Middlesex University

CST 3990

Undergraduate Individual Project

2024 - 2025

Title: SignSpeak – A Real-time ASL Interpreter

Name: Mithra Iyengar Ramprabu

MISIS No.: M01031074

Supervisor: Dr. Maha Saadeh

Module Leader: Dr. Fehmida Hussain

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

Sign Language has been the most accessible and essential method of communication for the deaf and hard-of-hearing for millennia. Its documented history dates to the 5th century BCE, when the philosopher Socrates, in Plato's *Cratylus,* speculated on the usage of hand, head and body gestures as a form of language for those without tongue or voice. However, formal recognition of sign language did not emerge until the 18th century, when the 'Father of Sign Language' Charles-Michel de l'Épée standardized a system for manual communication, later acknowledged as the French Sign Language (LSF), thereby laying the foundation for the American Sign Language (ASL) in the early 1800s, popularized by Thomas Hopkins Gallaudet and Laurent Clerc.

Despite the vast significance of these achievements, sign language is often marginalized and while officially recognized in over 70 countries, access to formal instruction in the language remains limited. In the 1880 Milan Conference, educators—predominantly hearing individuals—declared oralism (oral education) superior, leading to a ban on the usage of sign language in schools and causing a global decline in sign language education that persisted for nearly a century.

The World Federation of the Deaf (WFD) estimates that over 70 million deaf individuals worldwide use sign language today, yet fewer than 2% have access to education in their native sign language.

In an increasingly digitized world, the integration of Artificial Intelligence and Machine Learning into sign language interpretation represents a leap towards inclusivity in a society wherein vast communication gaps persist in healthcare, education and daily interactions with hearing individuals often lacking the means to understand sign language. By leveraging AI and ML-powered systems with computer vision, we may perceive a future of seamless communication and widespread understanding of sign language. However, such a future holds many challenges, including the variations in over 300 distinct sign languages, the complexity of markers like facial expressions, the ability of the system to identify dynamic and quick-moving signage and the need for high-speed processing to ensure fluid communication.

The implementation of AI-driven sign language recognition software has the ability to break down linguistic barriers and contribute vastly to the deconstruction of ableist principles. By integrating these tools into public services, emergency response systems, and digital communication platforms; society can foster a more inclusive and accepting environment for the deaf and heard-of-hearing community. Recognizing the knowledge of sign language as more than just an auxiliary skill, but an essential linguistic system imperative to achieving true inclusivity allows us the opportunity to rectify centuries of exclusion and ensure that sign language is treated with the same importance as any other spoken language, and the path to such drastic but critical measures is sure to be cleared through the advancement of AI and ML-based technologies.

INTRODUCTION

Sign Language is the primary means of communication for the hard-of-hearing community of the world, and utilises the movement of hands, fingers and facial expressions to express opinions. Today, there are more than 300 different sign languages in the world, spoken by more than 72 million deaf or hard-of-hearing people worldwide. Regardless of its applicability and necessity in today's day and age, Sign Language is not a widely spoken language by the hearing and is not taught in schools or other educational facilities. Considering the importance of networking and communication, it should be made possible for People of Determination to express themselves freely and integrate into society without the constraints of their disability.

With the tech-revolution, this can be made possible.

By leveraging Machine Learning Techniques and Artificial Intelligence models, the system I intend to create will detect ASL gestures in real-time and interpret them. The model will be trained on multiple datasets (both customised and publicly available), and the system will be presented as an aesthetic, user-friendly interface.

This project follows a **BUILD + MODEL** approach, integrating deep learning with practical application development. The system will use **Neural Networks** to improve accuracy, feature extraction techniques and multiple related libraries (TensorFlow, Scikit-Learn, etc.) to ensure optimisation of model performance.

AIM

The penultimate aim of the project is to attempt to provide a means of communication for the deaf and hard-of-hearing that encourages inclusivity and efficiency. Moreover, with the success of such a system; assuming it could potentially be implemented on a larger scale; the learning of sign languages may become widespread and essential, thereby breaking barriers between the differently abled and others, and discouraging ableism in society.

Further, the project attempts to utilize deep-learning and machine-learning/Artificial Intelligence concepts to interpret the signs. This is not to replace preexisting professional sign language interpreters; rather, it would enhance their abilities, allowing more people to learn sign languages, and creating more demand for their skills; signers being essential for the development of such complex deep learning systems.

Communication is a necessity, not a luxury; and therefore, cannot be impeded by frilly user interfaces requiring complicated manoeuvres. The GUI used to display the application must be accessible, efficient and easy-to-use.

Finally, the project must work in real-time. This necessitates the use of a video-based system that captures frames and compiles them to interpret the sign.

OBJECTIVES

The objectives of the project are few but complex, each possessing multiple technical barriers that complicate the task at hand. They are as follows:

Collect and preprocess accurate datasets of ASL gestures: The dataset is required to be sufficiently diverse (in terms of lighting, positioning etc.), handle class imbalances and in the case of preexisting datasets, be of sufficient quality.

Train a suitable ML model for gesture classification: Training and building a model requires carefully balancing between accuracy and efficiency while avoiding overfitting. Selecting an appropriate architecture, optimizing hyperparameters and ensuring robustness against variations and background noise are all details that vastly affect the performance of the system, and achieving a compromise between the accuracy and scope of the system, while preventing the model's overfitting is a crucial step in the project creation process.

