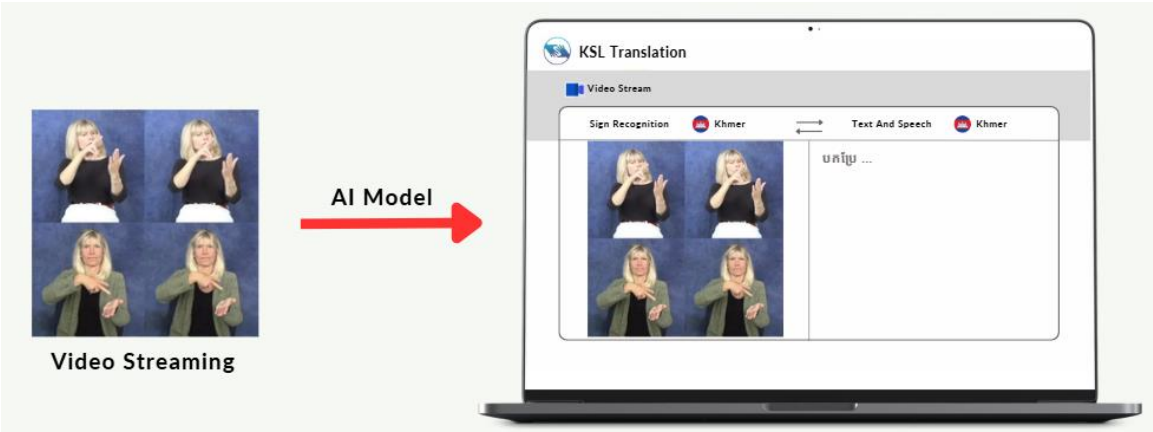


**Figure 1.1** Demostrate Interface for Sign Language Translation

### 1.3. Scope of Work

The scope of this project is:

- Data Collection: Capturing sign language, feature extraction for hands, and pose landmarks then convert into sequential data.
- Build and Train Model: Using LSTM for sequence model and tuning hyper-parameters.
- Evaluation and Develop model: Evaluate model accuracy and performance. Optimize model.
- Demo Interface: Create interface to demonstrate the model's capabilities in real-world setting by hosting it in local network.

### 1.4. Outline of the Project

There are four main chapters in this project:

- **Introduction**: The project begins by identifying the challenges faced by the deaf and mute communities in Cambodia. Specifically, discussing the project's objectives, scope, and bridging communication gaps through Deep Learning Technology.
- **Methodology**: This chapter describes the process of the project, including data collection, model building using LSTM for gesture recognition, model development, creating a web user interface using Flask, and deploying the system in real-time via a web application.
- **Results and Discussion**: Present the results of the model's performance, evaluating the model's accuracy and its effectiveness in real-time sign language translation and speech generation.
- **Conclusion and Future work:** Summarize the project's findings and discuss the limitations of the current system and propose directions for future research.

## 2. METHODOLOGY

In this section, we describe the technical pipelines of the recognition model and text to speech translation. In-depth details on the data collection, selecting model, model training and inference stages for web application deployments.

### 2.1. System Overview

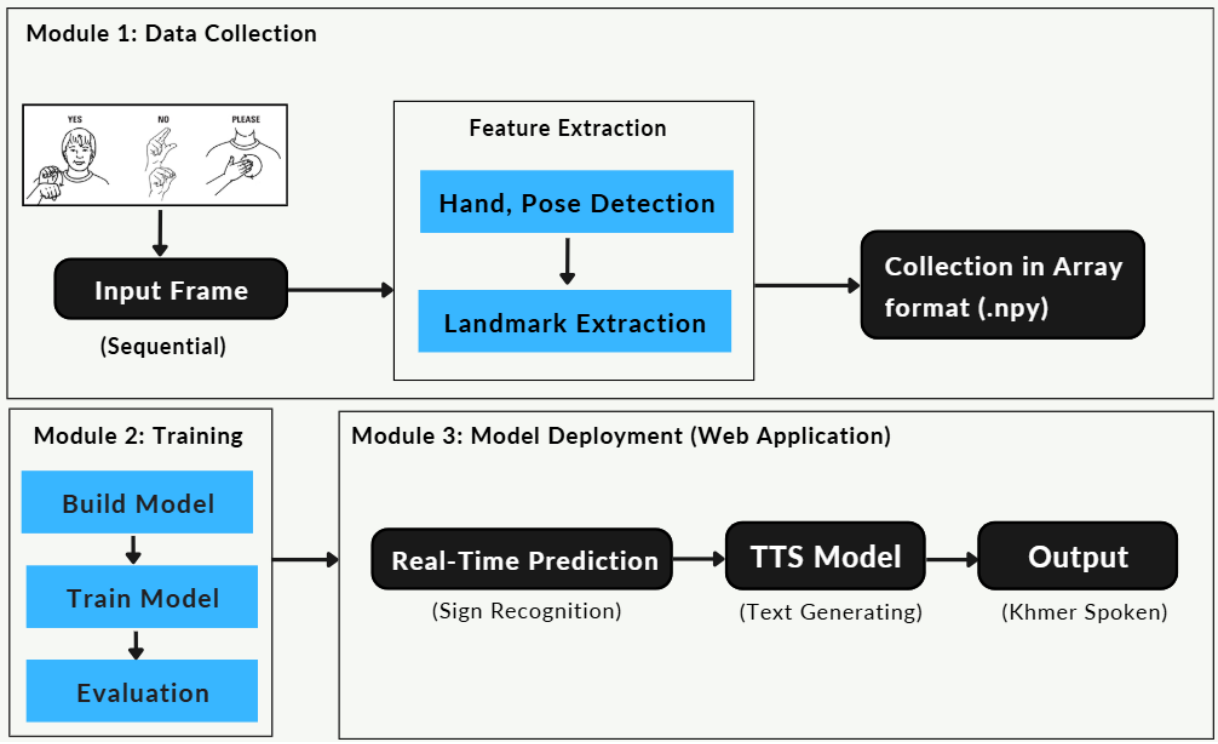


**Figure 2.1 Flow Diagram**

As shown in **Figure 2.1**, the project was constructed into 3 district function modules, Data Collection, Training Procedure, and Model Deployment in Web Applications.

- **Data Collection:** The data collection process involves capturing sequential data from video streams. We utilized the **Media Pipe** library to extract hand landmarks from each frame in the video. These landmarks represent key points on the hands, allowing us to focus on relevant features for sign recognition. The extracted landmark data is then saved in a structured array format (.npy files), which is efficient for storage and training purposes.
- **Training Procedure:** For training the sign language recognition model, we employed a Long Short-Term Memory (LSTM) network, which is well-suited for sequential data such as hand movements in sign language. After training, we evaluated the model's

performance using standard metrics and conducted extensive testing to ensure the model's reliability before deployment. This evaluation step helps us fine-tune the model for better performance during real-time inference.

- **Model Deployment:** The model was integrated into a web application to provide an intuitive and user-friendly interface for real-time sign language recognition. The web application's front-end was developed using **Next.js**, a React-based framework known for its fast rendering and seamless user experience. For the back end, we used **Flask**, a lightweight Python framework, to handle the recognition process. The trained LSTM model runs on the server-side, where it processes incoming video frames, performs predictions, and sends the recognized sign back to the client interface in real time.

