Faculty of Engineering
Software Engineering and Information Technology Department

Video Chat Application with Bilateral Arabic Sign Language Translation

“Eary”

Graduation Project

Submitted By:

Name	**ID**
Alaa Osama Yahya	191800029
Alaa Tarek Abbas	191800179
Farah Muhammed Hamdy	191800117
Haidy Ahmed Ibrahim	191800169
Nathalie Nader Nabil	191800001

Under the Supervision of:

Dr. Mohamed Taha Dr. Soha Safwat

Class of 2023

Acknowledgement

We extend our sincere gratitude to Dr. Soha Safwat and Dr. Mohamed Taha for their invaluable support, guidance, and unwavering patience throughout our endeavor. Additionally, we express our deep appreciation to T.A. Rahma Amin and T.A. Ghada Adel for their consistent willingness to assist us and provide invaluable advice. Furthermore, we would like to express our heartfelt thanks to Dr. Amin Mohamed El Sharaky, a lecturer at the Kafr El Sheikh Social Service Institute, for his significant contribution in aiding us with sign language.

Abstract

Successful communication requires the efforts of all people involved in a conversation. Communication with a hearing disability can be challenging if one doesn't know sign language. They always need someone to translate for them. Sign language allows them to learn, work, access services, and be included in their communities, but it also limits their communication with those who do not know sign language. Out of the Middle East’s 350 million people, over 11 million have disabling hearing loss, comparable to other regions. But services for deaf citizens have not kept pace. The dearth of sign language interpreters is partly a problem of inadequate recruitment and training, and partly a reflection of broader obstacles for deaf citizens. By 2050 nearly 2.5 billion people are projected to have some degree of hearing loss and at least 700 million will require hearing rehabilitation.

In developing countries, children with hearing loss and deafness often do not receive schooling. Adults with hearing loss also have a much higher unemployment rate. Among those who are employed, a higher percentage of people with hearing loss are in the lower grades of employment compared with the general workforce. Everyone should be able to access information equally and communicate freely.

In this document, we propose a video chat application with a bilateral Arabic sign language system to translate the sign language of the impaired person into Arabic text and translate the Arabic language voice into sign language using an avatar to simulate the sign, so hearing impaired people can communicate without being limited by other people's knowledge of sign language. To apply the system, we developed our dataset containing 30 Arabic words, 100 videos for each word, at the beginning to train and test the model. To get high accuracy, we will combine Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM). They will be trained on the key points extracted from each frame using mediapipe.

Table of Contents

List of Figures

Figure **Page**

Chapter 1

Introduction

1.1 Motivation

Sign language is the language used by deaf people to communicate with one another and with common people. This enables them to be involved in the traditional educational and pedagogic process. An interpreter is typically required when an ordinary person wishes to communicate with a deaf person. Visual sign language recognition is a dynamic area of research in computer vision [16]. The law of multiple proportions states that 5% of the populations who speak Arabic struggle with hearing impairments which is a significant number. This demonstrates why Arabic Sign Language is significant (ArSL).

Arabic sign language is divided into two parts: The first part is a complete language where each word is represented by a sign. In the second part, the Arabic sign language alphabets are used to represent each letter of the Arabic alphabet (ArSLA) [17]. Due to the importance of Arabic sign language, which was previously discussed, it is an important area of study. It helps to eliminate barriers between the deaf and the community, so researchers and practitioners have become interested in the challenge of creating Arabic sign language recognition systems. However, all the current solutions are just basic translation apps either for words or letters, but that still limits hearing-impaired community's interactions and social life experiences. A functional sign language recognition (SLR) system can provide an opportunity for a deaf person to communicate with non-signed persons without the need for an interpreter. It can be used to produce speech or text that makes the mute more autonomous. As a result, we have to think of developing a video chat application with real-time bilateral Arabic sign language translation. This will give hearing-impaired people a better opportunity to express themselves and communicate with the world around them. In the meantime, spoken people would be able to naturally communicate with them as there would be an avatar translating what they are saying into signs.

1.2 Problem Definition:

Singing has always been an important part of human communication. Deaf people use it to communicate with one another, but the problem arises when a deaf person tries to communicate with a normal person who does not understand sign language. This limits deaf people's interactions and social life experiences because they can only communicate with a few people. According to WHO, currently more than 1.5 billion people (nearly 20% of the global population) live with hearing loss and 11 million of the 350 million people living in the Arab world suffer from hearing loss [13]. That's why researchers have tried to concentrate more on this issue and look for a solution that would allow the use of AI and deep learning to help facilitating the interaction of the deaf community with those who do not understand sign language. Despite their best efforts most of sign language recognition research is simulation-based and does not work for real-time or it is just a basic translation apps which still not good enough for effective communication these interpreter’s performance can be affected by small differences such as the performer himself as the interpreter can be designed to be user-dependent or user-independent. In the first case, the interpreter depends on the person while in the latter one, the user is not a problem anymore.

This is due to so many challenges that face researchers starting from finding a good dataset that considered one of the biggest challenges. To collect a good and clean dataset, you have to overcome these challenges:

- Viewpoint Difference: as people can take the same sign with different hand kinematics and poses.
- Environment: The background, lighting, landmarks, and other elements can exist in the video all these factors may affect the quality of the dataset.
- Facial Expressions: A facial expression can be part of a sign. The system may be implied by the face, which may have accessories like earrings and glasses [18].

Researchers have tried to solve those challenges using mainly two approaches sensor-based or vision-based. In sensor-based, the user wears sensors, such as colored or special gloves and a motion capture system captures the sign, but this method has different drawbacks as it was ineffective in real-world scenarios because it requires the user to wear such sensors that depend on the continuous power supply, wires, and other requirements. In the vision-based approach, the system captures videos using mobile or laptop camera, there is no need for any external camera or device then, predicting the processed data using machine learning and deep learning techniques [18]. Convolutional neural networks, artificial neural networks, hidden-Markov models, and recurrent neural networks (RNN) are some examples of these techniques. This method is obviously low cost as it is not dependent on any other device. That's why we decided to continue with this approach.

1.3 Objectives

Assistive technologies for the Deaf community in Arab countries are numerous and limited. The complexity of Arabic and ArSL is the main reason for this limitation. Arabic language is much wealthier than English and is considered as one of the most complex natural language. As for ArSL, you can find a standard dictionary with 1600 of the basic and most common signs [19]. Nevertheless, each country has its own modified version of this dictionary that holds new signs for its colloquial language and modified signs for most of the existing words. Even more, you can find for the same word a different sign in distinct cities of the same country. As a conclusion, deaf individuals in developing Arabic countries can communicate with vocal people only through an ArSL human interpreter who is very hard to find and in all the cases will break the conversation privacy.

Our goal is to develop a video chat application with real time bilateral Arabic sign language translation to fill the gap between the deaf and vocal people and help them communicate with each other.

To reach our goal, we are expecting to deliver the following:

- A new dataset of 30 words with 100 videos per word.
- A deep learning model trained on this dataset with a high accuracy rate to translate the signs to text.
- A dictionary contains an avatar simulates each word on the dataset.
- A video chat application that uses this model to translate their signs and helps deaf people communicate easily with vocal people.

1.4 Document Organization

- Chapter 2 introduces:
    - Project terminologies and concepts
    - The related work done.
    - A detailed description of the project.
    - Functional and non-functional requirements.
    - System users
- Chapter 3 introduces:
    - System overview
- Chapter 4 introduces:
    - A detailed description of the proposed dataset
    - A detailed description of all the techniques and algorithms implemented
- Chapter 5 introduces:
    - A detailed description of the system analysis.

Chapter 2

Background

2.1Project Terminologies and Concepts:

In this section, we clarify the important concepts used in this document. We cover various concepts such as Deep Learning, Convolutional Neural Network (CNN), Long short-term memory (LSTM), Mediapipe, Blender, Speech Recognition, Google Speech-to-Text API, and Zegocloud.

2.1.1 Deep learning:

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy [20].

Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) as well as emerging technologies (such as self-driving cars).

2.1.2 Convolutional neural network (CNN):

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area [21].

2.1.3 Long short-term memory (LSTM):

LSTM used in the field of Deep Learning. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems. LSTM has feedback connections, i.e., it is capable of processing the entire sequence of data, apart from single data points such as images. This finds application in speech recognition, machine translation, etc. LSTM is a special kind of RNN, which shows outstanding performance on a large variety of problems [22].

2.1.4 Mediapipe:

MediaPipe is a Framework for building machine learning pipelines for processing time-series data like video, audio, etc. This cross-platform Framework works on Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano [23].

2.1.5 Blender:

Blender is a free and open-source 3D creation software used for creating animated films, visual effects, video games, 3D models, and more. It features a powerful and flexible interface for creating and editing 3D models, animations, and simulations using a variety of tools and techniques. Blender supports a wide range of file formats for importing and exporting 3D models, textures, and animations, and it can also be extended using Python scripting for customization and automation. [33]

2.1.6 Speech Recognition:

Speech recognition is a technology that allows machines to understand and transcribe human speech into text or commands. It involves the use of algorithms and machine learning techniques to analyze and decode spoken language, transforming it into a format that can be processed and understood by computers. This technology has a wide range of applications, including voice-enabled digital assistants, transcription services, language translation, and speech-to-text dictation software. Speech recognition has made significant advancements in recent years, and its accuracy and reliability continue to improve, making it an increasingly important tool for human-machine interaction.

2.1.7 Google speech-to-text API:

Google Speech-to-Text API is a cloud-based solution that enables developers to convert spoken audio into text in over 120 languages and variants with an accuracy of up to 95%. This API uses deep neural networks to process audio data and provides real-time streaming or batch processing capabilities. It supports a wide range of audio formats, including raw audio, WAV, FLAC, and MP3. The API is customizable, allowing developers to fine-tune recognition models for specific use cases. It can be used for a variety of applications, such as automated transcriptions, voice commands, and voice search. The API is part of the Google Cloud Platform and can be accessed through a REST API or client libraries for popular programming languages. [34]

2.1.8 Zegocloud:

ZEGOCLOUD is a library/platform where you can find different products like Video/Audio calling, Live Streaming, In-App Chatting and much more. By using this library you can instantly develop app with these different features using just chunk of code. [31]

2.2 Related Work

Research work on sign language recognition is reported for many languages such as Arabic [1], Indonesian [2], American [3], Indian [4] and Thai [5]. Sign language recognition systems are based on one of two ways to detect sign languages 'gestures. They are sensor-based recognition systems and image-based recognition systems [25]. In sensor-based systems, sign language recognition is based on sensors that detect the hand's appearance. For this kind of system, two types are considered, which are the glove-based systems [27] and the Kinect-based systems [29]. Glove based systems [27] use electromechanical devices to recognize hand gestures. Hearing and speech impaired signers are required to wear a glove that is linked to some sensors that gather information [26]. Although this technique can offer good results, it can be inconvenient to the signers [26]. For the second category, Kinect sensors are used to detect sign language gestures. Originally, these sensor devices were developed by Microsoft for their Xbox game as an input device to interact with video games without using any remote controllers. Nowadays, the use of this device is expanding to include recognition systems like sign language recognition. On the other hand, image-based systems use images or videos along with image processing and machine learning techniques to recognize sign language gestures [26]. These systems fall into two categories. The first depends on using gloves containing visual markers to detect hand gestures, such as colored gloves [25]. However, this method prevents sign language recognition systems from being natural, where naturalness is expected from similar HCI systems [25]. The second category depends on images capturing hand gestures of the sign language [26]. When using these image-based recognition systems, hearing and speech impaired do not need to use any sensors or gloves with visual markers, which eliminates any inconvenience and makes the system more convenient [25].

In the past 20 years, deep learning have been used in sign language recognition by researchers from all around the world. Convolutional Neural Networks (CNNs) have been used for video recognition and achieved high accuracies last years.

- B. Garcia and S. Viesca at Stanford University proposed a real-time ASL recognition with CNNs for classifying ASL letters, described in Reference [28] . After collecting the data with a native camera on a laptop and pre-training the model on GoogLeNet architecture, they attained a validation accuracy of nearly 98% with five letters and 74% with ten.

- CNNs have also been used with Microsoft Kinect to capture depth features. Reference [29] proposed a predictive model for recognizing 20 Italian gestures. After recording a diverse video dataset, performed by 27 users with variations in surroundings, clothing, lighting and gesture movement, the model was able to generalize on different environments not occurring during training with a cross-validation accuracy of 91.7%. The Kinect allows capture of depth features, which aids significantly in classifying ASL signs.

- A.E.E.El Alfi, M.M.R.El Basuony, S.M.El Atawy [24] introduce an Arabic speech to ArSL intelligent conversion system. The system is based on a knowledge base to resolve some of the Arabic language problems (e.g. derivational, diacritical and plural). According to the authors' evaluation, the system has accuracy up to 96%. Conversely, the system has two main problems. It is a desktop application and the system output is a sequence of still images without any motion. Therefore, it is more suitable for educational purposes, not for real-time translation.

- Keskin et al. [7] developed a Random Decision Forests (RDFs) model trained on a 3D hand models that represented the hand with 21 different parts. RDF classified each pixel and assign it to a hand part. Then they estimate the joint locations for the hand part using a local mode finding algorithm They also developed a support vector machine (SVM) model to recognize the Arabic sign language digits using the same technique. They achieved a high recognition rate on live depth images in real-time.