Implement a real-time video processing system to detect hand gestures: The system must efficiently detect hand movements, assign landmarks and features, and recognize said movements with minimum latency. This includes handling external factors like lighting, reducing noise, and maintaining accuracy regardless of differences between the model's training input and the user's interactive dynamic input.

Develop an application with a user-friendly interface: The application must seamlessly integrate the gesture recognition system with a suitably intuitive design that prioritizes accessibility and efficiency (as mentioned above).

SCOPE AND SIGNIFICANCE

This project sets out to create an AI-based solution for real-time detection and translation of American Sign Language (ASL) with the goal of improving communication for the Deaf and Hard-of-Hearing (DHH) community. Currently, around 430 million people globally require some form of aid for hearing loss and this number is expected to rise to 700 million by the year 2050. The system will incorporate machine learning methods for gesture recognition to help improve interactions among ASL users and non-signers. This project stems from the emerging interest to utilize Artificial Intelligence for accessibility purposes, as shown with Nvidia's release of a free tool that teaches ASL (Signs) using AI. This project was also inspired by other applications like AI Sign that uses machine learning to identify over a hundred ASL signs.

The importance of this project lies in its ability to help lessen the communication barriers of the DHH community with the rest of the world. In the United States alone, studies indicate that almost 5% of adults aged 45-54 years old experience severe hearing loss, which increases to 55% of people over the age of 75. Although ASL is the third most popular language in the United States, resources remain inadequate for teaching or interpreting it, this insufficiency being acutely felt during the development of this project. By developing a system that accurately interprets ASL, the project hopes to improve the quality of life for DHH individuals, facilitating better integration into educational, professional, and social environments.

LITERATURE REVIEW

Terminologies: ASL (American Sign Language), Machine Learning, People of Determination, Hard of Hearing, Accessibility, Dataset

1. INTRODUCTION

This review assesses the scope and benefits of digital, automatic Sign Language translation. It will elucidate the advantages of integrating dynamically developing fields like Artificial intelligence and Machine Learning into ASL Interpretation and evaluate the ethical and computational concerns that these systems could pose upon and during development.

SLR (Sign Language Recognition) has gained significant traction over the past few years as an efficient means of bridging the gap between the deaf and hearing. In particular, ASL poses significant challenges due to its heavy reliance on facial expressions, hand gestures and body movements (Koller et al., 2019). More traditional SLR systems relied upon sensor-based techniques, such as gloves and motion capture systems, which prove highly impractical for real-world use.

Recent advancements in deep learning and computer vision have significantly improved video-based ASL interpretation systems. Models like YOLO (Redmon et al., 2016) and Mediapipe have been explored for real-time gesture detection, enhancing the efficiency of SLR systems.

Despite these advancements, various challenges lay in the path to successfully creating a system that can reliably translate gestures into words and then sentences. Signer variability, background noise, and differences in video quality affect model performance (Zhou et al., 2021) drastically. Further, differentiating between static and dynamic signs, as well as recognizing subtle variations in hand configurations, requires robust feature extraction methods. The following review shall discuss a few existing techniques in ASL recognition in order to provide insight into the development of an efficient and effective interpreter.

2. REVIEW OF EXISTING TECHNOLOGIES

2.1. SENSOR-BASED RECOGNITION SYSTEMS

Earlier, Sign language recognition systems relied on wearable sensors, like gloves equipped with accelerometers, gyroscopes, and flex sensors (Liang & Ouhyoung, 1998). Devices like the CyberGlove captured precise finger movements, while Microsoft Kinect and Leap Motion provided depth-based gesture tracking (Ghosh et al., 2019). Using these sensor-gloves were considered a better idea over cameras as they give the user the flexibility of moving around freely within a radius limited by the length of wire connecting the glove to the computer; unlike a camera within which the user must stay in position camera. The effect of light, electric or magnetic fields or any other external disturbance does not affect the performance of the glove (Mehdi et al., 2002). While these methods were highly accurate, they were also impractical due to high cost of resources and portability concerns.

Figure 1: Sensor gloves for SLR

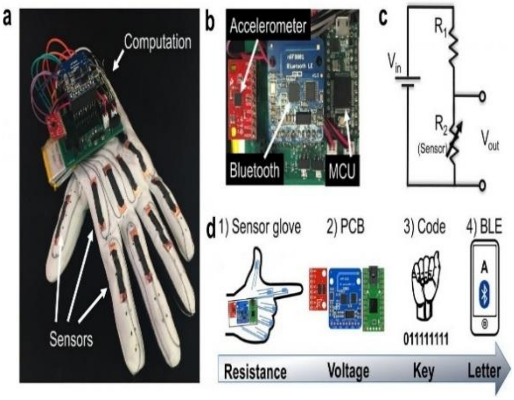


Figure 2: Parts of a Sensor-Glove

2.2. COMPUTER VISION-BASED APPROACHES

With recent advancements in deep learning, vision-based SLR has come to be the predominant approach. CNNs (Convolutional Neural Networks) are widely used for static sign recognition, extracting features from hand images (Pigou et al., 2015). However, ASL consists of both static and dynamic signs, requiring models that can capture temporal dependencies (The relation between different gestures and features at different points in time and their influence on one another).

Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks were introduced to model sequential information in sign language videos (Camgoz et al., 2018). More recently, Transformer-based models, such as

SignBERT, have demonstrated improved performance in recognizing complex sequences of signs. These models leverage self-attention mechanisms (Reviews the entirety of the model at once and then prioritising the frames and/or features) to capture both spatial and temporal dependencies more effectively than RNNs.