## 2.2. Data Collection

The data collection process is a critical step in the development of the Khmer sign language recognition and text-to-speech translation system. It involves capturing high-quality data of sign languages, extracting meaningful features, and ensuring the dataset represents diverse sign language gestures performed by different individuals in real-time sign translation. This section outlines the tools, framework, techniques, and methodologies used for data preparation and feature extraction.

### 2.2.1. What is MediaPipe?

MediaPipe is an open-source framework developed by **Google** that provides a set of pre-built tools and libraries for real-time perception tasks.


**Figure 2.2 MediaPipe Framework**

In this project, MediaPipe's **Hand Detection**, **Pose Detection**, and **Face Detection** solutions are used to capture 59 key landmarks for each hand and additional features from the face and upper body. These landmarks represent critical points like fingertips, joints, and the center of the palm. This extracted landmark data is then used as input for training the sign language recognition model. MediaPipe's efficiency and accuracy in real-time processing make it an invaluable component of this project.

### 2.2.2. Feature Extraction

Feature extraction is a key aspect of the data collection pipeline, converting raw video frames into structured, meaningful data. For this project, the MediaPipe library is used to extract **spatial coordinates** of keypoints representing the hands, pose, and face. These features are used as inputs for training and testing the Khmer sign language recognition model.

**Table 2.1 Properties and Key Feature**

| Region | Keypoint | Indices |
|---|---|---|
| Left Hand | 21 | 0-20 |
| Right Hand | 21 | 0-20 |
| Pose | 9 | [0, 11, 12, 13, 14, 15, 16, 23, 24] |
| Face | 8 | [0, 13, 14, 17, 61, 267, 269, 291] |

The extracted features are categorized as follows:

- **Hand Landmark:** Hand Landmarks Each hand contributes 21 landmarks, including fingertips, knuckles, and the wrist as shown in **Figure 2.3**.

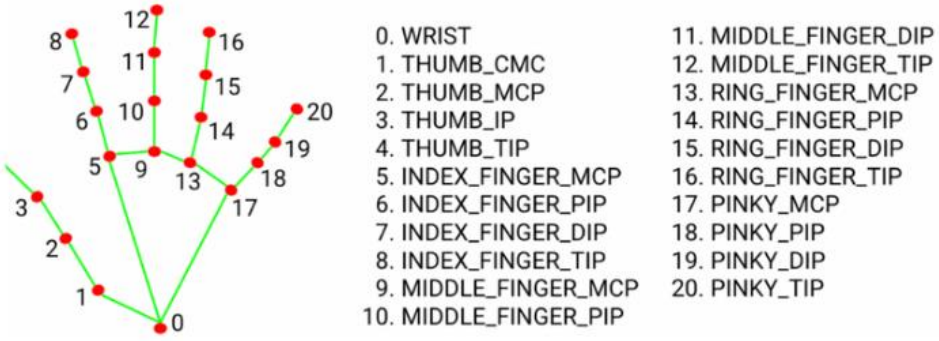


**Figure 2.3 Hand Landmark Coordinate**

- **Pose Landmark:** Nine pose landmarks [0, 11, 12, 13, 14, 23, 24] are used, focusing on key joints in the upper body, such as shoulders, elbows, and wrists. For facial landmarks provide reference 8 points, aiding in distinguishing gestures involving facial movements or expressions as shown in **Figure 2.4**.

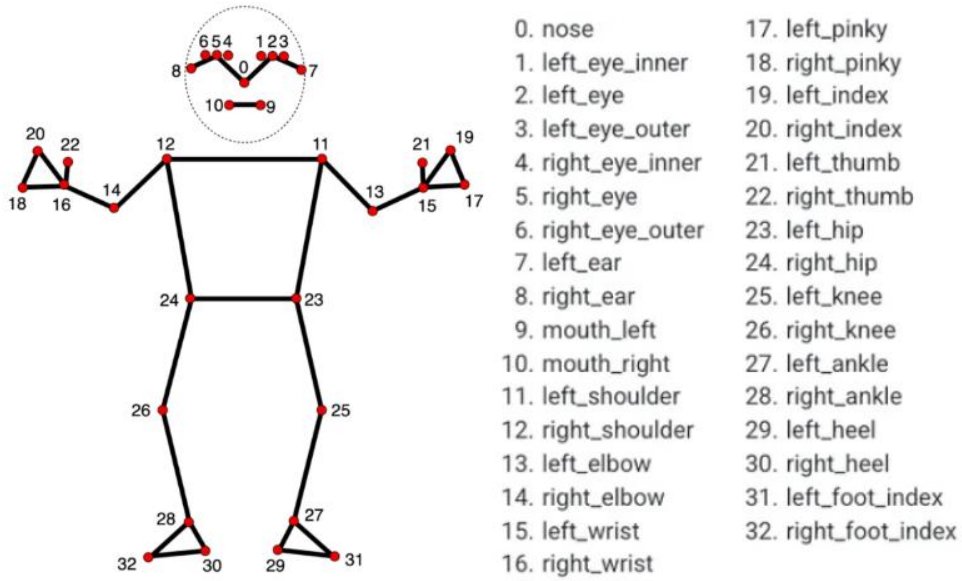


**Figure 2.4 Pose Landmark Coordinate**

### 2.2.3. Dataset Preparation

To ensure the model performs well across a variety of conditions, data is collected from multiple participants, each performing gestures in different environments, lighting conditions, and camera angles. Each gesture is captured in sequences of 30 frames, stored as **.npy** files for efficient processing.

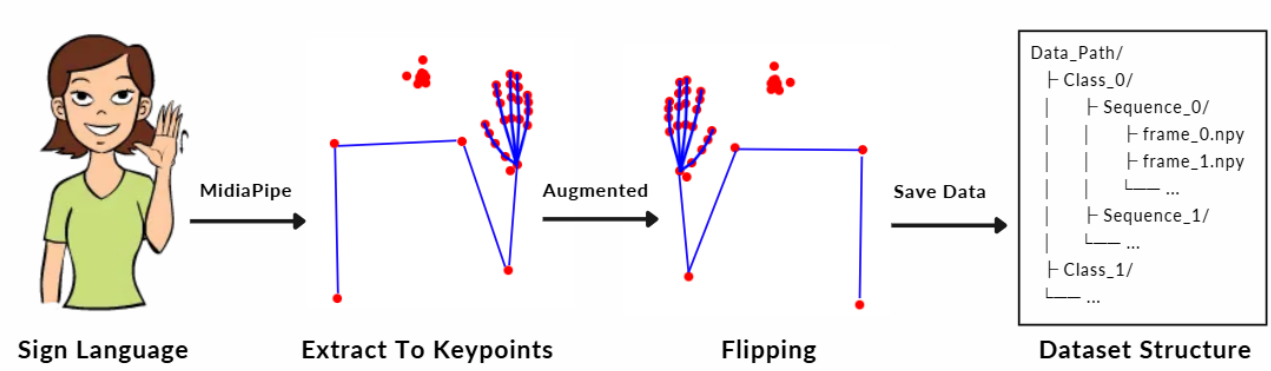


**Figure 2.5 Feature Extraction**

As illustrated in **Figure 2.5**, the dataset includes sign language gestures for words such as ['hello', 'I', 'go', 'home', 'like', 'today']. The input data is processed as sequential data, where each gesture is represented by a sequence of frames. We then perform feature extraction to convert raw images into key-points, capturing critical information about hand, pose, and face

landmarks. To enhance the dataset and improve model generalization during training, we apply data augmentation techniques such as flipping, rotation, and adding noise. These augmentations help create a more robust model capable of handling variations in gestures, camera angles, and lighting conditions. Finally, all the data that is collected we store it in file .**npy** as the array form.

