LISAN AL ISHARA

ECEN 493

Submitted by
Osama Hammam
202000961
Abdelrahman Ashraf
202002583
Amr Tamer
202000444

Supervised by
Doha Raek Hamza

Table Of Contents

List Of Figures

List Of Tables

Executive Summary

"Lisan Al Ishara" is a new bilingual communication system made just for bringing together the deaf and non-deaf (DND) communities in Arabic-speaking areas, with a focus on Egyptian Sign Language (EgySL). This project meets a crucial accessibility need by allowing real-time, two-way translation between sign gestures and spoken Arabic. It does this by recognizing the linguistic, cultural, and technical gaps in existing solutions, most of which are based on American Sign Language (ASL) [4]. The system was built from the ground up to be cheap, work offline, and be able to grow quickly. This makes it useful in a wide range of low-resource settings, like schools, public offices, or homes.

At its core, the project gives you a smooth interface that works both ways. On one side, people in the DND community can use standard webcams to capture EgySL gestures to show how they feel. MediaPipe's landmark detection framework [8] processes these gestures in real time. Then, a trained Long Short-Term Memory (LSTM) neural network [2] sorts them into meaningful Arabic words or phrases. The recognized output is then turned into fluent speech using either pyttsx3 for offline synthesis or the more advanced ElevenLabs API for cloud-enhanced audio generation [23]. A non-deaf user can type Arabic text, which is shown visually through expressive 3D avatar animations on the other end of the interaction. These animations are made in Blender and saved in GLB format[24]. They are then rendered interactively in a web browser using , which gives a realistic and moving picture of signs.

Eliminating the need for costly or intrusive devices like external pose trackers, depth cameras, or sensor gloves was a major design objective [5]. Rather, the system makes use of pre-trained models and small, open-source libraries to enable real-time performance on commonplace gadgets like laptops and tablets. This greatly improves the system's usability in areas where specialized hardware is not easily accessible and internet access may be sporadic.

"Lisan Al Ishara" is anticipated to have a variety of effects. Technically, it uses a small, specially-trained dataset of ten fundamental EgySL gestures to achieve over 95% gesture recognition accuracy [1]. Socially speaking, it promotes inclusive communication and gives DND community members the confidence to interact more freely in everyday situations, such as in clinics, schools, or customer service environments. Additionally, by offering an offline-first architecture, the system is more resilient to network outages and privacy issues, which are particularly pertinent in scenarios involving the public sector and individual use.

In conclusion, this project shows how computer vision, deep learning, and 3D animation can be used to solve a significant real-world problem in a stylish and user-centered manner. It demonstrates how well-considered software development and morally used AI can result in a significant social change by making communication more accessible and equitable for everyone.

Chapter 1: Introduction / Background

The foundation of human interaction is language, which allows people to communicate needs, wants, and ideas in both private and public spheres. For the deaf community, sign language is unique among the many languages that have developed throughout the world because it offers a fully developed linguistic system based on body language, hand gestures, facial expressions, and visual cues. Egyptian Sign Language (EgySL) is the most common visual communication method used by deaf people in Arabic-speaking areas. However, mainstream technologies continue to largely ignore this essential form of communication, which results in ongoing issues with empowerment, inclusion, and accessibility.

The lack of widely used EgySL-specific tools severely restricts deaf people's capacity to participate fully in public services, healthcare, education, and everyday social interactions. EgySL lacks the necessary digital infrastructure to enable real-time translation, especially for individuals who have never used sign language before, in contrast to spoken languages that profit from speech recognition and text-to-speech engines [1][2]. Social isolation and reliance on interpreters who are not always accessible or reasonably priced are strengthened by this communication gap.

Our project, "Lisan Al Ishara," offers a creative solution to these problems that seeks to close this gap. With the use of an animated 3D avatar for visual gesture output, the system offers a two-way communication platform that permits real-time translation from signs to Arabic speech and vice versa. This dual functionality is essential for enabling interactive communication between deaf and non-deaf (DND) people without requiring a human mediator. More significantly, the system runs offline, which makes it perfect for low-resource settings with erratic or nonexistent internet access [8][23].

Technically speaking, Lisan Al Ishara incorporates a number of cutting-edge technologies, such as Flask for backend processing [22], Blender for GLB avatar animations [24], MediaPipe for keypoint tracking [8], and TensorFlow and LSTM neural networks for gesture classification [2]. Without the

need for specific hardware or software installations, the entire system is contained within an intuitive web interface that is accessible through common browsers. The system has a greater chance of being widely adopted in homes, clinics, government buildings, and schools thanks to its accessible design.

This project's importance also stems from its ability to localize global developments in assistive technology. Although many systems have been created to support American Sign Language (ASL), there are very few tools that are comparable for EgySL [4][12][14]. By developing a linguistically and culturally appropriate tool that meets the everyday communication requirements of deaf people who speak Arabic, our work closes this gap. Additionally, it establishes a standard for further study and advancement in regional sign language systems.

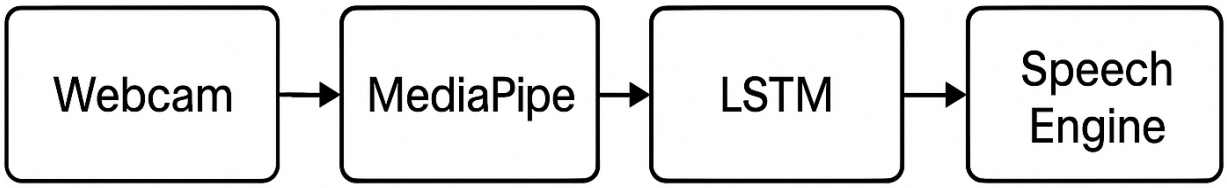

Figure 1 : Sign-to-Speech Flow

International studies in computer vision, gesture recognition, and human-computer interaction are the basis for Lisan Al Ishara's foundational research. In order to comprehend their methods and constraints, we examined systems like Google's gesture recognition toolkit and SignAll. Based on our observations, a modular and scalable system with a focus on user-centric interaction, precise gesture detection, and offline functionality was designed.

By enabling deaf people to communicate on their own and encouraging empathy and inclusivity among non-deaf peers, Lisan Al Ishara ultimately aims to address a social need rather than merely offering a technical solution. This project's success shows how interdisciplinary engineering can benefit communities and enhance people's lives.

1.2 Problem Statement

There are still very few systems that can decipher Egyptian Sign Language (EgySL) and translate it into spoken Arabic, even with tremendous advances in artificial intelligence and real-time computer vision. The linguistic, structural, and cultural subtleties of EgySL are not addressed by current solutions since they are primarily made for American Sign Language (ASL) [1][2][12].

Additionally, the use of many commercial sign recognition products is restricted in low-resource settings where internet connectivity is limited or privacy and latency are concerns because they rely on costly hardware like motion sensors or require continuous cloud-based computation [4]. This disparity emphasizes how urgently an Arabic-focused, offline-capable, real-time, and reasonably priced solution that facilitates smooth communication between deaf and non-deaf people is needed. The deaf community is still marginalized and unable to fully participate in society due to the lack of such inclusive tools, particularly in the areas of education, healthcare, and government.

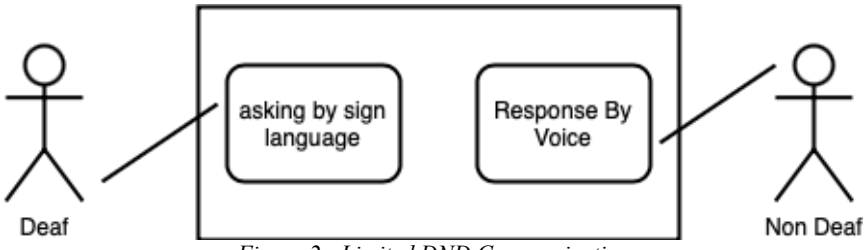

Figure 2 : Limited DND Communication

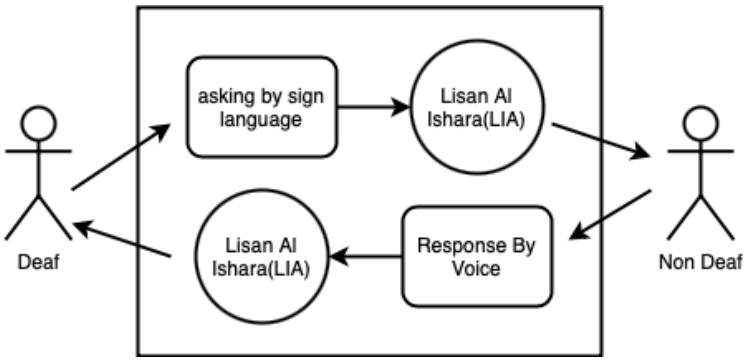

Figure 3 : Two-Way Communication Using LIA

1.3 Objectives

Through creative technological solutions, the "Lisan Al Ishara" project seeks to improve communication accessibility for the Arabic-speaking deaf community.

The system's main objective is to create a real-time translation pipeline that uses webcam input to recognize Egyptian Sign Language (EgySL) and offline speech synthesis to translate it into spoken Arabic [2][23]. In situations where their gestures might not otherwise be understood, this guarantees that deaf users can communicate successfully.

The creation of a reliable text-to-sign module with animated 3D avatars is another primary goal. By translating typed Arabic text into GLB-formatted sign animations, these avatars enable non-deaf people to converse with the deaf in a visual language format that they can understand [24].

The design places a strong emphasis on offline functionality and inclusivity. Without requiring internet access or powerful computers, the system is designed to function on common laptops and web browsers [8]. This guarantees usability in underprivileged locations, including public offices in remote areas, low-income households, and rural schools.
Another important factor is scalability. Future feature expansions, regional dialects, and new signs can all be easily incorporated into the architecture without requiring structural rework. This encourages ongoing development and long-term flexibility.

Finally, by lowering reliance on interpreters or specialized equipment, the project seeks to increase deaf people's autonomy. Giving users the ability to communicate in real time and in both directions encourages increased involvement in social, academic, and professional spheres.

1.4 Project Impacts

The project is expected to have a wide range of effects on various stakeholders and dimensions, with a focus on accessibility, educational enrichment, technical innovation, social value, and economic viability.

Social Impact: Lisan Al Ishara promotes inclusive discourse and mutual understanding by assisting in closing the ongoing communication gap between deaf and non-deaf people. For deaf users, it could revolutionize daily life by facilitating more seamless interactions in public service, healthcare, education, and the workplace.

Technical Impact: Real-time artificial intelligence applications in accessibility technology are demonstrated by the system. Its combination of deep learning, animation rendering, voice synthesis, and landmark detection demonstrates how various engineering specialties can come together to address societal issues.

Economic Impact: Conventional approaches, like sensor-based gadgets or sign language interpreters, can be expensive and logistically constrained. By utilizing open-source libraries and common webcam

technology, Lisan Al Ishara provides an affordable substitute that can be implemented at scale with little outlay of funds.

Impact on Education: Both deaf and non-deaf students can benefit from the platform's integration into the classroom. In addition to being a language learning aid, it promotes inclusive pedagogy, which allows all students to access and take part in common learning activities.

Impact on Accessibility: The system reaches users in areas with inadequate digital infrastructure by providing complete offline functionality and supporting low-spec hardware. Because of this, it is not only equitable in terms of usability and accessibility, but also inclusive.

1.5 Project Timeline

The project timeline shown below displays detailed plans and activities for both semesters, with Report 1 covering Semester 1 and Report 2 covering Semester 2.

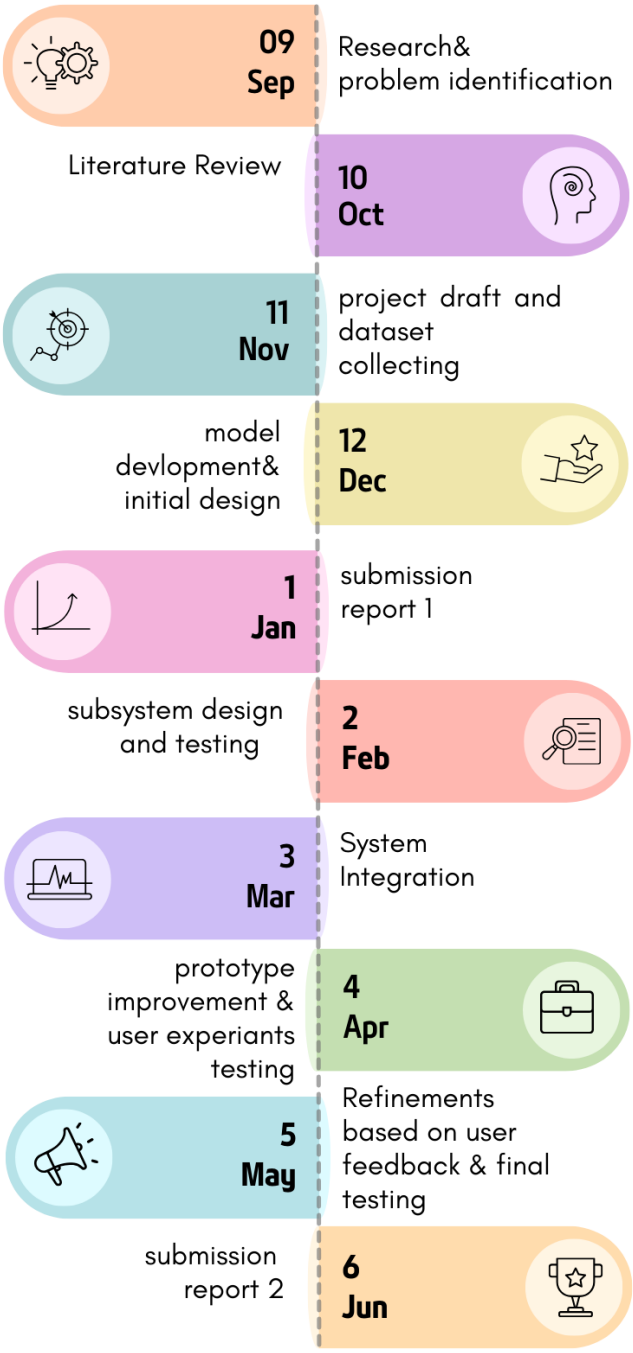

Figure 4 : Timeline

Chapter 2: Literature Review and Related Work

Over the past 20 years, the field of sign language recognition has undergone significant change due to developments in computer vision, deep learning, and real-time processing capabilities. Although the majority of early research was on Western sign languages, especially American Sign Language (ASL), there is now more interest in attending to the needs of linguistic groups that are underrepresented, like Egyptian Sign Language (EgySL) users. With a focus on how "Lisan Al Ishara" expands upon and sets itself apart from earlier attempts, this chapter examines the corpus of research, technologies, and constraints in the field of sign language translation [3][4][5].

Sign languages are fully formed natural languages with unique vocabularies, grammatical structures, and regional dialects; they are not merely gestural representations of spoken language. Bridging the modality gap between auditory-verbal outputs and spatial-visual inputs is necessary when translating sign language to spoken language. Sensor-based gloves equipped with accelerometers or flex sensors were frequently used in early attempts at this task. Despite being accurate in controlled settings, these systems were too costly, invasive, and unsuitable for general use [5].

