SCHOOL OF COMPUTING AND ENGINEERING SCIENCES

Real-Time Sign Language Gesture Recognition and Translation to Text System using CNN LSTMs.

STUDENT NUMBER: 146173

An Informatics and Computer Science Project Proposal Submitted to the School of Computing and Engineering Sciences in partial fulfillment of the requirements for the award of a Degree in Bachelor of Science in Informatics and Computer Science

Date of Submission: January 2025

Declaration and Approval

We declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of our knowledge and belief, the research proposal contains no material previously published or written by another person except where due reference is made in the research proposal itself.

Student Name:	Traciebel Wairimu
Admission Number:	146173

Student Signature: [illegible] Date: 16/12/2024

The Proposal of **Traciebel Wairimu** has been reviewed and approved by **Mr. Stephen Oketch Obonyo**

Supervisor Signature: [illegible] Date: 16/12/2024

Abstract

Savvy Signing will be designed to mainly break the communication barriers between the hearing impaired and those that can hear. Sign language being a mode of communication used by hearing impaired people that involves the use of gestures and signs. This system will be designed to recognize these gestures and signs as an individual makes the hand motions and make sense of these hand motions producing actual spoken or written words and sentences that are meaningful. This application will assist individuals who use sign language to effortlessly communicate with anyone from anywhere. The application fosters inclusivity by providing a convenient tool for seamless conversation. Signing is not a language that is normally taught in schools, it is a language that if an individual wants to learn it, they must look for an institution or individual that teaches the skill. We meet a lot of people who have hearing impairment and the only way we can understand each other and have a conversation with them is through making gestures and body movement. Nevertheless, it can be challenging to communicate with people who have hearing disabilities and learning sign language can be complicated, and it is not a skill that most people have. There are hearing aids that have been introduced to help people with hearing disabilities to hear sounds and be able to hear like a normal person by wearing these hearing aids but not everyone has access to these resources or even information on how to get these hearing aids installed or prescribed. An application that can be found online by searching its name when browsing on your laptop is quite convenient. Learning and using sign language is not only an act of inclusivity but also a way to foster understanding, empathy, and equal opportunities for all. This system will allow deaf people to fully participate in various domains of life, including education, employment, healthcare, and social interactions. Rather than viewing deafness as a limitation, embracing deafness entails recognizing it as a part of an individual’s identity and acknowledging the richness of deaf culture and sign language.

KEYWORDS: application, Sign language, hearing impaired/disabilities, communication barriers, convenient tool, inclusivity, conversation, laptop, spoken or written

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

SL – Sign Language

CNN – Convolutional Neural Networks

SLT – Sign Language Translators

ML – Machine Learning

SLR – Sign Language Recognition

RNN – Recurrent Neural Networks

LSTM – Long Short-Term Memory

MT – Machine Translation

ST – Speech Translation

RAD – Rapid Application Development

WebRTC – Web Real-Time Communication

JSON – JavaScript Object Notation

JWT – JSON Web Token

CSRF – Cross-Site Request Forgery

XSS – Cross-Site Scripting

CSS3 – Cascading Style Level 3

HTML5 – Hypertext Markup Language Version 5

IDE - Integrated Development Environment

UI – User Interface

Chapter 1 : Introduction

1.1 Background Information

Sign language is mostly used by the hearing impaired/disabled and few people who understand the sign language that usually live associated with the hearing impaired such as families, activists, and teachers. Deafness can be caused by genetics, complications at birth, infectious diseases, chronic ear infections, use of drugs, excessive noise exposure, and aging. According to the World Federation of the Deaf, there are about 72 million deaf people worldwide, more than 80 percent of whom live in developing countries. Collectively, they use more than 300 different sign languages. Sign language is divided into 2; natural gestures and formal cues Natural gestures are the unconscious physical expressions that accompany signs and add variation. These might include facial expressions indicating intensity, eyebrow movement to indicate emphasis, or head tilts for questioning.

In contrast, formal cues are deliberate/intentional and standardized when being developed, they have the same language as the spoken language of the community, example is the American Sign Language (ASL), which is the most widely used sign language in the world with the method of fingerspelling as a representation of the alphabets on cues (hand positions show each letter of the Latin alphabet). The fingerspelling is used as a complementary if there is no SL for a single word, then it is used to mention names appropriately or when people are unsure of sign language for a particular word.

In this regard, there have been advances in similar tasks such as translation between different spoken languages in the automatic machine translation (MT) and the speech translation (ST) tasks. These allowed the translation between different spoken languages (using text and audio) so that people that do not share a common language can now communicate. Thanks to these advances in the MT and ST tasks, automatic translations between two given languages can be easily obtained. These approaches can be extended to Sign language translations too. In fact, there has been research into lightweight models that can be stored and used from smartphones, for sign language recognition (SLR) and translation. Sign languages are expressed through articulators, i.e. parts of the body used to convey information. These articulators can be classified between manual (hand configuration, place of articulation, hand movement, and hand orientation and non-manual, for example, face or body movement

The set of possible articulators for which SL recognition systems work are defined as the Gesture Parameter Set (GPS), including the movement, location, orientation, and shape of hands, and the head and facial expressions. There two methods of signing the finger spelling and word representation- allows a signer to convey the meaning of words using the previously mentioned articulators In SL translation there are various tasks to complete to translated SL to text, such as detection, identifying if SL is being used, identification, which is identifying which SL is being used, segmentation, distinguishing the temporal boundaries to segment phrases or individual signs, recognition, translation, and production.

Sign language translation into text involves the recognition of these hand sign gestures and mapping them to text or audio output that can be understood by those that can hear. The most common method of capturing these hand gestures is using a standard video camera that will give a 2-dimensional image. This method is more common than others because most people have a smartphone with a built-in camera. However, the image quality depends on the camera so it’s better to be preprocessed first. For example, a low-quality camera has too much noise on the captured image that might interfere with the feature extraction. Also, the resolution of the image depends on the camera so usually an image preprocessing is needed so it can be consistent to the classifier. Some researchers have used Kinect which has sensors and a camera to capture colored images including the depths of the objects. This provides more detailed data that can improve the classification. Data can also be obtained from datasets already created on the internet, which are ready to use and usually come in large amounts of good quality. For example, use of the Kaggle dataset which contains 27000 sign language images. With existing good quality datasets, the research and development of this application software can focus on other parts of SLR and if data collected is not enough then data augmentation can be done to acquire more data. Data augmentation is a technique used to artificially increase the size and diversity of a training dataset for machine learning models. Imagine existing data through various modifications.

SLT systems present today use ML and Deep learning methodologies integrated into working application software. For SLR, the most important part is the hand gesture which is acknowledged by the hand movements. Therefore, image segmentation is applied to remove unwanted data like background and other objects as the input, that might interfere with the classifier calculation. Image segmentation works by limiting the region of the data, so the classifier will only look at the Region of Interest (ROI). The most common feature extraction is by using the convolutional layers of CNN, which accounts for 11 out of 22 mentioned research. The convolutional layers can extract the important features of the image. Also, CNN is common because of its accuracy, which can reach up to 90% or more for SLR tasks. Unlike Artificial Neural Network (ANN), CNN adds extra convolutional layers for feature extraction. Most of the previous CNN needs a lot of testing to get the layers right. Some apply transfer learning, so it does not have to worry about the layer and focus on the training and predicting part. Transfer learning specifically in ML, is a technique where a pre-trained model on one task is leveraged as a starting point for a new related task, which is useful for reusing of knowledge, efficiency since not much resources and work is put into training model. This type of learning is useful for image recognition, natural language processing (NLP) and speech recognition. Some of the researchers noticed a few drawbacks with some of these techniques, some models with the highest accuracy only allow static image input. In ASL, for example some alphabets require hand movement, and therefore, need real-time, live translator. This is a drawback that this project specifically wants to solve, having real-time translations from live video footage as input sources.

1.2 Problem Statement

Deaf and hard-of-hearing people utilize sign language, a visual language that depends on articulators including body language, facial expressions, and hand gestures. Nonetheless, there remains a substantial communication gap between the hearing challenged and the public, which is mostly caused by the latter's lack of broad sign language proficiency. People who are deaf or hard of hearing have obstacles in their daily lives related to education, work, healthcare, and social relationships. While there are some solutions, like the use of human sign language interpreters and simple sign recognition software, there are drawbacks as well. For example, there are few and often expensive human sign language interpreters available, and SL recognition software frequently focuses on translating individual signs into words or brief phrases. They are unable to interpret intricate words or decipher the subtleties of body language and facial expressions in sign language grammar. To close the gap between the deaf community and the broader public, this initiative suggests developing a deep learning-based sign language translation tool. The application will be created utilizing well-known computer vision and deep learning methods.

1.3 Objectives

1.3.1 General Objectives

To come up with an application that translates sign language to text accurately and uses deep convolutional neural networks, computer vision techniques and other deep learning techniques, the following questions must be answered in the process.

1.3.2 Specific Objectives

i. Develop and test an application that achieves a word and sentence accuracy rate exceeding current market leaders by leveraging advanced deep learning techniques and a comprehensive sign language gesture dataset.

ii. Enable offline sign language recognition and translation, allowing users to communicate effectively even in areas with limited internet connectivity.
iii. Develop and test a user interface that adheres to best practices for accessibility, ensuring a smooth and intuitive experience for both deaf and hearing users, including integration with existing phone accessibility features.
iv. Implement robust security measures to protect user data privacy.
v. Develop a comprehensive testing plan that evaluates the application for accuracy, performance, and usability across various devices and scenarios.
vi. Develop an application that has cross-browser compatibility, which ensures the web application functions consistently across different browsers and devices.

1.3.3 Research Objectives

i. What are some of the known existing sign language translation applications in the market, and what features do they offer?
ii. How current sign language translator applications utilize neural networks for gesture recognition and what neural networks are viable to the development of this application?
iii. What is the average word or sentence accuracy rate of existing sign language translators and are there any emerging technologies that could improve the accuracy and efficiency of sign language translation applications?
iv. Do any of the current existing sign language translators offer offline functionality for gesture recognition and translation, and how can I design and develop such a feature?
v. Who is the primary target audience for this type of applications apart from those that are deaf and hard-of-hearing community?

1.4 Justification

The ability to communicate effectively is a fundamental human right and essential for social inclusion, education, and employment opportunities. However, a significant communication barrier exists for millions of deaf and hear-of-hearing individuals who rely on sign language. This barrier rises due to the limited availability of sign language interpreters and lack of widespread SL fluency in the general population. The project proposes the development of deep learning-based SL translator application to address the gap. By researching and developing an effective solution, I aim to research on shortcomings of existing solutions, enhance accessibility and inclusion, breakdown communication barriers, address data collection challenges and diversity, ethical considerations and user-centered designs, and design advanced technological innovation that helps bridge the gap between the deaf community and the general population.

1.5 Scope

Real-time translation, the system should be able to process SL gestures swiftly and provide text output almost instantaneously. This real-time functionality is crucial for facilitating seamless communication in various settings, such as educational environments, workplaces, and social interactions. Use of deep learning techniques such as Convolutional Neural Network (CNN) and sequence to sequence(seq2seq) models, the system should continuously learn and improve its translation accuracy. Developing a user-friendly interface for both signers and non-signers. The interface should be intuitive, visually appealing, and accessible. Training data and model optimization, to achieve high accuracy, gathering extensive training data comprising of diverse SL gestures is crucial. Additionally, continuous model optimization and refinement based on user feedback and real-world usage scenarios are essential.

1.6 Limitations

Sign languages have regional dialects and variations, hence building a comprehensive dataset that encompasses these variations requires extensive collaboration with the deaf and hard-of-hearing community.

Variability in gestures, wide range of gestures, expressions, and nuances that can vary regionally and contextually. SL involves not just hand gestures but also facial expressions and body language. Capturing these subtleties and their contribution to meaning will be crucial for accurate translation.

Chapter 2 : Literature Review

2.1 Introduction

This chapter discusses the existing systems used in Sign Language Recognition (SLR), the methods and approaches used in related works from previous and current researchers.