### 2.3. Sequential Model

A sequence model processes input as a sequence, where each data point is dependent on the previous one. This allows the model to learn complex patterns over time.

In this project, we've built neural network by using a Many-To-One architecture, where a sequence of inputs (representing a series of sign language gestures or actions) is processed to generate a single output, and the output would be a single label or action corresponding to the entire gesture sequence.

#### 2.3.1. RNN Model

Recurrent neural networks (RNN) are a class of neural networks that are powerful for modeling sequence data such as time series or natural language. This diagram below **Figure 2.6**, represents the structure of a Recurrent Neural Network (RNN) Architecture and how it processes sequences.

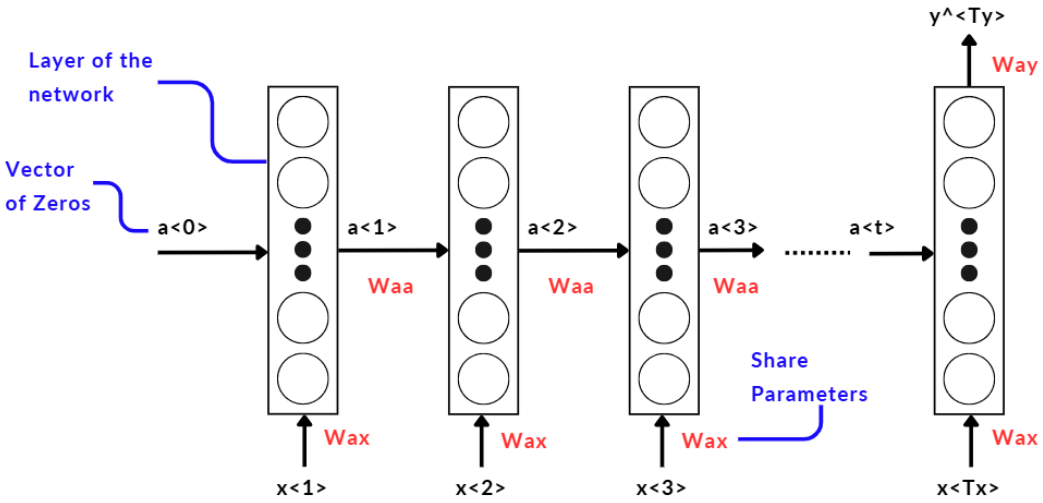


**Figure 2.6 Recurrent Neural Network (RNN) Architecture**

Let us provide a more detailed explanation:

- Input Sequence ($x^{<t>}$): The RNN takes sequential inputs $\{x^{<1>}, x^{<2>}, \dots, x^{<T_x>}\}$, where $T_x$ is the sequence length.
- Hidden State ($a^{<t>}$): At each time step, the hidden state ($a^{<t>}$) combines the current input ($x^{<t>}$) and the previous hidden state ($a^{<t-1>}$) to store information. The initial state ($a^{<0>}$) is usually set to zero.
- Weights:
    - $W_{ax}$: Transforms input ($x^{<t>}$) to a usable format.
    - $W_{aa}$: Transfers information between hidden states.
    - $W_{ay}$: Maps the hidden state to the output ($\hat{y}^{<t>}$).
- Output Sequence ($\hat{y}^{<t>}$): At each time step, the RNN generates an output based on the current hidden state.

### 2.3.2. Vanishing and Exploiding Gradient

In this project, the sign recognition model requires processing large amounts of sequential data. However, RNNs are not well-suited for capturing long-range dependencies, especially in lengthy sequences, due to inherent challenges in their architecture:

- **Backpropagation:** Gradients are calculated and propagated through time, involving repeated multiplication across layers.
- **Derivative (vanishing)**: When gradients become very small, the network forgets earlier information, leading to poor learning for long-term dependencies.
- **Derivative or Slopes are too large (exploding)**: When gradients grow excessively large, the network becomes unstable, making training ineffective.

### 2.3.3. LSTM Model Architecture

To address the vanishing and exploding gradient problems in RNNs, Long Short-Term Memory (LSTM) networks were introduced. LSTMs are a specialized type of RNN designed to capture long-range dependencies effectively.

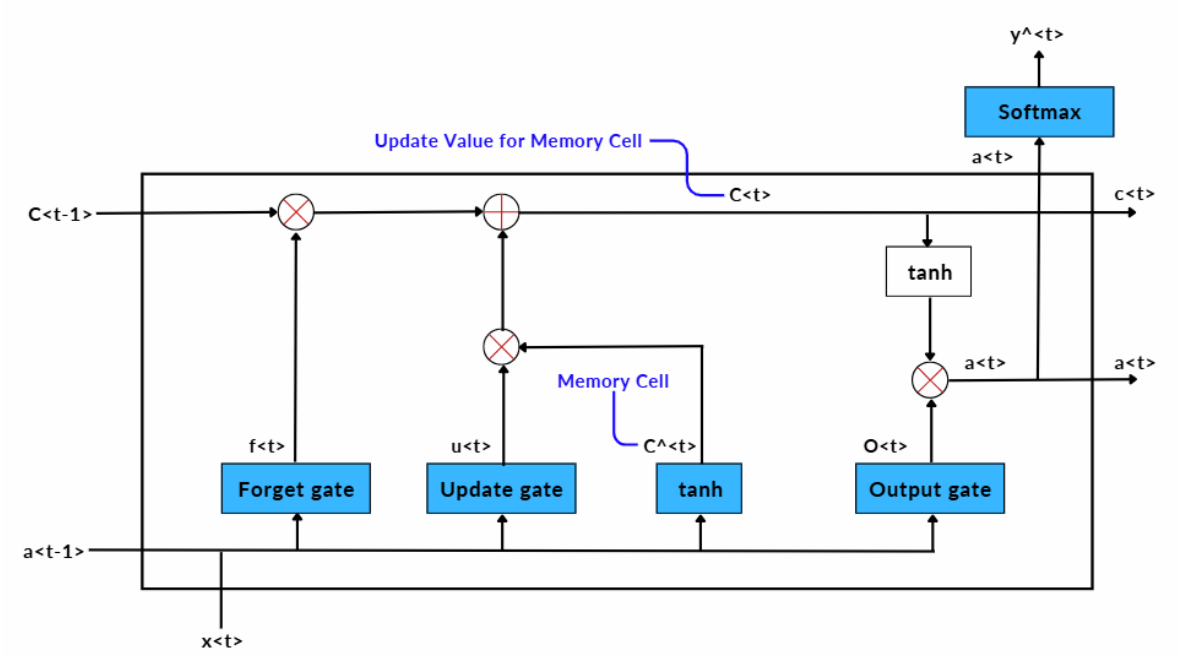


**Figure 2.7 LSTM Architecture**

As shown in **Figure 2.7**, LSTMs are specifically designed to handle sequential data by maintaining a memory cell that preserves relevant information over time. The key components of the LSTM architecture that make it effective for sign recognition include:

- **Forget Gate**: Removes irrelevant information from earlier frames, ensuring the model focuses on essential features of the gesture.
- **Input Gate:** Integrates meaningful information from the current frame to update the cell state, adapting to the evolving gesture.
- **Output Gate**: Generates predictions based on the updated cell state, capturing the gesture's meaning at the current step.