2.3. OBJECT DETECTION AND POSE ESTIMATION

Object detection models like YOLO and Faster R-CNN are popular systems to explore when attempting to interpret ASL in real-time. These models can detect hands and face landmarks, providing critical information for gesture recognition. Additionally, MediaPipe Hands (Zhang et al., 2020) and OpenPose (Cao et al., 2019) have been used to extract skeletal key points, allowing models to focus on the hand shape, movement, and facial expressions without processing entire video frames, thereby making the project less resource-intensive and relatively less expensive.

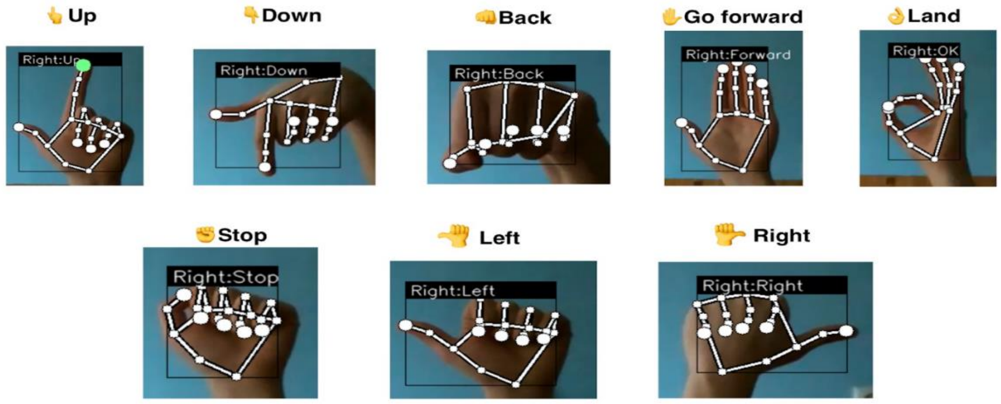


Figure 3: Mediapipe Skeleton Recognition

2.4. THE AVAILABILITY OF LARGE-SCALE DATASETS

The availability of large datasets has been highly influential in advancing the field of SLR research. The WLASL dataset (Li et al., 2020), one of the largest ASL datasets in the world, contains over 2,000 sign classes (words) with a diverse set of signers, enabling the deep learning models to generalise better. Other datasets, such as RWTH-PHOENIX-Weather (Forster et al., 2012) and LSA64 (Ronchetti et al., 2016), have contributed to training models on diverse linguistic variations as well.

3. DIFFERING VIEWPOINTS

Multiple arguments arise when discussing a field as complex and dynamic as SLR systems. The most predominant of these disagreements would be the use of physical data collection tools like sensor-based gloves opposed to that of a camera-based system (YOLO, MediaPipe etc.). A popular view would be that glove-based systems are just more straight-forward. They eliminate the need to process raw data into meaningful values by directly reporting relevant and required input data in terms of voltage values or resistances as in flex sensors' case to the computing device. On the other hand, vision-based systems need to apply specific tracking and feature extraction algorithms to raw video streams, increasing the computational overhead. (S. A. E. El-Din, M. A. A. El-Ghany., 2020). While a computer-vision-based system would be more computationally expensive, they are far more convenient, portable and inexpensive, requiring no additional tools to be purchased, thereby making the system accessible to all financial backgrounds.

Another argument would be the advantage of using a feature-based system over a deep-learning based one. Although the latter has proven highly capable and developed rapidly over the last few years, it is computationally expensive and difficult to implement. Neural network algorithms are also incredibly time-consuming when processing digital images and fail to perform ideally in different backgrounds, skin colours and coloured clothes. Such obstacles, however, are being rapidly overcome with the advancement of the M.L, D.L and A.I fields. Today, many researchers are inclined towards Deep-Learning methods due to its vast scope for improvement, adjustment and its high performance in video-related tasks. As SLR is a task closely related to computer-vision, it makes sense that most recent approaches tackling the issue have shifted to this direction.

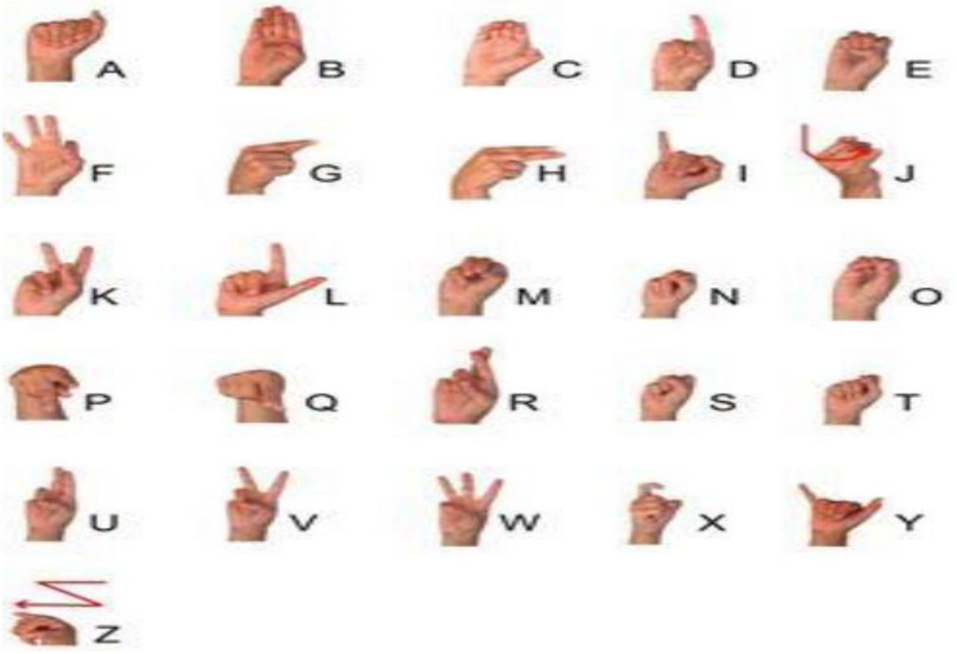


Figure 4: ASL Hand Gestures

Given below are the various opinions of researchers on different matters related to the subject.