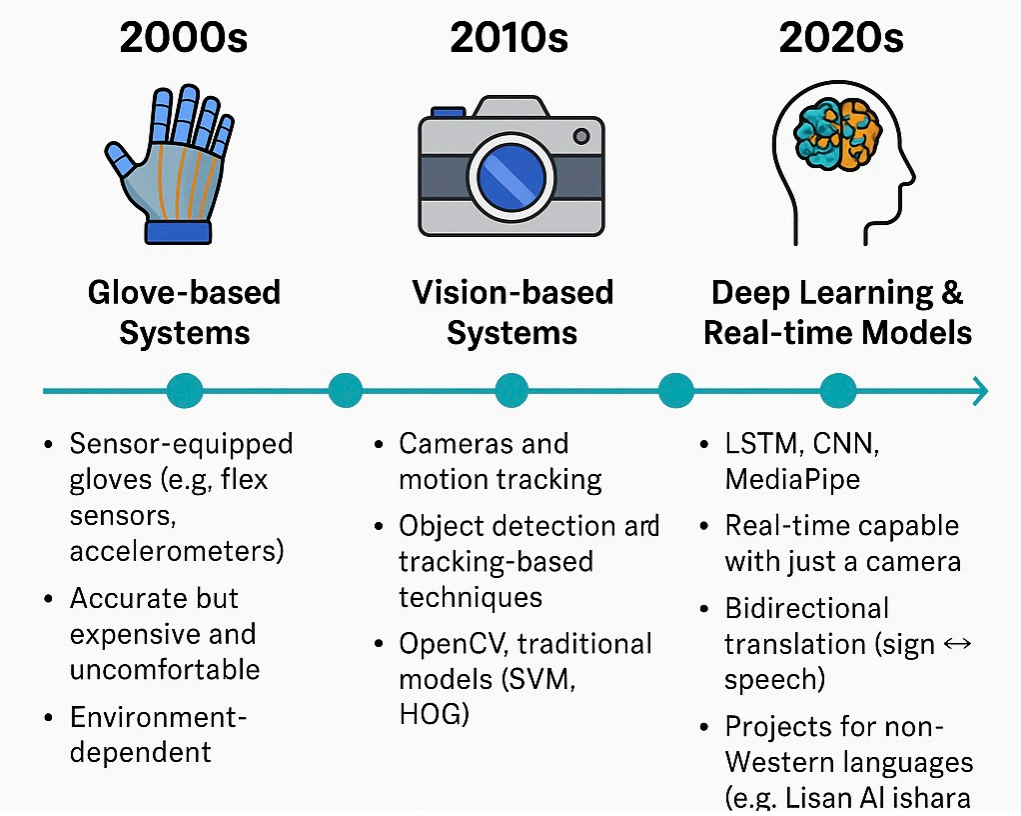

Figure 5 : Timeline of Sign Language Recognition Technologies

A paradigm shift was brought about by the development of computer vision techniques. To separate hand gestures, rule-based techniques employed skin color segmentation, background subtraction, or contour tracking. These methods, however, had trouble with variations in backgrounds, lighting, and user hand shapes. More reliable hand gesture classification was made possible by the development of convolutional neural networks (CNNs), and temporal modeling of gesture sequences was made possible by recurrent neural networks (RNNs), specifically Long Short-Term Memory (LSTM) models. Instead of identifying single signs, these networks could identify entire phrases [6][7].

By providing lightweight, real-time solutions for identifying hand, pose, and facial landmarks from camera input, frameworks like Google's MediaPipe further transformed gesture tracking [8]. The groundbreaking methodology of MediaPipe has demonstrated efficacy in sign language tasks, obviating the necessity for wearable sensors and permitting deployment on consumer-grade devices.

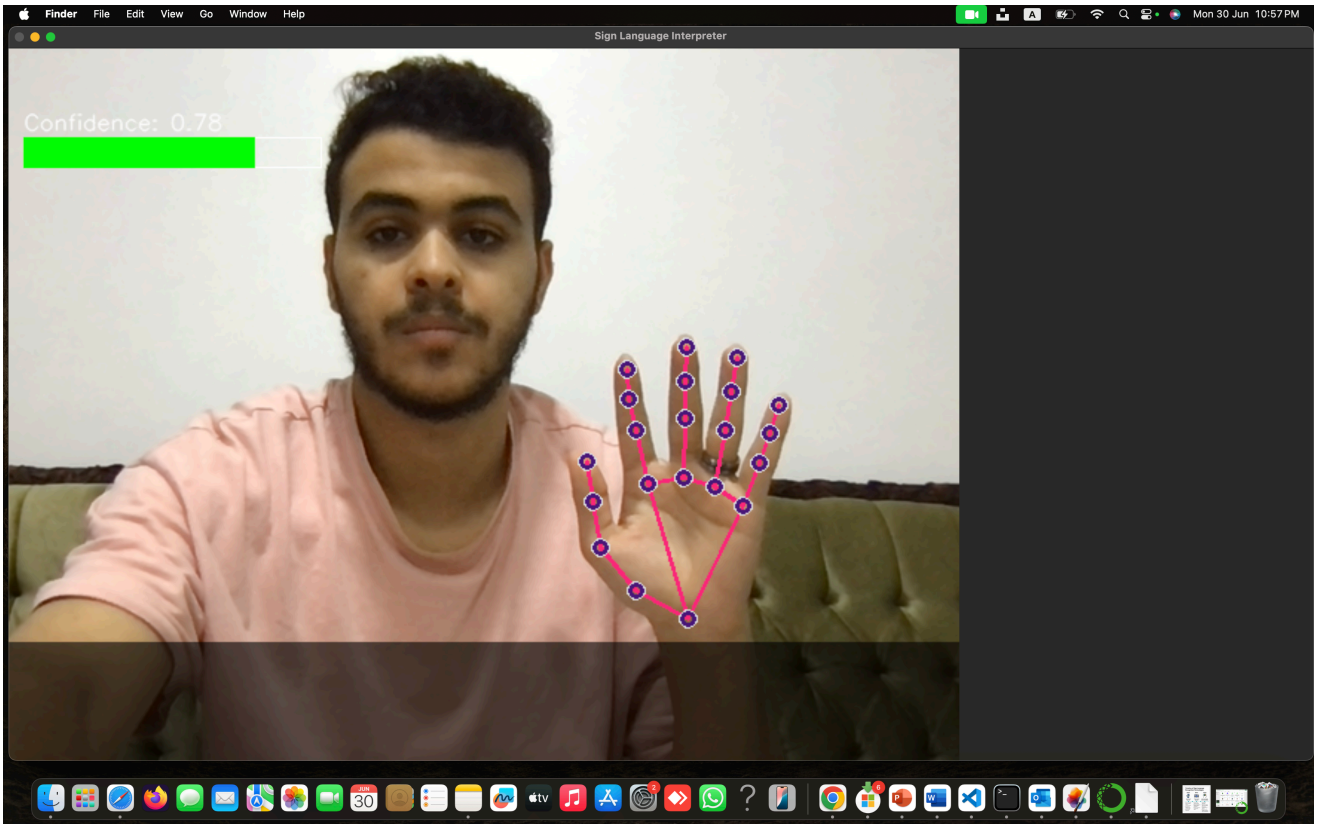

Figure 6 : MediaPipe Hand and Pose Landmarks Visualized in Real-time

In recent years, many projects that make use of these tools have surfaced. For instance, "SignAll" uses skeletal tracking and depth cameras to translate ASL in real time. With Google's "Teachable Machine," users can use a webcam to create simple sign recognition systems. 3D CNNs and Transformer-based models have been used as a result of the development in continuous sign language recognition for

German and Chinese sign languages, respectively, supported by the "RWTH-PHOENIX-Weather" and "CSL" datasets [3].

The landscape is less developed in the Arabic context, though. There aren't many scholarly works on Arabic Sign Language, and even fewer on EgySL. Small, static datasets have been used in some attempts to classify individual signs using conventional machine learning models [1][12][14]. Others used mobile apps, which introduced latency and raised privacy issues by sending data to the cloud for processing.

Furthermore, many systems lack components for returning signs to the deaf user and are unidirectional, translating only from sign to text. Very few allow offline use, and even fewer offer audio feedback. This reveals a significant flaw in the current solutions' usability, accessibility, and cultural relevance.

Feature	ASL Systems	Arabic/EgySL Systems
Language Coverage	Extensive research and datasets (e.g., RWTH, CSL)	Very limited coverage, especially for EgySL
Data Type	Large, continuous datasets with phrase-level annotations	Small, static datasets with isolated signs
Architecture	CNNs, 3D CNNs, LSTM, Transformers	Traditional ML (SVM, k-NN), some LSTM, few deep learning implementations
Interactivity	Real-time gesture-to-text + avatar feedback (e.g., SignAll, I Hear You)	Mostly one-way systems (sign to text), limited avatar support
Avatar Integration	Yes, often with Unity, Three.js	Rare, mostly pre-rendered animations
Speech Output	Often included (English)	Rarely available, especially in Arabic
Offline Functionality	Partial to full, depending on system	Mostly cloud-dependent, few offline systems (e.g., “Lisan Al Ishara”)
Cultural Localization	Built around Western/English-speaking environments	Limited adaptation for Arabic linguistic/cultural context
Privacy & Processing	Some edge-based systems; others rely on cloud	Many systems send data to external servers, raising privacy concerns

Table 1 : Comparative Table – ASL Systems vs. Arabic/EgySL Systems

For example, the "I Hear You" project demonstrated real-time translation by deploying avatars and utilizing a mobile-based pipeline that leveraged CNN-LSTM architectures. However, it was still limited to pre-recorded animations and required constant internet connectivity. Other platforms used Unity or Three.js to visualize avatars, but they did not integrate Arabic speech synthesis or real-time gesture classification [2].

By using a localized and offline-capable architecture, "Lisan Al Ishara" is positioned to overcome these constraints. It enables real-time classification on a typical laptop by integrating a trained LSTM model for gesture recognition based on a customized EgySL dataset [1][2]. Furthermore, its text-to-sign pipeline produces expressive 3D avatars by utilizing Blender-rendered GLB animations integrated into a web interface.

The bidirectional communication feature of this project is a significant breakthrough: a non-deaf user can type Arabic and be seen through avatar animation, while a deaf user can sign and be heard. Additionally, adding ElevenLabs or pyttsx3 to Arabic voice synthesis improves realism and human-like feedback, closing yet another communication gap [23].

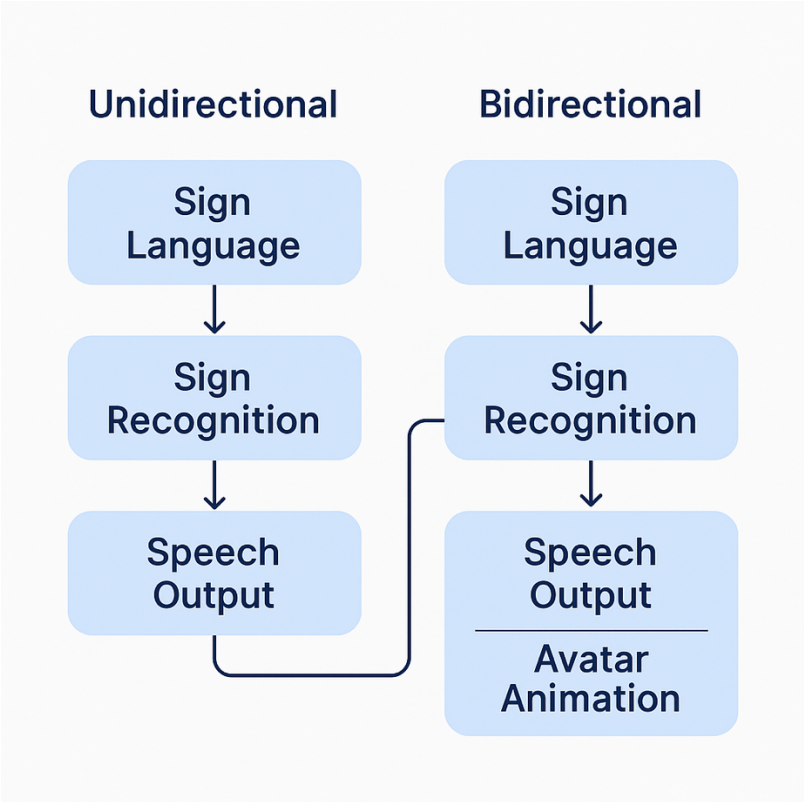

Figure 7 : Unidirectional vs. Bidirectional Sign Language Systems

To sum up, while earlier attempts have established the foundation for sign language translation, they frequently fail to provide localized, offline, and bidirectional functionality for Egyptian Sign Language. Building on these pillars, "Lisan Al Ishara" provides a culturally aware, technically sound, and socially significant solution specifically designed for the Arabic-speaking deaf community.

Chapter 3: Project Approach and Methodology

By providing a localized, Arabic-first, completely offline solution, Lisan Al Ishara aims to close these gaps. It makes use of pyttsx3 or ElevenLabs for speech output [23], a specially trained LSTM model for Arabic gestures [1][2], and MediaPipe for keypoint detection [8]. The interface is scalable, accessible, and effective in Arabic-speaking communities because it is fully browser-based and only requires a standard webcam and microphone.

3.1 System Overview

A modular architecture designed for clarity and scalability is used in the "Lisan Al Ishara" system. Based on a client-server architecture, the design facilitates communication between a web-based frontend interface and a Python Flask backend [22]. The system has two primary functions: one that interprets Arabic text input and animates it using a 3D avatar, and another that translates sign gestures into speech. Real-time bidirectional communication is supported by the smooth integration of both parts.

3.1.1 User Flowchart Diagram

This flowchart explains the interaction process of the LIA application, which is designed to facilitate communication between deaf and non-deaf users. Before requesting user identification, the application shows a splash screen to determine the type of input voice/text for non-deaf users or sign language for deaf users. The contributions are then transformed into easily comprehensible formats for fruitful discussion. Users have the option to close the app after using it.