In the context of this project, the LSTM processes sequences of extracted keypoints from **MediaPipe** (such as hand and pose landmarks). The network learns to identify patterns in these sequences, enabling it to classify gestures like (“Hello”, “Home”, or “Go”) with high accuracy. By leveraging LSTMs, the model ensures that both short-term and long-term dependencies within the gesture sequences are accurately captured.

### 2.4. Build and Train Model

In this section, we describe the process of building and training a deep learning model for sign language recognition using the **“Tensorflow.Keras”** framework. The goal of this

model is to recognize sequential patterns in sign language data effectively, leveraging the capabilities of LSTM networks to capture temporal dependencies.

### 2.4.1. Setup Model Architecture

Below, we outline the architectural setup and the key components of the model:

**Table 2.2 Create Layer of LSTM Model**

| Layer | Type | Unit | Activation | Output Shape | Additional |
|---|---|---|---|---|---|
| 1 | LSTM | 128 | tanh | (batch_size, time_steps, 128) | return_sequences=True, input shape = input_shape |
| 2 | Dropout | - | - | - | Dropout rate = 0.3 |
| 3 | LSTM | 128 | tanh | (batch_size, time_steps, 128) | return_sequences=True |
| 4 | Dropout | - | - | - | Dropout rate = 0.3 |
| 5 | LSTM | 128 | tanh | (batch_size, 256) | return_sequences=False |
| 6 | Dropout | - | - | - | Dropout rate = 0.3 |
| 7 | Dense | 128 | ReLU | (batch_size, 128) | L2 regularization (l2(0.01)) |
| 8 | Dense | n_class | Softmax | (batch_size, num_classes) | Final output for classification |

As shown in **Table 2.2,** the model comprises 8 layers, each serving a specific purpose in the learning process:

- **First LSTM Layer**: Configured with 128 units, this layer uses the `tanh` activation function and returns sequences to pass the temporal patterns to the next layer. It also accepts the input shape as a parameter.
- **Second LSTM Layer**: Another 128-unit layer, designed similarly to the first, to further process sequential dependencies.
- **Third LSTM Layer**: This 256-unit layer does not return sequences, preparing the output for the dense layers.
- **Dropout Layer:** Each LSTM layer is followed by a Dropout Layer with a dropout rate of 0.3 to prevent overfitting by randomly deactivating neurons during training.
- **First Dense Layer**: This fully connected layer with 128 neurons uses the **`relu`** activation function and applies an L2 regularization of 0.01 to penalize complex weight configurations, enhancing generalization.

- **Output Dense Layer**: The final layer uses a **`softmax`** activation function with a few neurons equal to the number of classes in the dataset, enabling probabilistic output for multi-class classification.

The following code defines the architecture of the LSTM model:

```
def create_model(input_shape, num_classes):
    model = Sequential([
        LSTM(128, return_sequences=True, activation='tanh',
input_shape=input_shape),
        Dropout(0.3),
        LSTM(128, return_sequences=True, activation='tanh'),
        Dropout(0.3),
        LSTM(256, return_sequences=False, activation='tanh'),
        Dropout(0.3),
        Dense(128, activation='relu', kernel_regularizer=l2(0.01)),
        Dense(num_classes, activation='softmax')
    ])
    return model
```

### 2.4.2. Hyper-Parameters

The training process is optimized with the following hyperparameters to enhance model performance and generalization:

**Table 2.3 Parameters Tuning**

| Parameters | Purpose |
|---|---|
| Dataset | Complex Signs: 1,900 sequences (20 **classes**) |
| Loss Function | Categorical Crossentropy |
| Optimizer | Adam |
| Batch Size | 32 |
| Epochs | 100 |
| Learning Rate | 0.001 |
| Train set/Test set | Split to 80 % and 20 % |

**Explanation of Hyperparameters**

This model is trained with **Categorical Crossentropy** and **Adam Optimizer**, making it suitable for multi-class classification tasks like sign language recognition.

- **Loss Function**: Categorical cross-entropy is a loss function commonly used for multi-class classification problems. It's calculated by comparing the predicted probabilities of each class to the true class labels.

  Which formula is:

$$Loss = -\sum_{i=1}^{C} y_i \,.\log\,(p_i)$$

  Where:

  - C is the number of classes.
  - $y_i$ is the true label (one-hot encoded), which is either 0 or 1.
  - $p_i$ is the predicted probability of the model for class $i$.

  Example:

  - True label: Class of 3 (one-hot encoded as [0, 1, 0] => [I, You, Go])
  - Predicted probabilities: [0.2, 0.7, 0.1]
  - The categorical cross-entropy loss would be:

$$Loss = -(0 \times \log(0.2) + 1 \times \log(0.7) + 0 \times \log(0.1))$$
$$Loss = -(-0.357) = 0.357$$

- **Optimizer**: The Adam optimizer updates the model's weights during training to minimize the loss. After calculating the loss, Adam uses the gradients to adjust the weights by combining two techniques: momentum (which smooths updates) and adaptive learning rates (which adjust step sizes for each weight). This makes training faster and more stable, helping the model learn efficiently.
- **Batch Size**: 32 samples are used per training batch to balance computational efficiency and model performance.

### 2.5. Text-To-Speech Generation

In the Text-to-Speech (TTS) generation module, is designed to convert the text output from the sign language recognition model into spoken Khmer, facilitating real-time communication. The process follows a series of steps to ensure high-quality speech generation.

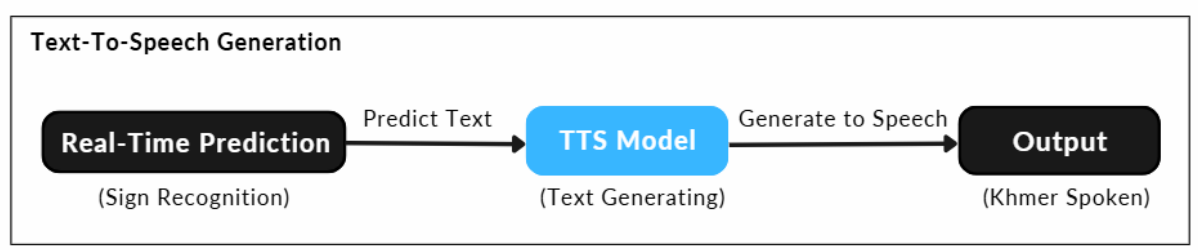


**Figure 2.8 Text To Speech Generation**

As we can see in **Figure 2.8**, the Text-to-Speech (TTS) system [3] converts the recognized sign language gestures into Khmer spoken, enabling real-time communication. The model used in this project is the **"facebook/mms-tts-khm"** from VITS model [4], which is designed for the Khmer language. The process involves several key steps as illustrated in **Figure 2.9**:

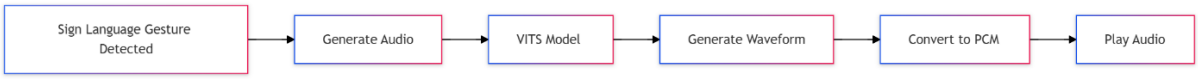


**Figure 2.9 Text To Speech Flow Diagram**

- **Recognition Model:** After recognizing a Sign Language from gesture, the system predicted the detected sign into Khmer Text using a predefined dictionary.
- **Generate Audio**: Once the recognition model predicted the sign languages to output label (Text), the system uses the **VITS model** to generate the corresponding audio waveform. The model tokenizes the Khmer text and processes it through the neural network to produce a waveform of the speech.
- **Generate waveform:** Converted into a compatible audio format using the **PyDub** library
    - The **tokenizer** from the Hugging Face library is used to tokenize the text, transforming it into a format suitable for the TTS model.
    - The **VITS model** processes the tokenized input and generates a waveform.
    - The waveform is scaled and converted into **16-bit PCM format**, which is suitable for playback. The 16-bit PCM (Pulse Code Modulation) format refers to a method used to digitally represent analog audio signals.
- **Playing Audio**: The generated audio is stored in memory and played using **PyDub**, ensuring quick playback without the need to write the audio to disk. This is done asynchronously to keep the system responsive.

To use this VITs Model for Text-To-Speech Generation, we can install the latest version of the library by following command below:

```
pip install --upgrade transformers accelerate
```

Then, we can run inference with the following code-snippet from Hugging Face:

```
from transformers import VitsModel, AutoTokenizer
import torch

model = VitsModel.from_pretrained("facebook/mms-tts-khm")
tokenizer = AutoTokenizer.from_pretrained("facebook/mms-tts-khm")

text = "some example text in the Khmer language"
inputs = tokenizer(text, return_tensors="pt")

with torch.no_grad():
    output = model(**inputs).waveform
```

The output result of waveform can be saved as a **.wav** file. To integrate with Khmer sign recognition system, we need convert and scale it into **16-bit PCM format**, which is suitable for playback as audio. Then use **PyDub** to playback the audio as a Khmer spoken.

### 2.6. Interface

The interface of the system is designed to provide seamless interaction between the client and server components, enabling real-time Khmer sign language recognition, translation, and text-to-speech generation. The server provides a modular implementation of sign recognition and text-to-speech generation that is accessible via a WebSocket. The client is a web application that serves as user-friendly interface for interacting with the two components which is sign recognition and text to speech translation.

#### 2.6.1. Server

The server is implemented in Python using the Flask web framework. It provides a WebSocket API that allows clients to interact with the sign recognition and text-to-speech Translation components. The server is designed to be modular, allowing for easy integration with other existing applications. The server captures video streams, processes the frame using the recognition model, and emits the sign letter or text to the client through the WebSocket messages. This architecture allows for real-time sign recognition.

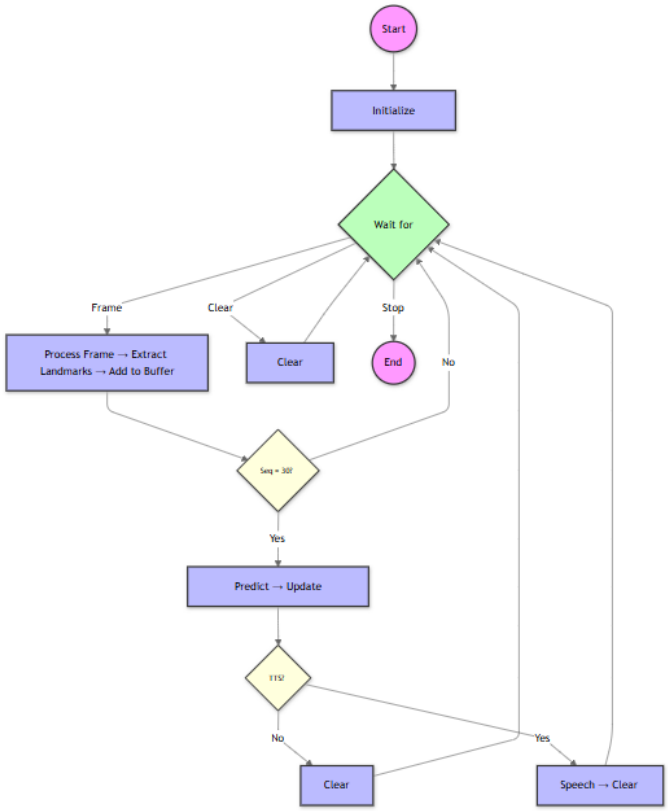


**Figure 2.10 Overall Flowchart of Server**

In **Figure 2.10** illustrates the flowchart representing the server's workflow for the real-time sign language recognition system. This flowchart demonstrates the step-by-step process followed by the server to process input, predict signs, update sentences, and provide feedback to the user. Below is a detailed explanation of each component:

- **Start and Initialization:** The system begins its operation when the server is launched. Once started, the system initializes the required models, such as:
    - The Sign Language Recognition Model to process input frames and make predictions.
    - The Text-to-Speech (TTS) Model is used to convert recognized sentences into spoken language when required.
- **Wait for Input:** The server enters a waiting state, where it continuously listens for input from the client, including:
    - A frame from the client’s camera for processing.
    - A Clear command to reset the system's data.
    - A Stop command to end the server's operation.

- **Frame Processing:** When the frame is received, the system starts analyzing the incoming video frame. Then, extract landmarks by using MediaPipe Library. After that, store the extracted landmarks in a buffer to construct a sequence. The buffer helps in capturing the context of movements over time.
- **Sequence length check:** The system checks whether the buffer contains 30 frames (the required sequence length). If **"No"**, the system returns to waiting for more frames, as a complete sequence is needed for accurate predictions. If **"Yes"**, the system proceeds to the **Prediction** step.
- **Prediction and Update sentence:** Once a full sequence of 30 frames is collected:
    - The system uses the sequence to predict a sign (word).
    - The predicted sign is appended to the current sentence. If the same word is repeated unnecessarily, the system avoids adding it to maintain sentence clarity. This step ensures the server constructs meaningful sentences from the recognized signs.
- **Text-To-Speech Translation:** The system evaluates whether the updated sentence needs to be spoken aloud using TTS.
    - If "**No"**, the sequence buffer is cleared, and the system waits for new input.
    - If "**Yes"**, the server converts the sentence into speech, plays it back to the user, and then clears the buffer.

### 2.6.2. Client

The client is a web application front-end developed using **Next.js**, a React-based framework, to manage both the front-end interface and server-side functionality. TypeScript was used to write the front-end code, with ". tsx" files defining the user interface components. To enable interaction with the server, the client establishes a connection via **WebSocket**, allowing real-time communication with the recognition and text-to-speech translation components. The client displays a live webcam feed for sign recognition, overlaying recognized text directly on the stream. Additionally, transcribed words from the server, generated by the sign recognition model, are displayed on the right side of the interface in real time, providing an intuitive and dynamic user experience as shown in **Figure 2.11**.