2.2 Existing techniques used for Sign Language Translation Software

Sign language translation software is a rapidly evolving field, but a major challenge lies in accurately capturing the sequential nature of sign language. Unlike spoken languages with linear word order, sign language incorporates hand movements, facial expressions, and body language in a continuous stream. Understanding the meaning requires analyzing the sequence of signs and their grammatical relationships.

Current systems leverage Recurrent Neural Networks (RNNs) and their more advanced cousin, Long Short-Term Memory (LSTM) networks, to address this challenge. RNNs are adept at handling sequential data, processing each sign in the sequence while considering the information from previous signs. However, standard RNNs struggle with long-term dependencies, meaning they might have difficulty remembering signs from earlier in a complex sentence.

LSTM networks address this limitation by incorporating internal memory cells that can store information about previous signs for extended periods. This allows the model to better understand the context and grammar of the entire sign sequence.

Several studies have explored the effectiveness of RNN and LSTM architectures for sign language translation. For instance, Jin et al. (2016) achieved promising results using LSTMs for Korean Sign Language recognition, demonstrating the potential of these networks in capturing sign language grammar. Similarly, Davydov and Lozynska (2017a) investigated the use of LSTMs for sentence-level sign language translation, highlighting the importance of sequence analysis for accurate translation.

2.2.1 How LSTMs work?

Long short-term memory networks are a special kind of recurrent neural networks. It avoids the long-term dependency problem, which involves the gap between the relevant information and the point where it is needed. Long Short-Term Memory (LSTM) networks rely on a cell state to store information over time. This cell state acts like a conveyor belt, carrying information throughout the network. However, LSTMs can control this flow of information using special structures called gates. These gates function like valves, deciding how much information gets added to, removed from, or simply passes through the cell state. Each LSTM has three gates that work together to carefully manage this information flow.

LSTMs manage information flow through a cell state using specific gates. (*Understanding LSTM Networks -- Colah's Blog*, n.d.)The first gate, the "forget gate", decides what information to discard from the cell state. It analyzes the previous cell state and the current input, assigning a value between 0 and 1 to each piece of information. A value closer to 1 signifies keeping the information, while 0 indicates discarding it.

The next step involves determining what new information to store in the cell state. An "input gate" identifies which values to update, while another layer creates potential new values. These steps are then combined to create an update for the cell state. Finally, an output gate determines what information from the cell state is used. It analyzes the cell state and creates a filtered version based on its importance to the current task. This filtered information is then used by the LSTM for prediction or further processing.

2.2.1.1 Variants on LSTMs

i. Peephole LSTM: This variant allows the gate layers to directly access information from the cell state.
ii. Coupled Forget and Input Gates: This variation combines the decision-making process for forgetting and adding new information. The network forgets only when it intends to add something new, and vice versa.
iii. Gated Recurrent Unit (GRU): This is a more prominent variant that simplifies the LSTM structure by merging the forget and input gates into a single "update gate". It also combines the cell state and hidden state.
iv. Some less-discussed variants are like the Depth Gated RNNs and Clockwork RNNs.

2.2.2 Challenges Facing the Current State of Sign Language Recognition and Translation

i. Sequential Nature: Unlike spoken language with linear word order, sign language incorporates hand movements, facial expressions, and body language in a continuous flow. Analyzing the sequence of signs and their grammatical relationships is crucial, which current systems are still mastering.
ii. RNN Limitations: Standard Recurrent Neural Networks (RNNs), a common technique for sequence analysis, struggle with long-term dependencies. They might forget earlier signs in a complex sentence, impacting the understanding of context and grammar.
iii. Accuracy and Robustness: While LSTMs, a more advanced type of RNN, offer improvements, current systems still require significant refinement to achieve high accuracy across diverse sign languages and complex grammatical structures.
iv. Regional Variations: Sign languages have dialects with variations in signs and grammar. Training models to handle these variations effectively is an ongoing challenge.
v. These limitations hinder the ability of current sign language translation software to fully capture the richness and complexity of sign language communication. Continued research and development are needed to create more robust and accurate translation systems for improved accessibility.

2.3 Related Works

2.3.1 Wearable-tech glove – translates sign language into speech in real time

UCLA researchers have developed a glove-like device that translates American Sign Language (ASL) into spoken English in real-time using a smartphone app. This aims to bridge the communication gap between deaf individuals and non-signers. The gloves contain thin sensors that pick-up hand and finger movements representing letters, numbers, words, and phrases. Facial expressions are also captured by additional sensors. The device is lightweight, comfortable, and inexpensive, addressing limitations of previous bulky ASL translation wearables. In testing, the system achieved a one-word-per-second translation rate with high accuracy for basic signs. While the research paves the way for future commercial applications, further development is needed to expand vocabulary and increase translation speed.

2.3.2 SlAIT – Real time Sin Language Translator with AI

SLAIT is an application that transcribes sign language gestures into text in real-time. SLAIT stands for Sign Language AI Translator. It offers the following features: Transcribes American Sign Language (ASL) to text. Transcribes spoken language to text. Provides video communication. Use cases include daily communication, career development, healthcare, and education1.Available for Web, Android, and iOS platforms. How it works is that you sign in front of a camera, and the program records the video then sign recognition whereby, AI analyzes the video, identifying hand shapes, movements, and facial expressions then finally text generation, so based on the recognized signs, the program generates written text representing the translated sentence.

2.3.3 Hand Talk Sign Language Translator App

The Hand Talk app, developed by a company aiming to improve accessibility for the deaf community, translates spoken languages (English and Portuguese) into sign languages (American Sign Language and Brazilian Sign Language) using an AI-powered avatar. This free app allows users to learn signs, save translations, and customize their experience. With over 3 million users (about the population of Arkansas) already, the addition of English to American Sign Language translation promises to reach a wider audience and break down communication barriers between the deaf and hearing communities globally.

2.3.4 SignAll – Sign Language Translation

The application's main aim is to bridge the communication barrier through translation and teaching to create an inclusive society for all. The application is intuitive which makes it simple and easy to use. Lightning-fast, meaning it provides real-time instant translation. Accessible, it only requires a mobile phone and internet connection to use it. Accurate, SignAll achieves over 99% accuracy on ASL alphabet. SignAll provides a diverse range of features that make SL communication a reality for those part of the hard-of-hearing and non-verbal communities. Some of these features includes multi-media support – translates a multitude of inputs including images, pre-recorded video, and real-time video cloud infrastructure – access fast, accurate translation anytime, anywhere through the power of cloud computing, and a learning toolkit – interact with personalized SL lessons with real-time feedback on your SL skills.

2.4 Related Methods and Approaches

Most methods and approaches used in designing and developing a sign language translator are majorly AI based. Some of these techniques involve deep learning, computer vision, convolutional neural networks (CNN), natural processing language (NLP). For hand sign language recognition there are several sub-areas that go into it, which are, hand detection, hand pose estimation, real-time hand tracking, hand gesture recognitions, and hand pose recovery. The most common means of recording sign gestures is through visual sensors that can capture fine-grained information such as facial expressions, and body postures. There are designs that have employed a Kinect sensor to simultaneously capture red-green-blue (RGB) image, depth and skeletal information towards recording of a multimodal dataset with Brazilian sign language.

Another sensor used is the Leap Motion, which can capture 3D positions of hand and fingers at the expense of having to operate close to the subject. They used this sensor to record sign language gestures. On the other hand, wearable sensors have been adopted to capture sign language gestures. Galea et al. used electromyography (EMG) to capture electrical activity produced during arm movement. In the generation of datasets, increasing datasets means involving more signers, as well as containing high resolution videos captured under various and challenging illumination and background conditions. Also recording sign language videos using many signers is very important, since each person performs signs with different speed, body posture and facial expressions, making the data collected diverse and inclusive. Datasets with videos captured under different conditions enable deep networks to extract highly discriminative features for SL classification. Another model that has been mentioned is the sequence-to-sequence model which allows for the mapping of input data which in this case is video input and an output in this case is the text, direct meaning of the SL gestures.

2.5 Conceptual Framework

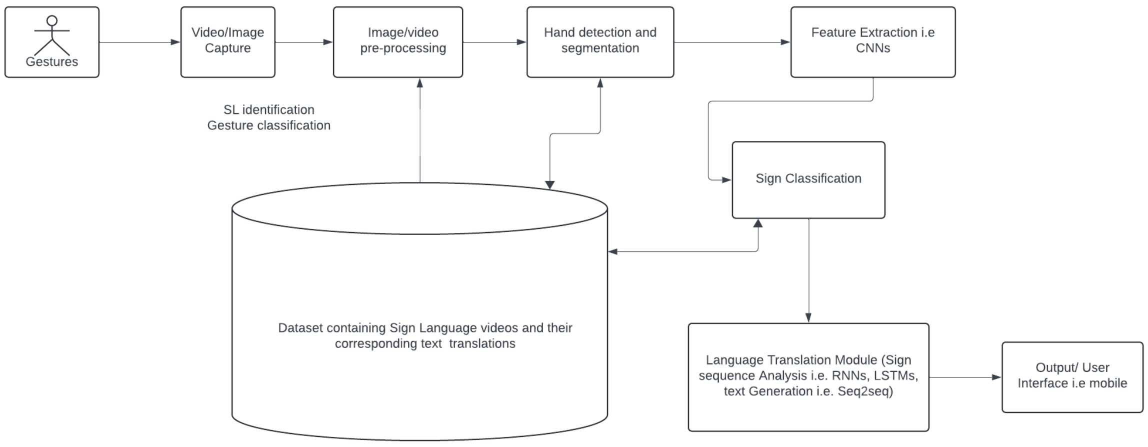

Gestures
Video/Image Capture
Image/video pre-processing
Hand detection and segmentation
Feature Extraction i.e CNNs
SL identification
Gesture classification
Sign Classification
Dataset containing Sign Language videos and their corresponding text translations
Language Translation Module (Sign sequence Analysis i.e. RNNs, LSTMs, text Generation i.e. Seq2seq)
Output/ User Interface i.e mobile

Figure 1 Conceptual Framework

A user who wants to translate sign language into text to be able to understand an individual signing, they point the camera towards the individual signing, the camera captures the sign language gestures, input can be either video or image. The video or image is processed and mapped to their respective output text referring to the dataset containing the sign language videos and images and respective text values. The gestures captured are then detected and segmented into their respective classes then proceeds to feature extraction whereby the input image is reduced to a set of feature maps through convolution and pooling layers. The features are passed into fully connected or hidden layers and output layer. Then the entering the sign language translation module, where techniques like RNN are used in analyzing the sequence features, leveraging it memory capabilities to understand the relationship between frames. This helps recognize the individual signs being presented. Then the mapping of the recognized signs to a corresponding word or sentence in the target language. Finally, the text is output to the user interface, and the user can read the word or sentence assisting the user in understanding what the deaf or hard of hearing individual has signed.

Chapter 3 : Methodology

3.1 Introduction

This chapter details the methodology used in developing a real-time sign language gesture recognition and translation to text web application using convolutional neural networks (CNN) and long short-term memory networks (LSTM). The chapter includes a structured approach to the project outlining the steps taken from initial requirements gathering to the final deployment. The proposed methodology will ensure the effective delivery of a robust and user-friendly application that meets the project’s objectives.

3.2 Software Development Methodology

Prototyping integrated with agile development emphasize on an iterative development and regular user feedback, making them well suited for real-time applications like sign language gesture recognition, which requires frequent adjustments based on user interaction. One particular use of prototyping is rapid application development (RAD). It is an object-oriented approach with three phases: requirements planning, the RAD design workshop, and implementation. Prototyping allows one to build functional versions of the web app, while Agile’s sprints enable regular updates and feature improvements. The use of Agile ensures that development is divided into manageable cycles, with each iteration involving design, development, testing, and evaluation of specific features. The synergy between Prototyping and Agile allows for continuous feedback from users, allowing an individual to refine the application, improve model accuracy, and enhance the user experience throughout the development lifecycle.