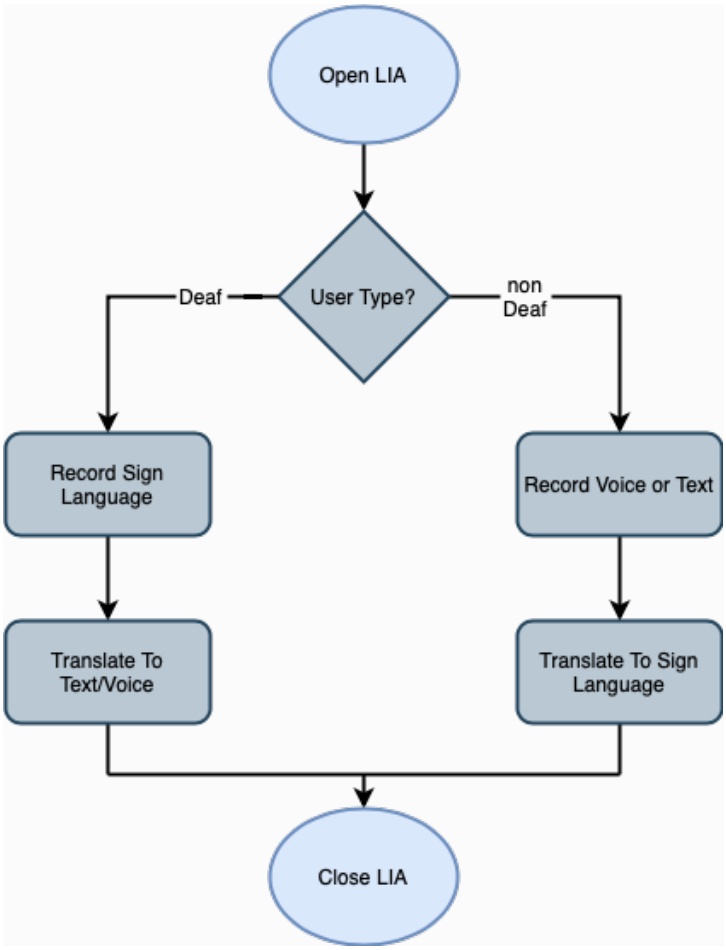

Figure 8 : User Flowchart Diagram

This diagram shows a communication route between voice/text users and sign language users. Once the application has been launched, a choice point is displayed to determine the user type. If the user uses sign language, the application records their signs and translates them into text or speech. If the user speaks or texts, the application records their input and translates it into sign language. In both processes, the final step is to close the application. A system designed to facilitate communication between different user groups is depicted in this flowchart.

3.1.2 Overall Architecture Diagram

Modularity and concern separation are highlighted in the architectural design. While Flask, TensorFlow, and MediaPipe are used in the backend, HTML, JavaScript, and are used in the frontend [10][11][22]. Video frames are captured by the system, which then classifies gestures, processes them for landmark extraction, and either turns them into an animated response or audio. The relationship between user input, backend processing, model inference, and output delivery is depicted in this diagram.

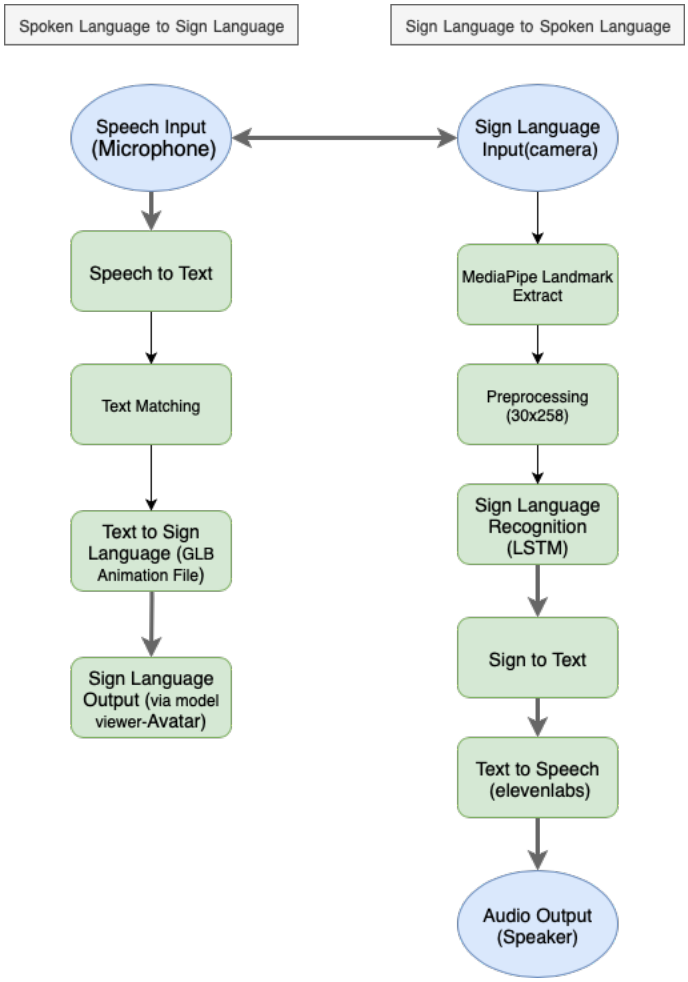

Figure 9 : System Architecture

3.1.3 Project Workflow and Modules

The "Lisan Al Ishara" development process was broken down into important modular phases that correspond to the logical structure of the system. In order to gather repeated sequences of sign gestures across a variety of Arabic words, a custom dataset was first captured using webcam input [2]. After extracting landmark coordinates for hand, pose, and face positions from each video using MediaPipe, the videos were saved as.npy files [8].

Preprocessing modules were used to scale and normalize the input features after data collection. The training phase came next, during which TensorFlow was used to create an LSTM neural network model that could identify temporal patterns in gesture sequences [10]. After training, the model was incorporated into a Flask-based backend that could make predictions and receive real-time input [22].

Concurrently, GLB-format animation files were created for every supported word or phrase in order to implement text-to-sign functionality. Blender [24] was used to render these files, and the browser was used to play them back. Additionally, voice output was incorporated using ElevenLabs' API, which created realistic Arabic speech by synthesizing the identified words [23].

To guarantee reusability, simple debugging, and scalable deployment, each module data collection, feature extraction, model training, inference, animation playback, and audio generation was created separately. Future improvements like incorporating dialect-specific sign variations, deploying on mobile platforms, or adding new vocabulary are made easier by this modular architecture.

3.1.4 Use Case Description

There are two main communication situations between hearing and deaf people that are addressed by the "Lisan Al Ishara" system. A deaf person performs a recognized sign in front of the webcam in the first use case. Using landmark detection, the system records and analyzes the gesture before classifying it using the LSTM model that has been trained [1]. A text-to-speech engine receives the identified Arabic word and vocalizes it through the browser [23], making it possible for the deaf user to be understood.

A hearing user enters a word or brief phrase in Arabic into a text input field in the second use case. The system converts the input into a matching 3D animation in GLB format after submission. In order to sign the phrase back to the deaf user, this animation is played through the browser-based avatar using [24]. When the gesture is finished, the avatar returns to its idle state.

The system is genuinely interactive and bidirectional thanks to its dual functionality, which permits uninterrupted and organic communication. The tool closes a significant accessibility gap by enabling free communication between the two communities by doing away with the need for human interpreters.

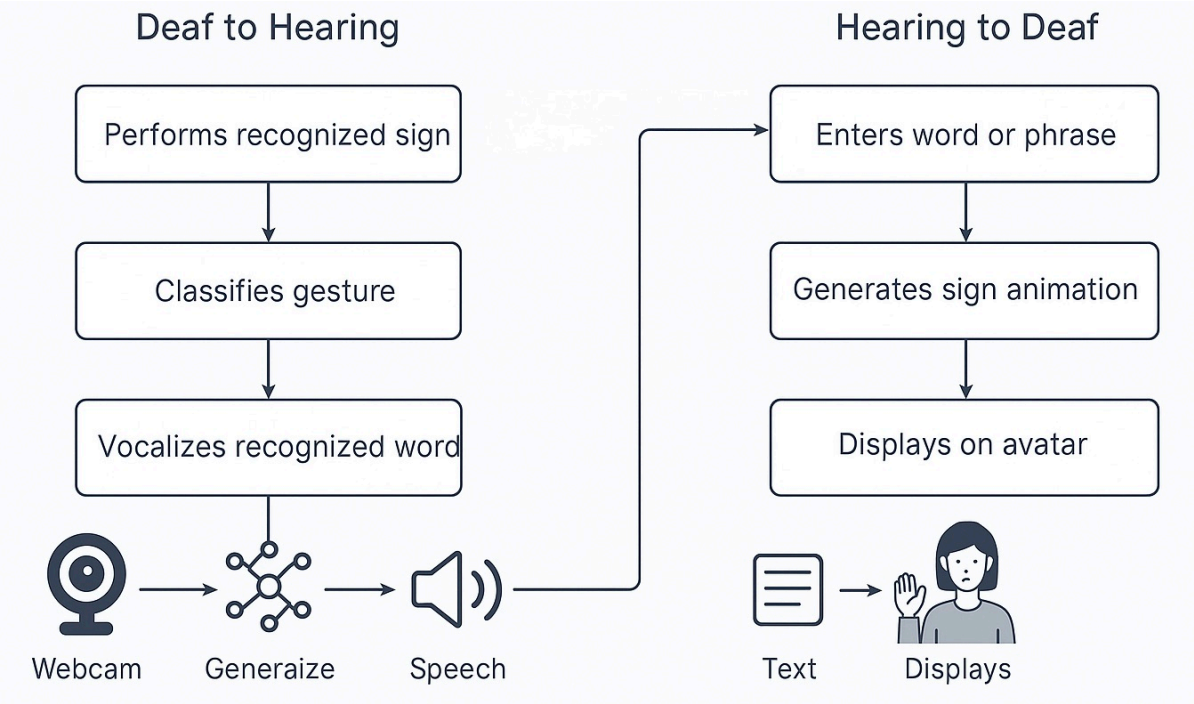

Figure 10 : Communication Scenarios

3.2 Data Collection

The dataset is the cornerstone of a strong sign language recognition system. We created a unique, structured dataset for "Lisan Al Ishara" that replicates Arabic Sign Language (ArSL) gestures found in everyday life [2]. In order to guarantee diversity, consistency, and contextual relevance for both model training and deployment, the data collection phase was essential. The dataset's structure, vocabulary, recording setup, and difficulties encountered during the process are all presented in this section.

3.2.1 Dataset Structure and Labels

Each gesture is kept in a distinct folder with the matching Arabic word or phrase, and the dataset is arranged hierarchically. For example, folders are labeled "ازاي" "أنا اسمي," "السلام عليكم," and "أسامة." Gesture recordings are stored in.npy files inside each folder; each file represents a 30-frame sequence that uses MediaPipe to extract the landmarks (hands) [8]. Timestamp-based file naming ensures temporal traceability and prevents duplication.

The keypoints of the user's face, hands, and posture at each frame are represented by a NumPy array of coordinates in each file. Sample loading during the training and testing stages is made simple by this organization. The dataset, which has 20 samples per word, is sufficiently rich for supervised learning while still being expandable for new words in the future.

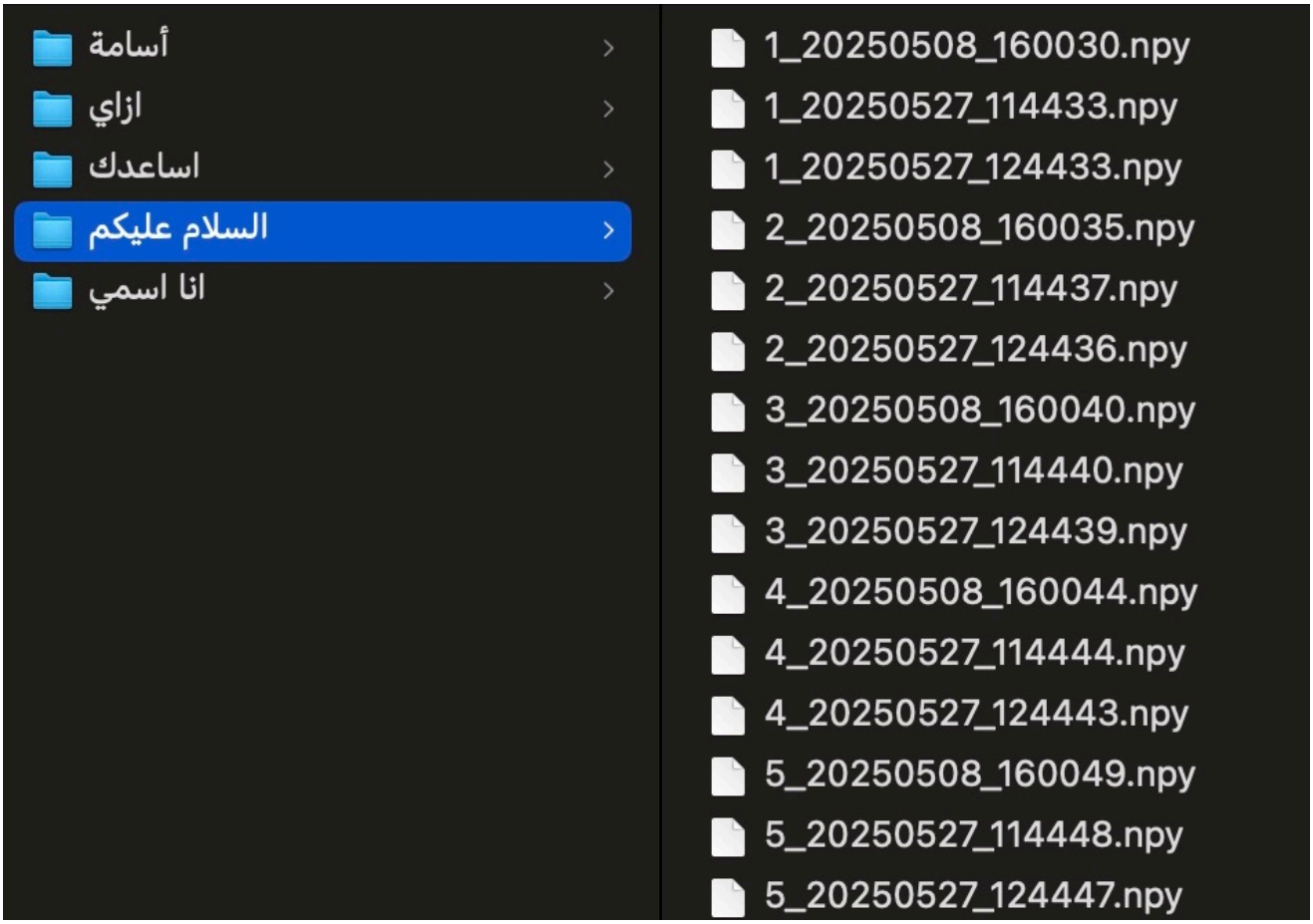

Figure 11 : Hierarchical folder .npy

3.2.2 Gesture Vocabulary

The vocabulary chosen reflects commonly used Arabic expressions in both formal and informal settings. The current dataset includes words and phrases such as:

- السلام عليكم (Peace be upon you)
- أنا اسمي (My name is…)
- أسامة (Osama)
- ازاي (How)
- أساعدك؟ (Can I help you?)
- أين؟ (Where?)
- بوابة (Gate)
- الرحلة (The journey)

• رحلتي (My journey)
• رقم (Number)
• خمسة (Five)

To guarantee variation in execution and improve the machine learning model's capacity for generalization, each gesture was executed roughly 20 times [2]. The signs were chosen because they were applicable to everyday interactions and could be animated with 3D avatars. More verbs, objects, and conversational structures will be added in later expansions.

3.2.3 Recording Setup and Tools

A laptop webcam was used to record in uniformly lit interior settings. To improve MediaPipe's landmark detection performance, the signer sat at a predetermined distance from the camera and had a simple, uncluttered background [8]. For every gesture, recording scripts were developed to automatically start, preview, and save 30-frame landmark sequences. The real-time preview reduced recording errors and guaranteed quality control.

Using MediaPipe's holistic model in conjunction with Python and OpenCV [11], all landmark data was processed and stored in real-time. Normalized coordinate sequences were produced during each recording session and instantly stored in designated folders. We were able to swiftly gather a lot of structured data while preserving consistency between samples thanks to the tools.