- Mehdi et al. [8] developed a sensors-based method. He used 7 sensor gloves of the 5DT company to get the input data of the hands' movements. Then an artificial neural network

(ANN) used to recognize the signs' gestures. This approach achieved an accuracy rate of 88%. Lopez- ´Noriega et al. [11] followed their same approach and also offered a graphical user interface made with ".NET".

- Starner et al.[12] used Hidden Markov Model (HMM) based that worked effectively in continuous and real-time sign language recognition tasks. They used color gloves to capture hand shape, orientation, and trajectory and then fed it to HMM-based system to recognize the sentence-level ASL. This system achieved high word accuracy results.

- Nandy et al. [10] proposed a new India sign language dataset and classify based on the feature of a direction histogram that appealed for illumination ad orientation invariance using two different approaches for recognition, the Euclidean distance and K-nearest neighbor metrics.

- A speech to sign language interpreter system (SSLIS) was made where signed English (SE) model is employed, as it is described by O. O. K. a. H. E. Khalid Khalil El DarymLi.[35] Simply, the recognized output text of the speech recognition engine is going to be inputted to the American Sign Language (ASL) database.[35] A project that uses template-based recognition as the main approach, in which the voice to sign (V2S) system first needs to be trained with speech patterns based on some generic spectral parameter set.[36]

Most of these systems achieved high accuracy rate, but they are simulation-based and does not work for real-time applications. However, with the advent of mobile Apps, true deployment of sign language translation systems is profiling. For instance, a Vision-based Android App for Indonesian sign language using the OpenCV computer vision library is proposed in [2]. The App recognizes alphabets of the Indonesian sign language in real-time. Likewise, a mobile App for recognizing alphabets of the American sign language is proposed in [6]. The App uses a vision-based approach for the sign language recognition.

2.3 Project Description

In this document we introduce a novel video call application that is universally accessible, particularly beneficial for communication between individuals with hearing impairments and those who are vocal. The application incorporates a deep learning model that integrates Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, without the need for sensors or gloves. This approach allows for more convenience, mobility, and usability of the system, as the model relies solely on recognizing facial and hand gestures. The proposed model is user-independent, as it can recognize different users and is not affected by the surrounding environment. To achieve this, we used the Mediapipe solution to track high-fidelity body poses by inferring 33 3D landmarks and background segmentation masks on the whole body from RGB video frames. [30]

Although many available datasets in Arabic sign language that focus on letters or words are based on specific conditions such as: (i) the user must wear gloves or (ii) many images refer to static words, making it difficult to achieve the primary goal of independence from unnecessary features related to specific users or the surrounding environment. Thus, we created our dataset with a focus on being user-dependent, utilizing smartphones without stabilization tools to capture videos in different environments with varying backgrounds and resolutions. The dataset contains 30 words with 100 videos for each word.

Our approach involves extracting video frames and key points using Mediapipe, which offers fast and accurate results for sign language recognition, as it recognizes gestures and integrates separate models for pose, face, and hand components. [23] These key points are then passed to the model, which is a combination of CNN and LSTM, to compute the final classification. CNN is capable of extracting important features from input images without human supervision, while LSTM works on the present input while taking into account the previous output and storing it in its memory for a short period of time, allowing us to understand the context of a video frame relative to the frames that came before it

• CNN does not require human supervision to identify the important features. It is made up of layers of neurons that have learnable weights and biases. Adjusting these weights properly, the layers are able to extract features from the input image. They are very accurate at image recognition, also minimizing computation in comparison with a normal neural network, and also make use of the same knowledge across all image locations. Convolutional neural networks are great for a 1 to 1 relation; given an image of a sign, it generates fixed-size label, like the class of the sign in the image. However, What CNNs cannot do is accept a sequence of vectors. That’s where Recurrent Neural Networks (RNNs) are used

•LSTM is a special type of RNN. it works on the present input while taking into consideration the previous output and storing it in its memory for a short period of time, so it allows us to understand the context of a video frame, relative to the frames that came before it.

The pre-trained CNN for feature extraction from input data, along with LSTMs for sequence prediction, is used in our final goal, which is a video call application that translates sign language into Arabic language to facilitate communication between individuals with hearing impairments and those who are vocal.

For the translation from voice to sign, we used Blender to create an avatar that simulates the signs of each word on the dataset. These videos are stored on Firebase. For voice recognition, we used Google speech-to-text API as it offers a robust and scalable solution for audio-to-text transcription needs, with high accuracy, speed, and reliability in transcribing various audio formats, including audio recordings, live audio, and phone calls. Additionally, it supports various languages and dialects, making it a good option in our case. So during the call, once Google API recognizes any speech, it will convert it to text and the avatar that matches this text will be displayed.

2.4 Functional Requirements

- **User can:**
  - If a user wants to register, he can do so by creating a new account or by signing up using a Google or Facebook account.
  - User can sign in if he already has an account.
  - User can add new contact.
  - User can select whom to call from his contacts.
  - User can delete any contact from his contact list.
  - User can search for any contact from his contact list.
  - User can start a video call with any of his contact list.
  - User can answer any incoming video call.
  - User can edit his profile picture and username.
  - User can reset his password

- **System should:**
  - During the video call, System can translate the Arabic sign language into text.
  - System can also simulate the speech into Arabic sign language

2.5 Non-Functional Requirements

- Performance: the system should return the result as fast as possible.
- Scalability: the system performance should not change with higher workloads
- Reliability: the extent to which the system run without a failure for a given period of time under predefined conditions
- Availability: accessibility of the system to a user at a given point in time.
- Security: assuring all data inside the system will be protected against malware attacks or unauthorized access.
- Usability: the system should be easy to use and easy to learn Privacy: protecting personal information and undesired access to personal space.

2.6 System Users

- **Intended Users:**
  - Any person who wants to communicate with a deaf person through video call.
  - Any person who wants to communicate with a deaf person through video call can use the system. The system is particularly designed to facilitate communication between hearing and deaf individuals.

- **User characteristics:**
  - Users should be able to operate a smartphone with a camera and an internet connection.
  - Users should have a basic knowledge of how to make a video call with another person using their smartphone.

Chapter 3

Analysis and Design

3.1 System Overview

In this section we provide a high-level description of the system through eight diagrams. These diagrams illustrate the system's use cases, activities, sequences, states, classes, flow charts, block diagrams, and system architecture.

3.1.1 Use Case Diagram

The following figure represent the actions that the user can directly perform on the system namely:

- Registering if he has no account
- Logging in if he already has an account
- Editing his profile by editing one or more of the following: (username, password, profile pic).
- Starting a video call
- Answering an incoming video call
- Adding a new contact
- Searching any contact from his contacts
- Deleting any contact from his contacts
- Logging out his account.

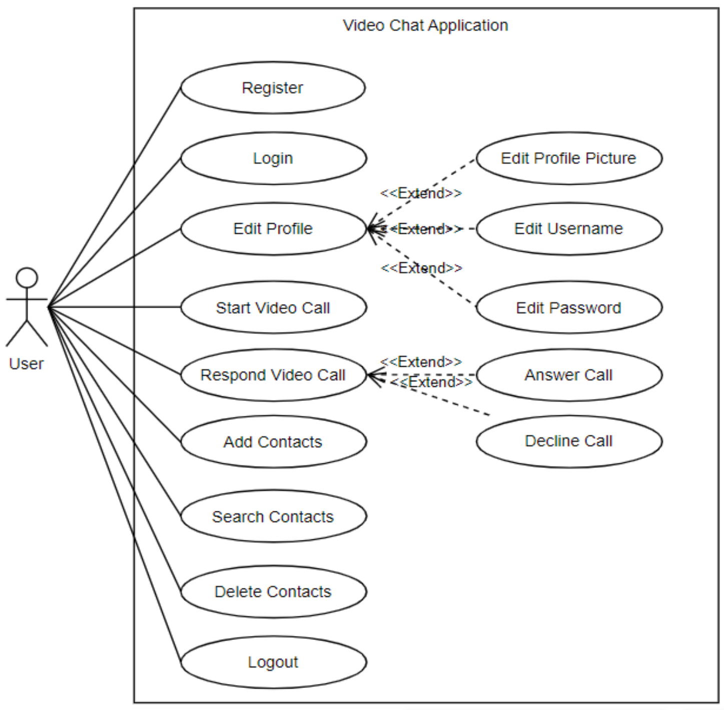


Figure 1: Use case diagram

3.1.2 Activity Diagram

The following figure represent the flow of activities that the user can do:

- It shows the flow of Login and Register activities, user can choose whether to login if he has account or register to create new account to access the home page. User will not be able to access the home page without verifying.
- After logging in user can edit profile, add contact, delete contact, search contact, start video call, answer video call and logging out his account.

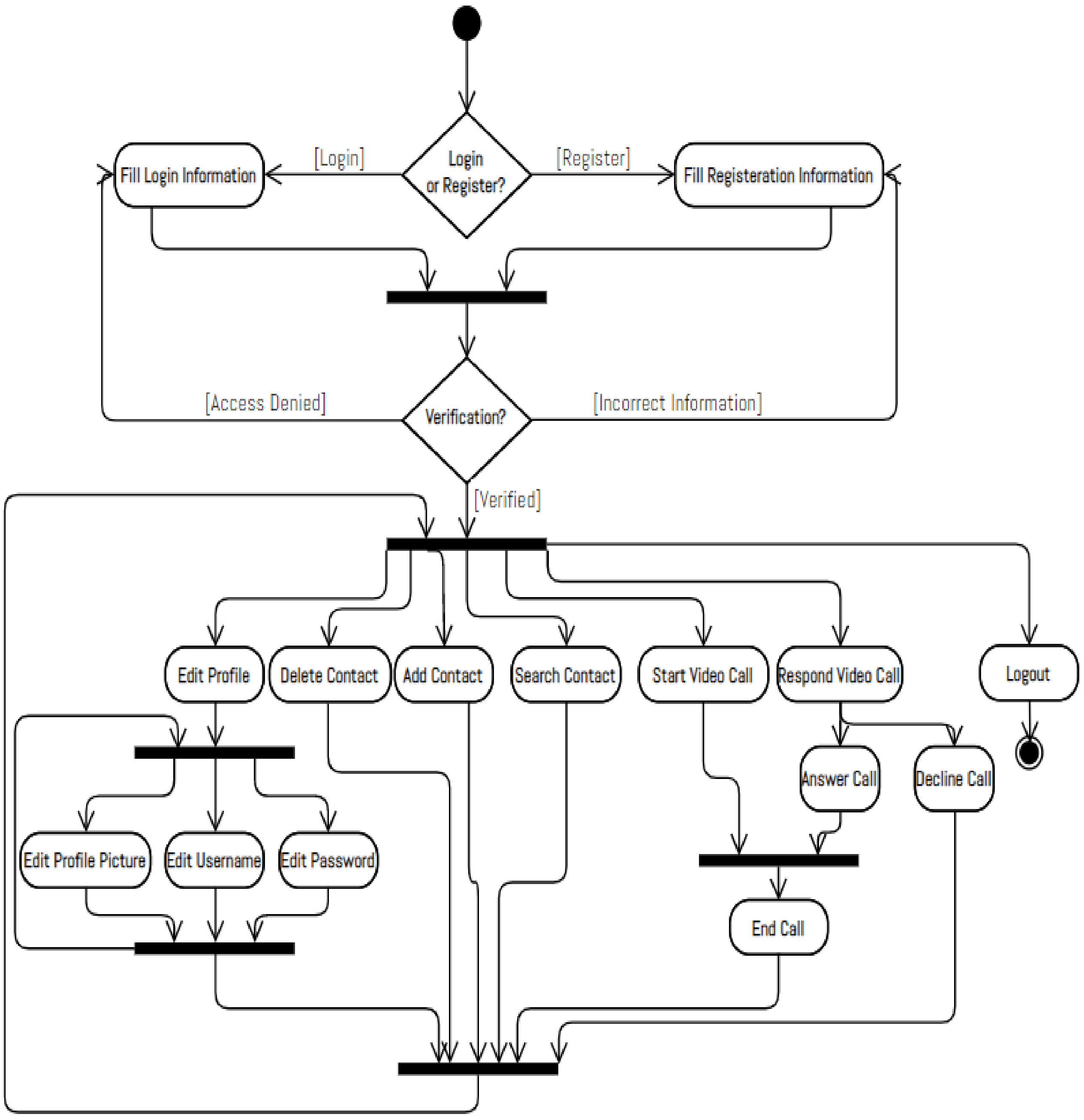


Figure 2: Activity diagram

3.1.3 Sequence Diagram

This diagram describes the sequence of actions that a user can make starting by logging in if he already has an account. After providing his username and password, the system will check with the database if they are valid or not. On the other hand, if he is a new user, he can also register and definitely the system will add his account to the database if it is valid. Once the user logged in successfully, he can edit his profile picture, password, or username and the system will update them in the database. He also can add any contact and if this contact has an account on the system the database will add this contact to his contact list. Moreover, the user can search for any contact and if he is already in his list, the database will return his profile and the user can start a video call or delete him. Furthermore, the user can answer any incoming call or end a call.