3.2.1 Justification of the Methodology

Prototyping facilitates the expeditious creation of working prototypes, which in turn permits preliminary testing and validation. Risks can be reduced, and expensive rework can be avoided by seeing possible problems early in the development process. By encouraging active user participation, prototyping makes sure that users’ requirements and expectations are met by the finished product. The real-time nature of this SL gesture recognition system requires ongoing feedback from users, particularly the deaf and hard-of-hearing communities. Given the complexity of CNNs and LSTMs for real-time recognition, prototyping allows for flexibility in adjusting algorithms, optimizing models, and refining user interactions.

3.2.2 Methodology Diagram (Prototyping)

Prototyping is the process of simulating how end users will interact with and use a real system. It serves as a tool for user testing, design input, and concept exploration for developers. Incremental prototyping was used in this instance, in which the system was divided into smaller components and developed separately. Ultimately, the component pieces were created and merged into a single system according to a predetermined sequence. Early end-user feedback is made possible by this model, which can assist to direct the development process and guarantee that the finished product fulfills the needs of the consumers. The prototype model is as seen **Figure *2***

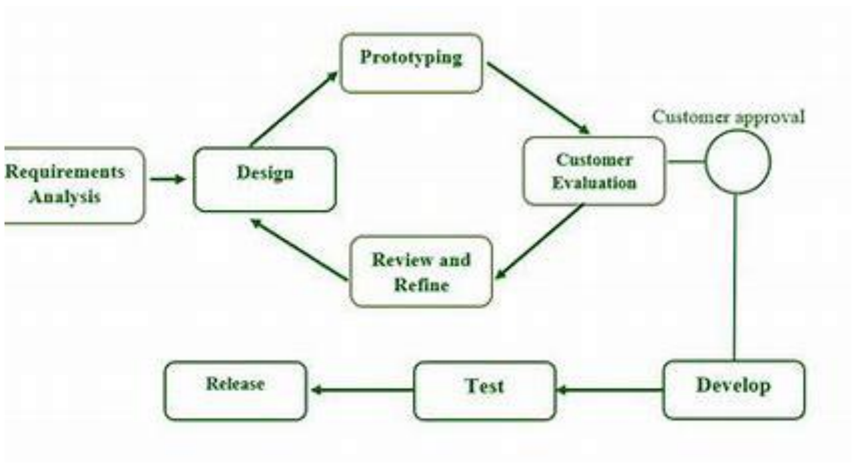


Figure 2 Prototype model

3.2.2.1 Requirements Gathering and Analysis

Here, the developer can specify the goals and scope of the project, which is a crucial stage in software development. This procedure entails determining, comprehending, and recording the end users' requirements and expectations for the system. Effective requirements management and collection is crucial because it establishes specific goals for the software solution.

Gathering requirements includes consulting potential users, sign language experts, and web accessibility specialists to gather user requirements. The developer will also users' stories to define the goals and expectations of the web application. From the system's objectives, the developer identifies essential functions of the web application such as real-time gesture recognition, gesture to text conversion, user authentication, and accessibility features.

Outlining non-functional requirements is very important, such as the performance goals, including low latency, high recognition accuracy, and cross-browser compatibility which is the ability of the web application to function consistently across different browsers and devices.

3.2.2.2 System Design

The system design can be split into two, architectural design and user interface design. The architecture design is centered on building a robust, scalable, and efficient system. The core of the system revolves around the CNN-LSTM hybrid model. The CNN component is responsible for extracting spatial features from each frame of the videos, capturing essential visual details such as hand shapes, positions, and movements. The LSTM network complements CNN by capturing temporal dependencies, which are crucial for recognizing the sequence of gestures over time. Together, CNN extracts the visual features, and the LSTM processes the temporal flow of these gestures, ensuring accurate recognition of SL phrases.

For data management, a database will be structured to store crucial information such as user profiles, mappings of SL gestures, and logs of recognition results. The database enables efficient data retrieval and provides the foundation for future analytics, such as tracking user progress or analyzing common errors in gesture recognition.

The user interface will be designed with a focus on simplicity and accessibility. UI wireframes will be created using design tools such as Figma to plan the layout and structure of the application. Special attention will be given to ensuring that the design caters to users with varying abilities, making the application easy to navigate for people who rely on sign language. The interface will be optimized to be intuitive, allowing users to interact with the app seamlessly, whether they are signing a gesture, viewing the recognition results, or navigating through other features.

3.2.2.3 Implementation and Testing

The implementation of the CNN-LSTM model for sign language gesture recognition is carried out using TensorFlow and integrated into a Django-based backend for model inference. The backend handles the model’s predictions by processing video input and returning the recognized gestures in real time. For the front-end, React is used to deliver real-time feedback to users, allowing them to see the results of their gestures as they interact with the app. Video input is managed through WebRTC, enabling smooth, low-latency transmission of the video feed from the user to the backend for gesture recognition.

The front-end is built as a responsive web interface using HTML5, CSS3, and JavaScript (React). This ensures that the interface adapts seamlessly to various devices, providing an accessible user experience. The WebRTC framework handles the real-time video feed, capturing the gestures from the user’s webcam and sending the data to the backend where the model performs gesture recognition.

Unit testing will be incorporated whereby each individual module of the system, including the gesture recognition engine, text translation logic, and user interaction components, is tested separately to ensure that they work as expected. System testing whereby the real-time gesture recognition feature is tested across different devices and web browsers to confirm compatibility and minimal latency, ensuring a smooth user experience regardless of the platform. User Acceptance Testing (UAT), whereby the end users from the target community, particularly those who rely on sign language, are invited to test the prototype. Their feedback is gathered to ensure that the app meets their needs in terms of functionality, accuracy, and ease of use, particularly concerning accessibility features. This development process ensures that the app is not only functional but also user-friendly and accessible across various environments.

3.2.2.4 User Evaluation

After each iteration of the prototype, user evaluation is conducted to collect valuable feedback in several key areas. First, users assess the system's gesture recognition accuracy, testing its ability to correctly identify a wide range of sign language gestures. This helps gauge how well the model performs in real-world conditions. Next, feedback is gathered on the app's usability and accessibility, with particular attention to how easy the system is to use, especially for users with disabilities. This ensures that the interface is intuitive and accessible to all users.

Additionally, users evaluate the response time of the system to determine whether the app delivers real-time performance that meets their expectations. Fast and responsive feedback is crucial for the system to be effective in real-time applications. Finally, users are encouraged to provide suggestions for improvement, offering ideas for new features or enhancements based on their experiences with the prototype. These suggestions are then considered for future iterations to improve the overall user experience.

3.3 Deliverables

3.3.1 Final Documentation

The final documentation for the project will include several key components to provide a comprehensive understanding of the system. First, the technical documentation will outline the codebase, system design, architecture, and model configuration. This section will provide developers and technical users with a detailed explanation of how the app is structured and how its components interact. The user documentation will serve as a guide for end-users, explaining how to navigate and use the application. It will also include instructions for troubleshooting common issues and provide information on how to contact support if needed. The model training documentation will describe the process used to train the CNN-LSTM model. It will cover the dataset used, the steps involved in training the model, and the performance metrics that evaluate the model's effectiveness. Finally, the testing documentation will present the results from all testing phases, including unit tests, integration tests, and user acceptance testing. This section will demonstrate the thoroughness of the testing process and validate the system's reliability and performance across different stages of development.

3.3.2 Authentication Module

The authentication module is designed to ensure secure access to the web app by implementing several key security features. It utilizes the OAuth 2.0 protocol, allowing users to securely sign in using third-party services such as Google or Facebook. This reduces the need for users to create new credentials while maintaining high levels of security.

Additionally, the module incorporates session management, which enables users to stay authenticated across multiple sessions without needing to log in repeatedly. This ensures a seamless user experience while maintaining secure access.

To further enhance security, JWT are employed to transmit user data securely between the client and server. JWTs ensure that the data is encoded and verified, preventing unauthorized access or tampering during communication.

3.3.3 User Module

The user module is responsible for managing interactions between the application and its users, providing several key features to enhance the user experience. One of the primary functions is profile management, which allows users to create, update, and manage their profiles within the app. This includes storing essential user information and preferences. Users can also customize their experience through the preferences section. Here, they can select options such as preferred language, video quality settings, and other features that personalize their interactions with the app. In addition, the user module maintains activity logs that record user actions, such as recognized gestures and their corresponding translations. These logs are stored for future analysis, helping both the users and developers understand patterns of usage and system performance.

3.4 Tools and Techniques

A variety of tools and techniques will be utilized during the development of the real-time sign language recognition app to ensure a smooth and efficient workflow. The programming language that will be used in the application's backend is Python, while JavaScript (React) will be used for the frontend, along with HTML5 and CSS3 for creating a responsive user interface. The core of the application's functionality lies in several key libraries and frameworks. This system will use libraries such as TensorFlow/Keras which will be employed to build the CNN-LSTM model for sign language recognition. These libraries handle the computational backend, efficiently managing model training, processing large datasets, and enabling GPU acceleration. Django will be used as the main open-source framework for developing the backend, handling API requests, and serving the machine learning model. An advantage to using Django as the framework for this system is that its user authentication system includes XSS, CSRF, and SQL injection protection, as well as HTTPS. The protection protects the web pages from attackers who try to inject malicious scripts into web pages viewed by other users, which may lead to theft of data, hijacking sessions, or performing unauthorized actions on behalf of the user.

The dataset contains 87000 images for training and testing, 29 labels; A - Z, space, delete, and nothing Label encoding; A = 0, B = 1, nothing = 28 for training. 25,000 annotated videos for real-life sign language. Videos must be downloaded separately, and they are mostly hosted on platforms like YouTube or cloud repositories. To work with images, download the videos from the platforms, extract frames from these videos to create image datasets for training the CNN-LSTM model. The files include information about the video, such as the gesture label, the path to the video, and metadata like the start and end time of the gesture within the video.

A pre-trained model, performance-wise, transfer learning models beat traditional deep learning models because the TL models include data (features, weights, etc.) from previously trained models, possessing a comprehensive grasp of the features. 87000 images in the MS-ASL dataset will be divided into training and testing data. 8000 images for training data and 7000 images for testing. Merging datasets, normalizing and standardizing labeling across the datasets being used. Renaming the classes for uniformity and unify labels for overlapping gestures. Creating labels for unique, non-overlapping classes. Resizing images to a consistent size (e.g., 64x64 or 224x224 pixels). Apply data augmentation including rotation, flipping, zooming. Normalize pixel values i.e., scale between 0 and 1. Concatenate image data and labels using libraries like pandas, numpy, or TensorFlow. Shuffle dataset to prevent model bias towards one dataset.

Both datasets should use the same number of frames per sequence. Pad Sequences, if the number of frames per video differs across datasets, there will be need to pad or truncate sequences to a fixed length before feeding them into the LSTM. The LSTM network complements CNN by capturing temporal dependencies, which are crucial for recognizing the sequence of gestures over time. Together, CNN extracts the visual features, and the LSTM processes the temporal flow of these gestures, ensuring accurate recognition of SL phrases. Unit testing will be incorporated whereby each individual module of the system, including the gesture recognition engine, text translation logic, and user interaction components, is tested separately to ensure that they work as expected. Testing across different devices and web browsers to confirm compatibility and minimal latency, ensuring a smooth user experience regardless of the platform. Combine and Split: After merging, split the combined dataset into training, validation, and testing sets. You can use tools like train_test_split from scikit-learn to maintain class balance across splits. Split into 70% training, 15% validation, and 15% testing. Cross-Validation: Using k-fold cross-validation to evaluate the model on different subsets of the data for better generalization. The sample size will be from the direct end users who are the deaf and hard-of-hearing community. A diverse representation in terms of age, proficiency in sign language, etc. Participants interact with the app in real-time, and gestures are translated to text. Metrics used to test performance will include recognition accuracy, response time, user feedback. Some controlled variables include same lighting conditions, frame rate, and input quality during testing

Web Real-Time Communication (WebRTC) was integrated to manage real-time video capturing and streaming, allowing users to interact with the app through their webcam. It allows audio, video, and data sharing directly between devices without needing an intermediary server, providing low-latency, peer-to-peer connections. It is commonly used for applications like video conferencing, voice calls, and real-time data sharing.