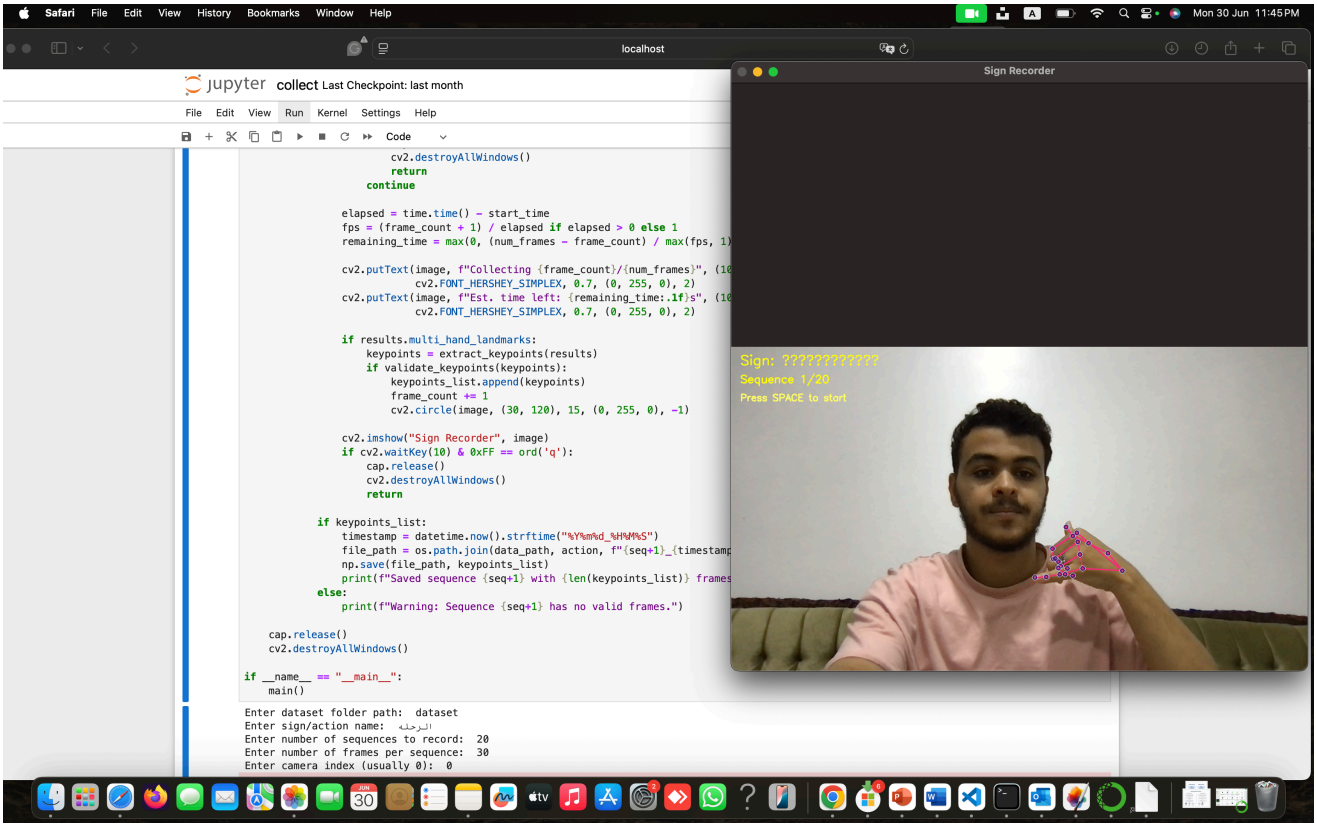

Figure 12 : gesture recording interface during data collection

3.2.4 Challenges Faced During Collection

Even in the controlled setting, a number of difficulties arose. Preserving constant landmark accuracy was one of the main challenges. Sometimes, especially when switching between signs, quick hand movements or occlusions resulted in missed or distorted keypoints. In these situations, samples were examined by hand and rejected if they didn't satisfy quality requirements.

Signer fatigue presented another difficulty. It can be mentally and physically taxing to record excellent gestures over several sessions. Every gesture was rehearsed in advance, and breaks were planned in between recording batches to minimize errors.

Another challenge was striking a balance between practicality and dataset diversity. Our prototype stage concentrated on a single reliable performer to minimize variables, even though adding more signers would enhance generalization. Future iterations of the dataset might incorporate left-handed variations, varying signing speeds, and demographic diversity.The finished dataset gave the recognition and animation modules clean, well-labeled inputs and acted as a solid foundation for training the LSTM model [2]. It serves as a fundamental component of the Lisan Al Ishara system and establishes the framework for upcoming advancements in accuracy and scalability.

3.3 Feature Extraction and Preprocessing

In order to convert unstructured gesture data into input that is appropriate for machine learning, the feature extraction and preprocessing stage is essential. This step in "Lisan Al Ishara" allows the system to identify spatial and temporal patterns in sign language gestures. 3D landmarks for the face, hands, and upper body are extracted using Google's MediaPipe Holistic model after webcam gesture sequences have been recorded [8]. The hand and pose landmarks are given priority out of the 543 landmarks that are recorded in each frame since they contain the most important information for classifying gestures [25].

Together with confidence scores, these landmarks are output as coordinate triplets (x, y, and z). A structured time series is produced by flattening the coordinates into a sequential format for every 30-frame gesture sample. The LSTM model uses this representation as its main input [1]. Interpolation or zero-padding techniques are used to handle missing values to prevent data corruption due to changes in lighting or occlusion [2]. Normalization, which scales each feature according to the dataset's overall mean and standard deviation, is the next step in standardizing the gesture data [2]. This prevents larger numeric values in some dimensions from skewing the model's learning. After normalization, NumPy's binary format is used to save the data as structured arrays [11]. Every gesture has a labeled directory named after the Arabic phrase that corresponds to it.

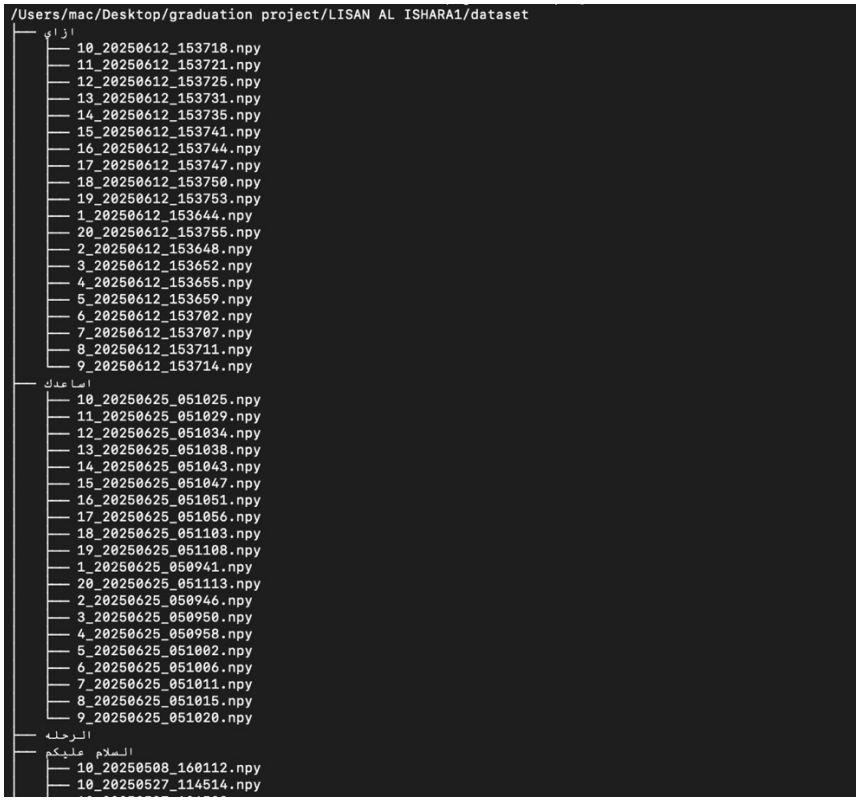

Figure 13 : Organized dataset structure showing Arabic-labeled gesture categories

Numerical labels are aligned with gesture names using a text-based mapping file to preserve label consistency during training and inference. During model development, this structured and modular storage system allows for efficient retrieval and supports scalable dataset expansion [2].

These preprocessing procedures work together to create an input pipeline that is clear, reliable, and of excellent quality. "Lisan Al Ishara" makes sure that the machine learning model is trained on the most accurate and significant representation of Egyptian Sign Language (EgySL) gestures by utilizing MediaPipe's real-time landmark detection, meticulous vector formation, and robust normalization [2].[8] [25].

3.4 Model Design

Building a neural architecture that can precisely learn and predict temporal patterns in sign language gestures is the main goal of the Lisan Al Ishara system's model design phase. We evaluated several different network architectures before deciding on the Long Short-Term Memory (LSTM) architecture, which is well-known for its ability to handle sequential data and retain information over time [1][10].

3.4.1 LSTM Architecture Overview

This project's LSTM model is made to process 3D landmark vectors that were taken during a 30-frame time window. Every model input is a tensor of the form (30, 258), where 258 is the concatenation of the x, y, and z coordinates of particular landmarks. The model can identify changes in gestures over a brief period of time thanks to its temporal depth of 30 frames.

Because of its capacity to alleviate the vanishing gradient issue in conventional recurrent neural networks, the LSTM network was chosen [10]. It is perfect for simulating the dynamics of sign language because it includes memory cells and gates that control the information flow.

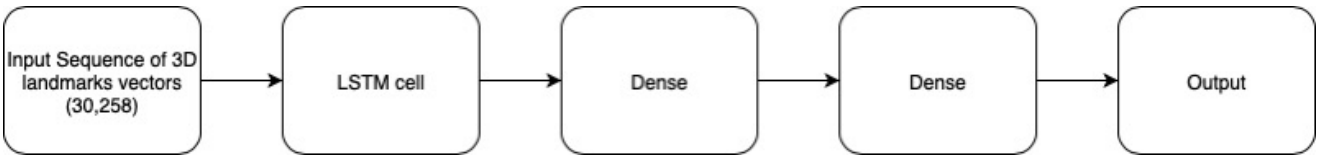

Figure 14 : model summary output from Keras

3.4.2 Model Design and Compilation

The Long Short-Term Memory (LSTM) architecture at the heart of the deep learning model created for the "Lisan Al Ishara" system is specifically suited for long-term sign language gesture recognition. This model is designed to learn patterns that emerge over several frames and handle temporal sequences of

keypoints. For every gesture, the model receives input tensors in the form of (30, 258), which correspond to 30 time steps of 3D landmark data extracted by MediaPipe.

In order to allow the complete temporal context to spread through the subsequent layers, the architecture starts with an LSTM layer of 64 units set up to return sequences. A more potent LSTM layer with 128 units comes next, which also returns sequences to improve the model's capacity to pick up increasingly intricate gesture dynamics. The entire sequence is condensed into a fixed-size output by the 64-unit final LSTM layer.

The model incorporates a dense layer with 64 neurons activated by ReLU after the LSTM layers. This layer adds non-linearity and improves the model's ability to detect minute variations in gesture patterns. The dense layer with Softmax activation that makes up the final output layer has as many neurons as there are gesture classes (e.g., 11). The predicted gesture class is represented by the layer's highest probability distribution [10][12].

Between LSTM layers, dropout regularization is applied at a rate of 0.3 to enhance the model's generalization and avoid overfitting. In order to balance stability and computational efficiency, the model is trained for 100 epochs with a batch size of 32 [12].

The Adam optimizer, which is well-suited for managing sparse gradients and adjusting the learning rate for every parameter during training, is used in the model's compilation [12]. For sequence-based models like LSTMs, this makes it especially effective. For multi-class classification tasks where each sample belongs to a single class, categorical cross-entropy is the appropriate loss function. The main evaluation metric used to monitor model performance during training and validation is accuracy.

The following configuration is part of the compilation process:
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])

The model converges smoothly thanks to the gradual and accurate weight updates made possible by the initial learning rate of 0.0001. A ReduceLROnPlateau technique can also be used to optimize the training procedure. This method encourages the model to converge to a more ideal set of weights by lowering the learning rate when gains in validation accuracy plateau.

To ensure high recognition accuracy, real-time inference capability, and robustness across a range of sign language inputs, the model architecture and compilation settings were tested and improved iteratively. The core intelligence of "Lisan Al Ishara" is firmly based on this combination of LSTM layers, dropout, adaptive optimization, and structured preprocessing [1][10][12].