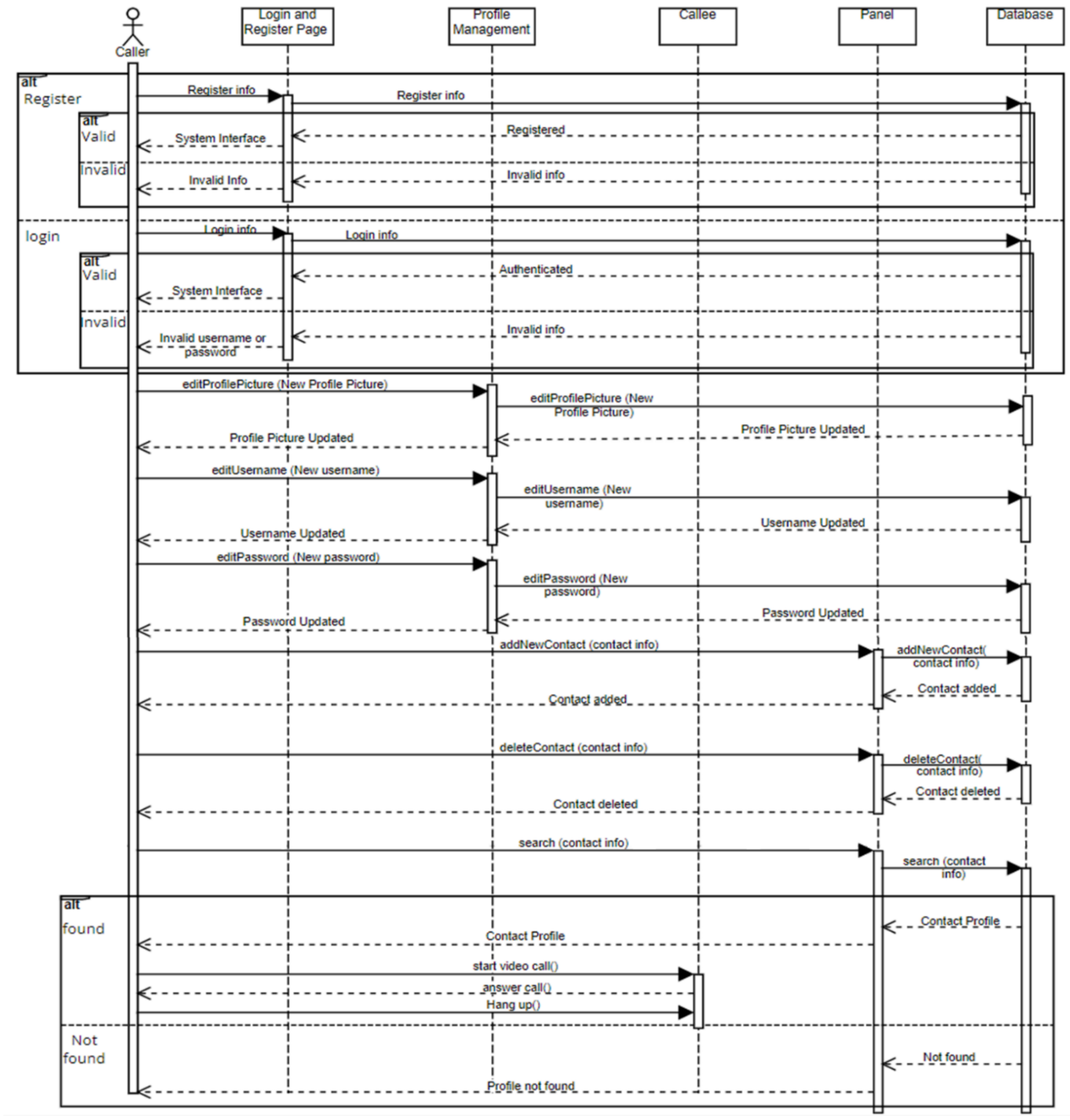


Figure 3: Sequence diagram

3.1.4 State Diagram

This Diagram shows the response of all the classes in the system to the user actions. Initially, the system waits for the user to either log in if they already have an account or register if they are a new user. If the user is new, they will fill out the registration form to create an account, and the system will verify the information provided by the user. If the information is valid, the user will be allowed to access the home page. If the user already has an account, they can log in and directly access the home page. Once the user has logged in to the home page, they can perform various actions, including editing their profile, starting, or answering a video call, adding or searching for a contact, deleting a contact, and logging out of their account.

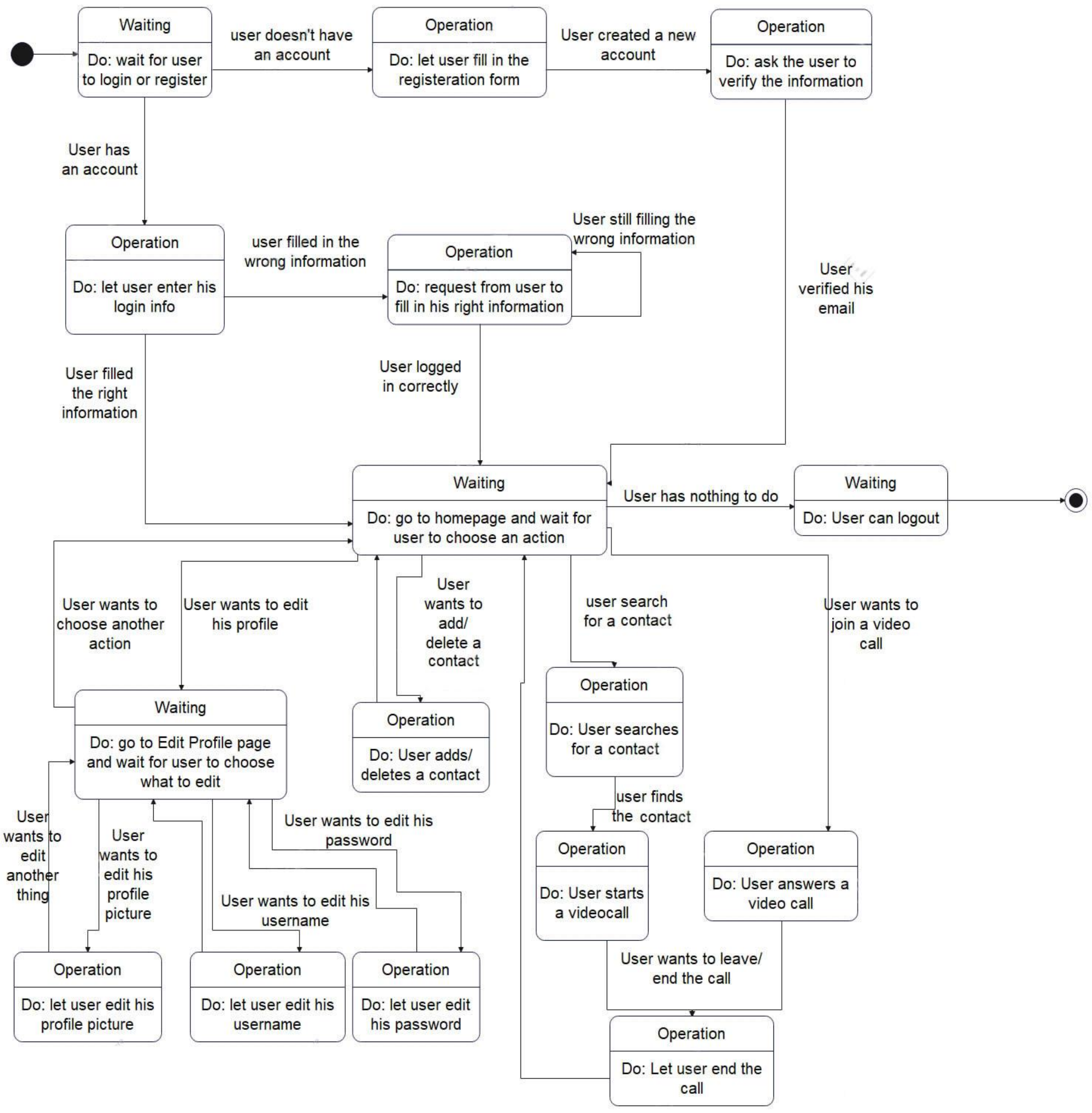

Waiting
Do: wait for user to login or register
user doesn't have an account
Operation
Do: let user fill in the registeration form
User created a new account
Operation
Do: ask the user to verify the information
User has an account
Operation
Do: let user enter his login info
user filled in the wrong information
Operation
Do: request from user to fill in his right information
User still filling the wrong information
User verified his email
User filled the right information
User logged in correctly
Waiting
Do: go to homepage and wait for user to choose an action
User has nothing to do
Waiting
Do: User can logout
User wants to choose another action
User wants to edit his profile
User wants to add/ delete a contact
user search for a contact
User wants to join a video call
Waiting
Do: go to Edit Profile page and wait for user to choose what to edit
Operation
Do: User adds/ deletes a contact
Operation
Do: User searches for a contact
user finds the contact
User wants to edit another thing
User wants to edit his profile picture
User wants to edit his password
User wants to edit his username
Operation
Do: User starts a videocall
Operation
Do: User answers a video call
User wants to leave/ end the call
Operation
Do: let user edit his profile picture
Operation
Do: let user edit his username
Operation
Do: let user edit his password
Operation
Do: Let user end the call

Figure 4: State Diagram

3.1.5 Class Diagram

The following figures represents the functions and attributes of the four main classes that the system use:

- User
- Profile
- Contact
- Panel

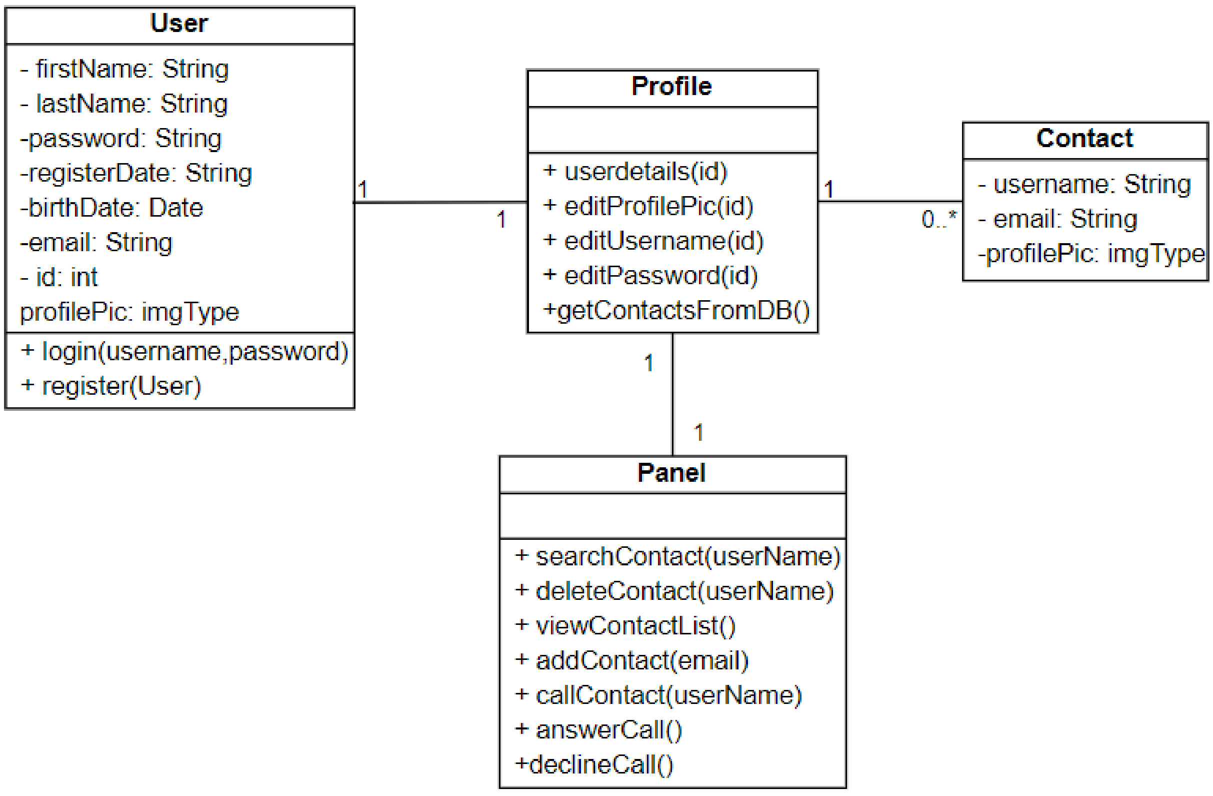


Figure 5: Class diagram

3.1.6 Flow Chart

In this section we provide a detailed description of the various functions in the system through flow charts. Each flow chart represents a specific function and outlines the steps involved in performing that function. The section covers the following functions: register and login, add or delete contact, start video call, answer video call, edit profile, search contact, logout, and video call analysis.

3.1.6.1 Register and Login

The following figure shows the steps of register and login function:

- The user can either register to create new account or login by his account.
- User can login by entering username and password.
- The system will let the user access the home page if the information is valid.