THEME	Source 1 (Nikolas Adaloglou et al.,2021)	Source 2 (Butt et al., 2019)	Source 3 (Buttar et al., 2023)	Source 4 (Jiang et al., 2021)
Computational Efficiency	Deep learning is computationally expensive but tractable after optimization. (Page-2)	Feature-based methods are lightweight and work on low-resource devices. (Page-2)	Not stated explicitly.	Deep Learning models are computationally costly, but the improvement in model effectiveness renders it worthwhile. (Page-3)
Identification of Static vs. Dynamic Signs	End-to-End models can process both categories but require large datasets (Page-2)	Feature-based methods are suitable for static signs but struggle with sequences and dynamic signs. (Page-2)	Hybrid deep learning improves dynamic sign recognition. (Page-19)	Skeleton-based deep learning models significantly improve dynamic sign recognition. (Page-2)
Dataset Size Requirements	Deep learning requires vast datasets but performs better than a feature-	Small datasets are sufficient for feature-based methods. (Page-3)	Large, annotated datasets improve deep learning performance.	Data augmentation strategies can help curb the

	based system with them. (Page-4)		(Page-3)	requirement for huge, labelled datasets in deep learning algorithms. (Page-6)
Generalization Across Signers	Neural networks handle signer variation better than traditional methods (Feature-based). (Page-3)	Handcrafted features may not generalize well across different signers. (Page-5)	Deep learning models can generalize well with enough data. (Page-7)	Multi-modal deep learning can adapt to different signers more effectively. (Page-2)
Robustness to Background Noise	Deep learning models are capable of filtering out undesired background information. (Page-4)	Feature extraction may be affected by background variation. (Page-2)	CNNs and RNNs help deep learning filter out noise. (Page-5)	Multi-modal approaches enhance robustness to environmental variations. (Page-8)
Interpretability & Explainability	Handcrafted features are interpretable and transparent. (Page-1)	Deep learning models have complex, non-human-readable structures and are harder to interpret. (Page-1)	Model interpretability can be improved with feature visualization techniques. (Page-4)	Ensemble models improve explainability by integrating multiple data sources. (Page-6)
Training & Implementation Complexity	Pre-trained models in deep learning reduce complexity. (Page-3)	Feature-based approaches need manual feature engineering but are simpler to implement. (Page-2)	Deep learning requires significant expertise and computational resources. (Page-6)	Training of deep learning models is Computationally expensive but provides greater accuracy. (Page-9)
Conclusion	Greater preference for deep learning due to better adaptability. (Page-12)	Feature-based methods are practical for low-resource scenarios. (Page-6)	Deep learning is preferred for recognizing both static and dynamic signs. (Page-19)	Deep learning, particularly with multi-modal input, outperforms feature-based methods. (Page-9)

4. GAPS IN EXISTING WORKS OF RESEARCH

Existing works on SLR have multiple gaps particularly in dataset diversity, real-time processing and model generalization. **Butt et al. (2019)** and **Buttar et al. (2023)** rely on smaller datasets, making it difficult to generalize across different, diverse signers, backgrounds, and spontaneous signing. **Jiang et al. (2021)**, while using a skeleton-based approach (MediaPipe), does not evaluate the system's performance on unseen signers or spontaneous gestures (real-time). Computational efficiency remains a challenge, as **feature-based methods (Butt et al., 2019)** struggle with complex signage, while **deep learning methods (Buttar et al., 2023; Adaloglou et al., 2020)** demand high resources but lack optimizations such as model pruning or edge AI. Additionally, interpretability remains a concern, as deep learning models are often black-box systems (complex internal structures), making it difficult to debug or justify their decisions in practical applications. Future research should focus on diverse datasets, lightweight models for real-time use, and hybrid approaches that balance accuracy, efficiency, and interpretability.

5. EMERGING TECHNOLOGIES

5.1. BERT (Bidirectional Encoder Representations from Transformers)

BERT, a transformer-based model originally designed for NLP (Natural Language Processing) systems, is gradually becoming predominant in SLR. By adapting BERT or any similar transformer models, researchers can capture the context and semantic nuances of sign language better. This is particularly useful for processing sequences of gestures, where understanding the context (which sign follows which) is important. BERT's bidirectional attention mechanism allows for more accurate recognition and translation by considering not just the previous sign but the entire context of the sign language sequence (sentence). (Devlin et. al., 2019).

5.2. Virtual Reality (VR) and Augmented Reality (AR)

VR and AR create immersive environments that allow for efficient sign language learning and communication. In VR, users can interact with virtual hands or avatars, helping them practice sign language in a controlled, monitored space. On the other hand, AR allows for real-world signs to be captured and processed by the device (smart glasses, smartphone etc.), providing immediate feedback or translations. These technologies allow learners to practice sign language in a more engaging and interactive way. (Rakhmadi et al., 2024).