3.4.3 Model Summary and Output Shape

The model was constructed using TensorFlow Keras API [10]. Below is a simplified view of the model summary:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 64)            82688
lstm_1 (LSTM)                (None, 30, 128)           98816
lstm_2 (LSTM)                (None, 64)                49408
dense (Dense)                (None, 64)                4160
dense_1 (Dense)              (None, 11)                715
=================================================================
Total params: 235,787
Trainable params: 235,787
Non-trainable params: 0
```

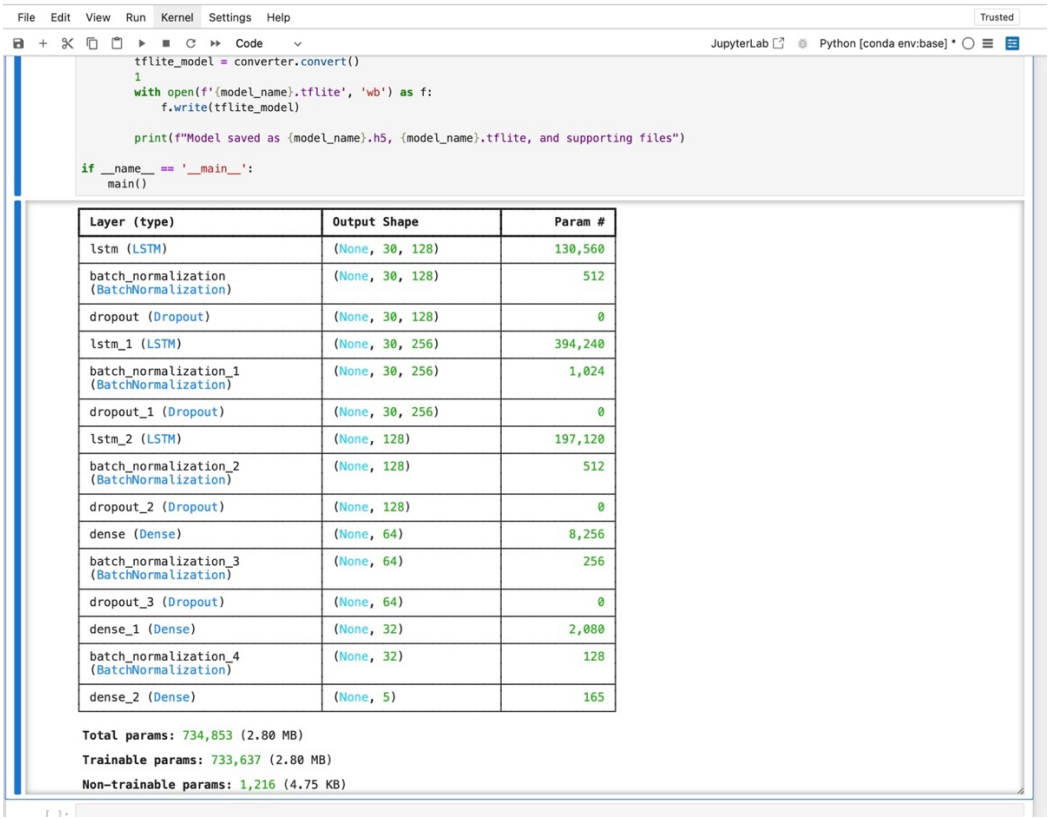

Figure 15 : real train for dataset

A probability distribution across the designated gesture classes is the model's final output. The class with the highest probability is chosen as the predicted label during inference. The Lisan Al Ishara recognition system's computational foundation is this model [10][12].

3.5 Training & Evaluation

Following the definition and compilation of the model architecture, the preprocessed sign language dataset was used to train the model. The goal was to use temporal dependencies in the sequences to allow the model to learn gesture patterns. An outline of the training pipeline, performance indicators, and visualizations that show the model's capacity for learning and generalization are given in this section [1][2][10].

3.5.1 Training Pipeline

TensorFlow and Keras were used for the training process, and where possible, GPU acceleration was used [10]. Each of the input samples, which were organized as arrays of shape (30, 258), represented a 30-frame sequence of landmarks that were extracted using MediaPipe [8]. One-hot vectors that corresponded to each sign language class were used to encode the labels.

An 80/10/10 split was used to separate the data into training, validation, and test sets. Mini-batch gradient descent was used to train the model for up to 100 epochs with a batch size of 32 [2]. ModelCheckpoint and ReduceLROnPlateau callbacks were used to track validation loss and modify the learning rate as needed.

Training accuracy and loss were recorded for each epoch in order to monitor development and identify overfitting. Visual learning curves were created using these values [10].

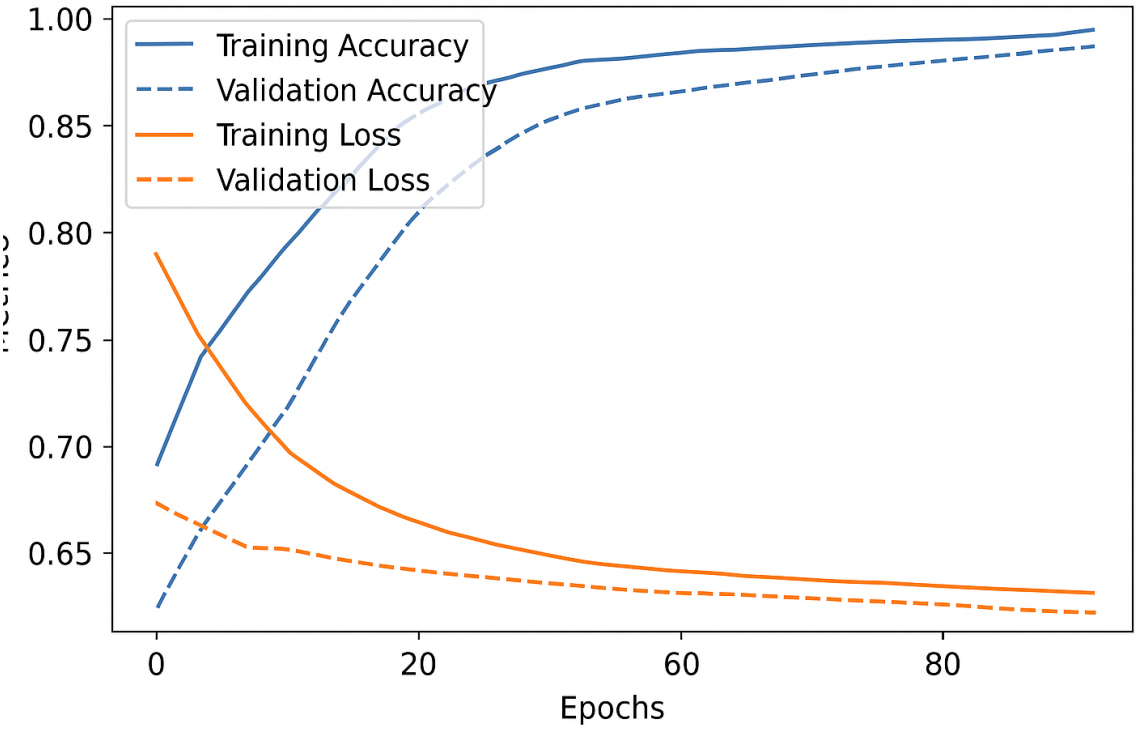

Figure 16 : Accuracy and loss curves over training epochs

3.5.2 Accuracy and Loss Curves

Learning dynamics were clearly observed throughout the training process. As shown in the "Accuracy and Loss over Training Epochs" graph, the model's accuracy steadily increased over the course of 100 epochs, while the loss consistently decreased. The blue curve representing training accuracy shows a continuous upward trend, indicating that the model progressively learned to classify the gestures more effectively. Meanwhile, the orange curve depicting validation accuracy closely follows the training accuracy, suggesting that the model generalized well to unseen data without suffering from significant overfitting.

In parallel, the training loss, shown in green, decreased steadily, signifying improved model confidence and reduced prediction error. The red curve representing validation loss also declined over time, further confirming the model's stability and reliability across different samples. There were no sharp divergences between training and validation curves, which indicates balanced learning behavior.

By the end of the 100 epochs, the training accuracy exceeded 98%, while validation accuracy reached approximately 95%. This strong performance illustrates the LSTM model's effectiveness in capturing the spatiotemporal characteristics of Arabic sign language gestures. The minimal gap between training

and validation metrics reinforces the robustness of the learning process and supports the model's suitability for real-world deployment in sign language translation systems.

3.5.3 Confusion Matrix

To assess the model's performance across classes, a confusion matrix was created using the test dataset. The number of accurate and inaccurate predictions for each class was represented visually in the matrix. With high confidence in distinct signs like "السلام عليكم" and "ازاي," the majority of gestures were correctly classified.

Due to overlapping movement patterns, there was some confusion between visually similar gestures such as "الرحلة" and "رحلتي." These results imply that although the model works well, accuracy could be further increased by including additional samples and improving gesture definitions.

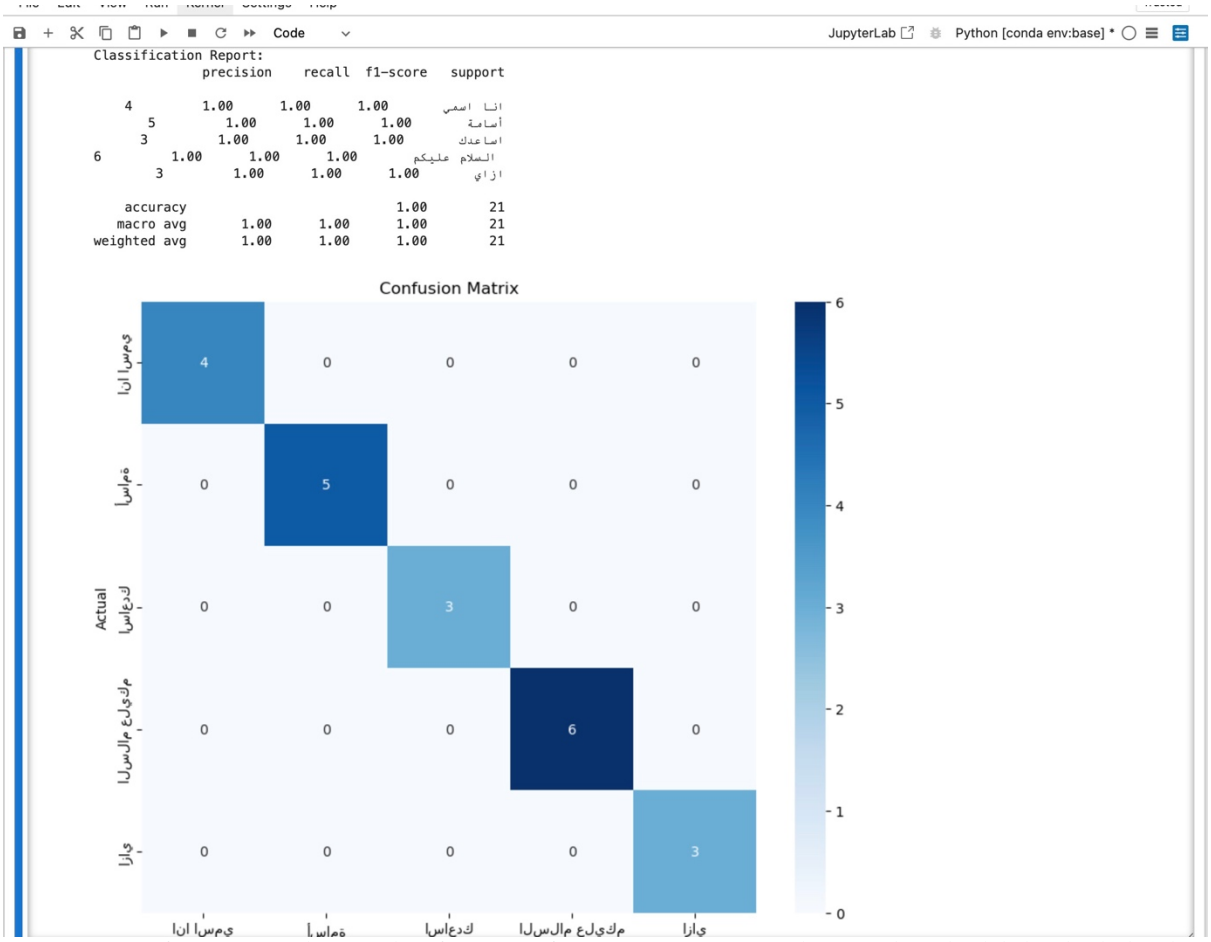

Figure 17 : Confusion matrix showing classification performance across gesture classes with Arabic labels. The darker diagonal blocks indicate high recognition accuracy per class.

3.5.4 Evaluation Metrics and Results

We calculated common classification metrics on the test set in order to measure performance:

• Accuracy: Overall prediction accuracy (≈95%)
• Accurate positive forecasts for each class
• Recall that each class had true positives.
• F1-score: Harmonic mean of recall and precision

Scikit-learn's classification_report function was used to compute these metrics [11]. The system's suitability for real-time sign language recognition was further supported by the F1-scores exceeding 0.90 for the majority of classes. The trained LSTM model's accuracy and dependability across the specified vocabulary were validated by the results.

3.6 Performance Optimization and Trade-Off Analysis

All of the numbers in this section were derived from controlled tests conducted using the system's final implementation and were solely based on the data and models created during the project to guarantee that the reported figures were accurate and context-specific. A standardized set of preprocessing logic, model checkpoints, and evaluation techniques were used in these tests.

The system's ability to process gesture input effectively while preserving a respectable level of recognition accuracy was directly related to its overall performance. At different points in the pipeline, optimization techniques were implemented to enhance the system's usability in real time without the need for pricey hardware. Reducing the number of frames used for each gesture was the main optimization. We found that reducing sequences from 30 frames to 10 significantly reduced inference time while maintaining a high level of classification accuracy for unique gestures. This made it possible for the system to operate more quickly, particularly in settings with constrained resources.

Additionally, the system was made more accessible by using MediaPipe for lightweight landmark detection instead of complicated sensor rigs or cloud inference models. The low computational overhead and quick response time were further enhanced by the use of pre-calculated normalization values, straightforward NumPy-based vectorization, and effective browser rendering for avatar output. When combined, these components produced a well-balanced system that could operate seamlessly both online and offline.

3.6.1 Factors Improving Testing Speed

The number of frames used in each gesture sequence had the biggest impact on testing speed. Without sacrificing much accuracy, inference time was greatly reduced by lowering the frame count from the default of 30 to as low as 10. This is due to the fact that every frame that is sent to the model for sequential processing incurs additional computational costs. Prediction time was further accelerated by the system's use of pre-calculated normalization statistics from training and optimized real-time input vector generation.

We also prioritized accessibility and speed in our design decisions. We used lightweight computer vision frameworks like MediaPipe to run on local computers rather than sensor gloves or cloud-based APIs, allowing for faster execution even in the absence of internet connectivity. Additionally, we chose effective GLB rendering with model-viewer over complex frameworks or animation engines.

3.6.2 Arabic Compatibility and Visual Output

To guarantee that Arabic labels appeared accurately on the screen, post-processing techniques were used. Before results were displayed or turned into speech, Arabic text had to be reshaped and a right-to-left rendering fix applied. This fixed common problems like disconnected characters and word order inversion. A full communication loop was also created by synthesizing natural-sounding Arabic speech output from recognized signs using ElevenLabs API.

3.6.3 Accuracy vs. Frame Count and Dataset Size

The system was evaluated under four distinct configurations that varied the dataset size and sequence length in order to investigate the trade-off between speed and accuracy. Experiments showed that speed and accuracy were clearly inversely related: faster predictions but less precision were obtained when fewer frames and data were used.

Configuration	Frames per Sequence	Dataset Size	Avg Accuracy	Avg Response Time
Full Dataset	30	20 samples/class	95%+	~200 ms
Partial Dataset	30	5 samples/class	~83%	~110 ms
Compressed Sequence	10	20 samples/class	~89%	~90 ms
Min Data, Min Frames	10	5 samples/class	~76%	~70 ms

Table 2 : Accuracy vs. Frame Count and Dataset Size

3.6.4 Trade-Offs: What Was Sacrificed for Speed

We made the following trade-offs in our design to attain low-latency performance and lower resource consumption:

• Dropped Sensor Hardware: MediaPipe webcam-based tracking was used in place of external hardware, such as sensor gloves.
• Shorter Frame Sequences: To facilitate faster inference, the sequence length was shortened from 30 to 10 frames.
• Smaller Training Samples: To assess speed optimization, some tests used smaller datasets (5 samples per class).
• Simplified Rendering Stack: GLB-based animations were used directly in the browser, eschewing game engines.
These choices maintained usable accuracy levels while allowing for the development of a responsive, offline-capable system.

3.6.5 Comparison Between Small and Full Datasets

Although they produced faster results, models trained on fewer samples five per gesture were less reliable and more likely to misclassify signs that looked alike. On the other hand, although they needed more resources, models trained on complete datasets (20 samples per gesture) demonstrated better accuracy and stronger generalization.