The tools that will support development tasks include, Jupyter Notebooks which will be used for model training, experimentation, and data visualization during the initial stages of building the CNN-LSTM model. The Visual Studio Code will serve as the main IDE for both front-end and back-end development. GitHub is a cloud-based platform for version control and collaboration. It is built around Git, and an open-source version control system. It will be used to track changes in the codebase and facilitate collaboration. The system uses MySQL to store user data, activity logs, and model-related logs, ensuring efficient data management and retrieval.

For deployment, the web application is set to be hosted on scalable platforms like Amazon Web Services (AWS) or Heroku, providing accessibility and ensuring the app can handle user traffic efficiently.

Chapter 4 : System Analysis and Design

4.1 Introduction

The content of this chapter is particular to viewing the application through its technical architecture, user cooperation and interaction with design methods that help to deploy the application competently. In addition, the set of system analysis diagrams, use case diagrams and system design are provided to increase understanding of the application's processes and relationships. With respect to the methodology of the project, Object-Oriented Analysis and Design (OoAD) approach is used, which ensures that the system is thought out and built as a single system of interacting elements.

4.2 System Requirements

Some of the system requirements reviewed in the project include:

4.2.1 Functional Requirements

These are product features or functions they will implement to enable users to accomplish their tasks. Here are the functional requirements for this application:

i. Authentication, which includes the user's registration and profile management that ensure secure access to the application while providing a personalized user experience. New users can create accounts by entering basic credentials such as first name, last name, email and password. User login that enables existing users to log in securely using their credentials.

ii. Real-Time gesture recognition and translation to text, which is the core functionality of this application, enabling users to translate sign language gestures into text in real-time. Utilizing a pre-trained machine learning model to identify hand gestures captured via the device's camera. Converting recognized gestures into corresponding text displayed on the screen.

iii. Learning basic signs from the signs posted on the application, these resources will help users learn and practice basic sign language gestures. Presenting a categorized list of commonly used signs, such as alphabets, numbers, and everyday phrases.

iv. Tutorials and manuals, to serve as a guide for users to understand how to interact with the application effectively. Providing step-by-step instructions on using various features of the application. Explaining how to position the camera and hands for optimal gesture recognition.

4.2.2 Non-Functional Requirements

They are not related to the system functionality, rather define how the system should perform. Here are the non-functional requirements for this application:

i. Performance

   The application will deliver real-time recognition and text translation with minimal latency to ensure a seamless experience. With the increase in the number of users and higher computational demand, the system will process the gestures without degradation in performance. Minimal load time allows users to quickly access features like gesture translation, tutorials, and learning web pages.

ii. Usability and User Experience

   A clean, intuitive design, making it easy for users to navigate. Providing options to display instructions on how the system works in simple terms.

iii. Security and Privacy

   Encryption of user data during storage and transmission to ensure confidentiality and integrity. Secure login mechanisms like two-factor authentication to protect accounts from unauthorized access.

iv. Reliability and Availability

   Stability by minimizing crashes, errors, and interruptions during usage. Maintain an uptime of 99.9%, ensuring that the application always remains accessible to users, including during updates.

v. Integration and Compatibility

The application functions across multiple platforms, including Windows, macOS, Android, and iOS, while adapting to various screen sizes.

vi. Performance Monitoring and Analytics

Key metrics to continuously monitor the gesture recognition speed, translation accuracy, and system resource usage to optimize performance. Collects anonymized data on user behavior, such as frequently used gestures and interaction patterns, to refine translation models and enhance the user experience.

4.3 System Analysis Diagrams

Here are the system analysis diagrams:

4.3.1 Use Case Diagram

The diagram explains the features of the system by providing information on the basic interactions of the user and the services that the system provides. The use-case diagram demonstrates how the users are able to utilize Savvy Signing within the application by indicating its main features, which allow the users to facilitate the communication better and improve their signing skill.

The use case diagram for the application Savvy Signing consists of some important elements which illustrate interaction with the users by the system. The user is the primary role who interacts with the application in order to use its functionalities. Important use cases are Register/Login, allowing users to sign up or authenticates themselves, Enable Camera, which activates the camera of the device for video call on gesture recognition, and Perform Gesture Translation, in which users can do sign language gestures that will be converted to text vice versa in real time. Other features are Access Tutorials, to assist users in using the application correctly; These elements together represent the limits and features of the application so that they can be easily interacted and accessed.

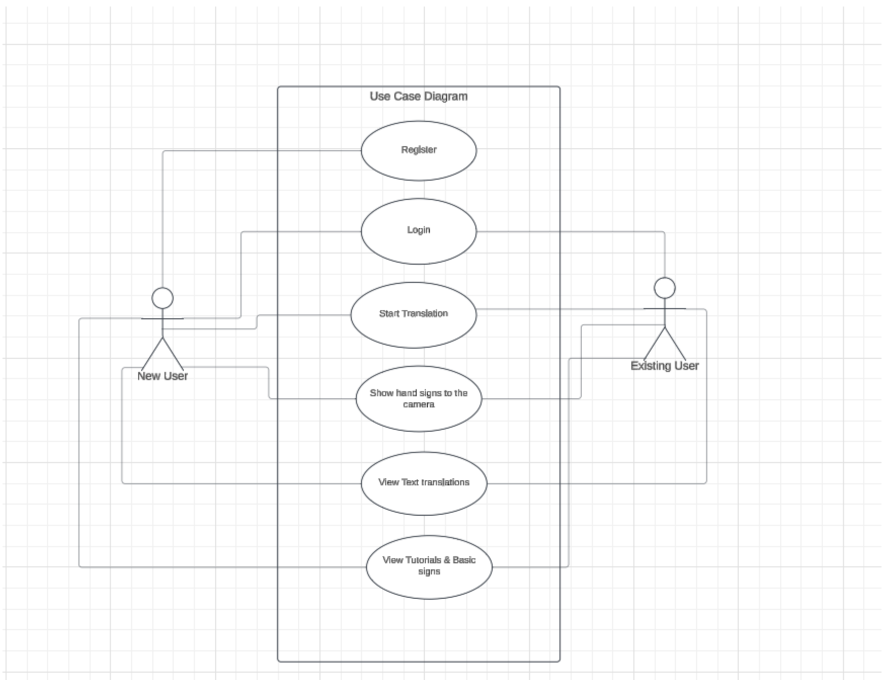


Figure 3 Use Case Diagram

4.4 System Design Diagrams

4.4.1 Database Schema

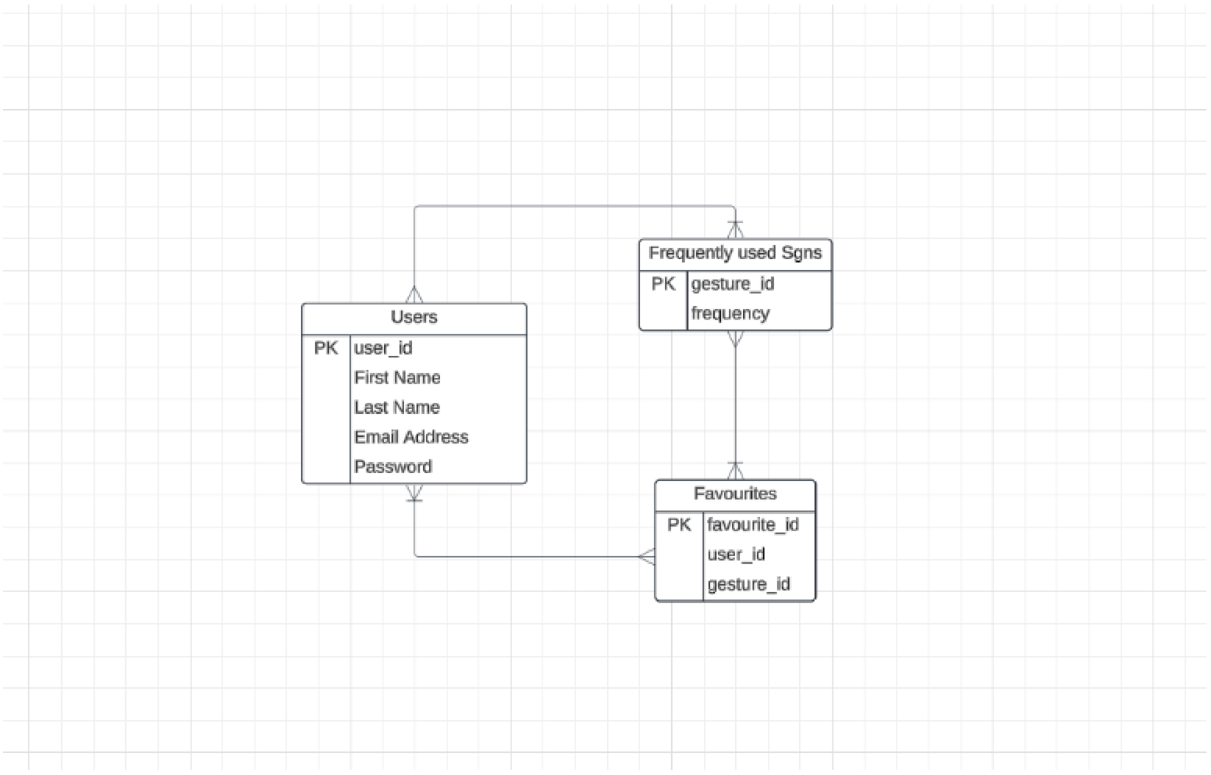
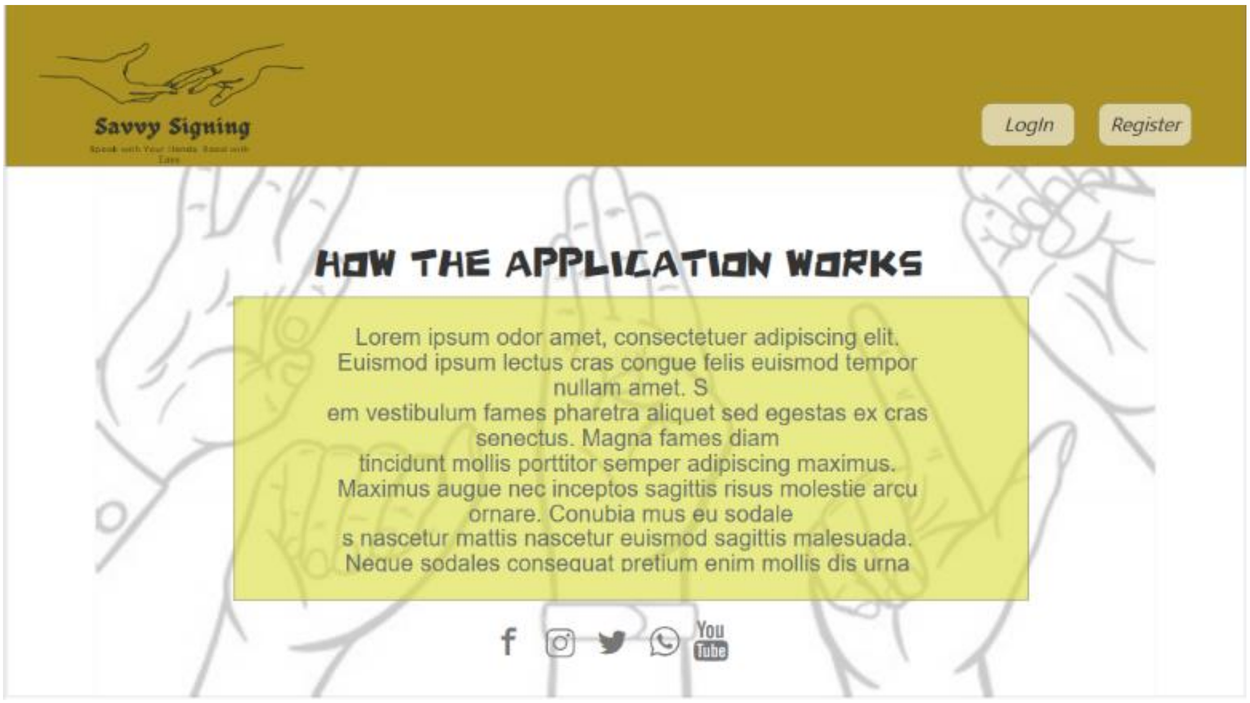