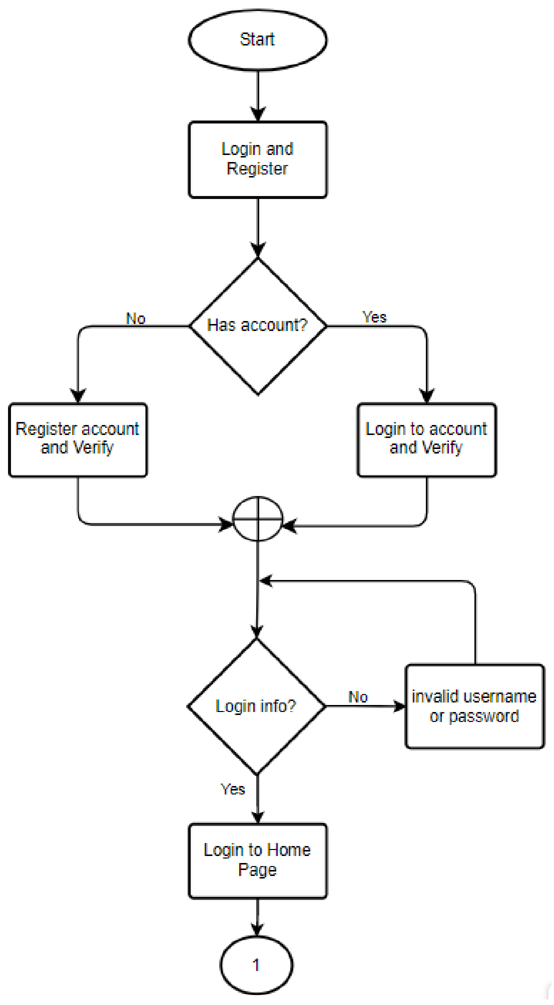

Start
Login and Register
Has account?
No
Yes
Register account and Verify
Login to account and Verify
Login info?
No
invalid username or password
Yes
Login to Home Page
1

Figure 6: Register and Login flow chart

3.1.6.2 Add or Delete Contact

The following figure shows the steps of add and delete contact function:

When user wants to add a new contact or delete an existing contact, he can search for the contact by entering his email or username if the contact found user can add it to the contact list or delete it from his contact list.

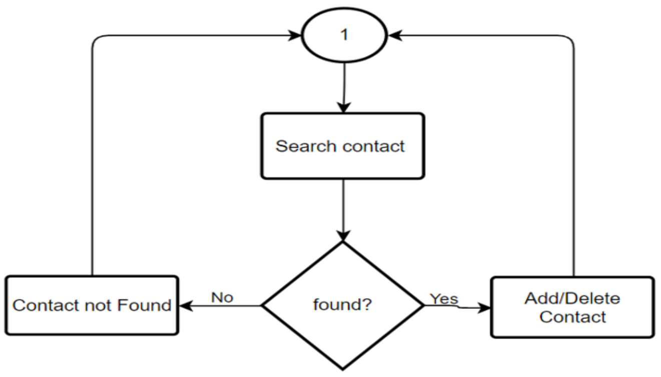


Figure 7: Add or Delete Contact flow chart

3.1.6.3 Start Video Call

The following figure shows the steps of starting a video call:

- When the user wants to start a video call, he can select any contact from his list then starting the video call.
- If the call ended user will return to the home page.

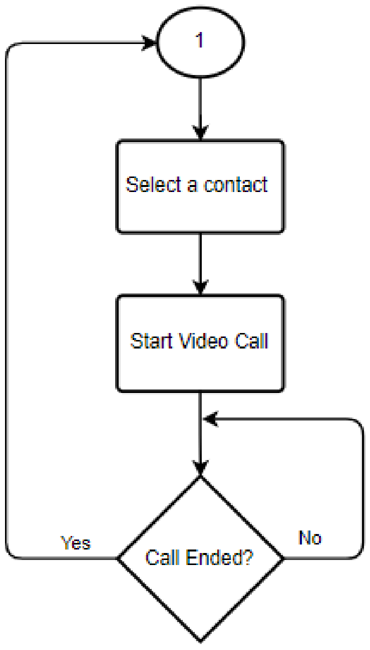


Figure 8: Start Video Call flow chart

3.1.6.4 Answer Video Call

This figure shows the workflow of answering video call function:

When the user has an incoming video call, he has the option to answer it and during the call he can end it, or he can hang it up in either case the user will return to the homepage again.

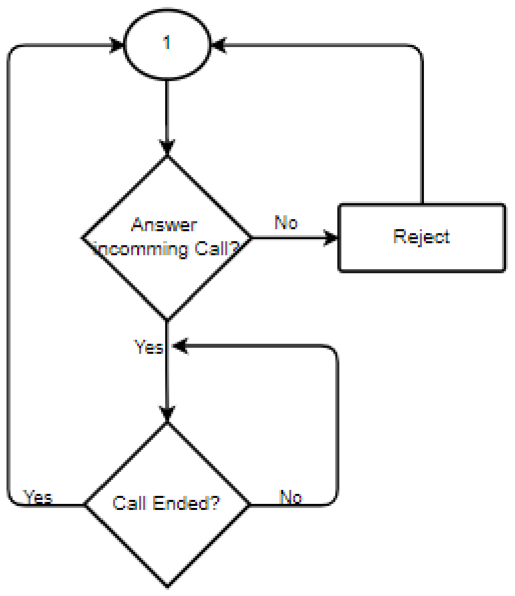


Figure 9: Answer Video Call flow chart

3.1.6.5 Edit Profile

This figure shows the workflow of edit profile function:

- When the user chooses to edit his profile, he can edit his password, pofile picture , or password.
- If his edits are valid the system will return the user to the homepage and if there was an error or the edits are invalid, the system will return the user to the edit profile screen again.

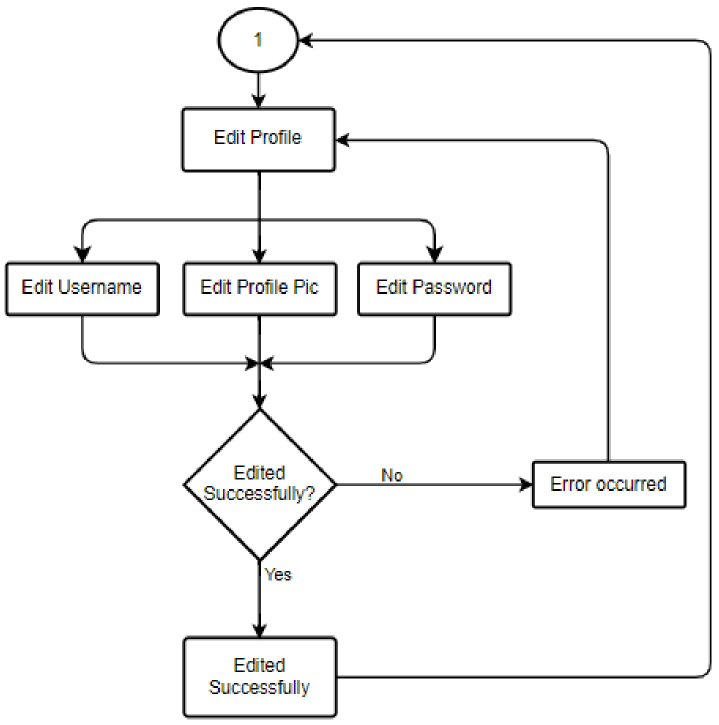


Figure 10: Edit Profile flow chart

3.1.6.6 Search Contact

This figure shows the workflow of search contact function:

- When user wants to search for a contact, he can type his username or email in the search bar.
- If the database found it in his contact list, the system will return his profile and contact information.
- If the contact was not found in the database the system will return to the homepage.

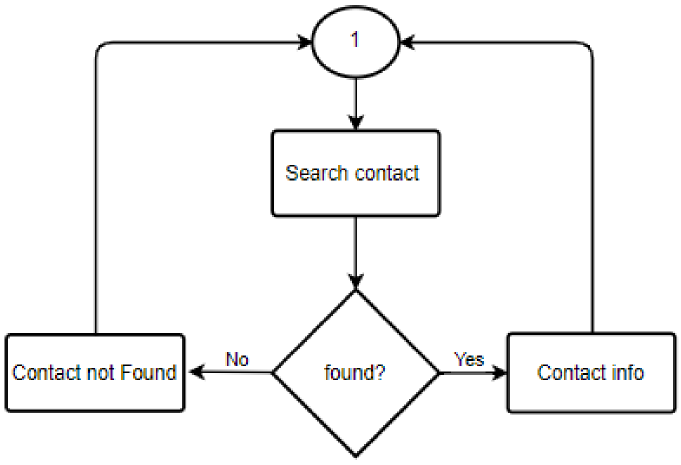


Figure 11: Search Contact flow chart

3.1.6.7 Logout

This figure shows the workflow of the logout function:

- The Logout flow chart outlines the steps involved in logging out of the user's account. The flow chart starts with the user being on the home page and wanting to log out. The user initiates the logout process by clicking the logout button.
- Once the user clicks the logout button, the system logs them out of their account and clears their session.

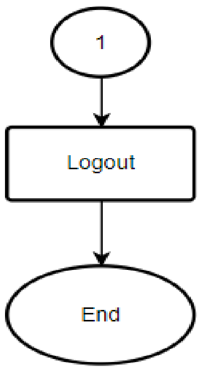


Figure 12: Logout flow chart

3.1.6.8 Video call

The video call flowchart captures the key steps involved in analyzing video frames during a call. It starts with extracting frames, drawing landmarks, and extracting keypoints. These keypoints are then sent to a model for analysis. If a sign is recognized, the model predicts the corresponding word. If not, the process continues with frame extraction until the call ends.

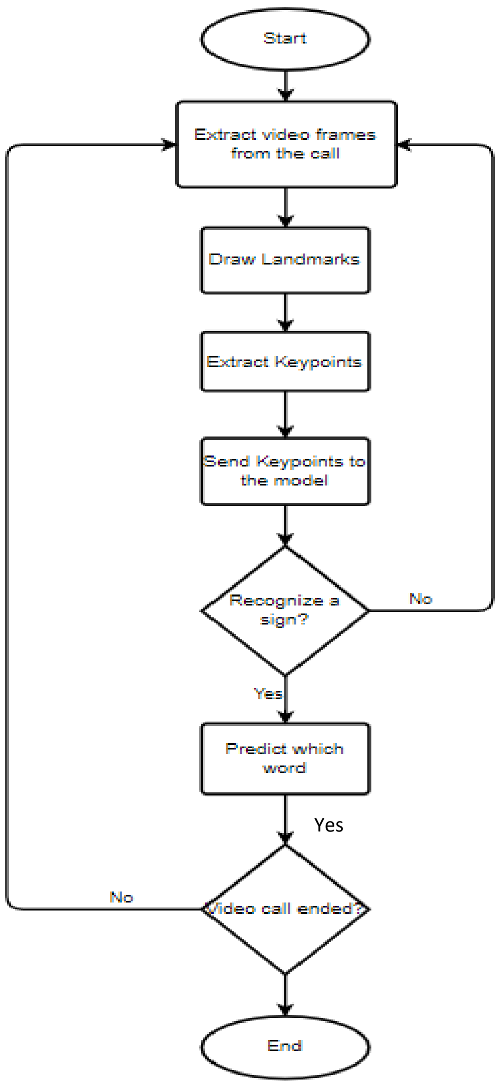


Figure 13: Video call flow chart

3.1.7 Block Diagram

The block diagram illustrates a system with two pathways for communication. In the "Voice to Sign" pathway, spoken words are converted to text and mapped to a sign language avatar for display. In the "Sign to Text" pathway, video frames of sign language gestures are classified to determine the corresponding text displayed on the screen.

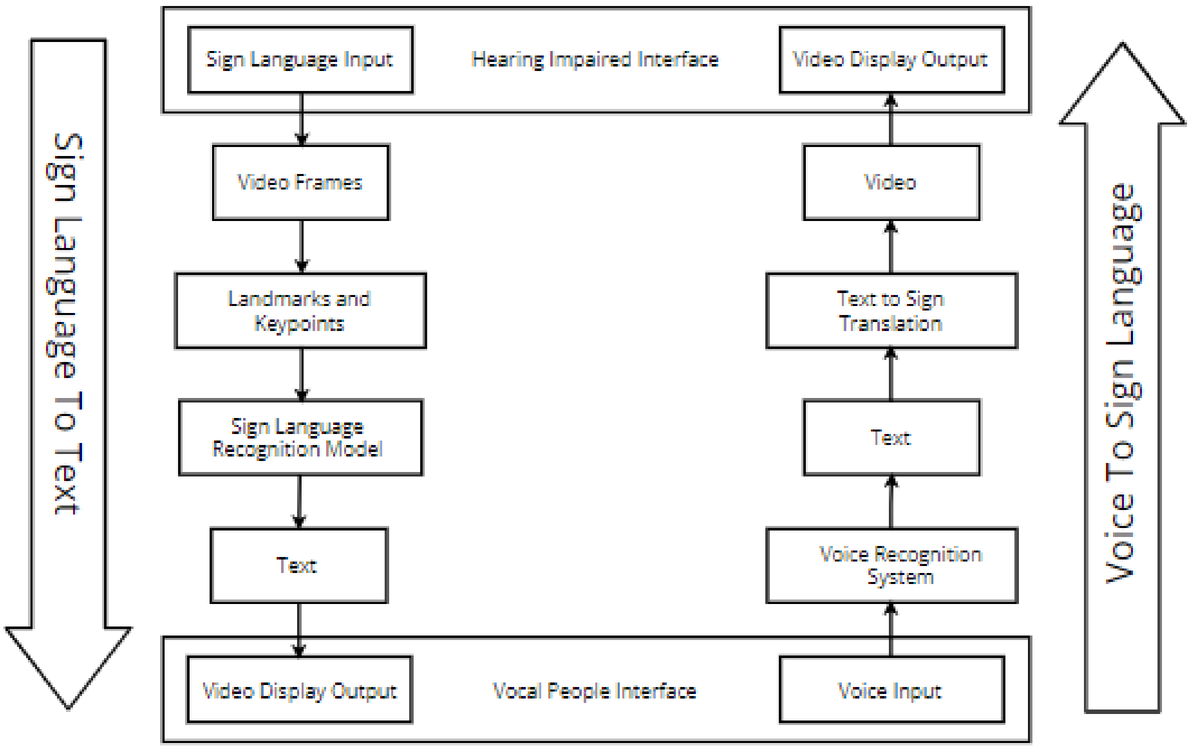


Figure 14: Block Diagram

3.1.8 System Architecture

The system architecture diagram is divided into four layers. The top layer includes the "Mobile User Interface" and "Video Call App." The second layer is the "Configuration Service," followed by the "Application Services" layer. The bottom layer consists of "Utility Services." This architecture ensures a user-friendly interface, customizable configurations, essential application functionalities, and supporting system services.