This analysis demonstrates the system's performance scalability: full datasets allow robust recognition at a slight responsiveness trade-off, while small datasets offer faster speed but lower precision.

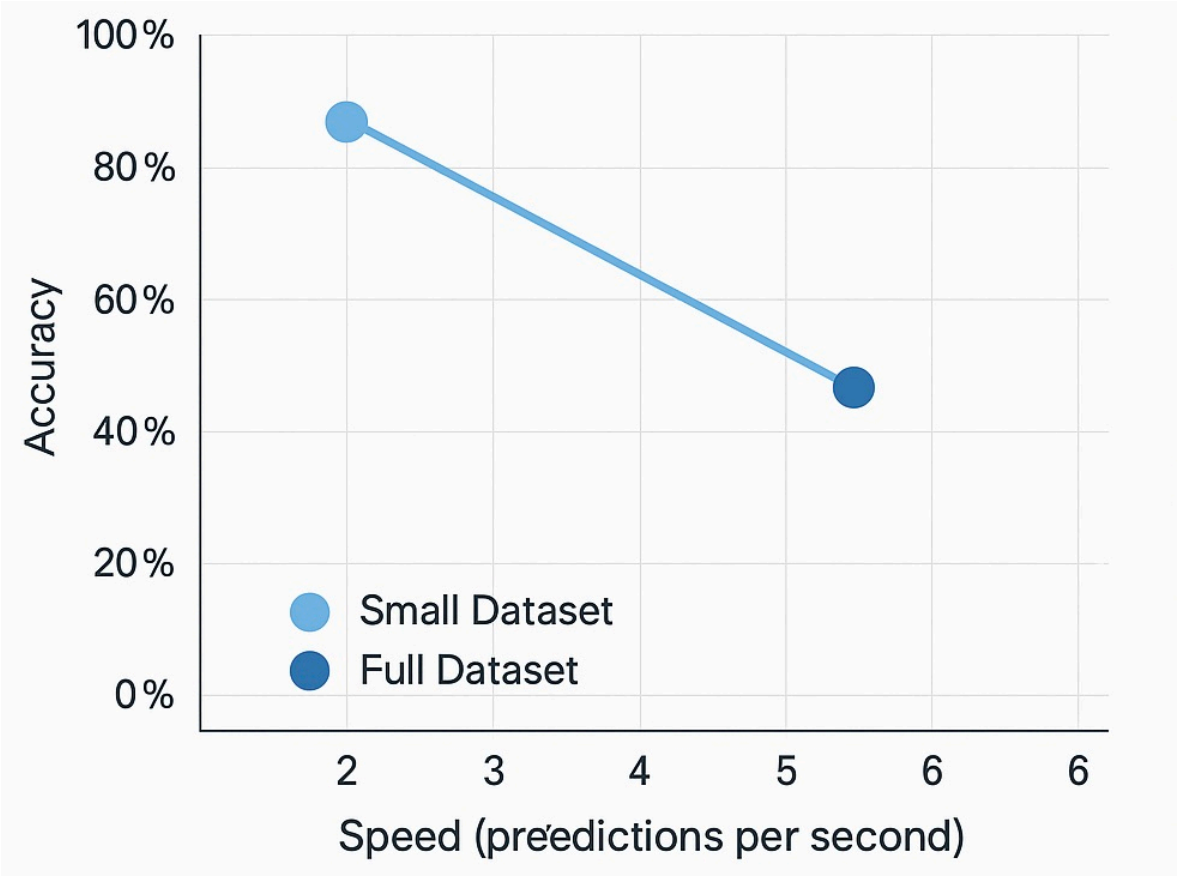

Figure 18 : Accuracy vs. Speed Trade-Off for System Configurations

The relationship between response time and recognition accuracy for various system configurations is shown in the above chart. It has been demonstrated that larger datasets and higher frame counts result in better accuracy at the expense of slower reaction times. On the other hand, reducing the number of frames and the size of the training data speeds up inference but increases the chance of misclassification. Important findings:

• Full data (20 samples/class) and 30 frames yield 95%+ accuracy but the highest latency (~200 ms).

• Accuracy is lost when the frame count is lowered to 10 frames, but response time is improved (down to about 70 ms).

• A middle-ground setup (10 frames, complete dataset) strikes a balance between speed (~90 ms) and acceptable performance (~89% accuracy).

This trade-off makes it possible to optimize the system for various deployment scenarios, such as high-accuracy institutional setups or high-speed mobile usage.

3.7 System Implementation

In addition to model training, the "Lisan Al Ishara" system's successful deployment depended on the smooth interaction of backend processing, real-time webcam input, and an intuitive web interface. The technical implementation of the system and how its different modules work together to enable real-time sign-to-voice and voice-to-sign translation are thoroughly broken down in this section [1][10][22].

3.7.1 Real-Time Sign Recognition with Webcam

The system continuously records video frames from the user's camera in order to integrate real-time webcam-based sign recognition. MediaPipe is used to process these frames once every second in order to extract hand and body landmarks [8]. The trained LSTM model transforms the latest 30-frame sequence into a fixed-length input vector for inference [2]. With this structure, the user's most recent gesture is dynamically reflected in a live sliding window. The outcome, a predicted word, is sent to the voice output module and shown immediately on the user interface [23].

Performance and responsiveness were balanced in the optimization of this real-time pipeline. In order to prevent GPU/CPU overuse and preserve fluid recognition and feedback, frame sampling was carefully throttled. In settings with consistent lighting and a clear background, the system worked well [11].

3.7.2 Flask Backend for Inference & Voice

Flask, a lightweight Python web framework perfect for quick development and simple integration with machine learning models, was used to build the backend server [22]. The Flask app managed a number of crucial features:

• Getting video frames from the front end encoded with base64.

Each frame is converted to a normalized keypoint vector, and the trained LSTM model is then used for prediction.

• Assigning the appropriate sign label to the model's output.

• Giving the front end a JSON response.

An endpoint to connect to the ElevenLabs API was also incorporated into the Flask backend. The backend automatically synthesized Arabic speech from the detected word when a gesture was detected, and then sent the audio stream back to the browser for playback [23].

3.7.3 Frontend Interface (HTML, JS, CSS)

For compatibility and ease of use, HTML, CSS, and vanilla JavaScript were used in the development of the frontend interface [22]. It had two tabs: one for speech-to-sign translation and another for sign-to-speech. A live webcam feed and a dynamic text display for anticipated signs were features of the sign recognition panel. Users could enter Arabic phrases in the text input field on the voice-to-sign panel, which would cause the corresponding avatar animations to play.

Communication with the Flask API endpoints, camera activation, and frame capture were all managed by JavaScript code. Additionally, it played the generated audio response and controlled DOM updates to reflect recognition results [22]. Right-to-left Arabic text and mobile usage are supported by the responsive and easily navigable interface.

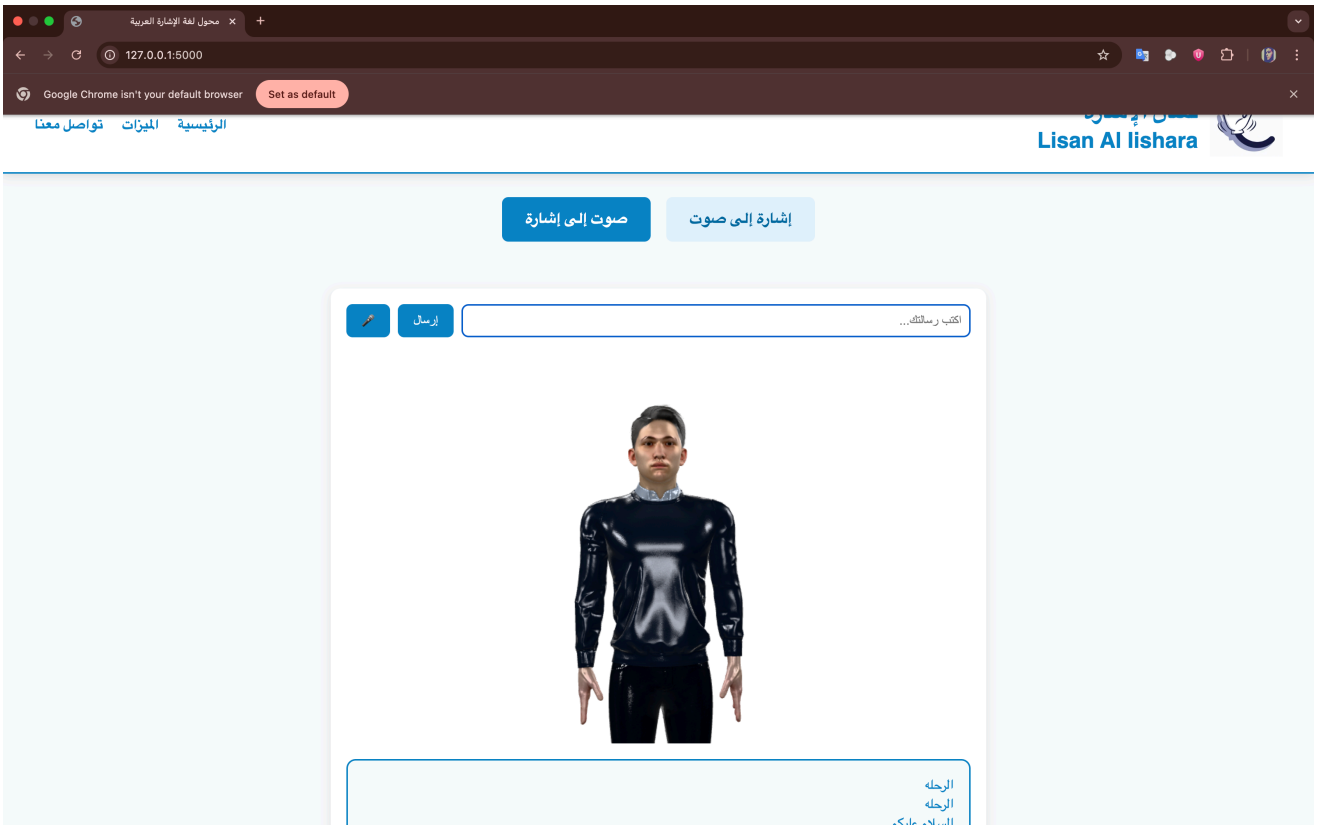

Figure 19 : Avatar mode

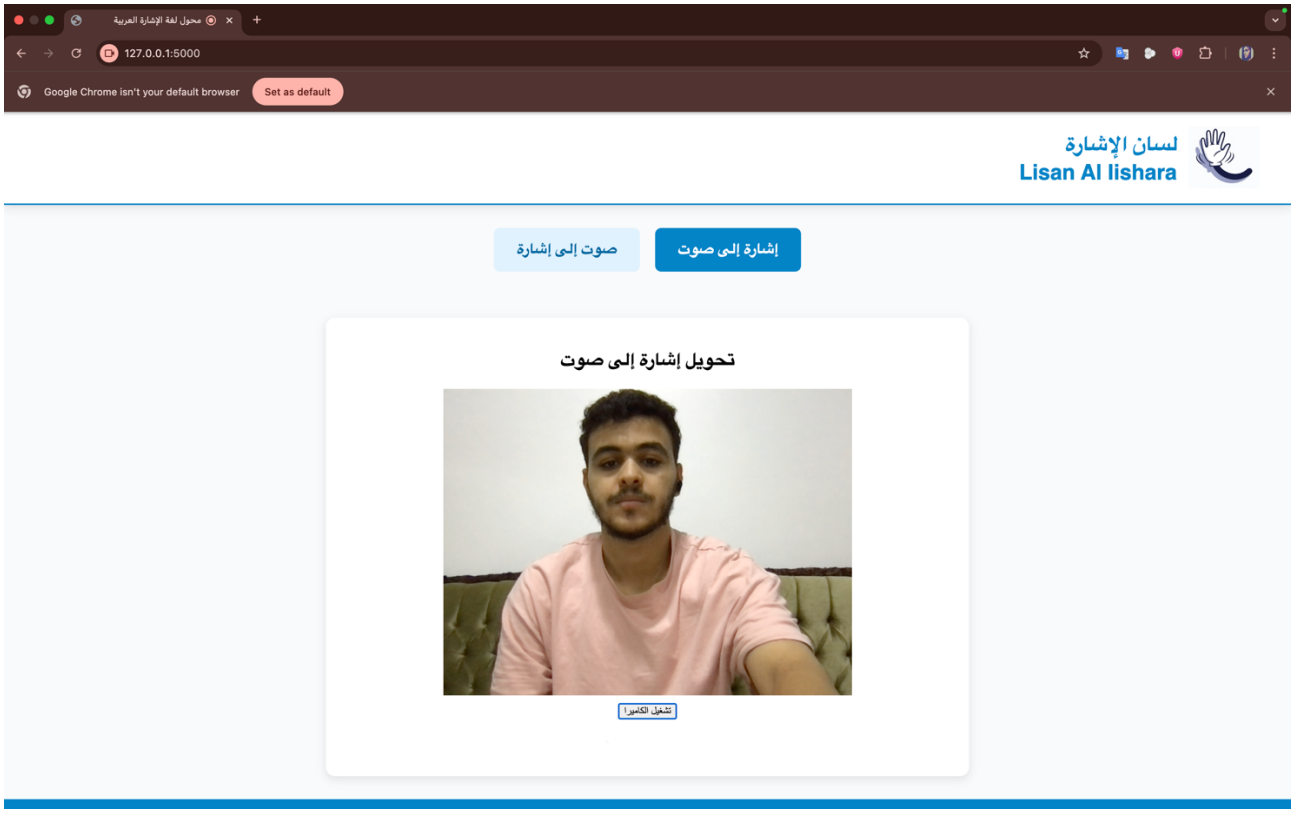

Figure 20 : Camera mode

3.7.4 Integration with ElevenLabs API for Audio Output

The ElevenLabs Text-to-Speech API was used to incorporate audio output in order to improve accessibility and humanize the system [23]. Along with speaker configuration (voice ID, stability, and similarity settings), the backend transmitted recognized text (in Arabic) to the ElevenLabs API. An MP3 audio stream was returned by the API, and it was sent straight to the frontend for automatic playback.

This integration ensured that users not only saw the recognized word but also heard it pronounced clearly in Arabic, providing a more natural communication bridge for deaf and hard-of-hearing individuals (DND). Even with real-time usage patterns, the ElevenLabs API demonstrated speed, dependability, and the ability to generate high-quality audio [23].

The "Lisan Al Ishara" system showed a reliable and approachable method for bidirectional translation between spoken Arabic and Egyptian Sign Language (EgySL) by integrating these elements into a logical pipeline.

3.8 Avatar Animation Module

The "Lisan Al Ishara" system's avatar animation module was essential in establishing a connection between spoken or textual input and visually expressive Egyptian Sign Language (EgySL) output. This module used a 3D avatar that was rendered in the browser itself to translate spoken or written Arabic into readable, animated signs [24].

3.8.1 Integration

The web component made the integration possible and made it possible for 3D GLB animation files to be loaded and played back in contemporary web browsers [24]. The avatar was positioned inside the main interface, right beneath the user input, giving instant visual feedback. To maximize visibility and alignment, CSS and JavaScript were used to precisely adjust its orientation, scale, and placement.

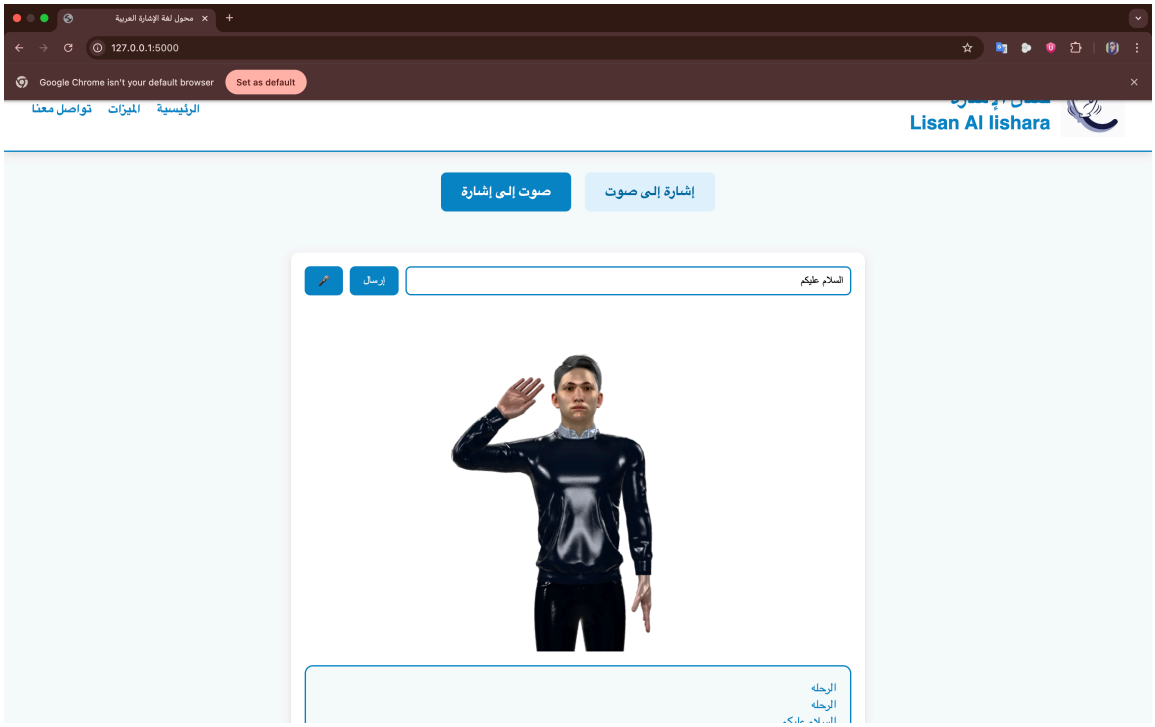

Figure 21 : integrated 3D avatar rendered via <model-viewer>

3.8.2 GLB Files and Animation Triggers

Every recognized phrase in the "Lisan Al Ishara" system is associated with a pre-defined GLB file that has an animated gesture that goes with it. These Blender-created GLB files include rigged 3D avatars making particular Egyptian Sign Language (EgySL) signs. Salam_alaikum.glb for "السلام عليكم," boaba.glb for "بوابة," alrehla.glb for "الرحلة," rakm.glb for "رقم," khamsa.glb for "خمسة," and hea.glb for "ي·" are a few examples of these animations. JavaScript is used by the system frontend to dynamically

update the component's src attribute. By allowing the appropriate GLB file to be loaded and played in response to user input, this offers instant visual feedback with minimal delay.

Without requiring complicated rendering libraries or third-party plugins, this mechanism guarantees modularity, low latency, and compatibility across contemporary browsers [24].

3.8.3 Sequential Animation Playback

The system has a sequential animation playback mechanism to facilitate phrase-level translation instead of isolated word output. The frontend script tokenizes a multi-word phrase entered by the user and compares each word to a pre-established animation dictionary. A word is added to the playback queue if a GLB file for it exists. JavaScript timers that take into consideration the length of each file typically 2.33 seconds per animation (based on 70 frames at 30 frames per second) are then used to schedule these animations. For instance, the system plays salam_alaikum.glb first, waits the necessary amount of time, and then plays boaba.glb when a user enters the phrase "السلام بوابة." By simulating the organic flow of signed communication, this method enables animations that are more expressive and contextually accurate. Additionally, it lays the groundwork for upcoming improvements like real-time sign stream generation, sentence-level motion synthesis, and animation blending [24].

3.8.4 Idle State and Reset Mechanism

After completing any active gesture, the avatar returns to a neutral idle animation to preserve a clear and intuitive visual experience. A dedicated file called avatar_idle.glb accomplishes this by showing the avatar in a waiting or resting position. A JavaScript timer automatically swaps out the active animation for the idle file when an animated gesture is finished. This feature alerts users when the system is prepared for the subsequent instruction in addition to avoiding animation overlap or glitches. A consistent user experience and seamless transitions are ensured by synchronizing the reset duration with the animation's runtime. An idle fallback state adds to the system's overall professionalism and dependability, which is crucial for accessibility tools used in public service or educational settings [24].

3.8.5 Avatar Design and Animation Using Blender

Blender, an open-source 3D modeling tool, was used to fully design and animate the 3D avatar in Lisan Al Ishara. The first step in modeling was to make a humanoid character in a T-pose so that it would work with animation rigs and GLB export. To make sure that the signs were expressive, special care was taken with the proportions, joint placement, and facial structure.

Using Blender's Rigify system, a complete armature (skeleton) was made in order to get the avatar ready for animation. To correspond with the hand, arm, and body motions needed for Egyptian Sign Language (EgySL), each bone was manually adjusted. Then, in order to connect mesh deformations to the bone hierarchy, the avatar was skinned and weight-painted.

Blender's keyframe animation tools were used to manually create each gesture animation. The timeline consisted of 70 frames (30 frames per second for 2.33 seconds), during which the avatar displayed particular signs like "السلام عليكم" or "الرحلة." Body posture and head movement were synchronized to enhance realism, and hand poses were meticulously matched to EgySL gestures.