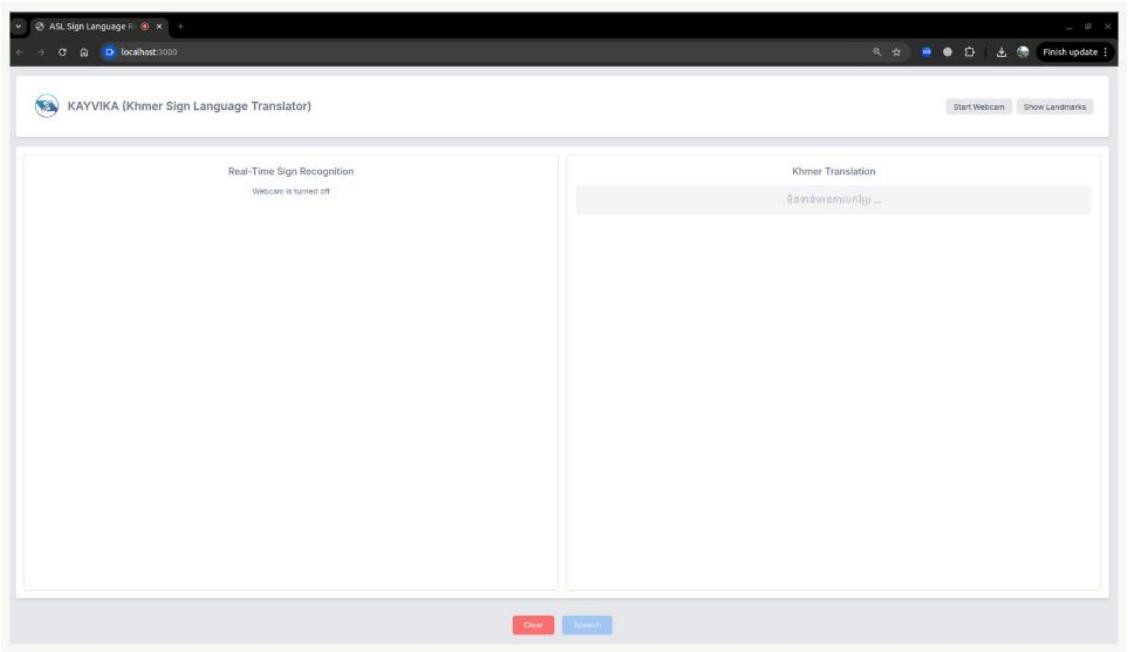


**Figure 2.11 Demostrate Web Interface**

Below is the flowchart of the client-side application in **Figure 2.12**, which illustrates the structure and workflow of its components. The application starts by initializing a socket connection with the server and transitions to the main interface.

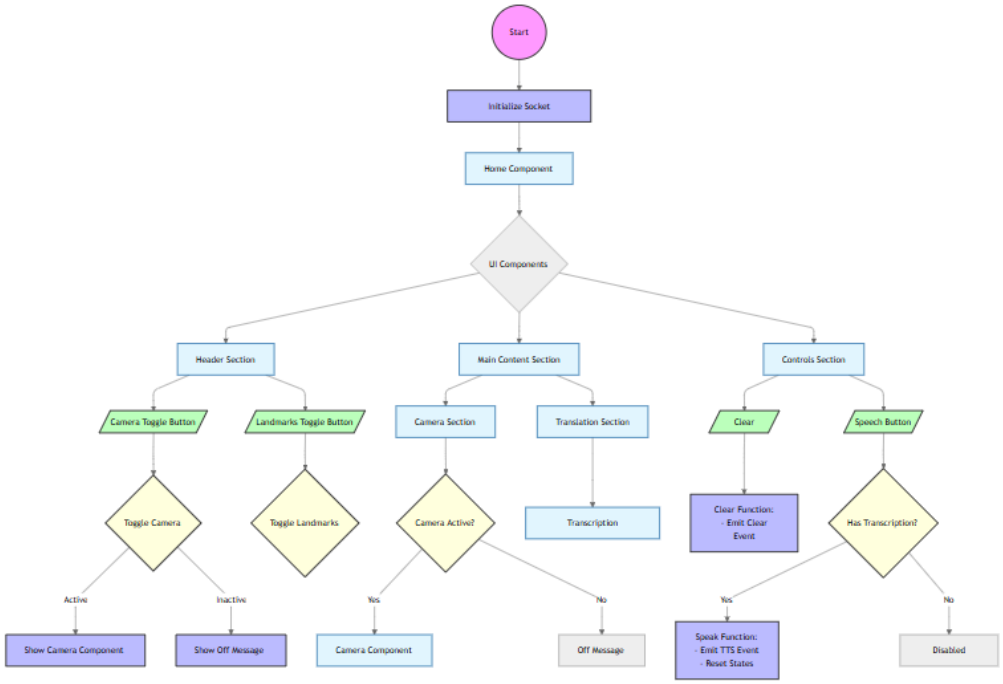


**Figure 2.12 Overall Flowchart of Client**

Once connected, the application transitions to the main interface as we can see in **Figure 2.11**, which is divided into several primary function:

- **Head Section:** These allow users to activate or deactivate the camera feed and landmark annotations, respectively.
    - The Camera Toggle Button manages the display of the camera feed. If toggled "on," the camera feed is activated and displayed in the Camera Component; if toggled "off," an off message is shown.
    - The Landmarks Toggle Button enables or disables the display of landmark annotations on the video stream.
- **Left Pane**: Displays the camera feed, where users can view the real-time video input.
- **Right Pane**: Displays the transcription results of the recognized sign language, updated in real-time as the system processes user inputs.
- **Clear Button**: Triggers a clear function that emits a clear event to the server, resetting all states and removing displayed text.
- **Speech Button**: Allows users to trigger the Text-to-Speech (TTS) component.
    - If there is transcription available, it emits a TTS event to the server and resets the states after the speech generation.
    - If “No” transcription is present, the button remains disabled.

As the results, the flowchart visually represents how these components interact, detailing the processes triggered by user actions and the corresponding application responses between client interactions with server which is business logic to perform recognition and translation tasks.

## 3. RESULT AND DISCUSSION

In this section, we will make a discussion about the results obtained from the Khmer sign language recognition Model and Web Interface. It provides an analysis of the model's performance and the overall functionality of the user interface, emphasizing their contributions to the project objectives.

### 3.1. Model Evaluation

The Khmer sign language recognition model was trained using a **TUF GTX 1650 Ti** graphics card with 4GB VRAM, involved a large dataset with 5,948 augmented sequences and involved splitting into “training set” and “validation sets”.

```
Data loading summary:
Total sequences loaded: 12552
Unique labels found: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]

Class distribution:
Class 0 (ខ្ញុំ): 553 sequences
Class 1 (អរគុណ): 570 sequences
Class 2 (ជំរាបសួរ): 578 sequences
Class 3 (ជួយ): 575 sequences
Class 4 (ចិញ្ចឹម): 579 sequences
Class 5 (ទៅ): 582 sequences
Class 6 (ញុំ): 595 sequences
Class 7 (ចាប់): 590 sequences
Class 8 (ទាំ): 590 sequences
Class 9 (ជួប): 593 sequences
Class 10 (ចូរចិត្តល្អ): 602 sequences
Class 11 (ចាស់ៗ): 606 sequences
Class 12 (ជីក): 610 sequences
Class 13 (ជូន): 595 sequences
Class 14 (ឈ្លក់ចិត្ត): 618 sequences
Class 15 (ញ៉ាំ): 610 sequences
Class 16 (លក់ដូរ): 621 sequences
Class 17 (ចង់រៀនគណិត): 611 sequences
Class 18 (មនុស្សល្អ): 622 sequences
Class 19 (ឥវ៉ាន់): 623 sequences
Class 20 (?): 629 sequences

Original data shapes:
Sequences shape: (12552, 30, 177)
Labels shape: (12552,)
Unique labels in original data: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]

Augmenting data...

Augmented data shapes:
Sequences shape: (18818, 30, 177)
Labels shape: (18818,)
Unique labels in augmented data: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]

One-hot encoded labels shape: (18818, 21)

Training/Validation split:
X_train shape: (15054, 30, 177)
y_train shape: (15054, 21)
X_val shape: (3764, 30, 177)
y_val shape: (3764, 21)
```