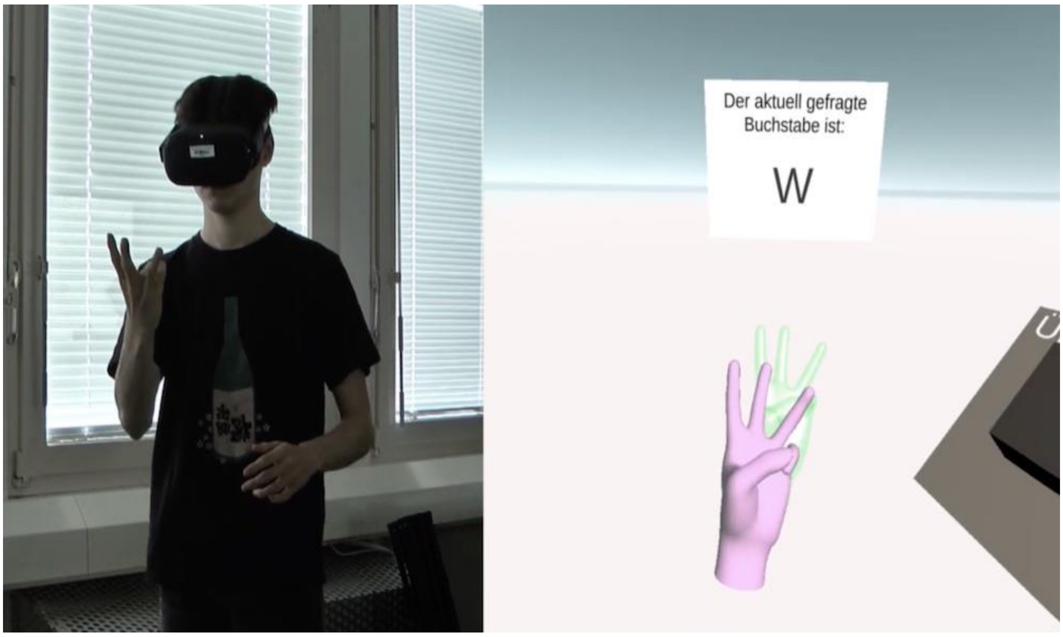


Figure 5: VR Headset used to teach SL

5.3. Natural Language Processing (NLP)

NLP techniques are evolving rapidly to handle the discreet nuances of sign language better, particularly in translating signs to text or speech. Sign language is often not a direct one-to-one translation with spoken language, so NLP is crucial in improving our understanding of the context, grammar, and meaning of sign sequences. Models are being designed to process multi-modal data (video + text), enabling a greater understanding of signs and improving real-time translation systems. (Jeyasheeli et. al., 2020).

5.4. Generative Pre-trained Transformers (GPTs)

GPT models, including GPT-3 and GPT-4, can be leveraged in Sign Language Recognition for contextual translation. By using these models, you can generate natural language interpretations of signs, even considering different cultural contexts and variations of ASL. For example, a GPT model could be trained to interpret ASL signs and convert them to meaningful responses in natural language, enhancing real-time communication. (Brown et. al., 2020).

6. ETHICAL DILEMMAS

The development of Automatic Sign Language Recognition systems presents various moral dilemmas that must be carefully addresses to ensure security, inclusivity and responsible usage of A.I and technology. A primary concern of ASLR is privacy. Most systems of this kind rely upon video-based data, which raises issues on security, surveillance and consent. Users may understandably be uncomfortable with the idea of being recorded constantly and analysed, especially if this data is stored and shared unwillingly. Ethical usage mandates data anonymity, encryption and clear written consent policies to protect user privacy and data.

Another major worry the field poses is employability. ASLR systems, while efficient and accessible, must only complement human interpreters, rather than replace them. Overly relying on D.L. machines to translate for people of determination could cause job displacements for professional sign language interpreters, potentially affecting the access of the deaf community to culturally and contextually aware communication that automatic systems struggle to replicate.

Not to mention the age-old open ethical question, “Who is responsible?”, which arises when errors are made during ASL interpretation in high-stakes situations with serious repercussions (legal and medical settings, emergencies etc.) wherein the blame is shifted between the developers, the deploying institution or A.I. itself.
There is also the possibility of bias, as the model may be better acclimatised to one skin tone, dialect, language, signing speed etc. over the other, thereby disadvantaging certain factions of the community.

In order to eradicate these dilemmas, ASLR research must prioritise fair, unbiased dataset collection, transparent decision-making, ethical data handling and overall community involvement in system design process, thereby ensuring that the technology empowers People of Determination, rather than overpowering them.

7. CONCLUSION OF LITERATURE REVIEW

The advancement of ASLR systems represents a significant step towards improving the lives of the Deaf community, ensuring an accessible and inclusive future. By utilising Machine Learning, Computer Vision and Natural Language Processing techniques, such systems can bridge the vast communication gap between signers and non-signers.
Despite the vast benefits of such technologies and recent progress in the field, challenges such as contextual understanding, cultural awareness, computational intensity, dataset biases, privacy concerns and ethical dilemmas continue to plague the system, preventing its widespread adoption.
Ultimately, the future of ASLR lies in the collaboration of AI researchers, linguists, and the Deaf community to create systems that are not only technologically advanced

but also morally correct, user-friendly, and truly beneficial to those who rely on sign language for communication.