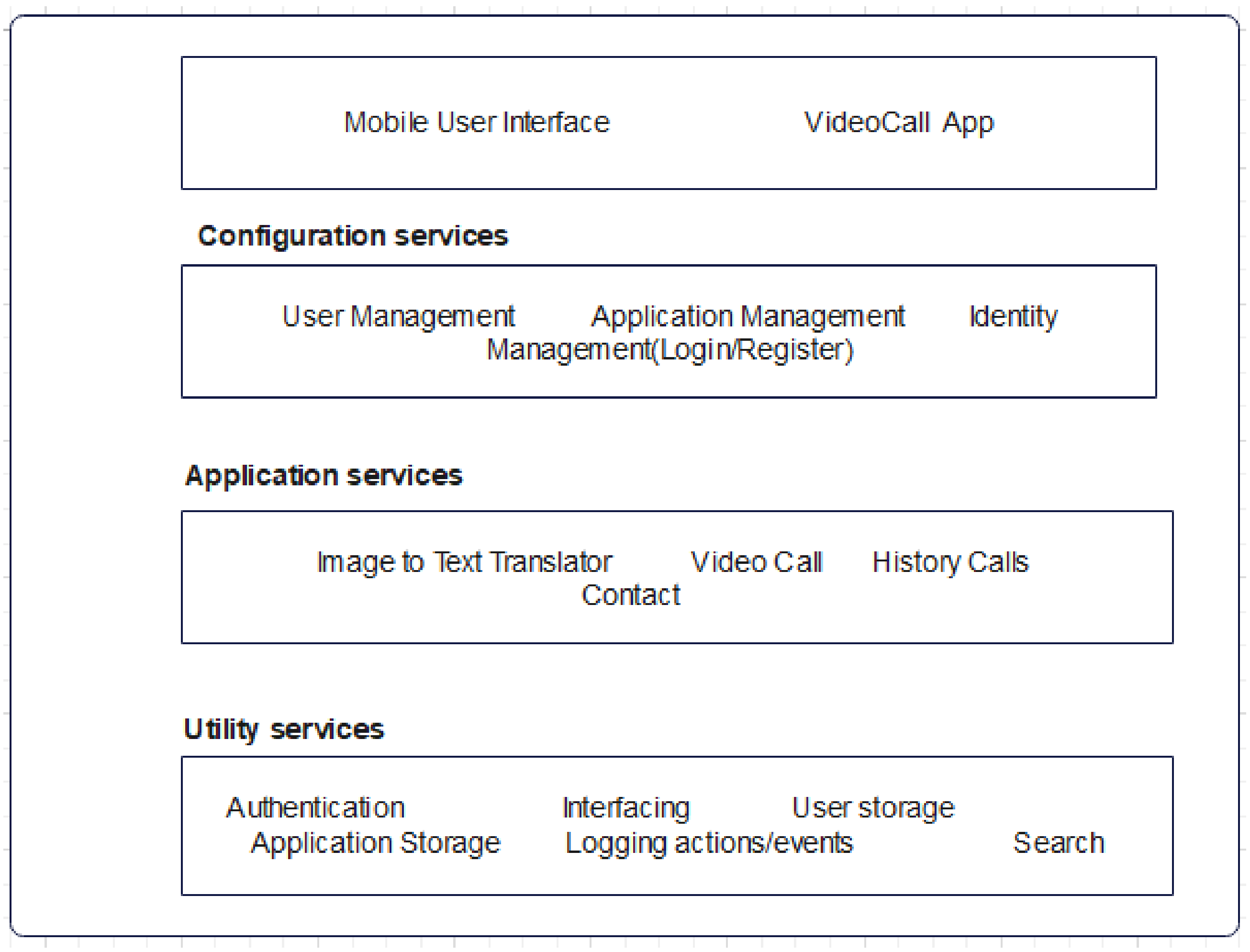


Figure 15: System Architecture

Chapter 4

Data and Implementation

4.1 The proposed Datasets

This section provides an overview of the datasets used in the project. The two subsections are the ArSL dataset, which consists of 3000 videos of 30 signs performed by 5 volunteers using a smartphone camera, and the Avatar dataset, which comprises 30 high-quality videos of single words represented through sign language gestures created using 3D animation technique.

4.1.1 ArSL Dataset

One of the primary goals of this study is to develop a dataset that accurately reflects natural circumstances and environments. According to 2020 statistics, nearly everyone owns a smartphone with a built-in camera [40]. To adhere to this trend, the dataset was created using smartphone videos without any hardware or software stabilization tools. These videos were captured in a range of resolutions and diverse locations, places, and backgrounds. A total of 3000 video were recorded for 30 signs from 5 volunteers, with each volunteer required to perform each sign 20 times (resulting in roughly 600 videos per volunteer).

4.1.2 Avatar Dataset

To accomplish the second part of our project, we utilized a 3D animation technique to produce avatar simulations of sign language gestures. This approach resulted in generating high-quality videos that accurately depict the hand and body movements required for voice-to-sign language conversion. The dataset comprises 30 videos, each featuring a single word represented through sign language gestures. This methodology allowed us to capture the precise positioning and movements of the hands and body, which are essential for creating an accurate sign language translation.

Table 1 provides an overview of all the words included in the dataset.

#	Word
1	اب
2	اعمل
3	التوفيق
4	الحمد لله
5	السلام عليكم
6	اليوم
7	ام
8	انا
9	انت
10	اين
11	تخرج

12	تنتهي
13	حزين
14	ذاهب
15	سعيد
16	شكرا
17	عفوا
18	على
19	عليكم السلام
20	قلق
21	كيف حالك
22	لماذا
23	متى
24	مسجد
25	مشروع
26	من عند الله
27	مناقشه
28	مهندس
29	و
30	وداعا

4.2 Sign to Text Translation

This section focuses on implementing the initial phase of our project, which involves translating sign language gestures into text. Through the use of computer vision and machine learning techniques, we aim to capture and analyze video frames of sign language gestures in real-time.

4.2.1 Extracting Key Points

In this section, we outline the steps required to prepare the dataset for training our sign language gesture detection model. The first step involves processing the dataset using the MediaPipe library to extract various key points, such as pose and hand landmarks. These key points serve as inputs to our model during training.

To prepare the dataset, we created a CSV file that listed the paths of all videos in the dataset, along with their corresponding labels. We then used the MediaPipe library to extract the key points from each video and save them as individual image frames. These frames were organized into sequences of a fixed length, which were used as inputs to our model during training.

By following these steps, we ensured that our dataset is properly formatted and ready to be used for training our sign language gesture detection model.

See appendix A.

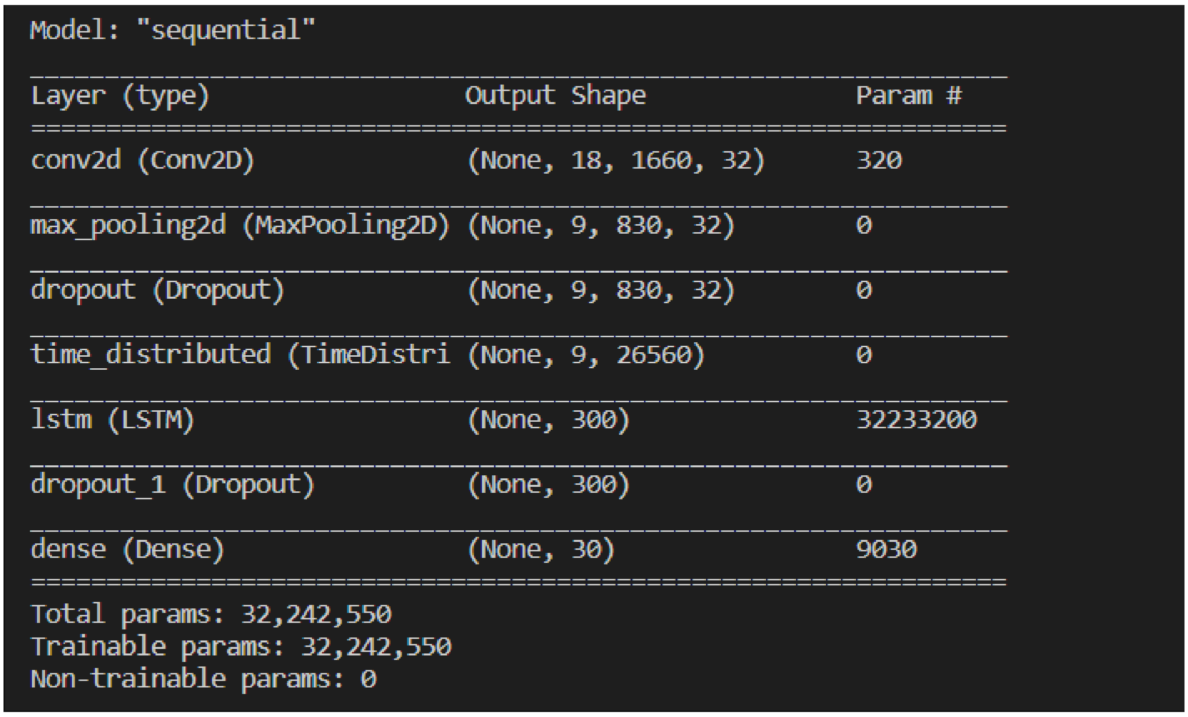
4.2.2 Model Training and Evaluation

In this section, we trained our model using the preprocessed dataset from the previous section. We used a deep learning model based on a Convolutional Neural Network (CNN) and a Long Short-Term Memory (LSTM) architecture to classify the sign language gestures. The model took sequences of fixed-length key point frames as input and output the predicted class label. We split the dataset into training and validation sets, and then trained the model using the training set while monitoring the validation loss to prevent overfitting. We also used early stopping to terminate the training process when the validation loss stopped decreasing. Finally, we evaluated the model's performance on the test set by calculating the accuracy. Then we saved the trained model for future use.

See appendix B.

4.2.3 Model Architecture

The model starts with a 2D convolutional layer that applies 32 filters of size 3x3 to the input images. This layer helps extract spatial features from the images. The output of the convolutional layer is then passed through a max-pooling layer with a pool size of 2x2, reducing the spatial dimensions of the features. To prevent overfitting, a dropout layer is added, randomly setting a fraction (0.05) of the input units to 0 during training. This regularization technique helps improve the model's generalization ability. Next, a TimeDistributed wrapper is used to apply the same operation, which is a flattened operation, to each time step of the input. This allows the model to handle sequential data effectively. An LSTM layer with 300 memory units is added to capture temporal dependencies and extract information from the flattened features. Another dropout layer with a dropout rate of 0.25 is included to further prevent overfitting and enhance the model's robustness. Finally, a dense layer with 4 units and a softmax activation function is added to produce the classification probabilities for the given image. This last layer maps the extracted features to the probabilities of belonging to each of the 4 classes.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)             (None, 18, 1660, 32)      320