Figure 4 Database Schema

4.4.2 UI Mockup – Wireframes



Lorem ipsum odor amet, consectetuer adipiscing elit. Euismod ipsum lectus cras congue felis euismod tempor nullam amet. S
em vestibulum fames pharetra aliquet sed egestas ex cras senectus. Magna fames diam
tincidunt mollis porttitor semper adipiscing maximus. Maximus augue nec inceptos sagittis risus molestie arcu ornare. Conubia mus eu sodale
s nascetur mattis nascetur euismod sagittis malesuada. Neque sodales consequat pretium enim mollis dis urna

Figure 5 Landing Page

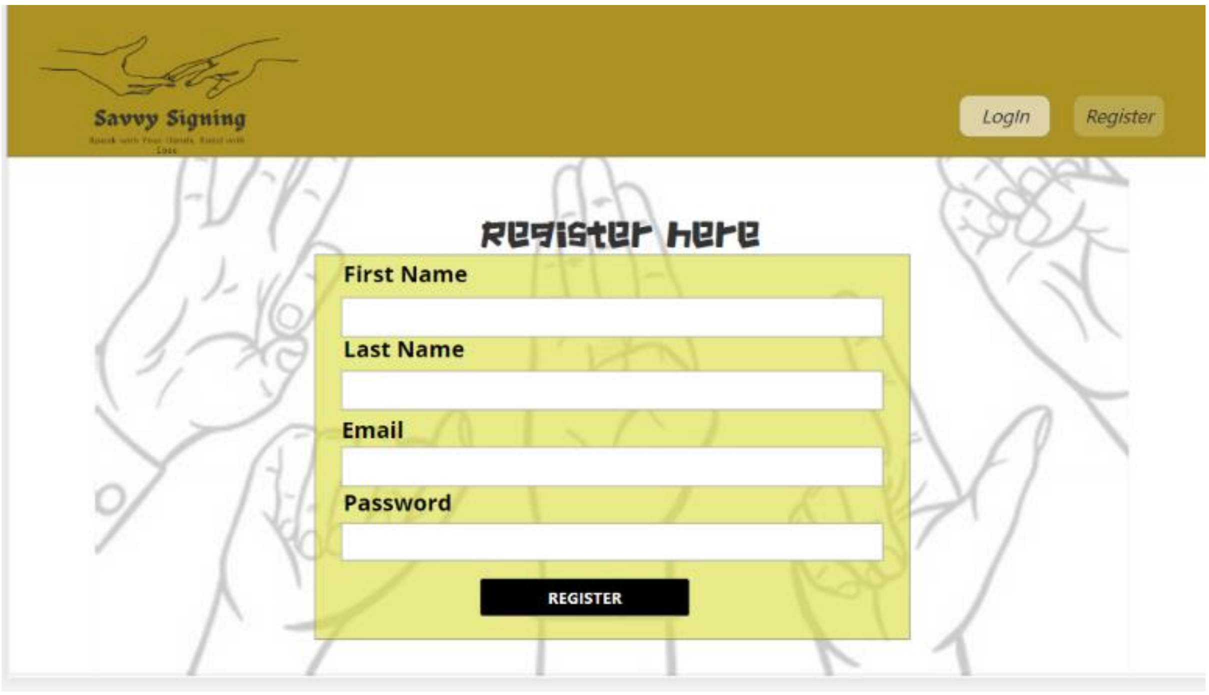


Figure 6 Registration

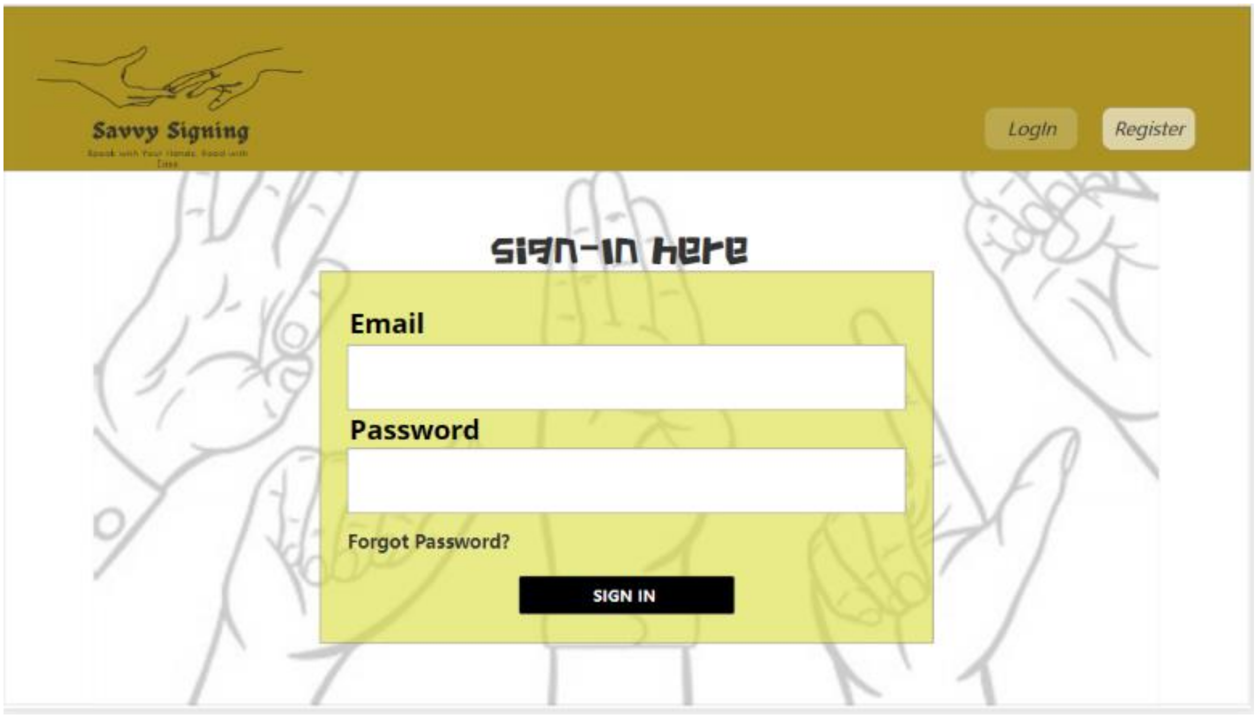
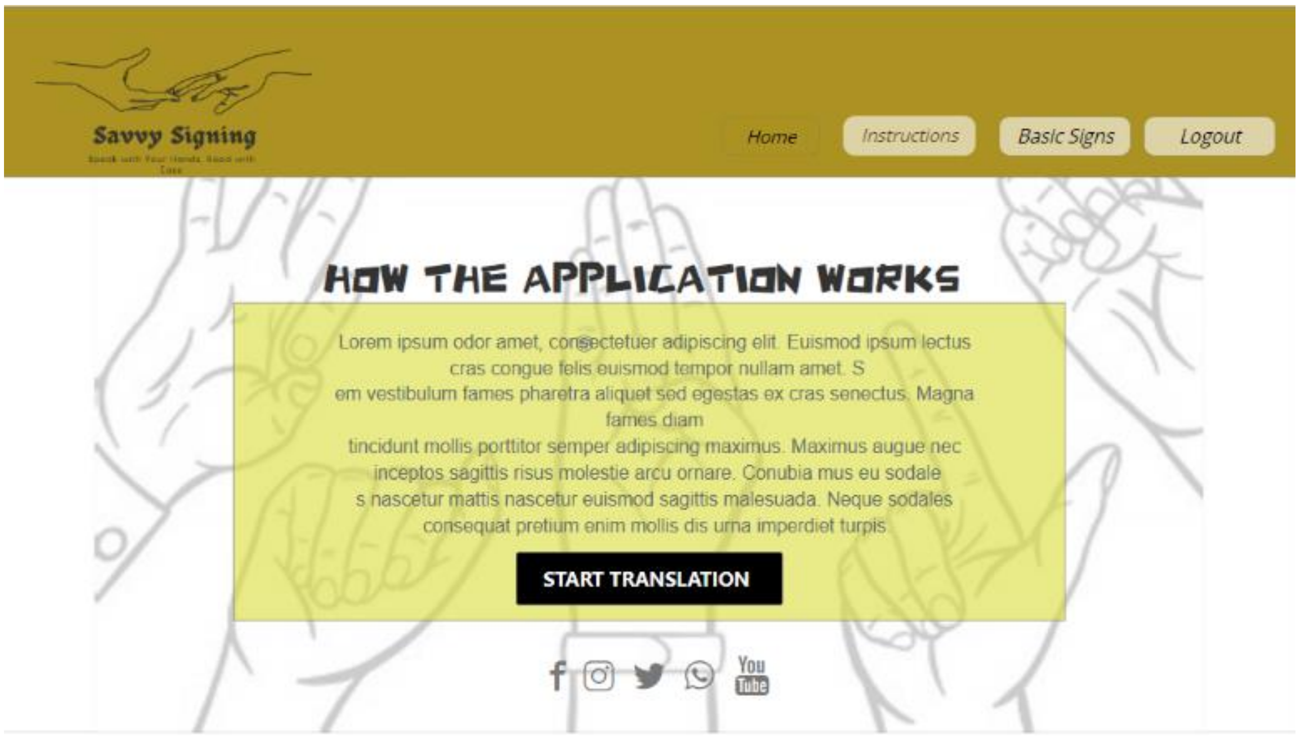
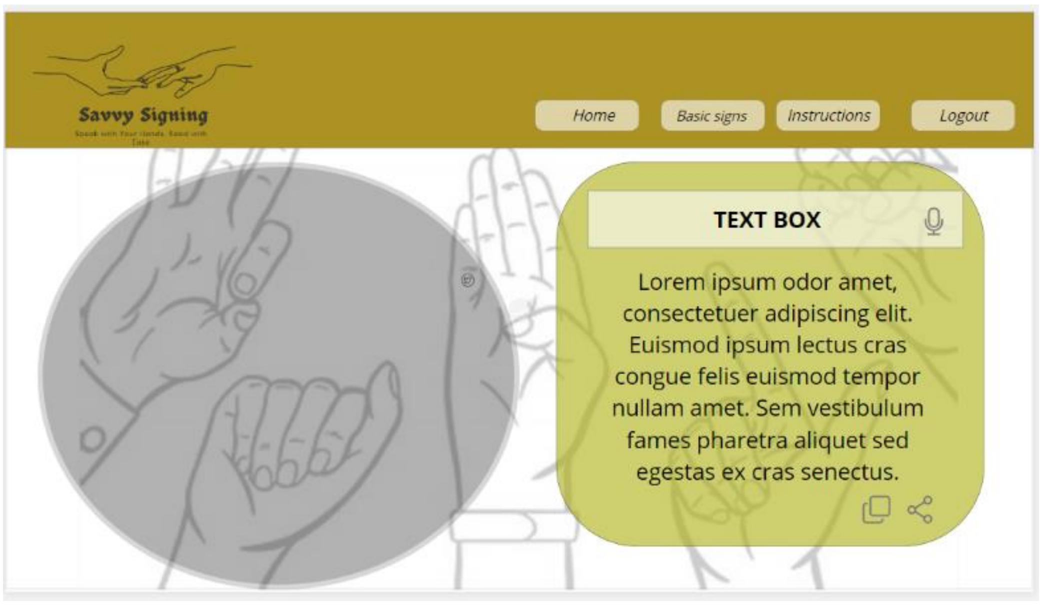


Figure 7 Login



Lorem ipsum odor amet, consectetuer adipiscing elit. Euismod ipsum lectus cras congue felis euismod tempor nullam amet. S
em vestibulum fames pharetra aliquet sed egestas ex cras senectus. Magna fames diam
tincidunt mollis porttitor semper adipiscing maximus. Maximus augue nec inceptos sagittis risus molestie arcu ornare. Conubia mus eu sodale
s nascetur mattis nascetur euismod sagittis malesuada. Neque sodales consequat pretium enim mollis dis urna imperdiet turpis.