8. Hardware and Software Requirements

8.1. Hardware Requirements

Computer with at least 16GB RAM, 256GB Storage and GPU (preferably). These requirements ensure a suitable balance between computational efficiency and cost-effectiveness of resources.

8.2. Software Requirements

- Programming Language: Python 3.11.x or below. (Most libraries, as of January 2025, are not compatible with Python versions higher than 3.11.x)
- Libraries & Frameworks: TensorFlow/Keras, Scikit-learn, Pandas, NumPy, Matplotlib, OpenCV, MediaPipe, TQDM, Python, Fasttext, BERT.
- Development Tools: Jupyter Notebook, VS Code
- Dataset: WLASL Dataset, ASL Sign Recognition Datasets (Kaggle)

9. Overview

This project follows a data-driven approach, utilising publicly available datasets and Deep-learning to create the ASLR system.

Data Collection: Using publicly available trustworthy datasets (e.g., WLASL Dataset, ASL Dataset) or creating a customised dataset.

Preprocessing: Images are assigned landmarks and features using Mediapipe, which are then stored in csv/npz files. Landmarks are mapped carefully to labels. Data quality and variety is ensured.

Model Training: Utilizes LSTM, Logistic Regression and/or CNN deep-learning techniques to train data to achieve highest possible accuracy during implementation while avoiding overfitting.

Real-Time Video Processing: Captures video input dynamically and applies model.

Testing & Validation: Evaluating the model’s accuracy and performance. To test generalisation of the model, it must be tested on different backgrounds, signing speeds and skin tones as well.

Enhancement: Attempt to enhance model using AI , Translate APIs etc. to improve versatility of application.

Application Development: Build a suitable user interface to create a hassle-free experience.

3. SYSTEM DESIGN

3.1 Non-Technical Information

The ASLR system is designed to bridge the ever-growing communication gap between the DHH community and hearing individuals through real-time sign language conversion to text. The system is intended for use in the education sector, in accessibility applications and for general communication between sign language users.

The fundamental operation of this vastly complex system relies upon computer vision and deep learning algorithms to accurately identify and interpret ASL gestures. The goal is to provide a natural experience, allowing users to sign before a camera and have their gestures interpreted accurately and efficiently.

This application targets:

- Deaf and hard-of-hearing individuals to ease communication in everyday life
- Teachers and students for assisting in ASL teaching by providing instant feedback
- Businesses and public services to enhance accessibility for patrons and staff using ASL

3.2 Functional Requirements

The functional requirements of this project are as follows:

- The ASL recognition system must process and understand hand gestures in real-time.
- The user must be able to perform ASL signs in front of a camera, and the system must recognize, classify, and interpret them correctly.
- The system must function effectively in environments different to that of the data the model was trained on.
- Recognized signs must be displayed as text.
- Gesture recognition must be performed based on a live video feed as opposed to pre-recorded videos.
- The system should begin interpreting as soon as user accesses the interface.

- Performance should be optimized to reduce lag between gesture input and recognition output.
- The model must distinguish between visually similar signs based on hand position and movement.
- The user interface should be simple, accessible, and responsive across various devices.
- The application must support future improvements, such as integrating translation APIs for multiple languages and expanding the system's vocabulary.

The ASLR system will be interactive, effective and uncomplicated. The user ought to see their video feed overlaid with the detected signs as text, and have it displayed neatly on the interface in large, bold font for clarity. The system design will ensure smooth gesture detection, with low latency between the video feed and appropriately detected sign.

3.3 Non-Functional Requirements

- The interface must be neat and professional, ensuring accessibility for all users.
- The system must provide real-time feedback in a low-latency environment.
- Recognition accuracy must remain high under different conditions, including lighting, backgrounds, and signer demographics.
- The model should be optimized for speed without compromising accuracy, but overfitting must be avoided at all costs.
- The system should be scalable to allow for future development, like an expanded vocabulary, or the inclusion of animations.

3.4 System Analysis

This section analyses the ASLR system by identifying the mandatory hardware, software, and overall procedure in order to successfully implement the system.

Materials Needed

Hardware:

- A computer with at least 16GB RAM, 256GB storage, and preferably a GPU for smooth processing.

Software:

- **Programming Language:** Python 3.11.x or earlier (ensuring compatibility with most machine-learning libraries).
- **Libraries and Frameworks:**
    - TensorFlow/Keras for deep learning
    - Scikit-learn for machine learning models
    - Pandas and NumPy for data processing
    - Matplotlib for visualization
    - OpenCV for image processing
    - MediaPipe for landmark detection
    - TQDM for progress tracking
- **Development Tools:** VS Code, Jupyter Notebook
- **Dataset:** WLASL dataset, ASL sign recognition datasets from Kaggle, Custom Dataset

WLASL Dataset: The WLASL (Word-Level American Sign Language) dataset is a huge video corpus consisting of thousands of videos of various signers performing ASL signs of words in the English language. The dataset is varied in terms of signer attributes, hand shapes, and signing speeds; however, it does not contain enough samples of each word to create a robust D.L. model.

ASL Sign Recognition Dataset (Kaggle): Provided on Kaggle, this dataset contains annotated images of ASL hand gestures, typically segmented for classification-based systems. Unlike WLASL, which is video-based, this dataset is primarily image-based, capturing discrete ASL hand gestures of letters, words, or phrases, and is ideal for the development of machine learning models, due to a greater number of samples provided for each category.