_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 9, 830, 32)       0
_________________________________________________________________
dropout (Dropout)           (None, 9, 830, 32)        0
_________________________________________________________________
time_distributed (TimeDistri (None, 9, 26560)         0
_________________________________________________________________
lstm (LSTM)                 (None, 300)               32233200
_________________________________________________________________
dropout_1 (Dropout)         (None, 300)               0
_________________________________________________________________
dense (Dense)               (None, 30)                9030
=================================================================
Total params: 32,242,550
Trainable params: 32,242,550
Non-trainable params: 0
```

Figure 16: Model Architecture

4.2.4 Testing in Real-time

In this section, we used our pretrained model to simulate a real-world video call application. First, the saved model was loaded, and the camera was opened to predict the data in real-time then the predicted words were displayed on the webcam.
See appendix C.

4.3 Voice to Sign Translation

This section covers the implementation of the second part of our project, which involves transforming voice into an avatar that simulates sign language gestures. By utilizing speech recognition technology.

4.3.1 Speech Recognition

We have used google speech-to-text API to recognize the speech and convert it to Arabic text. The voiceReco() function uses the speech_recognition library to recognize speech using the default system microphone. The audio is first adjusted for ambient noise and then the recognize_google() method is used to transcribe the audio into Arabic text. The Arabic text is then reshaped and formatted using the arabic_reshaper and bidi libraries, respectively.
See appendix D.

4.3.2 Mapping Text to Avatar

The recognized words are then extracted from the transcribed text and stored in the recognizedWords list. For each recognized word in the recognizedWords list, the script loads a corresponding video file from a specified directory and adds it to the clips list. The video clips in the clips list are concatenated into a single video file, which is then displayed.

See appendix E.

4.4 Application Design Interface

This following screen serves as the entry point for users to establish their credentials and gain access to the app's features and functionalities. Users who are new to the app can easily register. For returning users, there is a login option where they can enter their registered email address and password to access their personalized account.

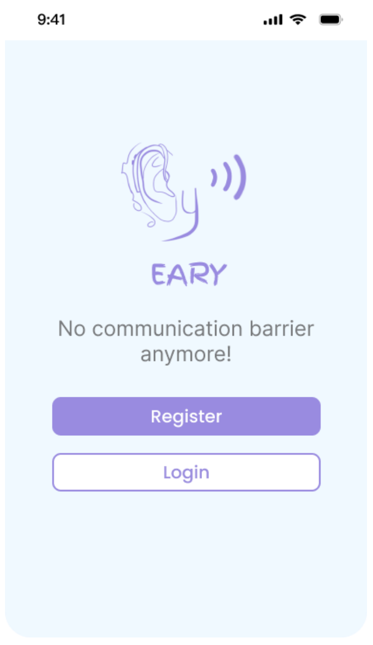


Figure 17: Login and register page

The following figure displays the user interface for creating a new account on the application. It includes a form that requires information about the user. Alternatively, they can sign up using their Google or Facebook account by clicking the corresponding button.

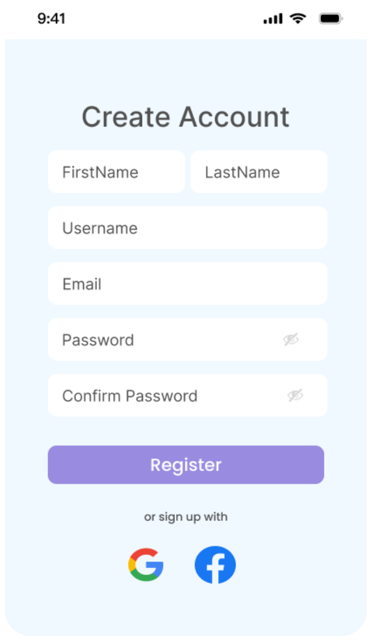


Figure 18: Register page

The following figure of the app is dedicated to user login. Users have the option to log in using their username and password, or they can choose to log in through their Google or Facebook accounts. This provides users with added convenience and flexibility, as they can utilize their existing social media credentials to access the app.

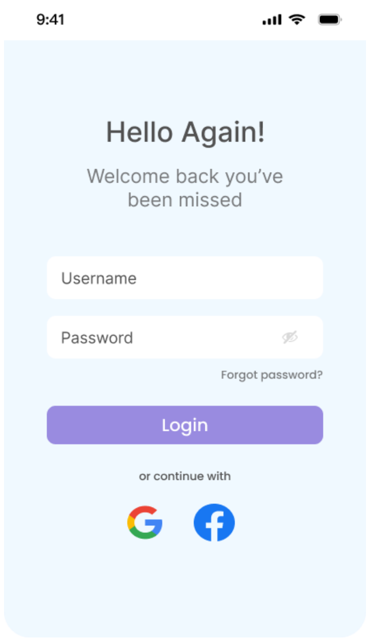


Figure19: Login page

The following figure displays the user interface for the main screen of the application. The page contains a contact list, which is ordered alphabetically, providing easy navigation for users. In addition, the home page includes a navigation drawer icon, user profile picture, and a search bar that allows users to search for contacts by email address to add them to their contact list.

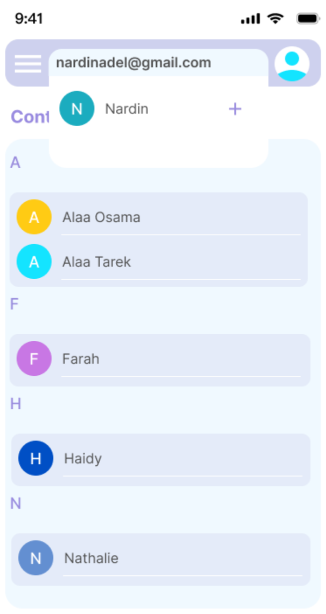


Figure 20: Search and add contact page

The following figure is the navigation drawer. It offers convenient access to various features and settings. Users can easily edit their profile, change language, and themes, access app settings, and log out. It serves as a central hub for customization and account management.

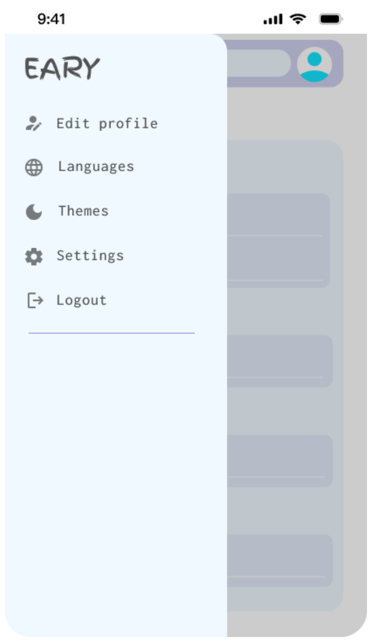
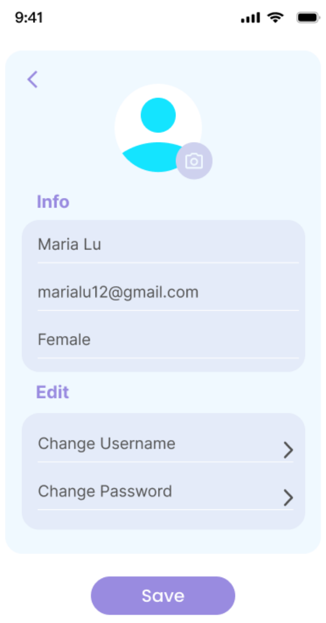


Figure 21: Navigation drawer

The following figure displays the user interface for the profile page. The page includes the user's profile picture that is editable, username, email, and gender information. In addition, the profile page includes an edit section where users can update their username and password. It includes also a save button to save the changes made.



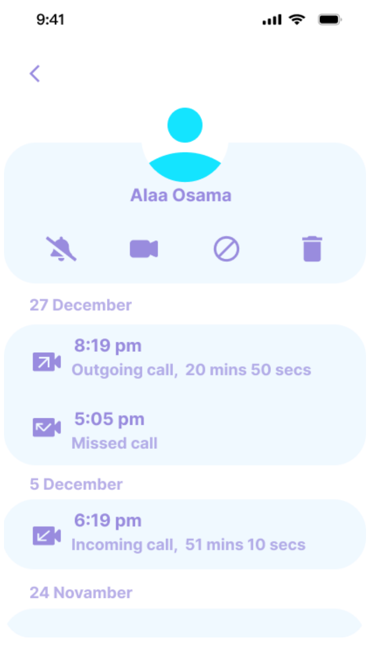
Maria Lu

marialu12@gmail.com

Female

Figure 22: Edit profile page

The contact info page displays the selected contact's name, profile picture, and call history. Users can call, delete, mute, or block the contact from this page. It provides a quick overview of the contact's details and facilitates efficient communication management.



9:41

Alaa Osama

27 December

8:19 pm
Outgoing call, 20 mins 50 secs

5:05 pm
Missed call

5 December

6:19 pm
Incoming call, 51 mins 10 secs

24 Novamber

Figure 23: Contact info page

The following figure displays the user interface for a video call on the application. The screen is divided into two sections, with the caller appearing in a small screen and the callee in the large screen. The Avatar also appears on the screen, simulating sign language translation from voice to sign. In addition, the interface includes a text box that displays the translation of the sign language into text. The video call also includes icons for ending the call, muting the microphone, and reversing the camera.

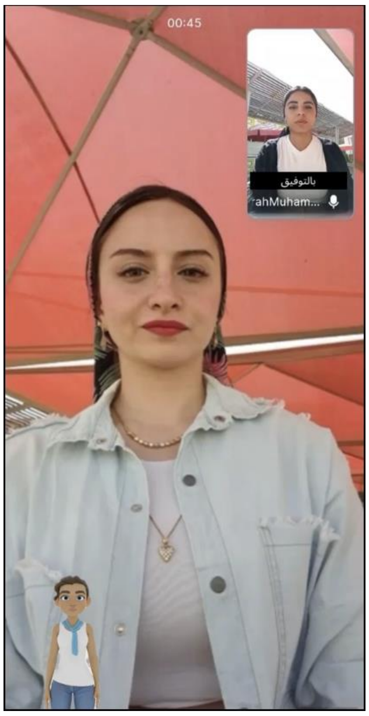


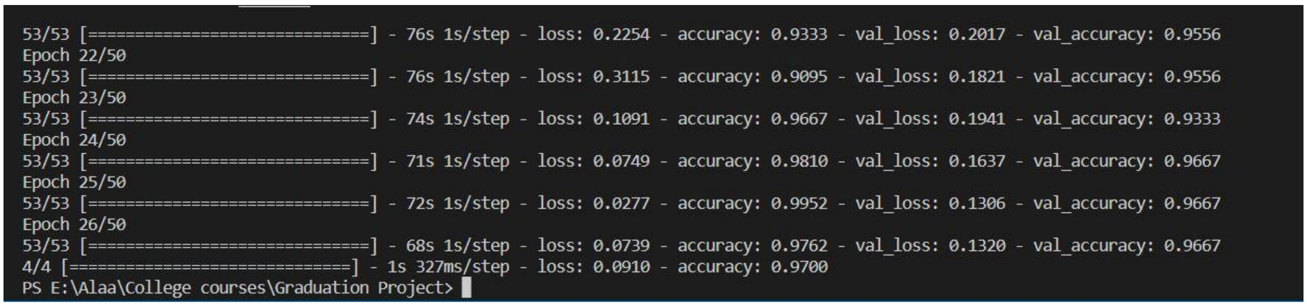
Figure 24: Video call page

Chapter 5

Testing and Maintenance

5.1 Model analysis

We have explored various model combinations to achieve optimal accuracy in our task. Initially, we utilized a 3D CNN due to the spatial and sequential components present in our data, which are crucial for accurate predictions. Our initial model consisted of three Conv3D layers, yielding an accuracy of 73%. However, this rate did not meet our expectations. Consequently, we experimented with a 3D ResNet101 model, resulting in an accuracy rate of 78%. Although this was an improvement, it still fell short of our desired accuracy. Subsequently, we explored alternative deep learning algorithms, such as the Skeleton-based Action Recognition Networks (SARNs) with LSTM, which yielded an accuracy of 86%. Finally, we decided to use a 2D CNN with LSTM and fine-tuned the hyperparameters to achieve an impressive accuracy of 97%.
The high accuracy rate of 97% achieved during testing further confirms the model's proficiency in recognizing sign language gestures and converting them into text. This indicates that our model is capable of understanding a wide range of gestures and consistently interpreting them with a high level of accuracy. The achieved loss value of 0.0910 indicates the model's ability to closely represent the true sign language gestures. This suggests that the model has successfully learned the underlying representations of sign language, enabling precise interpretation and conversion into textual representations. Additionally, the model's performance during both training and testing phases demonstrates its robustness and reliability. The consistent accuracy and low loss values obtained across multiple iterations highlight the stability and dependability of our model's performance.
The provided information is a screenshot taken from the terminal, displaying the accuracy and loss values obtained once the training of the model is completed.