**Figure 3.1 Sequential Data loading summary**

The data loading summary in **Figure 3.1,** In this study, we loaded a total of 12,552 sequences, each comprising 30 time-steps and 177 features. For each step, 177 features are extracted using the Mediapipe holistic model. These features represent the 3D coordinates (x, y, z) of keypoints detected for the pose and hands, capturing the spatial positions of the body and hand movements essential for sign recognition. The dataset includes 21 unique classes, with a relatively balanced distribution across each class. To enhance the robustness of the model, data augmentation was performed, increasing the total number of sequences to 18,818. The labels were one-hot encoded, resulting in a 21-dimensional binary vector for each sequence. The augmented dataset was then split into training and validation sets, with 15,654

sequences allocated for training and 3,764 sequences for validation. This preparation ensures a comprehensive and balanced dataset for model training and evaluation.

### 3.1.1. Confusion Matrix

The confusion matrix is a fundamental tool for evaluating the performance of classification models. A confusion matrix is a tabular representation that summarizes the performance of a classification algorithm. It displays the actual classes (true labels) against the predicted classes (model's predictions).

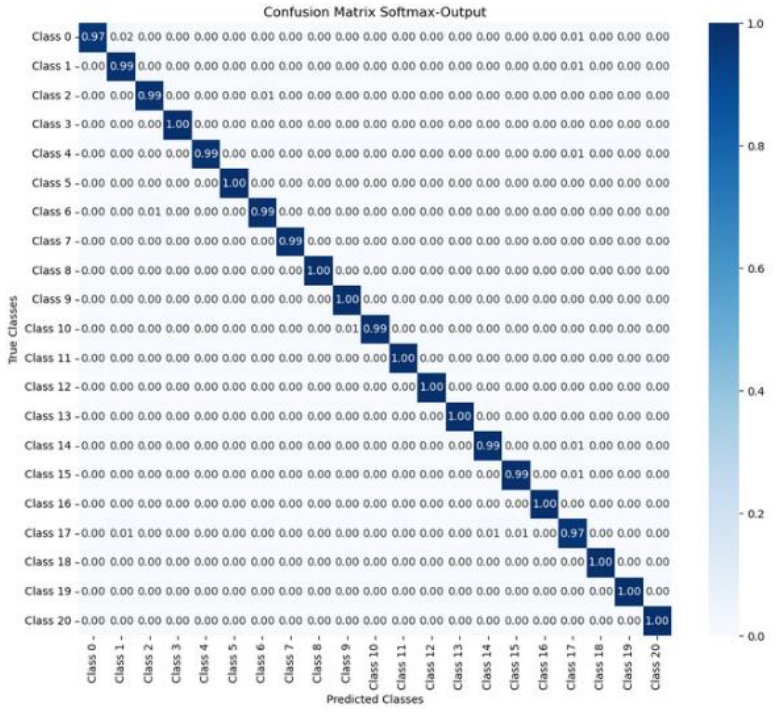


**Figure 3.2 Confusion Matrix Softmax Output**

As illustrated in **Figure 3.2**, each cell in the confusion matrix represents the number of instances where the actual class was classified as the predicted class. Since this study involves 21 classes (labeled 0–20), the confusion matrix is a 21×21 table. The diagonal cells (from top-left to bottom-right) indicate correctly classified instances, while off-diagonal cells represent misclassifications. By analyzing these misclassifications, we can identify which signs are frequently confused with each other. Potential reasons for these errors may include similar hand gestures, variations in lighting conditions, or dataset imbalance. However, in this case, the confusion matrix demonstrates strong performance, indicating that the model performs well in real-time recognition. This is evidenced by the high **softmax** probability values along the diagonal cells for each class.

### 3.1.2. Accuracy and Loss

The graphs you provided show the accuracy and loss of the model over epochs during training and validation:

- Training Accuracy: Starts around 0.5 (50%) and increases steadily over epoch. Reaches close to 1.0 (100%) by the end of training, indicating the model is learning well on the training.
- Validation Accuracy: Starts similarly around 0.5 but increases more slowly. Plateaus around 0.8 (80%) by the end of training, which is lower than the training accuracy.

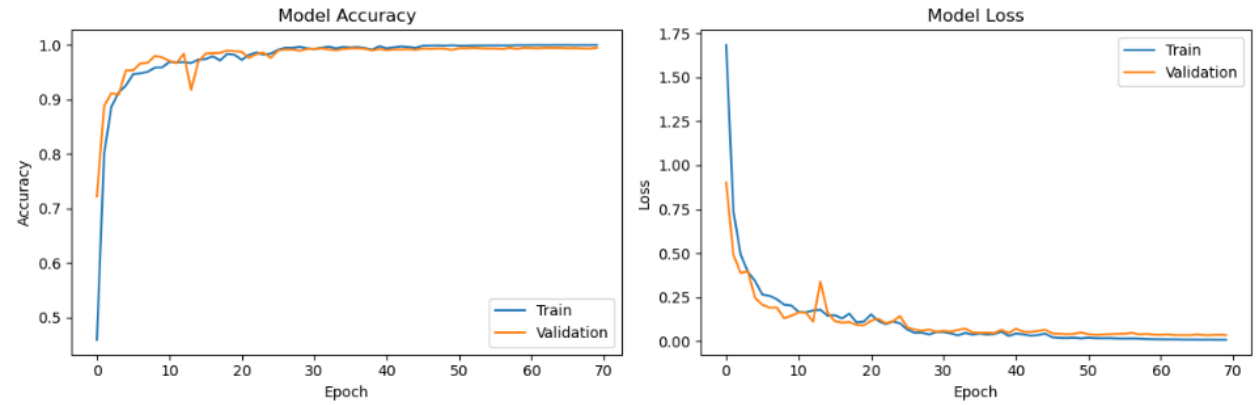


**Figure 3.3** Model Accuracy and Loss

### 3.1.3. Classification Reports

The classification report provides a detailed evaluation of the model's performance for each class in the dataset. It includes precision, recall, and F1-score, which are critical metrics for assessing the effectiveness of a multi-class classification model. As provided in **Figure 3.4** below, the model achieves high precision, recall, and F1-scores for all classes, indicating strong performance across the board. For instance:

- **Class 0:** Precision = 0.99, Recall = 0.97, F1-Score = 0.98.
    - This means that 99% of the predictions for Class 0 are correct (high precision), and 97% of the actual Class 0 instances are correctly predicted (high recall). The F1-score of 0.98 indicates an excellent balance between precision and recall.

- **Class 17:** Precision = 0.97, Recall = 0.98, F1-Score = 0.97.
    - While slightly lower than other classes, the performance is still strong, with a high F1-score indicating reliable predictions.