Custom Dataset: The dataset includes self-recorded ASL gesture videos created for the purposes of this project. It contains signs that weren't to be found in abundance in publicly available datasets, with variation in signing style and positioning. The custom dataset improves model performance by enhancing generalization to different methods of signing. Deep learning and Machine learning architectures like CNN and Logistic Regression are utilized to classify ASL signs with high accuracy. Real-time video processing is handled using OpenCV to ensure smooth interaction between the system and the user.

Future improvements include integrating the Google Translate API for multi-language support, expanding the vocabulary and fine-tuning model performance with external datasets while maintaining a high accuracy.

METHODOLOGY

This section covers how the model was implemented.

Dataset Collection and Preprocessing

Multiple datasets were used and also discarded in the making of this system. This is because a high-quality dataset with a distinct set of requirements is essential for the success of the project.

The requirements are:

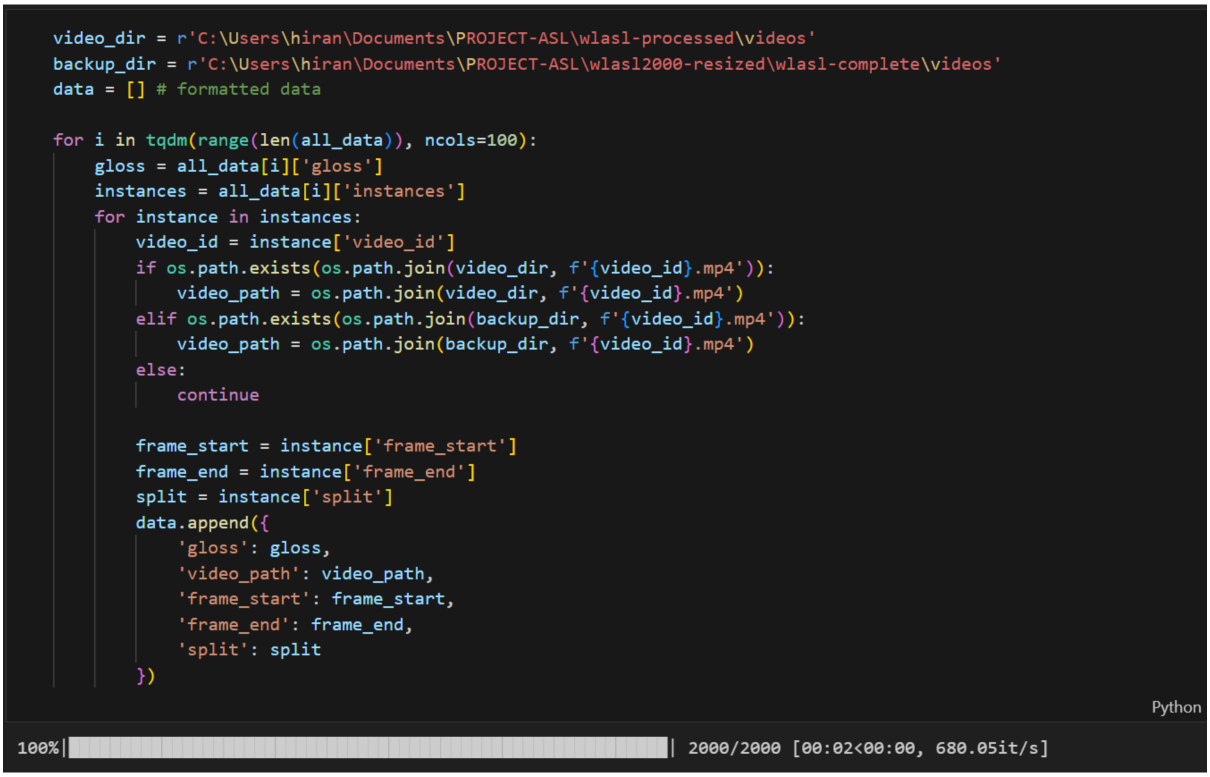
1. Sufficient Sample Size: A lesson learnt the hard way, without a sufficient sample size of at least 30+ samples per category, the model (regardless of its capabilities) will fail to function to its greatest capacity. The dataset must also be highly balanced across all categories to prevent bias, causing the model to learn from the more saturated category better then it would the rest, thereby causing the accuracy to be skewed.
2. High Quality Samples: The samples must be clear and high definition. Regardless of noise and variation, the samples must also clearly display all required landmarks within the camera frame at all times, requiring stable framing.
3. Diverse Data Collection: To always ensure that the system functions efficiently, the model must be trained on a variety of environmental changes. This includes, but is not limited to, slight changes of angles and positioning, lighting conditions and speeds. This ensures model robustness to unseen data, a necessary factor in a real-time system.
4. Ethical and Legal Compliance: With the rapid rise of M.L, D.L and A.I techniques in mainstream society in the past decade, moral dilemmas continue to rise, with legal systems struggling to cope with more lucrative digital solutions. Ownership, terms of use and citation are important to the usage of external datasets during model training, as well as choosing data that is unbiased and possesses no culturally offensive undercurrents.
5. Formats and Compatibility: Data can be structured in a multitude of formats. To ensure ease-of-use and reproducibility of system, it is best if the dataset is stored in structured directories (Dataset/WordName/Sample1.mp4), Compatible with **TensorFlow, PyTorch, OpenCV, or MediaPipe** processing pipelines, in one of the compatible formats like **MP4, AVI, PNG, JPG**, with corresponding metadata (if required), or stored altogether in a comprehensive JSON or CSV file.