Figure 8 Home Page



Lorem ipsum odor amet, consectetuer adipiscing elit. Euismod ipsum lectus cras congue felis euismod tempor nullam amet. Sem vestibulum fames pharetra aliquet sed egestas ex cras senectus.

Figure 9 Translation Page

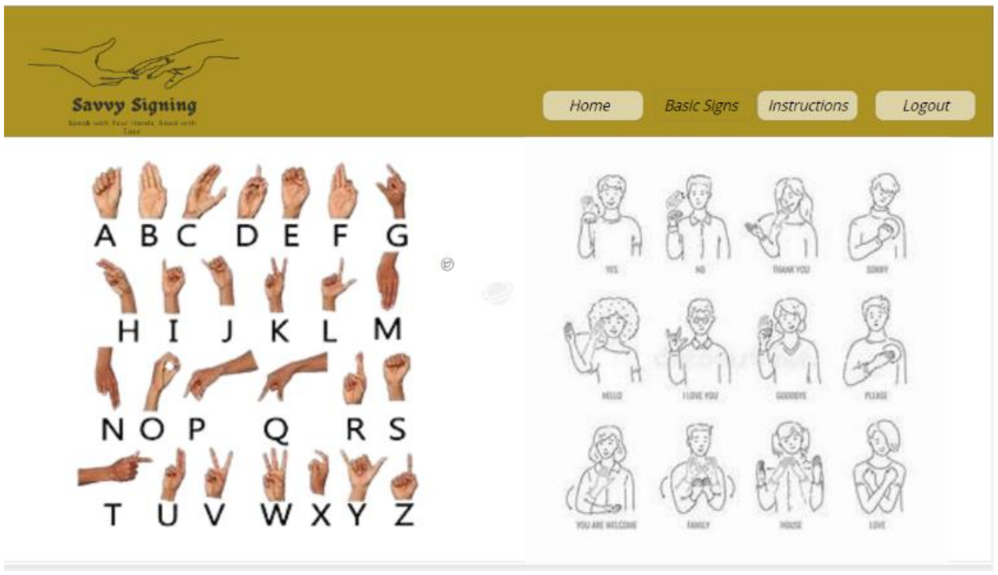
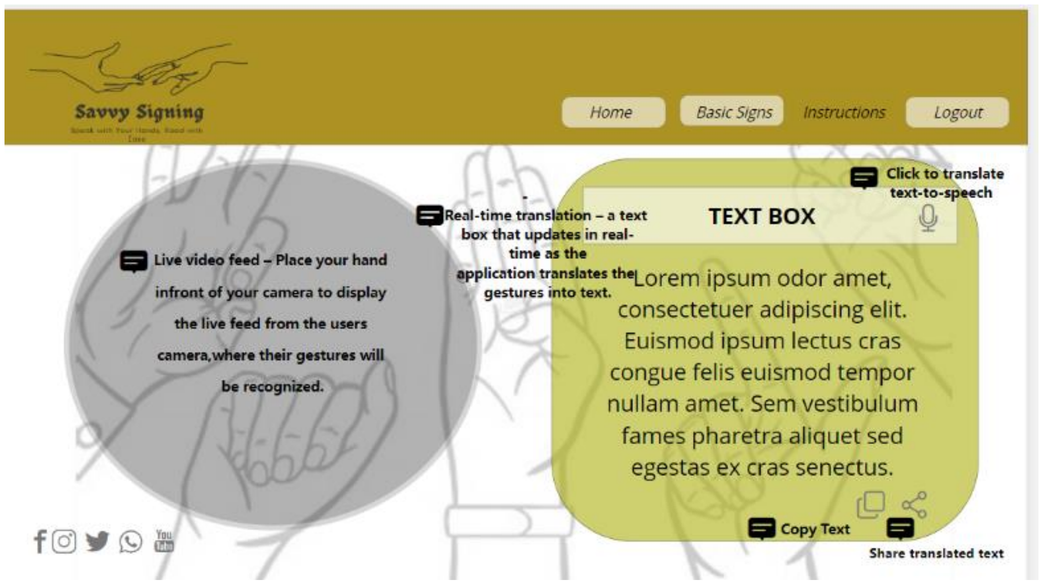


Figure 10 Basic Signs Page



Real-time translation – a text box that updates in real-time as the application translates the gestures into text.

Live video feed – Place your hand infront of your camera to display the live feed from the users camera,where their gestures will be recognized.

Lorem ipsum odor amet, consectetuer adipiscing elit. Euismod ipsum lectus cras congue felis euismod tempor nullam amet. Sem vestibulum fames pharetra aliquet sed egestas ex cras senectus.

Figure 11 Instructions Page

Chapter 5 : System Implementation and Testing

5.1 Introduction

This section examines the post-implementation testing of the system. System testing is the process of verifying that the various system modules operate as intended and that they meet the established functional and non-functional requirements. System implementation determines how a system should be constructed, making sure it is functional and utilized, and finally making sure it satisfies quality requirements.

5.2 Description of the implementation Environment

The implementation environment for this real-time gesture recognition and translation system is designed to ensure accessibility and seamless integration into everyday use. The product is available as a web application and can be accessed through searching the name of the application on your respective browsers. To implement the system, users need either a mobile device or any device that is equipped with a functional camera, sufficient processing power for real-time gesture recognition, and stable internet connectivity. The system can also run on personal computers with an external or built-in camera, provided they meet the software's hardware requirements. The platform's compatibility with various devices and operating systems enables widespread adoption and ensures users can access the service regardless of their preferred device.

5.2.1 Hardware Specifications

The hardware specifications below show the specifications of the computer used to develop and test the system

Table 1 Hardware Specifications

Item	**Specifications**
Processor	Processor 11th Gen Intel(R) Core (TM) i5-1135G7 @2.40GHz
RAM	RAM 8GB
Hard Disk Storage	Hard Disk Storage 1TB

5.2.2 Software Specifications

Savvy Signing as specifically been designed and developed for a web environment, utilizing technologies suitable for web development. The application uses a suitable database management system, known as dbsqlite3. The framework used, Django, which is suitable for compatibility with a wide range of devices including desktops, laptops, tablets, and smartphones connected to the internet.

Table 2 Software Specifications

Item	**Specifications**
Operating System Edition	Windows 11 Pro
OS Version	22H2
System Type	64-bit OS, x64-based processor

5.3 Dataset

5.3.1 Data Collection

The data for the real-time gesture recognition and translation system was collected manually using the LabelImg software, a graphical image annotation tool that facilitates the labeling of object detection datasets. LabelImg allows users to open images and draw bounding boxes around objects of interest. Each bounding box is assigned a label corresponding to the object or gesture being identified. The software then saves the annotations in XML format (Pascal VOC) or TXT format (YOLO), which is compatible with most machine learning frameworks. In this case, it saved the images in XML format. These labeled images were stored locally on a personal computer, providing a well-structured dataset for model training.

Once labeled, the images were split into two subsets: training (80%) and testing (20%). This division ensures the model learns effectively from the training data while its performance is evaluated on unseen testing data, improving generalization and accuracy

5.3.2 Features and Labels

In this system, the features represent pixel-based data from the images, such as shape, color, and texture. These features enable the model to identify gestures within the camera's frame. The labels correspond to the specific gestures or actions, such as “Hello,” “Thank You,” or other sign language gestures.

Each labeled gesture was encoded using one-hot encoding, a process where each label is represented as a binary vector with a single 1 cphorresponding to the gesture and 0s elsewhere.

For instance, if there are three gestures ("Hello," "Yes," "No"), "Hello" would be encoded as [1, 0, 0]. This approach ensures that the model treats each gesture as an independent class and avoids ordinal assumptions.

5.3.3 Data Preprocessing

Features were extracted from the annotated images using image preprocessing techniques such as resizing, normalization, and augmentation. Resizing standardizes the image dimensions, while normalization scales pixel values to a range of [0, 1] for consistent input. Data augmentation, such as flipping, rotating, and adjusting brightness, was applied to artificially increase the dataset size and diversity, enhancing the model's robustness.

5.3.4 Training

The training process leveraged the TensorFlow Object Detection API, a robust framework designed specifically for training object detection models. The preprocessed training dataset was input into a neural network model using supervised learning. An SSD (Single Shot Detector) architecture was selected due to its efficiency and accuracy in real-time object detection tasks. The TensorFlow Object Detection API simplified model configuration through its pipeline configuration files, allowing easy customization of key parameters such as the learning rate, batch size, and the number of training steps.

The API enabled seamless execution of training commands and handled complex operations like data augmentation, model checkpointing, and loss calculation. During training, the model iteratively updated its weights to minimize the loss function, which quantified the discrepancy between predicted and actual labels. This iterative process ensured the model progressively improved its detection accuracy, making it suitable for recognizing and translating real-time sign language gestures.

5.4 Testing

After training, the model was evaluated on the testing dataset to measure its generalization performance. Metrics such as accuracy, precision, recall, and F1 score were calculated to assess how well the model recognized gestures in unseen data. Cross-validation techniques were employed to further validate the model, ensuring consistency and robustness.

The validation process also included qualitative assessments by visually inspecting the model's predictions on real-time inputs to confirm its practical applicability and reliability. These steps collectively ensured the model's proficiency and readiness for deployment.

5.4.1 Testing Paradigm

The black box testing paradigm was employed as it is the most appropriate paradigm for this system. This method of testing concentrates on how the application behaves and what results it generates, without any regard to the internal code structure or its implementation. This is because the main purpose of the application in question is for the system to recognize gestures accurately and convert them into text. In this instance black box testing checked that the system was functioning as required from the users' point of view.

The reform applies to increased operational reliability because of considering only those principles and questions that are already solved and central. This is because the approach uses simulations of user interaction and ensures that the expectations are met by validating features-based motion tracking and real time speech translation. It is now possible to shift the focus of the evaluation from translation to its assemblies: as a transformation of objects occurs which is defined in the form of input - output coordinates. In its simplicity of realization, it is convenient for implementing an iterative approach, when functional tests can be passed at different development stages without deep understanding of the internal structure.

5.5 Testing Results

5.5.1 Authentication Module

Table 3 Authentication Module Testing Results

Test Case ID	**Description**	**Test Data**	**Expected Outcome**	**Actual Results**	**Status (Pass or Fail)**
TC001	Check user registration with valid data i.e. First Name, Last Name, email and Password	First Name = Traciebel Lat Name = Kinyari Email = Traciebel.kinyari@strathmore.edu Password = 12345	User should be able to input their user credential and register to create a new user account	As expected	Pass
TC002	Check user login with valid data	Email =Traciebel.kinyari@strathmore.edu	User should be able to automatically	As expected	Pass

			login into the application's home page		
TC003	Check user login with invalid data	Email =tracybelle.kanyari@strathmore.edu	The system displays an error message('invalid login credentials')	As expected	Pass
TC004	Check user registration with valid data	First Name = John Last Name = Doe Email = johndoe@gmail.com Password = john@Doe23	The system should notify the user of the successful registration with a toast message	As expected	Pass

5.5.2 Sign Language to Text Translation Module

Table 4 Sign Language to Text Translation Module

Test Case ID	**Description**	**Test Data**	**Expected Outcome**	**Actual Results**	**Status (Pass or Fail)**
TC005	Check if the device's camera is activated	Hand movements in front of the device's camera	The user should be able to view their hand moving on a frame in real time	As expected	Pass
TC006	Check if the translation frame is starting up	Starting up the backend	The user should be able to view a frame that picks up the hand gestures and their corresponding translations	As expected	Pass
TC007	Check if the application is picking up the hand signs	Display basic hand signs towards the camera and make sure	The user should be able to view the respective text to the hand signs being	As expected	Pass

		the hand is being seen	displayed on their screen		
TC008	Check if the translations are accurate	Display a specific hand sign, i.e. “Hello” sign towards the camera	The user should be able to view the corresponding text “Hello” on their screens	As expected	Pass

5.5.3 Basic signs and tutorials Module

Table 5 Basic signs and tutorials Module

Test Case ID	**Description**	**Test Data**	**Expected Outcome**	**Actual Results**	**Status (Pass or Fail)**
TC009	Check if the web pages display the basic signs and SL tutorials	Clicking on the Basic signs’ menu	The user should be able to view the basic signs and SL tutorials	As expected	Pass

5.5.4 Instructions/ How it works Module

Table 6 Instructions Module

Test Case ID	**Description**	**Test Data**	**Expected Outcome**	**Actual Results**	**Status (Pass or Fail)**
TC010	Check if the instructions are well displayed on the web page	Clicking on the instructions’ menu	The user should be able to view the web page that explains the different components and steps on how the	As expected	Pass

			translation works		

Chapter 6 : Conclusions, Recommendations and Future Works

6.1 Conclusions

In conclusion, the real-time sign language recognition and translation system provides a practical and innovative solution to bridge the communication gap between deaf or hard-of-hearing individuals and the public. By leveraging advanced machine learning techniques, particularly the TensorFlow Object Detection API, the system accurately recognizes sign language gestures and translates them into text in real-time, making communication more accessible. This approach not only solves the problem of language barriers but also enhances inclusivity by empowering users with a tool that promotes seamless interaction.