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Class 0 | 0.99 | 0.97 | 0.98 | 166 |
| Class 1 | 0.98 | 0.99 | 0.98 | 171 |
| Class 2 | 0.99 | 0.99 | 0.99 | 167 |
| Class 3 | 0.99 | 1.00 | 0.99 | 172 |
| Class 4 | 0.99 | 0.99 | 0.99 | 174 |
| Class 5 | 1.00 | 1.00 | 1.00 | 178 |
| Class 6 | 0.98 | 0.99 | 0.99 | 178 |
| Class 7 | 1.00 | 0.99 | 0.99 | 177 |
| Class 8 | 1.00 | 1.00 | 1.00 | 176 |
| Class 9 | 0.99 | 1.00 | 1.00 | 178 |
| Class 10 | 1.00 | 0.99 | 0.99 | 184 |
| Class 11 | 1.00 | 1.00 | 1.00 | 181 |
| Class 12 | 1.00 | 1.00 | 1.00 | 184 |
| Class 13 | 1.00 | 1.00 | 1.00 | 176 |
| Class 14 | 0.99 | 0.99 | 0.99 | 181 |
| Class 15 | 0.99 | 0.99 | 0.99 | 181 |
| Class 16 | 1.00 | 1.00 | 1.00 | 184 |
| Class 17 | 0.97 | 0.98 | 0.97 | 182 |
| Class 18 | 1.00 | 1.00 | 1.00 | 189 |
| Class 19 | 1.00 | 1.00 | 1.00 | 188 |
| Class 20 | 1.00 | 1.00 | 1.00 | 190 |
| accuracy | | | 0.99 | 3757 |

**Figure 3.4** Classification Report

### 3.1.4. Output layers and parameters

The LSTM-based model is designed to be computationally efficient while maintaining high accuracy in Khmer sign language recognition.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| lstm (LSTM) | (None, 30, 64) | 61,952 |
| dropout (Dropout) | (None, 30, 64) | 0 |
| lstm_1 (LSTM) | (None, 128) | 98,816 |
| dropout_1 (Dropout) | (None, 128) | 0 |
| dense (Dense) | (None, 64) | 8,256 |
| dense_1 (Dense) | (None, 21) | 1,365 |

**Total params:** 511,169 (1.95 MB)
**Trainable params:** 170,389 (665.58 KB)
**Non-trainable params:** 0 (0.00 B)
**Optimizer params:** 340,780 (1.30 MB)

```
Total inference time for 1000 samples: 1.6535 seconds
Average inference time per sample: 0.0017 seconds
Average inference time per sample: 0.0017 seconds
```

**Figure 3.5** Output Layer and Parameters of Model

The model consists of a total of **511,169 parameters**, which makes it relatively lightweight compared to larger deep learning architectures. The model processes 1,000 samples in just 1.65 seconds, which is extremely fast. It can handle high-throughput scenarios where multiple samples need to be processed quickly.

### 3.2. Interface

The interface is designed to provide real-time sign language recognition and translation. It allows users to perform Khmer Sign Language gestures, which are then recognized and translated into text or speech.

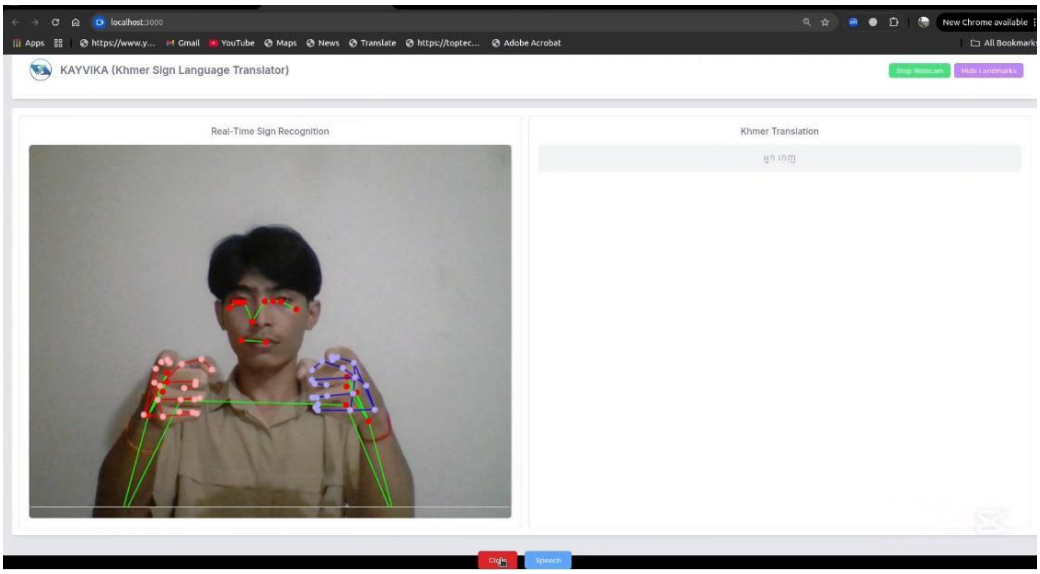


**Figure 3.6** Demonstration of Recognition Model Deployment

- Real-Time Performance:
    - The system processes input and provides translations in real-time, with an average inference time of 0.0059 seconds per sample run on CPU (for the quantized model).
    - Scalability: can be deployed on various platforms, including desktops, mobile devices, and embedded systems, the lightweight quantized model (0.35 MB).

# 4. CONCLUSION

## 4.1. Conclusion

This research explores the potential of integrating AI technology to bridge communication gaps for deaf and mute individuals. It demonstrates the feasibility of using deep learning models, specifically Long Short-Term Memory (LSTM), for real-time sign language recognition. Additionally, the model was successfully quantized, reducing its size by 89.4% (from 3.30 MB to 0.35 MB), making it highly suitable for deployment on edge devices.

## 4.2. Limitation

Despite the model's strong performance, certain limitations must be considered:

- The model was trained on a **limited dataset**, which may not encompass all variations of Khmer Sign Language gestures, potentially leading to misclassifications for unseen variations.
- Lack of diversity in the dataset could impact the model's ability to generalize across different users, environments, and lighting conditions
- When recognizing **complex gestures, fast hand movements, or occlusions**, which could hinder its ability to maintain consistent recognition accuracy.

## 4.3. Future Work

To further enhance the system, future work will focus on expanding the dataset to improve model accuracy and robustness. Additionally, experimenting with other neural network architectures could yield better performance or efficiency. The integration of Transformer models for word synthesis is also proposed to refine the recognition and generation of sign language. To enable seamless two-way communication, speech-to-text and text-to-speech functionalities will be integrated with sign animation, allowing both deaf/mute and hearing individuals to interact effectively. Finally, developing a mobile or web application will ensure easier deployment and broader accessibility, making the technology more user-friendly and widely available.

## 5. REFERENCES


[1] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, “Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7291–7299.

[2] J. Zelinka and J. Kanis, “Neural 3D Mesh Renderer for 2D-to-3D Mapping,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020, pp. 1–16.

[3] Kim, J., Kong, J., & Son, J. (2021). "VITS: Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech." *Proceedings of the 38th International Conference on Machine Learning (ICML)*

[4] Meta AI. (2023). "MMS: Massively Multilingual Speech." *Hugging Face Model Hub*. Retrieved from https://huggingface.co/facebook/mms-tts-khm