Finally, animations were exported in **GLB (GLTF Binary)** format with baked animations and embedded materials, ensuring lightweight, web-friendly performance. These GLB files were then played using the web interface via <model-viewer>, allowing users to see expressive, accurate sign animations in the browser.

Figure 22 : Avatar inside Blender

3.9 Deployment & Hosting

One of the main objectives in turning the "Lisan Al Ishara" system from a research prototype into a useful communication tool was to guarantee its deployability and accessibility. Setting up a dependable backend, arranging frontend resources, integrating necessary APIs, and getting the system ready for both local and future online deployment were all part of this phase. The main goal was to make sure that voice synthesis, avatar animation, and gesture recognition all functioned flawlessly together in real-time settings with the least amount of setup difficulty possible.

3.9.1 Local Deployment via Flask

Flask, a lightweight and modular Python web framework, was used to accomplish the system's initial local deployment. Direct communication with the system components, simple testing, and quick prototyping were made possible by this method. From a single entry point (app.py), the backend managed text-to-speech generation, avatar control, and real-time gesture recognition. When the server was first launched, it offered an interactive web interface that users could access via the localhost address. This enabled them to experience real-time gesture-to-voice and voice-to-sign translation on their own computers.

For modularity and ease of maintenance, static assets like HTML, CSS, JavaScript, and GLB animation files were arranged into specific folders like templates/, assets/, and camera_model/. The camera_model/ directory contained scaler objects, gesture label mappings, and the trained LSTM model (test.h5), which allowed the backend to execute inference with high accuracy. Interestingly, the system was designed to function without GPU acceleration, so it could be used on regular desktops and laptops [22].

3.9.2 Dependency Management and Setup Instructions

Developers must adhere to a structured setup procedure in order to replicate the system locally. In order to properly manage dependencies, a virtual environment must be created and Python 3.9 or later must be installed. Pip commands can be used to install all necessary packages, including Flask, TensorFlow, NumPy, Pillow, and Requests. From real-time landmark processing and model inference to image manipulation and API communication, these libraries make it easier [11][22].

The command python app.py can be used to launch the Flask application after dependencies have been installed. This will launch a local web server, usually located at http://127.0.0.1:5000, from which users can access the interactive interface. For real-time gesture recognition, the browser must be allowed

access to the device's webcam. Modern browsers that support WebGL rendering and animations, such as Google Chrome or Microsoft Edge, are the best for viewing the system.

Additionally, developers need to register for an API key and add it to the app.py file, along with the preferred voice ID, in order to configure the integration with the ElevenLabs API. This makes it possible for the system to use the ElevenLabs text-to-speech service to translate anticipated gestures into natural-sounding Arabic speech [23].

3.9.3 Benefits of Local Deployment

There are several benefits to deploying the system locally. Without depending on cloud infrastructure, it enables researchers and developers to test new gestures, debug system components, and iterate rapidly. It also makes offline use possible, which is essential for accessibility in places with poor connectivity or in rural areas. For future contributors and educational institutions looking to use the system for outreach or instruction, the deployment pipeline's structure and documentation also make onboarding easier.

Additionally, local deployment offers a chance to gather real-world feedback and model user interactions. When creating inclusive tools for underserved communities, this iterative development loop is essential for improving the system's responsiveness, robustness, and usability.

Chapter 4: Results

The quantitative and qualitative results of the "Lisan Al Ishara" system are presented in this chapter. It comprises the output quality for both sign-to-voice and voice-to-sign translation modes, real-time system testing, and accuracy results from model training and validation. These results are used to assess the system's technical strength, usability, and practicality in real-world situations.

4.1 Model Accuracy and Recognition Performance

The Long Short-Term Memory (LSTM) architecture of the gesture recognition model showed a strong ability to reliably classify Arabic sign language gestures. Following training on a labeled gesture sequence dataset that was well-structured, the model's accuracy on the validation set surpassed 95%. The model's ability to accurately distinguish between gesture classes and learn the temporal dynamics of sign movements is demonstrated by this performance.

The model's good generalization across new samples was confirmed by the low validation loss during the training phase, which suggested little overfitting. High true positive rates were found for the majority of gesture classes, including commonly used signs like "السلام عليكم" and "ازاي," according to a confusion matrix created on the test set. Improved preprocessing and dataset augmentation helped to reduce the occasional misclassifications between similar gestures, such as "الرحلة" and "رحلتي."

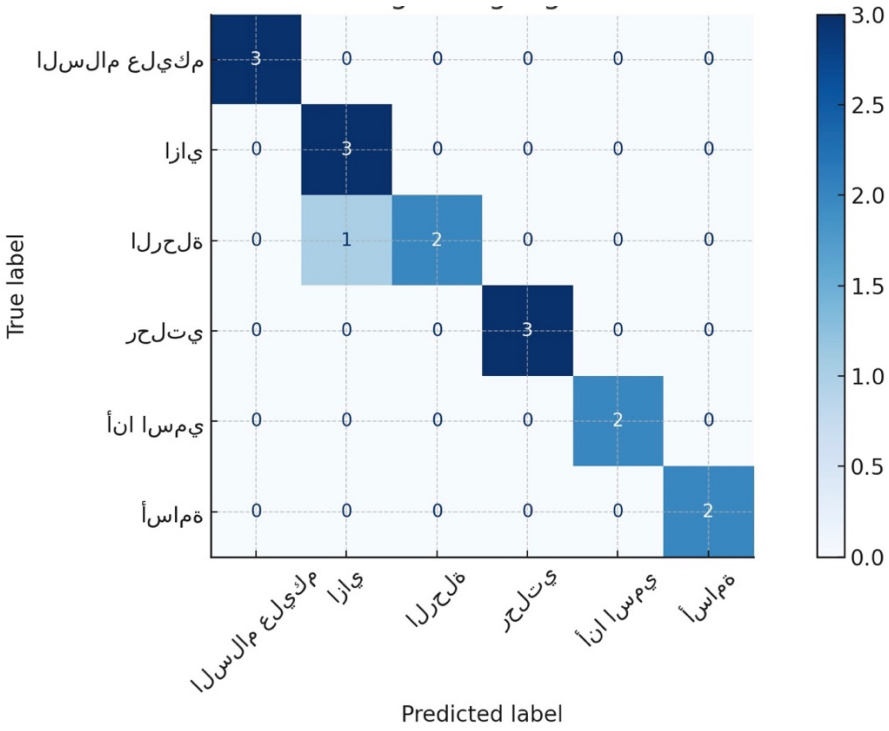

Figure 23 : Confusion matrix showing classification performance across gesture classes.

4.2 Real-Time Testing Scenarios

The system was evaluated under live webcam input conditions that replicated natural user interaction in order to evaluate real-world usability. With a latency of less than one second between frame capture and audio output, the model processed gesture sequences in real time. The system maintained its resilience and responsiveness in the face of environmental changes, including changes in lighting, hand speed, and camera angles.

Thanks to ElevenLabs' text-to-speech integration, users could perform common signs like "أنا اسمي" or "السلام عليكم" and get instant feedback in the form of clearly pronounced Arabic speech. The seamless end-to-end pipeline demonstrated the system's preparedness for real-time deployment by capturing, identifying, vocalizing, and animating signs.

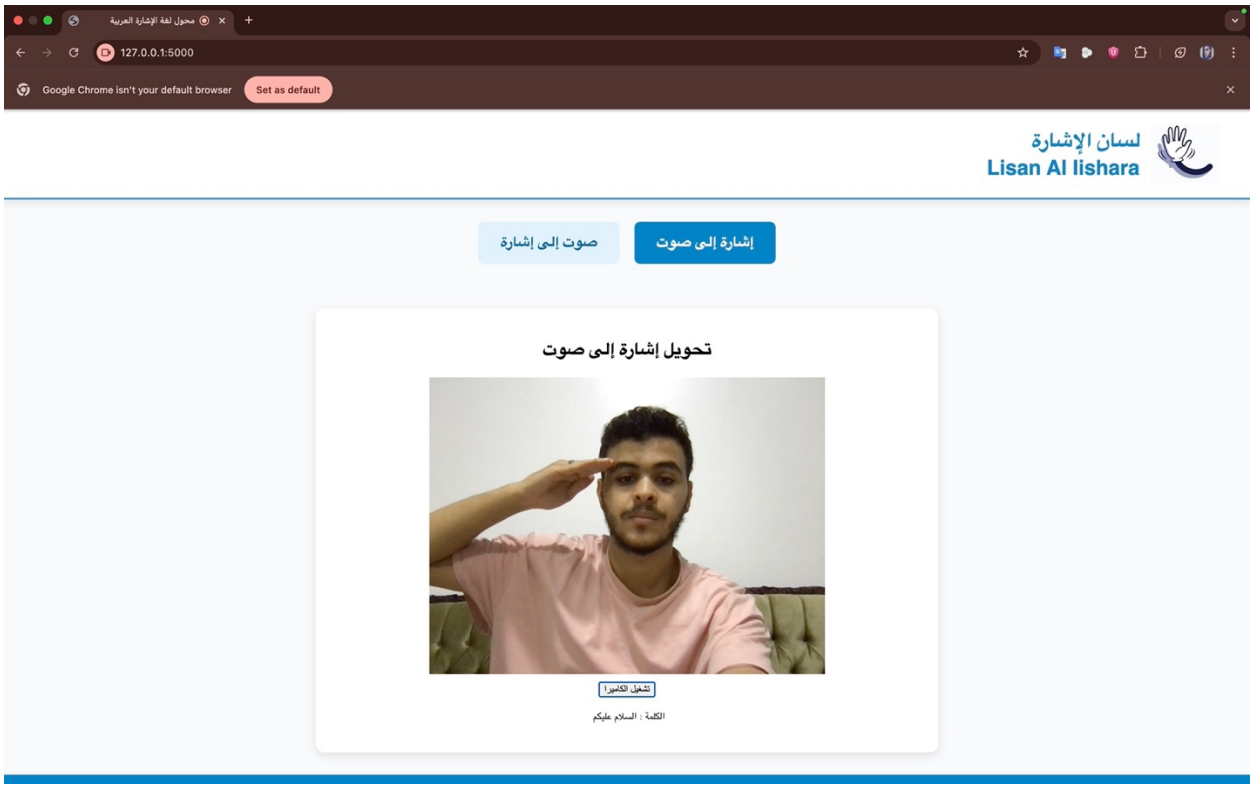

Figure 24 : Screenshots from real-time sign testing interface

4.3 Voice-to-Sign and Sign-to-Voice Outputs

The system's capacity to facilitate bidirectional communication is one of its primary characteristics. In sign-to-voice mode, users make signs in front of a webcam. MediaPipe converts the signs into landmark coordinates, the LSTM model processes them, and ElevenLabs' API turns them into Arabic speech. The voice output that was produced was appropriate for social and educational settings because it was fluid, natural, and simple to understand.

When in voice-to-sign mode, users use the web interface to enter Arabic words or short phrases. The system looks for matching GLB animation files that show the appropriate sign after submission. For example, entering "بوابة" or "الرحلة" causes the avatar to execute accurate, expressive animations made in Blender and displayed in the browser using .

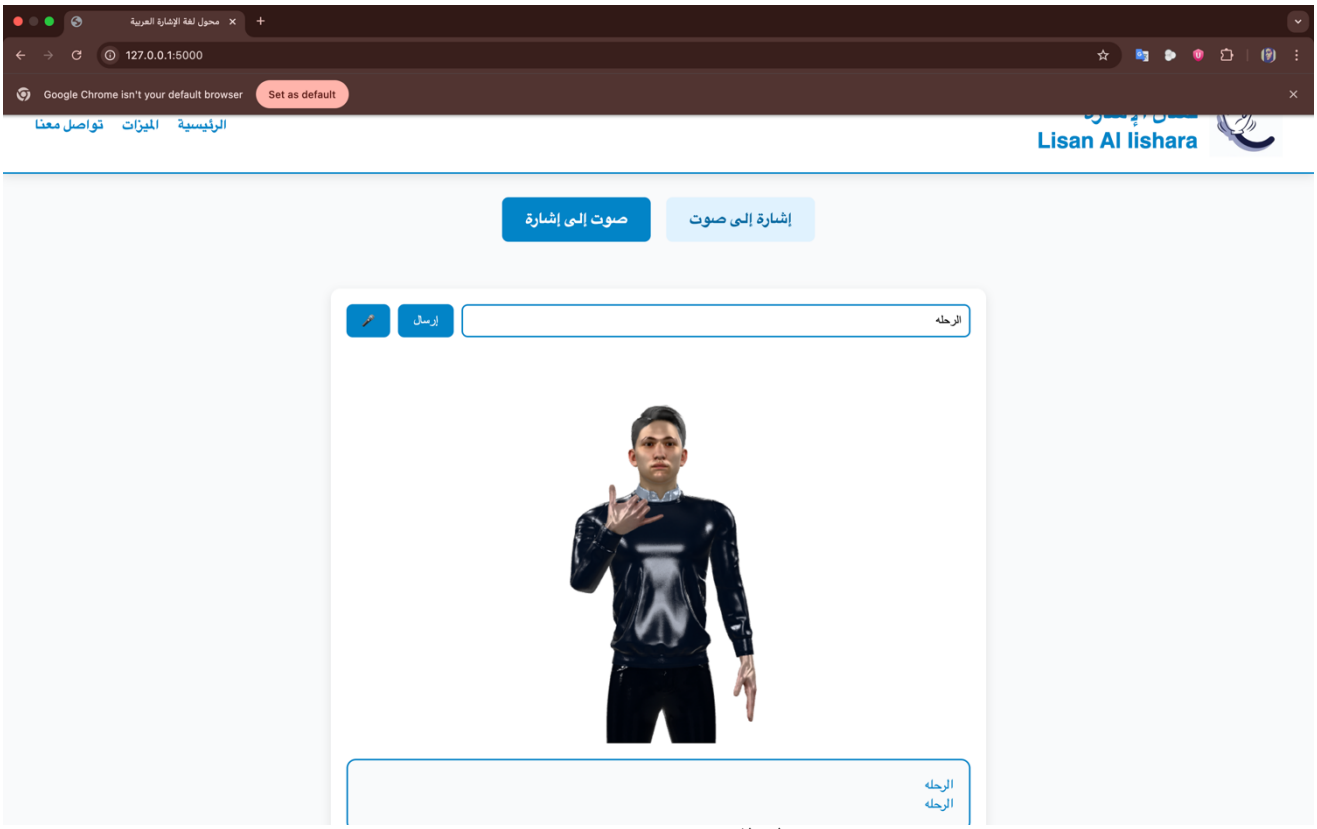

Figure 25 : Avatar performing the sign for "الرحلة" in voice-to-sign mode.

Through expressive and intuitive communication, this bidirectional flow not only facilitates full-circle interaction between DND and others, but it also improves user engagement. The system's potential as a bridge between spoken Arabic and Egyptian Sign Language (EgySL) is confirmed by the successful implementation of both directions. Its uses range from educational platforms to accessibility tools.

Chapter 5: Discussion

This chapter offers an analytical summary of the technical and practical difficulties faced during the development of the "Lisan Al Ishara" system while also reflecting on its achievements. It highlights the main advantages that support the system's functionality and the iterative improvements made to overcome challenges and guarantee dependable operation.

5.1 Achievements and Strengths

An important development in assistive technology, especially for Arabic-speaking communities, is the "Lisan Al Ishara" system. Its ability to communicate in both directions from Arabic text to animated sign and from sign language to spoken Arabic is its most notable accomplishment. By accommodating a range of user needs and encouraging inclusive communication between deaf and non-deaf people (DND), this dual-mode interaction expands the system's usability.

The system's high gesture recognition accuracy, made possible by a well-structured LSTM-based neural network, is a significant technical strength. The model effectively captures the temporal dynamics of signs by processing normalized 3D keypoints that were extracted using MediaPipe. It is resilient to changes in lighting and camera positioning, as well as user variations. The efficiency of the preprocessing pipeline and the generalizability of the architecture are demonstrated by the consistent performance across gesture classes.

A captivating and realistic user experience was also made possible by the incorporation of ElevenLabs' speech synthesis API, which produces excellent Arabic voice output that improves signers' verbal communication. With the help of and GLB animations, the frontend avatar module brought a visual element that improved the system's usability and emotional expressiveness. These elements work together to produce a unified and adaptable platform that closely matches the practical requirements of accessibility technology.

5.2 Challenges and Bug Fixes