The solution presents a significant improvement over traditional methods, such as relying on sign language interpreters or static translation systems, by offering a more dynamic and accessible means of communication. Its impact is particularly relevant to both the deaf community and society at large, fostering greater participation in social, educational, and professional settings. Moreover, the project's contributions to the IT community are notable, as it showcases the potential of real-time machine learning applications in solving real-world problems, encouraging further research and development in the field of assistive technologies. The application also demonstrates how AI and computer vision can be effectively applied to support social good, ultimately making the world more connected and inclusive.

6.2 Recommendations

The CPUs I was using to construct this project were just basic, and the software working with these machines was also basic. To begin crafting the multi-functional application, much better hardware would be required, perhaps something like an Intel Core I7 or I9 CPU. These types of CPUs have high clock speeds and are equipped with multiple cores which allow them to perform complex calculations without any issues.

My recommendation for real-time translation applications during Apple M1 laptop ML model training is the Apple MacBook Pro with an M2 Pro or M2 Max chip, 32GB of unified memory, and 1TB of SSD. This laptop has remarkable computational capacity because of its M2 series chip's integrated NPUs which enables it to perform model training and real-time translation faster. Together with the large Retine display, the high RAM and quick SSD facilitate multitasking which enables integration of filtering large datasets without any bottlenecks during heavy usage. The architecture of the MacBook Pro has high-performance which is required when working with AI or TensorFlow or Pytorch or other Libraries. It shortens the time used in training the models leading to

6.3 Future Works

Should offline capabilities be introduced, this will enable users to make use of the application without an Internet connection, making it usable in locations with poor Internet connection coverage and for users with limited access to data.

This could help, as users will be able to give feedback on areas where they find the gesture recognition feature failing and where it is inaccurate. Such information can be mined to improve the system over time and to ensure a more tailored user experience especially to the users of the app.

While the current application does recognize a few basic sign language gestures, expanding the scope to other elaborate gestures and regional variations would make the application more relevant to a wider audience. For this, the model could be trained using a richer dataset consisting of more sign language varieties.

To further widen its societal impact, the application may be made to collaborate with other assistive technologies like voice command devices to increase its practicality and reach.

References

Papatsimouli, M., Sarigiannidis, P., & Fragulis, G. F. (2023). *A Survey of Advancements in Real-Time Sign Language Translators: Integration with IoT Technology*.

Cayamcela MEM, Lim W. *Fine-tuning a pre-trained Convolutional Neural Network Model to translate American Sign Language in Real-time*. 2019 International Conference on Computing, Networking and Communications, ICNC 2019. 2019;: p. 100-104.

Núñez-Marcos, A., Perez-de-Viñaspre, O., & Labaka, G. (2023, March 1). *A survey on Sign Language machine translation.*

Papastratis, C., Konstantinidis, D., Dimitropoulos, K., & Daras, P. (2021, August 30). Artificial intelligence technologies for sign language. Sensors (Basel, Switzerland), 21(17).

United Nations [UN]. (2019, September 23). *Sign language protects 'linguistic identity and cultural diversity' of all users, says UN chief. UN News*.

Rastgoo, R., Kiani, K., & Escalera, S. (2021, February 01). *Sign Language Recognition: A Deep Survey*

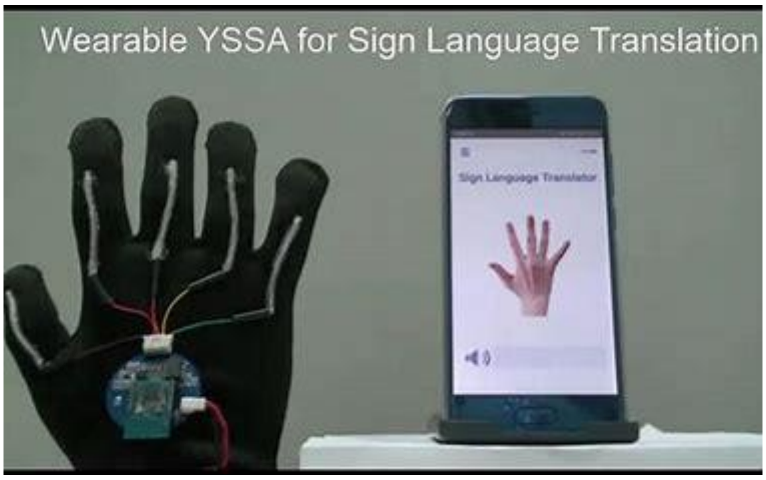
Andra, A., Brandon, H., Halim, M., Hanafiah, N., & Wibisurya, A. (2021, January 01). Systematic Literature Review: American Sign Language Translator.

Davydov, A., & Lozynska, A. (2017a, June). Sign language recognition with deep memory networks. In 2017 IEEE International Conference on Computer Vision (ICCV) (pp. 5344-5352). IEEE. https://ieeexplore.ieee.org/document/10048804

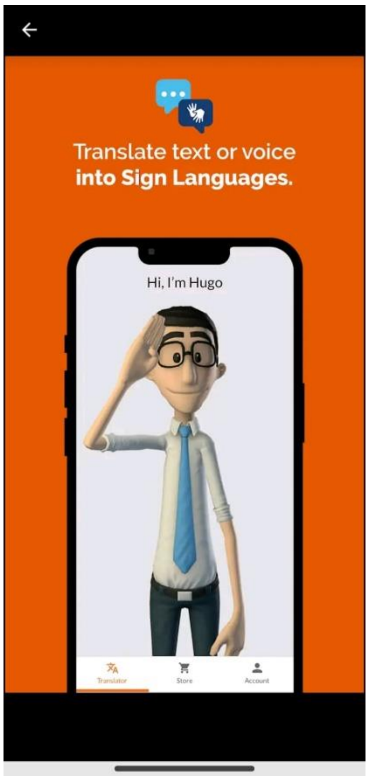
Jin, S., Kim, M., Kim, J., & Kim, H. (2016). *A deep learning approach for sign language recognition using recurrent neural networks.* In Proceedings of the 2016 International Conference on Machine Learning and Cybernetics (ICMLC) (Vol. 2, pp. 720-724). IEEE.

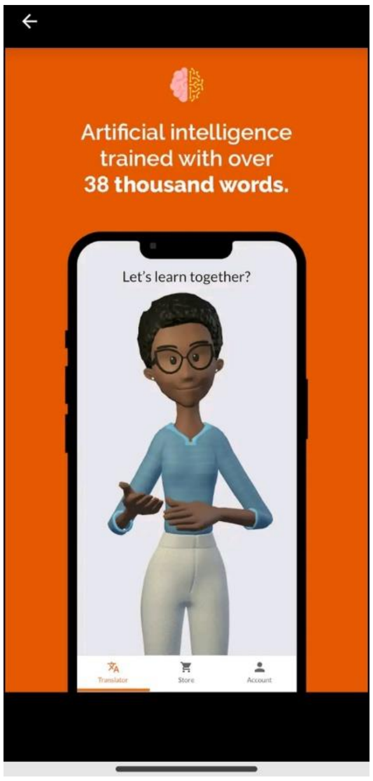
Appendix

Appendix 1. Wearable UCLA gloves

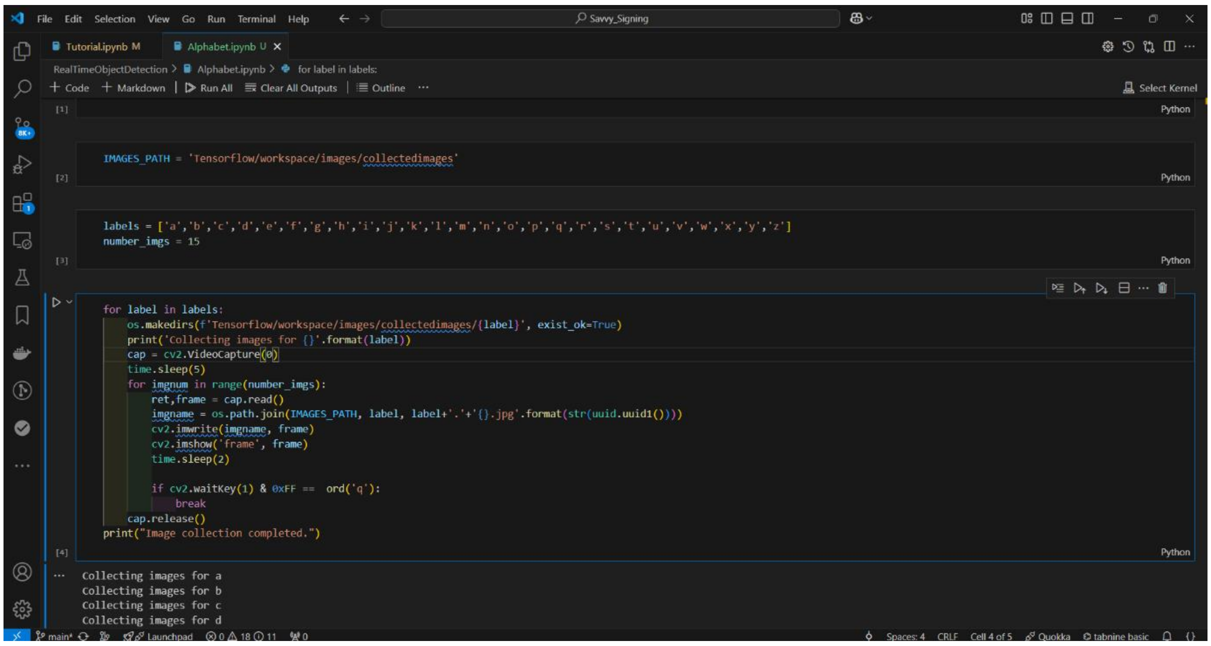


Appendix 2. Hand Talk App





Appendix 3. Labelling the dataset for training



```
IMAGES_PATH = 'Tensorflow/workspace/images/collectedimages'
```