```
53/53 [==============================] - 76s 1s/step - loss: 0.2254 - accuracy: 0.9333 - val_loss: 0.2017 - val_accuracy: 0.9556
Epoch 22/50
53/53 [==============================] - 76s 1s/step - loss: 0.3115 - accuracy: 0.9095 - val_loss: 0.1821 - val_accuracy: 0.9556
Epoch 23/50
53/53 [==============================] - 74s 1s/step - loss: 0.1091 - accuracy: 0.9667 - val_loss: 0.1941 - val_accuracy: 0.9333
Epoch 24/50
53/53 [==============================] - 71s 1s/step - loss: 0.0749 - accuracy: 0.9810 - val_loss: 0.1637 - val_accuracy: 0.9667
Epoch 25/50
53/53 [==============================] - 72s 1s/step - loss: 0.0277 - accuracy: 0.9952 - val_loss: 0.1306 - val_accuracy: 0.9667
Epoch 26/50
53/53 [==============================] - 68s 1s/step - loss: 0.0739 - accuracy: 0.9762 - val_loss: 0.1320 - val_accuracy: 0.9667
4/4 [==============================] - 1s 327ms/step - loss: 0.0910 - accuracy: 0.9700
PS E:\Alaa\College courses\Graduation Project>
```

Figure 25: Model Accuracy

5.2 Dataset Experimental Results

The experiments configurations are shown in Table 2.

Table 2 Experiments common configurations

Configuration	Value
Categories	30
Batch Size	4

Parameters Optimizer	Adam
Number of Epochs	50
Activation Function	Softmax
Performance Metrics	Accuracy, Loss
Training Environment	Windows 10, RAM 16 GB, GPU 8 GB, and Intel Core i7 Processor

The top-1 accuracies are reported for each class on validation and test sets. The results are shown in Table 3.

Table 3 Top-1 accuracies

Class Name	Validation %	Test %
اب	99%	97%
اعمل	95%	91%
التوفيق	100%	99%
الحمد لله	99%	98%
السلام عليكم	93%	89%
اليوم	98%	98%
ام	100%	99%
انا	97%	96%
انت	92%	90%
اين	97%	96%
تخرج	99%	98%
تنتهي	98%	97%
حزين	98%	96%
ذاهب	100%	98%
سعيد	100%	99%
شكرا	94%	94%
عفوا	98%	96%
على	100%	99%
عليكم السلام	93%	90%
قلق	100%	98%
كيف حالك	99%	99%
لماذا	98%	95%
متى	98%	96%
مسجد	100%	99%
مشروع	99%	96%
من عند الله	99%	97%
مناقشه	100%	99%
مهندس	97%	96%

و	93%	92%
وداعا	98%	97%

The results showed that five classes have very low accuracies in comparison to other classes. They are "و", "أنت", "عليكم السلام", "السلام عليكم", "أعمل". The reason behind these results is that almost all of them are similar in the kinematic movement and the sign performance. As a result, there was a conflict between these signs, which in turn caused these poor accuracy levels for these classes.

5.3 Voice-To-Sign

Initially, we used 2D avatars to simulate the signs of the recognized voice. These avatars were constructed based on the facial and hand mediapipe keypoints, as illustrated in Figure 18. However, we observed that this representation lacked a sense of realism, which was crucial for enhancing the user experience. In order to address this limitation and provide users with a more immersive and authentic interaction, we made the decision to transition to 3D avatars. The 3D avatars were meticulously crafted using the powerful 3D modeling software Blender, enabling us to create avatars that possess a heightened level of realism, thereby further enriching the user's engagement and overall experience.

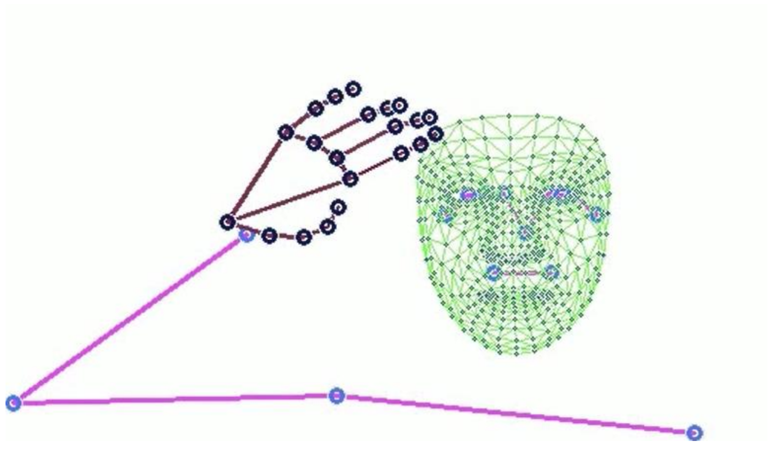


Figure 26: 2D avatar

Chapter 6
Conclusion and Future Work

7.1 Conclusion

In conclusion, our project presents a comprehensive and innovative solution for enhancing communication between individuals with hearing impairments and those who rely on vocal language. By leveraging advanced computer vision techniques, deep learning models, and speech recognition technology, we have developed a video call application that effectively translates sign language gestures into text and vice versa.
Through the integration of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, our model demonstrates remarkable accuracy in recognizing and classifying sign language gestures. The model is trained on a diverse dataset that we specifically created, focusing on user-dependent videos captured in real-world environments using smartphones. This user-dependent approach ensures adaptability to different users and environments, enabling more accurate and reliable gesture recognition.
Furthermore, our application incorporates voice recognition technology powered by the Google speech-to-text API. This allows for seamless translation of spoken words into text and the display of corresponding sign language gestures through an avatar. By simulating sign language visually, we enable effective communication between individuals who rely on different language modalities, bridging the gap between spoken and sign languages.
During our model analysis, we explored various architecture combinations and fine-tuned hyperparameters to achieve optimal performance. We experimented with 3D CNNs and 3D ResNet models, which showed promising accuracy rates but fell short of our desired results. We then turned to Skeleton-based Action Recognition Networks (SARNs) with LSTM, which improved the accuracy further. Ultimately, we settled on a 2D CNN with LSTM, achieving an impressive accuracy rate of 97%. The model's consistent accuracy and low loss values throughout training and testing phases validate its proficiency in recognizing and representing sign language gestures accurately.
The application's interface and user experience have been carefully designed to provide ease of use and accessibility. Features such as the navigation drawer with options for profile editing, language and theme customization, settings management, and logout, ensure a user-friendly and personalized experience.
Overall, our project showcases the potential of advanced technologies in empowering individuals with hearing impairments to engage in seamless communication. By facilitating the translation of sign language gestures into text and vice versa, our video call application enhances inclusivity and enables effective interaction in both personal and professional contexts. We are excited about the positive impact our project can have in promoting accessibility and breaking communication barriers for individuals with hearing impairments.

7.2 Future Work

In the ever-evolving landscape of technology, advancements are constantly being made to improve the performance and effectiveness of various applications. In the case of sign language recognition apps, there are several enhancements that can be implemented to create a more accurate, personalized, and user-friendly experience for individuals who rely on sign language as their primary means of communication. By enlarging the dataset, incorporating Egyptian Arabic, leveraging Natural Language Processing (NLP) techniques, utilizing 3D pose estimation, optimizing real-time communication, and fine-tuning the model, significant strides can be made in the field of sign language recognition.

First and foremost, enlarging the dataset used by the app can greatly enhance its performance. By adding new signs and involving more users in the data collection process, the system can become more adept at recognizing a wider range of signs and gestures. A diverse dataset that encompasses various sign languages and regional variations will enable the app to accommodate a larger user base and cater to a more extensive array of communication needs.
In addition to enlarging the dataset, a crucial enhancement to consider is the integration of Egyptian Arabic into the app's repertoire. While Standard Arabic is commonly used in sign language recognition systems, incorporating Egyptian Arabic can make the app more relevant and accurate for users who primarily use this dialect. This adjustment acknowledges the importance of catering to the specific linguistic needs and preferences of different user groups, ultimately providing a more inclusive and effective communication tool.
Another avenue for improvement lies in the utilization of Natural Language Processing (NLP) techniques to parse the recognized speech. By applying NLP algorithms, the app can extract more nuanced information from the recognized sign language, enabling a deeper understanding of the conveyed messages. This enhanced comprehension can lead to more seamless translation and interpretation, ultimately facilitating more effective communication between sign language users and non-sign language speakers.
Furthermore, exploring the use of 3D pose estimation instead of conventional 2D pose estimation can yield significant advancements in sign language recognition. 3D pose estimation involves capturing depth information, which can provide valuable context and enhance the accuracy of the recognition system. By incorporating depth perception, the app can better discern between similar signs and gestures, ultimately minimizing errors and improving the overall recognition accuracy.
Optimizing real-time communication is another vital aspect to consider when enhancing sign language recognition apps. Minimizing call latency, or the delay between sending a message and receiving a response, can significantly improve the user experience. Real-time communication is crucial in sign language interactions, and any delay can disrupt the flow of conversation and hinder effective communication. By leveraging cutting-edge tools and technologies, such as advanced networking protocols and optimized server infrastructure, the app can achieve near-instantaneous communication, providing users with a smooth and uninterrupted experience.
Lastly, fine-tuning the model is an essential step towards enhancing the accuracy and efficiency of the sign language recognition app. By experimenting with different architectural variations, adjusting hyperparameters, and incorporating regularization techniques, the model can be optimized to achieve better performance. Fine-tuning allows the app to adapt and improve over time, ensuring that it remains up to date with the latest advancements in the field of sign language recognition.
By implementing these various enhancements, our app can offer a more accurate, personalized, and user-friendly experience.

Appendix A:

```
import os
import cv2
import pandas as pd
import mediapipe as mp
import numpy as np

mp_holistic = mp.solutions.holistic  # Holistic model
mp_drawing = mp.solutions.drawing_utils  # Drawing utilities

def mediapipe_detection(image, model):
    # COLOR CONVERSION BGR 2 RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False                   # Image is no longer writeable
    results = model.process(image)                  # Make prediction
    image.flags.writeable = True                    # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)  # COLOR COVERSION RGB 2 BGR
    return image, results

def draw_styled_landmarks(image, results):
    # Draw face connections
    mp_drawing.draw_landmarks(image, results.face_landmarks, 
mp_holistic.FACEMESH_TESSELATION,
                              mp_drawing.DrawingSpec(
                                  color=(80, 110, 10), thickness=1, 
circle_radius=1),
                              mp_drawing.DrawingSpec(
                                  color=(80, 256, 121), thickness=1, 
circle_radius=1)
                              )
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, 
mp_holistic.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(
                                  color=(80, 22, 10), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(
                                  color=(80, 44, 121), thickness=2, circle_radius=2)
                              )
    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, 
mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(
                                  color=(121, 22, 76), thickness=2, 
circle_radius=4),
                              mp_drawing.DrawingSpec(
                                  color=(121, 44, 250), thickness=2, 
circle_radius=2)
```

```
                              )
    # Draw right hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(
                                  color=(245, 117, 66), thickness=2,
circle_radius=4),
                              mp_drawing.DrawingSpec(
                                  color=(245, 66, 230), thickness=2,
circle_radius=2)
                              )

def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten(
    ) if results.pose_landmarks else np.zeros(33*4)

    face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten(
    ) if results.face_landmarks else np.zeros(468*3)

    lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten(
    ) if results.left_hand_landmarks else np.zeros(21*3)

    rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten(
    ) if results.right_hand_landmarks else np.zeros(21*3)

    return np.concatenate([pose, face, lh, rh])

def create_csv(path):
    dataset_paths = os.listdir(path)
    rooms = []
    for item in dataset_paths:
        all_rooms = os.listdir(path + '/' + item)
        for room in all_rooms:
            rooms.append((item, str(path + '/' + item) + '/' + room))
    df = pd.DataFrame(data=rooms, columns=['label', 'video_path'])
    df = df.loc[:, ['video_path', 'label']]
    name = path.split("/")[-1].split(".")[0]
    df.to_csv(name + '.csv')

path = 'Dataset'
create_csv(path)
df = pd.read_csv("dataset.csv")
SEQUENCE_LENGTH = 20
```

```python
def create_dir(path):
    try:
        if not os.path.exists(path):
            os.makedirs(path)
    except OSError:
        print("Error: Creating directory with name {path}")

def save_frame(video_path, save_dir):
    label = video_path.split("/")[-2].split(".")[0]
    name = video_path.split("/")[-1].split(".")[0]
    save_path = os.path.join(save_dir, label, name)
    create_dir(save_path)
    # cap = cv2.VideoCapture(video_path)
    idx = 0
    # Read the Video File using the VideoCapture object.
    video_reader = cv2.VideoCapture(video_path)

    # Get the total number of frames in the video.
    video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))

    # Calculate the the interval after which frames will be added to the list.
    skip_frames_window = max(int(video_frames_count/SEQUENCE_LENGTH), 1)

    with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
        # Iterate through the Video Frames.
        for frame_counter in range(SEQUENCE_LENGTH):

            # Set the current frame position of the video.
            video_reader.set(cv2.CAP_PROP_POS_FRAMES,
                             frame_counter * skip_frames_window)

            # Reading the frame from the video.
            success, frame = video_reader.read()

            # Check if Video frame is not successfully read then break the loop
            if not success:
                break

            image, results = mediapipe_detection(frame, holistic)
            print(results)
            draw_styled_landmarks(image, results)
            keypoints = extract_keypoints(results)
            npy_path = f"{save_path}/{idx}"
            np.save(npy_path, keypoints)
            idx += 1

def prepare_all_videos(df):
    video_paths = df["video_path"].values.tolist()
    # labels = df["label"].values
    save_dir = "SignTranslator/Keypoints/"
    for path in video_paths:
        save_frame(path, save_dir)

prepare_all_videos(df)
```

Appendix B:

```
import os
import numpy as np
import random
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import TimeDistributed, Flatten

def seed_everything(seed=42):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)

# Set random seed
seed_everything()

# Define actions and corresponding labels
actions = np.array(['اب', 'اعمل', 'التوفيق', 'الحمد لله', 'السلام عليكم', 'اليوم',
                    'ام', 'انا', 'انت', 'اهلا وسهلا', 'اين', 'تخرج', 'تنتهي', 'حزين',
                    'ذاهب', 'رضيع', 'سعيد', 'شكرا', 'عفوا', 'على', 'عليكم السلام',
                    'قلق', 'كيف حالك', 'لماذا', 'متى', 'مسجد', 'مشروع',
                    'من عند الله', 'مناقشة', 'مهندس', 'و', 'وداعا'])

label_map = {label: num for num, label in enumerate(actions)}

# Load sequences and labels
sequences, labels = [], []
for action in actions:
    for sequence in range(100):
        window = []
        for frame_num in range(20):
            path = os.path.join("SignTranslator/Keypoints/", action,
                                f"{action} ({sequence+1})",
"{}.npy".format(frame_num))
            res = np.load(path)
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])

# Convert labels to one-hot encoding
y = to_categorical(labels).astype(int)

# Reshape sequences
X = np.array(sequences)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, shuffle=True, random_state=seed_everything())

X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], X_train.shape[2],
1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], X_test.shape[2], 1))

# Define the model architecture
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), input_shape=X_train.shape[1:4],
activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(0.05))
model.add(TimeDistributed(Flatten()))
model.add(layers.LSTM(300))
model.add(layers.Dropout(0.25))
model.add(layers.Dense(4, activation='softmax'))

# Print model summary
model.summary()

# Define Early Stopping Callback
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10, mode='min',
restore_best_weights=True)