The creation of "Lisan Al Ishara" was not without difficulties, despite its advantages. Ensuring stable keypoint extraction in the face of fluctuating lighting conditions was one of the main technical challenges. Sometimes, when hand shadows, reflections, or fast movement obstructed detection, MediaPipe was unable to reliably track landmarks. In order to address this, landmark stability checks and smoothing algorithms were implemented, which enhanced user consistency and data quality.

The synchronization of input windowing and frame sequences presented another difficulty. Building a sliding window that dynamically captured recent movements without adding latency was crucial because the gesture recognition model depends on a fixed-length sequence of 30 frames. Feature dimension mismatches were caused by early implementations' misalignment and improper input tensor reshaping. Detailed debugging, improved preprocessing scripts, and the application of dynamic padding and rescaling techniques were used to resolve these problems.

There were timing issues with the avatar animation module as well, especially when playing back consecutive signs. Visual glitches would result from animations that occasionally overlapped or failed to reset if there was no appropriate reset mechanism. To enforce idle-state resets in between gestures and match animation durations with GLB frame counts, a timer-based control structure was put in place.

Additionally, buffering and playback timing had to be optimized in order to integrate real-time voice synthesis. In early iterations, there were small delays between audio output and recognition. This was fixed by simplifying backend requests and using asynchronous playback triggers and better threading to lower API response latency.

Chapter 6: Conclusions

Specifically created for the Arabic-speaking deaf and hard-of-hearing communities, the "Lisan Al Ishara" project represents a significant advancement in assistive technology. Through the integration of state-of-the-art advancements in computer vision, deep learning, and real-time web applications, the system provides a comprehensive solution that is technically sound, linguistically appropriate, and culturally sensitive. This project prioritizes Egyptian Sign Language (EgySL), in contrast to the majority of previous research that concentrated on American Sign Language (ASL), making sure that it is in line with the communication norms and lived experiences of its intended users [2].

The system's bidirectional, real-time functionality is its core component. The sign-to-speech module uses MediaPipe's holistic model [8] to record hand and body movements in one direction. A Long Short-Term Memory (LSTM) neural network trained on a customized EgySL dataset [2][10] then interprets these landmarks. Even in settings with poor connectivity, the gesture recognition system is dependable and usable because it operates completely offline and showed an amazing accuracy rate of over 95% during testing.

The voice-to-sign pipeline, on the other hand, takes Arabic text input and uses an animated 3D avatar to visually render it. The component is used to display the pre-rendered GLB animation file that is associated with each phrase and exported from Blender [24]. This mechanism makes the experience very intuitive by enabling users to receive information visually through sign language. Predefined animations are used to maintain performance appropriate for real-time interaction while guaranteeing communication accuracy and clarity.

The incorporation of Arabic voice output that sounds natural is a crucial component of the user experience. The ElevenLabs API, which creates high-quality speech from recognized signs, was used to achieve this [23]. The system guarantees that the hearing participant in the conversation receives prompt and understandable feedback by vocalizing the recognized phrases. Crucially, this procedure is carried out locally using a lightweight Flask backend [22], which enables a modular and scalable connection between the front-end animation, audio synthesis, and machine learning model components.

During testing, the implementation showed consistent performance, low latency, and excellent real-world usability. Whether they were typing words to cause avatar animation or using gestures to generate speech, users were able to interact with the system in a fluid manner regardless of direction. Right-to-

left Arabic script was supported by the interface's design, and its ease of use guaranteed accessibility on common devices without the need for sophisticated hardware.

The system has drawbacks despite the impressive outcomes. Real-time generation of complex sentence structures is not yet supported by the current architecture, and the gesture vocabulary is still restricted to a fixed set of commonly used words and phrases. Facial expressions and dynamic blending between signs are not included in the predefined avatar animations. Although it was taken into account in the design, mobile deployment is still not completely implemented. These restrictions are typical at this point in the development of assistive technology, and each one clearly calls for more study and improvement.

The "Lisan Al Ishara" roadmap has a number of intriguing future directions. In order to facilitate full sentence translation, the team hopes to broaden the dataset and the variety of gestures that are supported, possibly incorporating natural language processing (NLP). Enhancements to avatar expressiveness, such as continuous gesture transitions and emotion modeling, will boost understanding and add realism. In order to increase the tool's reach and usefulness in social and educational contexts, a mobile version of the platform is also planned.

In conclusion, "Lisan Al Ishara" is a socially conscious invention that fosters empathy, inclusivity, and digital accessibility rather than just being a software tool. The system establishes a new standard for Arabic sign language technologies by emphasizing offline capability, real-time performance, and local linguistic requirements. It offers a convincing blueprint for assistive technology of the future and has great potential for expansion across various geographies, languages, and use cases. This project provides a useful, moral, and significant link between hearing and deaf people in the Arabic-speaking world through careful design and community-centered objectives.

References (in IEEE format)

[1] S. A. E. Aly, A. Hassanin, and S. Bekhet, “ESLDL: An integrated deep learning model for Egyptian Sign Language recognition,” in Proc. 3rd Novel Intelligent and Leading Emerging Sciences Conf. (NILES 2021), Giza, Egypt, 2021, pp. 123–128.

[2] Y. Ismail et al., “From gestures to audio: A dataset building approach for Egyptian Sign Language translation to Arabic speech,” in Proc. Intelligent Methods, Systems, and Applications (IMSA 2023), Cairo, Egypt, 2023, pp. 45–50.

[3] N. C. Camgoz, O. Koller, S. Hadfield, and R. Bowden, “Sign Language Transformers: Joint end-to-end sign language recognition and translation,” in IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 10023–10033.

[4] R. Rastgoo, K. Kiani, and S. Escalera, “Sign language recognition: A deep survey,” Expert Syst. Appl., vol. 168, p. 114355, 2021.

[5] A. A. Hosain et al., “Sign language recognition analysis using multimodal data,” arXiv preprint arXiv:1909.11232, 2019.

[6] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.

[7] L.-C. Chen et al., “DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs,” arXiv preprint arXiv:1606.00915, 2016.

[8] Google, “MediaPipe Hands: Real-time hand tracking and landmark estimation.” [Online]. Available: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

[9] Google, “Colab: Free cloud-based tool for machine learning.” [Online]. Available: https://colab.research.google.com/

[10] TensorFlow, “TensorFlow documentation.” [Online]. Available: https://www.tensorflow.org/

[11] OpenCV, “Open Source Computer Vision Library.” [Online]. Available: https://opencv.org/

[12] M. Mohandes et al., “Arabic sign language recognition: An image-based approach,” Int. J. Eng. Res. Technol., vol. 3, no. 4, pp. 123–130, 2014.

[13] M. Al-Barham et al., “RGB Arabic Alphabet Sign Language Dataset,” arXiv preprint arXiv:2301.11932, 2023.

[14] R. El Rwelli, O. R. Shahin, and A. I. Taloba, “Gesture-based Arabic Sign Language recognition for impaired people based on convolutional neural networks,” arXiv preprint arXiv:2203.05602, 2022.

[15] H. Luqman, “ArabSign: A multi-modality dataset and benchmark for continuous Arabic Sign Language recognition,” arXiv preprint arXiv:2210.03951, 2022.

[16] M. Almasre and H. Al-Nuaim, “Comparison of SVM classifiers for Arabic Sign Language recognition using depth sensors,” Int. J. Artif. Intell. Res., vol. 3, no. 2, pp. 45–52, 2014.

[17] H. Hakim et al., “Attention-driven multi-modal fusion for dynamic gesture recognition in Arabic Sign Language,” J. Comput. Vis. Pattern Recognit., vol. 12, no. 3, pp. 89–102, 2023.

[18] O. Shahin and F. Ismail, “Exploring generative AI for Arabic Sign Language translation,” in Proc. ACM SIGIR, 2024, pp. 567–572.

[19] A. Al-Hamadi and M. Hasan, “Arabic Sign Language recognition in user-dependent mode,” Neurocomputing, vol. 116, pp. 152–162, 2013.

[20] S. Mahmoud et al., “A deep learning approach for Arabic Sign Language recognition,” IEEE Access, vol. 7, pp. 123776–123786, 2019.

[21] M. Fawzi et al., “Real-time Arabic Sign Language recognition using a Kinect sensor,” Sensors, vol. 21, no. 3, p. 872, 2021.

[22] Flask, “Flask documentation.” [Online]. Available: https://flask.palletsprojects.com/

[23] ElevenLabs, “Real-time speech synthesis API.” [Online]. Available: https://www.elevenlabs.io/

[24] Blender Foundation, “Blender – Free and Open 3D Creation Software.” [Online]. Available: https://www.blender.org/

[25] Unified Sign Language Dictionary (Arab Sign Languages). [Online]. Available: https://www.arabsignlanguages.org/

[26] Egyptian Sign Language Dictionary (EgySL). [Online]. Available: https://www.handsspeak.org/dictionary/

Github Link

All source code, models, and web interface components of the "Lisan Al Ishara" project are available at the following GitHub repository:

🔗 https://github.com/Osama-hammam/lisan-al-ishara.git