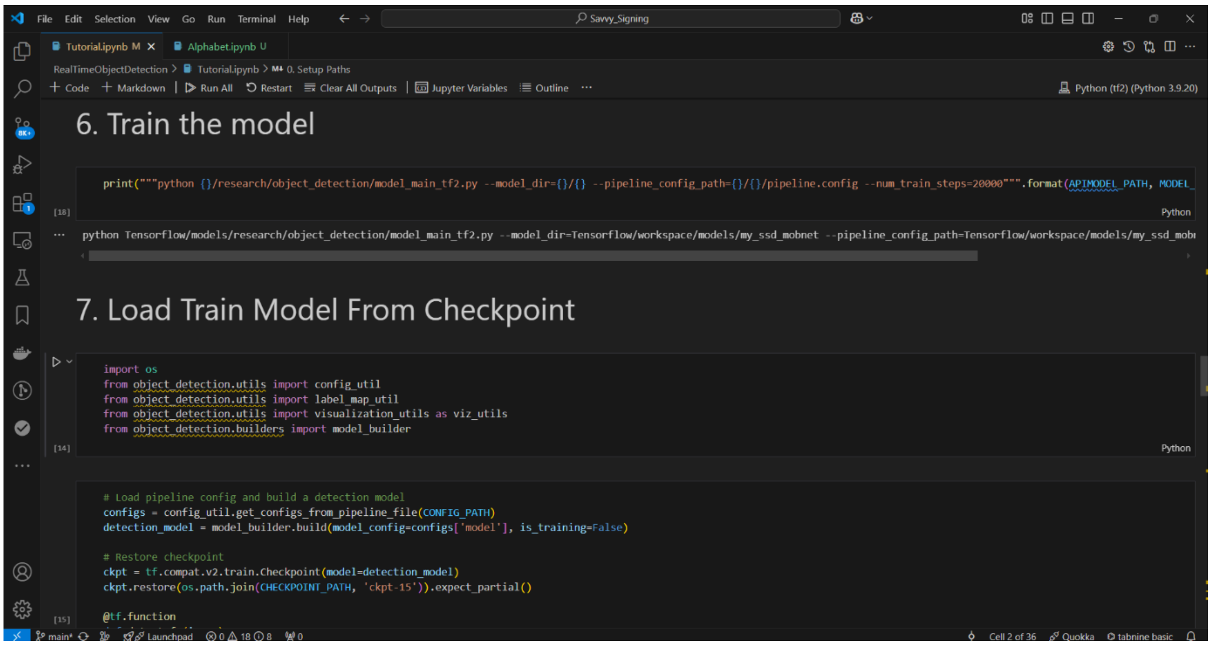
```
labels = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
number_imgs = 15
```

```
for label in labels:
    os.makedirs(f'Tensorflow/workspace/images/collectedimages/{label}', exist_ok=True)
    print('Collecting images for {}'.format(label))
    cap = cv2.VideoCapture(0)
    time.sleep(5)
    for imgnum in range(number_imgs):
        ret,frame = cap.read()
        imgname = os.path.join(IMAGES_PATH, label, label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        cv2.imwrite(imgname, frame)
        cv2.imshow('frame', frame)
        time.sleep(2)

        if cv2.waitKey(1) & 0xFF ==  ord('q'):
            break
    cap.release()
print("Image collection completed.")
```

```
Collecting images for a
Collecting images for b
Collecting images for c
Collecting images for d
```

Appendix 4. Training the Sign language Recognition and Translation Model



6. Train the model

```
print("""python {}/research/object_detection/model_main_tf2.py --model_dir={}/{} --pipeline_config_path={}/{}/pipeline.config --num_train_steps=20000""".format(APIMODEL_PATH, MODEL_
```

```
python Tensorflow/models/research/object_detection/model_main_tf2.py --model_dir=Tensorflow/workspace/models/my_ssd_mobnet --pipeline_config_path=Tensorflow/workspace/models/my_ssd_mobi
```

7. Load Train Model From Checkpoint

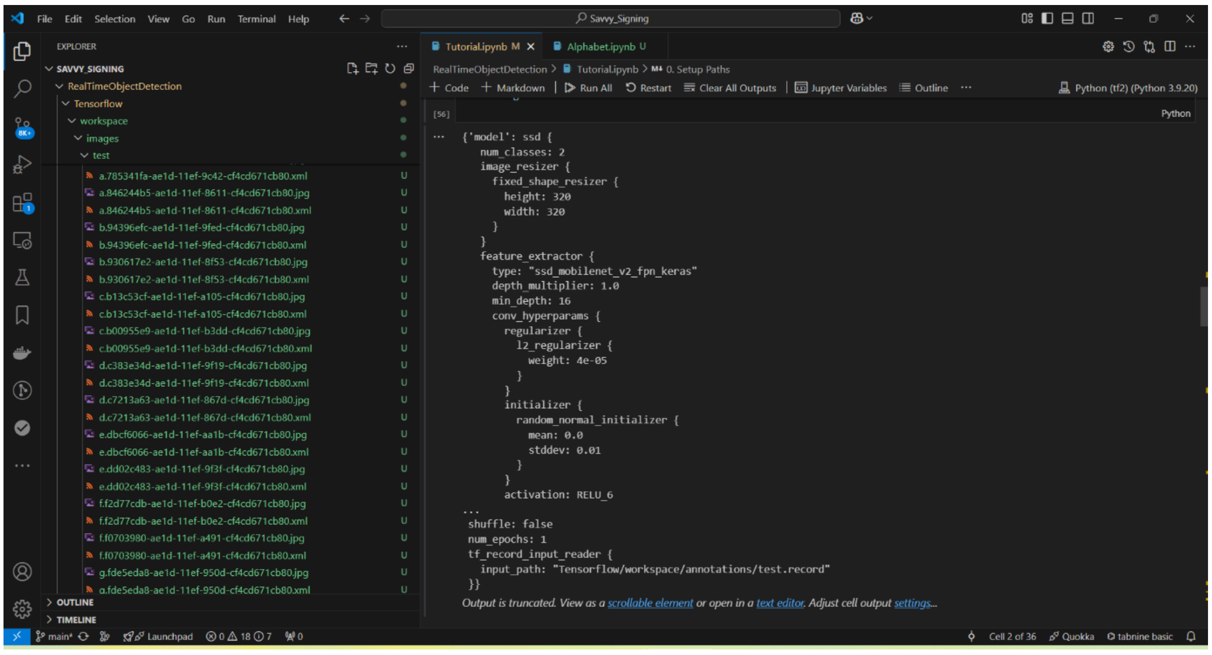
```
import os
from object_detection.utils import config_util
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
```

```
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(CHECKPOINT_PATH, 'ckpt-15')).expect_partial()

@tf.function
```

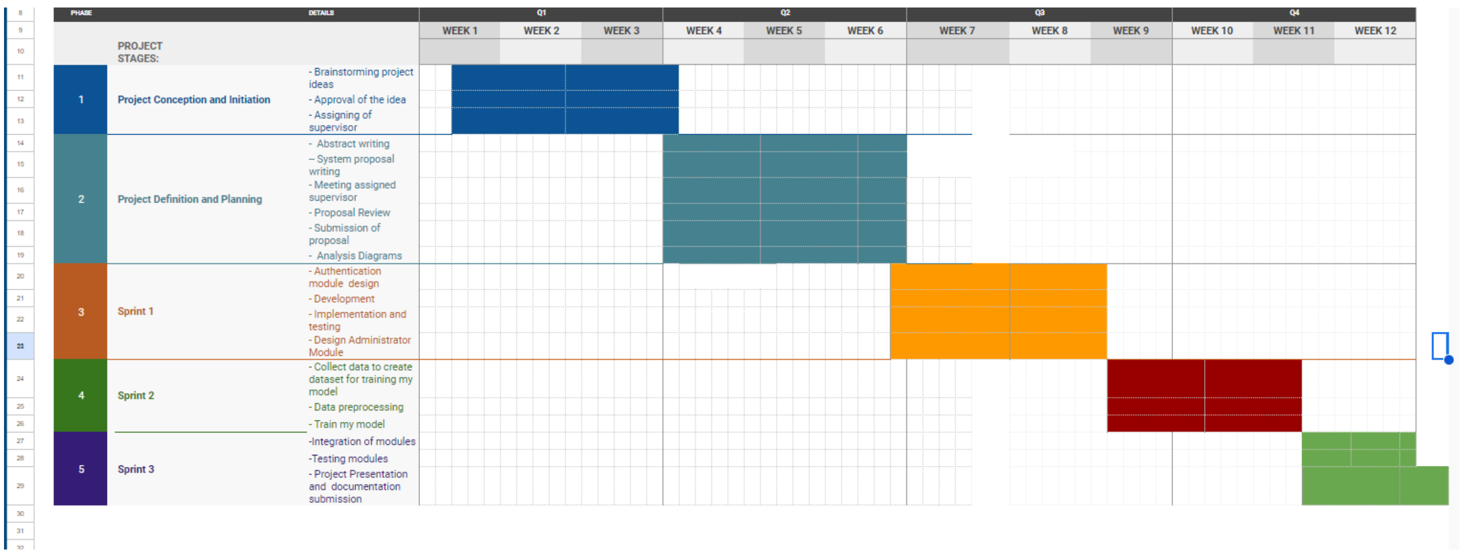
Appendix 5. Test Dataset in XML format (Images)



```
SAVVY_SIGNING
  RealTimeObjectDetection
    Tensorflow
      workspace
        images
          test
            a.785341fa-ae1d-11ef-9c42-cf4cd671cb80.xml    U
            a.846244b5-ae1d-11ef-8611-cf4cd671cb80.jpg    U
            a.846244b5-ae1d-11ef-8611-cf4cd671cb80.xml    U
            b.94396efc-ae1d-11ef-9fed-cf4cd671cb80.jpg    U
            b.94396efc-ae1d-11ef-9fed-cf4cd671cb80.xml    U
            b.930617e2-ae1d-11ef-8f53-cf4cd671cb80.jpg    U
            b.930617e2-ae1d-11ef-8f53-cf4cd671cb80.xml    U
            c.b13c53cf-ae1d-11ef-a105-cf4cd671cb80.jpg    U
            c.b13c53cf-ae1d-11ef-a105-cf4cd671cb80.xml    U
            c.b00955e9-ae1d-11ef-b3dd-cf4cd671cb80.jpg    U
            c.b00955e9-ae1d-11ef-b3dd-cf4cd671cb80.xml    U
            d.c383e34d-ae1d-11ef-9f19-cf4cd671cb80.jpg    U
            d.c383e34d-ae1d-11ef-9f19-cf4cd671cb80.xml    U
            d.c7213a63-ae1d-11ef-867d-cf4cd671cb80.jpg    U
            d.c7213a63-ae1d-11ef-867d-cf4cd671cb80.xml    U
            e.dbcf6066-ae1d-11ef-aa1b-cf4cd671cb80.jpg    U
            e.dbcf6066-ae1d-11ef-aa1b-cf4cd671cb80.xml    U
            e.dd02c483-ae1d-11ef-9f3f-cf4cd671cb80.jpg    U
            e.dd02c483-ae1d-11ef-9f3f-cf4cd671cb80.xml    U
            f.f2d77cdb-ae1d-11ef-b0e2-cf4cd671cb80.jpg    U
            f.f2d77cdb-ae1d-11ef-b0e2-cf4cd671cb80.xml    U
            f.f0703980-ae1d-11ef-a491-cf4cd671cb80.jpg    U
            f.f0703980-ae1d-11ef-a491-cf4cd671cb80.xml    U
            g.fde5eda8-ae1d-11ef-950d-cf4cd671cb80.jpg    U
            g.fde5eda8-ae1d-11ef-950d-cf4cd671cb80.xml    U
```

```
[56]
... {'model': ssd {
      num_classes: 2
      image_resizer {
        fixed_shape_resizer {
          height: 320
          width: 320
        }
      }
      feature_extractor {
        type: "ssd_mobilenet_v2_fpn_keras"
        depth_multiplier: 1.0
        min_depth: 16
        conv_hyperparams {
          regularizer {
            l2_regularizer {
              weight: 4e-05
            }
          }
          initializer {
            random_normal_initializer {
              mean: 0.0
              stddev: 0.01
            }
          }
          activation: RELU_6
    ...
     shuffle: false
     num_epochs: 1
     tf_record_input_reader {
       input_path: "Tensorflow/workspace/annotations/test.record"
     }}
    Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Appendix 6. Project Timeline



PHASE		DETAILS	Q1			Q2			Q3			Q4		
	PROJECT STAGES:		WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 5	WEEK 6	WEEK 7	WEEK 8	WEEK 9	WEEK 10	WEEK 11	WEEK 12
1	Project Conception and Initiation	- Brainstorming project ideas - Approval of the idea - Assigning of supervisor	🟦	🟦	🟦									
2	Project Definition and Planning	- Abstract writing – System proposal writing - Meeting assigned supervisor - Proposal Review - Submission of proposal - Analysis Diagrams				🟦	🟦	🟦						
3	Sprint 1	- Authentication module design - Development - Implementation and testing - Design Administrator Module							🟧	🟧				
4	Sprint 2	- Collect data to create dataset for training my model - Data preprocessing - Train my model									🟥	🟥		
5	Sprint 3	-Integration of modules -Testing modules - Project Presentation and documentation submission											🟩	🟩