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='Adam',
metrics=["accuracy"])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=4, validation_split=0.3,
          shuffle=True, callbacks=[early_stopping_callback])

# Evaluate the model
model_evaluation_history = model.evaluate(X_test, y_test)

# Save the model
model.save('model.h5')
```

Appendix C:

```
import cv2
import numpy as np
import mediapipe as mp
from keras.models import load_model
import arabic_reshaper
from bidi.algorithm import get_display
from PIL import ImageFont, ImageDraw, Image

mp_holistic = mp.solutions.holistic  # Holistic model
mp_drawing = mp.solutions.drawing_utils  # Drawing utilities

actions = np.array(['اب', 'اعمل', 'التوفيق', 'الحمد لله', 'السلام عليكم', 'اليوم',
                    'ام', 'انا', 'انت', 'اهلا وسهلا', 'اين', 'تخرج', 'تنتهي', 'حزين',
                    'ذاهب', 'رضيع', 'سعيد', 'شكرا', 'عفوا', 'على', 'عليكم السلام',
                    'قلق', 'كيف حالك', 'لماذا', 'متى', 'مسجد', 'مشروع',
                    'من عند الله', 'مناقشة', 'مهندس', 'و', 'وداعا'])

model = load_model('model.h5')

def mediapipe_detection(image, model):
    # COLOR CONVERSION BGR 2 RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False                  # Image is no longer writeable
    results = model.process(image)                 # Make prediction
    image.flags.writeable = True                   # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)  # COLOR COVERSION RGB 2 BGR
    return image, results

def draw_styled_landmarks(image, results):
    # Draw face connections
    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_TESSELATION,
                              mp_drawing.DrawingSpec(
                                  color=(80, 110, 10), thickness=1,
circle_radius=1),
                              mp_drawing.DrawingSpec(
                                  color=(80, 256, 121), thickness=1,
circle_radius=1)
                              )
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(
                                  color=(80, 22, 10), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(
                                  color=(80, 44, 121), thickness=2, circle_radius=2)
                              )
    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(
                                  color=(121, 22, 76), thickness=2,
circle_radius=4),
                              mp_drawing.DrawingSpec(
                                  color=(121, 44, 250), thickness=2,
circle_radius=2)
                              )
    # Draw right hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(
                                  color=(245, 117, 66), thickness=2,
circle_radius=4),
                              mp_drawing.DrawingSpec(
                                  color=(245, 66, 230), thickness=2,
circle_radius=2)
                              )

colors = [(245, 117, 16), (117, 245, 16), (16, 117, 245)]

def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten(
    ) if results.pose_landmarks else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten(
    ) if results.face_landmarks else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten(
    ) if results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten(
    ) if results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])

# 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.5

cap = cv2.VideoCapture(0)
# Set mediapipe model
```

```python
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)
as holistic:
    while cap.isOpened():

        # Read feed
        ret, frame = cap.read()

        # Make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)

        # Draw landmarks
        draw_styled_landmarks(image, results)

        # 2. Prediction logic
        keypoints = extract_keypoints(results)
        sequence.append(keypoints)
        sequence = sequence[-20:]

        if len(sequence) == 20:
            res = model.predict(np.expand_dims(
                np.expand_dims(sequence, axis=2), axis=0))[0]
            print(actions[np.argmax(res)])
            predictions.append(np.argmax(res))

        # 3. Viz logic
            if res[np.argmax(res)] > threshold:
                reshaped_text = arabic_reshaper.reshape(
                    actions[np.argmax(res)])
                bidi_text = get_display(reshaped_text)
                if len(sentence) > 0:
                    if bidi_text != sentence[-1]:
                        sentence.append(bidi_text)
                else:
                    sentence.append(bidi_text)

            if len(sentence) > 5:
                sentence = sentence[-5:]

        fontpath = "arial.ttf"  # <== https://www.freefontspro.com/14454/arial.ttf
        font = ImageFont.truetype(fontpath, 32)
        img_pil = Image.fromarray(image)
        draw = ImageDraw.Draw(img_pil)
        draw.text((50, 80), ' '.join(sentence), font=font)

        # Show to screen
        img = np.array(img_pil)
        cv2.imshow('OpenCV Feed', img)
        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Appendix D:

```
import speech_recognition
import arabic_reshaper
from bidi.algorithm import get_display

def voiceReco():
    recognizer = speech_recognition.Recognizer()
    with speech_recognition.Microphone() as mic:
        recognizer.adjust_for_ambient_noise(mic, duration=1)
        audio = recognizer.listen(mic)
        text = recognizer.recognize_google(audio, language='ar-AR')
        reshaped_text = arabic_reshaper.reshape(text)
        bidi_text = get_display(reshaped_text)
        return bidi_text, text

bidi_text, text = voiceReco()
```

Appendix E:

```
from moviepy.editor import VideoFileClip, concatenate_videoclips
import cv2


def showVideo(path):
    cap = cv2.VideoCapture(path)
    if (cap.isOpened() == False):
        print("Error opening video file")

    while (cap.isOpened()):
        ret, frame = cap.read()
        if ret == True:
            cv2.imshow('Frame', frame)
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break

        else:
            break
    cap.release()
    cv2.destroyAllWindows()
```

```python
def convert(lst):
    lst = lst.split()
    recognizedWords = []
    i = 0
    while i < len(lst):
        if lst[i] in sign_names:
            recognizedWords.append(lst[i])
            i += 1
        else:
            if i < len(lst) - 1:
                merged_word = lst[i] + ' ' + lst[i+1]
                if merged_word in sign_names:
                    recognizedWords.append(merged_word)
                    i += 2
                else:
                    i += 1
    return recognizedWords


sign_names = ['اب', 'اعمل', 'التوفيق', 'الحمد لله', 'السلام عليكم', 'اليوم',
                    'ام', 'انا', 'انت', 'اهلا وسهلا', 'اين', 'تخرج', 'تنتهي', 'حزين',
                    'ذاهب', 'رضيع', 'سعيد', 'شكرا', 'عفوا', 'على', 'عليكم السلام',
                    'قلق', 'كيف حالك', 'لماذا', 'متى', 'مسجد', 'مشروع',
                    'من عند الله', 'مناقشة', 'مهندس', 'و', 'وداعا']

recognizedWords = []
clips = []
lst = convert(text)
print(lst)
for w in lst:
    if w in sign_names:
        recognizedWords.append(w)
        print(recognizedWords)
    if (text == "اغلق"):
        break
for word in recognizedWords:
    path = 'F:/speech to text/animation/'+word+'.mkv'
    print(path)
    clip = VideoFileClip(path)
    clips.append(clip)
final_clip = concatenate_videoclips(clips)
final_clip.write_videofile("my_concatenation.avi", codec='libx264')

showVideo('my_concatenation.avi')
```

References

[1] T. Shanableh, K. Assaleh and M. AL–Rousan, "Spatio–Temporal feature extraction techniques for isolated Arabic sign language recognition," IEEE Transactions on Systems, Man and Cybernetics Part B, 37(3), June, 2007.

[2] I. Kryvonos, I. Krak and W. Wojcik, "Information technologies applications for sign languages investigations," Computer Science and Information Technologies (CSIT), 2015, Yerevan, 2015, pp. 148-150.

[3] J. Wu, Z. Tian, L. Sun, L. Estevez and R. Jafari, "Real-time American Sign Language Recognition using wrist-worn motion and surface EMG sensors," 2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN), Cambridge, MA, 2015, pp. 1- 6.

[4] N. Yadav, S. Thepade and P. H. Patil, "Noval approach of classification based Indian sign language recognition using transform features," 2015 International Conference on Information Processing (ICIP), Pune, 2015, pp. 64-69

[5] C. Chansri and J. Srinonchat, "Reliability and accuracy of Thai sign language recognition with Kinect sensor," 2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Chiang Mai, 2016, pp. 1-4.

[6] C. M. Jin, Z. Omar and M. H. Jaward, "A mobile application of American sign language translation via image processing algorithms," 2016 IEEE Region 10 Symposium (TENSYMP), Bali, 2016, pp. 104-109

[7] Keskin C, Kırac¸ F, Kara YE, Akarun L (2013) Real time hand pose estimation using depth sensors in consumer depth cameras for computer vision. Springer, pp 119–137

[8] Mehdi SA, Khan YN (2002) Sign language recognition using sensor gloves. In: Proceedings of the 9th international conference on neural information processing, 2002. ICONIP'02, vol 5. IEEE, pp 2204– 2206

[9] Yang S, Zhu Q (2017) Continuous chinese sign language recognition with cnn-lstm. In: Ninth international conference on digital image processing (ICDIP 2017). (International Society for Optics and Photonics), vol 10420, p 104200F

[10] Nandy A, Prasad JS, Mondal S, Chakraborty P, Nandi GC (2010) Recognition of isolated indian sign language gesture in real time. In: International conference on business administration and information processing. Springer, pp 102–107

[11] Lopez-Noriega JE, Fern ´ andez-Valladares MI, Uc-Cetina V (2014) Glove-based sign language recognition solution to assist communication for deaf users. In: 2014 11th International conference on electrical engineering, computing science and automatic control (CCE). IEEE, pp 1–6

[12] Starner TE (2008) Visual recognition of american sign language using hidden Markov models. Massachusetts Inst Of Tech Cambridge Dept Of Brain And Cognitive Sciences. Technical report

[13] Reading the signs: Diverse arabic sign languages. Reading the Signs: Diverse Arabic Sign Languages | Center for Strategic and International Studies. (n.d.). Retrieved December 26, 2022.

[14] UCSF Health. (2022, June 24). Communicating with people with hearing loss. ucsfhealth.org. Retrieved December 26, 2022.

[15] World Health Organization. (n.d.). Deafness and hearing loss. World Health Organization. Retrieved December 26, 2022.

[16] Alselwi, G., & Tasci, T. (n.d.). (PDF) arabic sign language recognition: A review - researchgate. Retrieved December 26, 2022.

[17] Alsaadi, Z.; Alshamani, E.; Alrehaili, M.; Alrashdi, A.A.D.; Albelwi, S.; Elfaki, A.O. A Real Time Arabic Sign Language Alphabets (ArSLA) Recognition Model Using Deep Learning Architecture. Computers 2022, 11, 78.

[18] Balaha, M.M., El-Kady, S., Balaha, H.M. et al. A vision-based deep learning approach for independent-users Arabic sign language interpretation. Multimed Tools Appl (2022).

[19] M.El-Gayyar, M., S.Ibrahim, A., & M.E.Wahed. (2016, May 18). Translation from Arabic speech to arabic sign language based on cloud computing. Egyptian Informatics Journal. Retrieved December 26, 2022.

[20] What is deep learning? IBM. (n.d.). Retrieved December 26, 2022.

[21] Saha, S. (2022, November 16). A comprehensive guide to Convolutional Neural Networks-the eli5 way. Medium. Retrieved December 26, 2022.

[22] says, K. L. (2022, December 13). What is LSTM - introduction to long short term memory. Intellipaat Blog. Retrieved December 26, 2022.

[23] Kukil. (2022, November 18). Introduction to MediaPipe. LearnOpenCV. Retrieved December 26, 2022.

[24] A.E.E.El Alfi, M.M.R.El Basuony, S.M.El Atawy Intelligent Arabic text to arabic sign language translation for easy deaf communication Int J Comput Appl, 92 (2014), pp. 22-29

[25] O. Al-Jarrah and A. Halawani, “Recognition of gestures in arabic sign language using neuro-fuzzy systems,” Artif. Intell., vol. 133, no. 1–2, pp. 117–138, Dec. 2001

[26] A. Tharwat, T. Gaber, A. E. Hassenian, M. K. Shahin, and B. Refaat, “SIFT-Based Arabic Sign Language Recognition System,” vol. 334, Nov. 2014.

[27] N. Tubaiz, T. Shanableh, and K. Assaleh, “Glove-based continuous arabic sign language recognition in user-dependent mode,” IEEE Trans. Hum.-Mach. Syst., vol. 45, no. 4, pp. 526–533, Aug. 2015.

[28] B.Garcia, S. Viesca, “Real-time American Sign Language Recognition with Convolutional Neural Networks” 2016.

[29] L. Pigou, S. Dieleman, P. Kindermans, B. Schrauwen. “Sign Language Recognition using Convolutional Neural Networks” 2015.

[30] Pose. mediapipe. (n.d.). Retrieved December 27, 2022.

[31] Ahmed, Z. (2023, January 9). ZEGOCLOUD video calling UIKits in flutter. DEV Community. Retrieved April 4, 2023.

[32] *Video calling product overview: Agora docs*. Video Calling Product overview | Agora Docs. (n.d.). Retrieved April 4, 2023.

[33] Foundation, B. (n.d.). *About*. blender.org. Retrieved April 4, 2023.

[34] Google. (n.d.). *Speech-to-text: Automatic speech recognition | google cloud*. Google. Retrieved April 4, 2023.

[35] O. O. K. a. H. E. Khalid Khalil El-DarymLi, “”Speech to Sign Language Interpreter System (SSLIS)”,” in IEEE International Conference of Computer and Communication Enginnering (ICCCE’06), Kuala Lumpur, Malaysia, 2006.

[36] T. J. L. a. W. W. L. Oi Mean Foong, “”V2S: Voice to Sign Language Translation System for Malaysian Deaf People “,” in International Visual Informatics Conference, Tronoh,Malaysia, 2009

[40] *39+ smartphone statistics you should know in 2023*. Review42. (2023, March 17).