The WLASL Dataset

The WLASL Dataset is one of the largest video datasets for American Sign Language Recognition, containing over 2,000 words in the language, being intended for academic and computational purposes only.

The dataset is presented in a Json file, which is then parsed to extract both file names and labels. Said JSON file contains:

- File names of ASL video clips
- Word labels corresponding to each video
- Metadata (e.g., signer ID, video duration, frame rate)



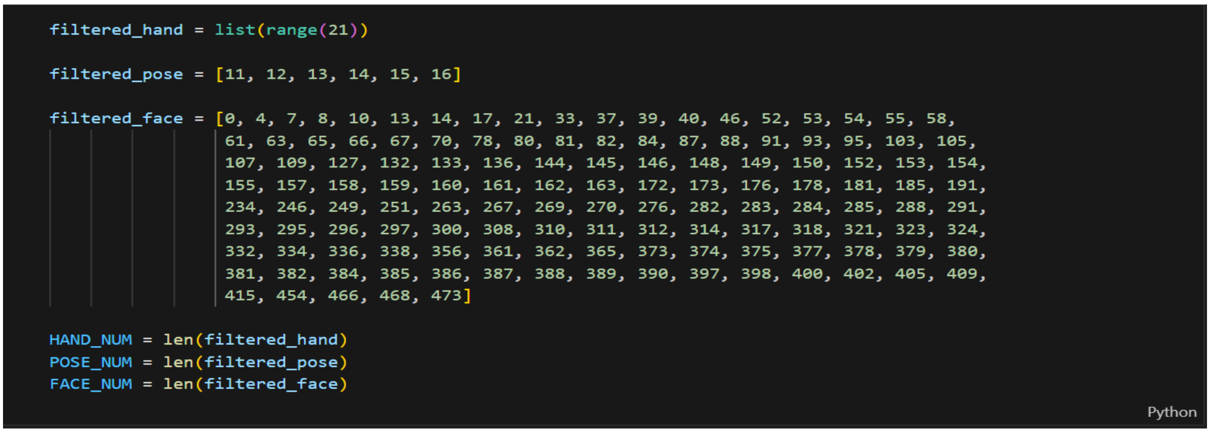
```python
video_dir = r'C:\Users\hiran\Documents\PROJECT-ASL\wlasl-processed\videos'
backup_dir = r'C:\Users\hiran\Documents\PROJECT-ASL\wlasl2000-resized\wlasl-complete\videos'
data = [] # formatted data

for i in tqdm(range(len(all_data)), ncols=100):
    gloss = all_data[i]['gloss']
    instances = all_data[i]['instances']
    for instance in instances:
        video_id = instance['video_id']
        if os.path.exists(os.path.join(video_dir, f'{video_id}.mp4')):
            video_path = os.path.join(video_dir, f'{video_id}.mp4')
        elif os.path.exists(os.path.join(backup_dir, f'{video_id}.mp4')):
            video_path = os.path.join(backup_dir, f'{video_id}.mp4')
        else:
            continue

        frame_start = instance['frame_start']
        frame_end = instance['frame_end']
        split = instance['split']
        data.append({
            'gloss': gloss,
            'video_path': video_path,
            'frame_start': frame_start,
            'frame_end': frame_end,
            'split': split
        })
```

```
100%|████████████████████████████████████████| 2000/2000 [00:02<00:00, 680.05it/s]
```

Figure 6: JSON file parsing and data extraction

The parsed data is saved in an organized list.

We use Mediapipe to track the movement of hands, face, elbows and shoulders. The landmarks to be recorded are chosen carefully, to not overload the system but still have enough to detect the words accurately. 42 landmarks are chosen on the hand, as they are the most critical to the accuracy of the system. 6 landmarks are chosen for the upper body, excluding the face. Out of 478 total landmarks upon the face, 132 are chosen, focusing on the lips, eyes, eyebrows and the outline of the face. This allows for future increments of the system’s functional vocabulary. The total landmarks are therefore 180, each with x, y and z coordinates.



```python
filtered_hand = list(range(21))

filtered_pose = [11, 12, 13, 14, 15, 16]

filtered_face = [0, 4, 7, 8, 10, 13, 14, 17, 21, 33, 37, 39, 40, 46, 52, 53, 54, 55, 58,
                 61, 63, 65, 66, 67, 70, 78, 80, 81, 82, 84, 87, 88, 91, 93, 95, 103, 105,
                 107, 109, 127, 132, 133, 136, 144, 145, 146, 148, 149, 150, 152, 153, 154,
                 155, 157, 158, 159, 160, 161, 162, 163, 172, 173, 176, 178, 181, 185, 191,
                 234, 246, 249, 251, 263, 267, 269, 270, 276, 282, 283, 284, 285, 288, 291,
                 293, 295, 296, 297, 300, 308, 310, 311, 312, 314, 317, 318, 321, 323, 324,
                 332, 334, 336, 338, 356, 361, 362, 365, 373, 374, 375, 377, 378, 379, 380,
                 381, 382, 384, 385, 386, 387, 388, 389, 390, 397, 398, 400, 402, 405, 409,
                 415, 454, 466, 468, 473]

HAND_NUM = len(filtered_hand)
POSE_NUM = len(filtered_pose)
FACE_NUM = len(filtered_face)
```

Figure 7: Extracted features (180 in total)

The above landmarks are then extracted from the frames of each video, there being 30 frames per video.

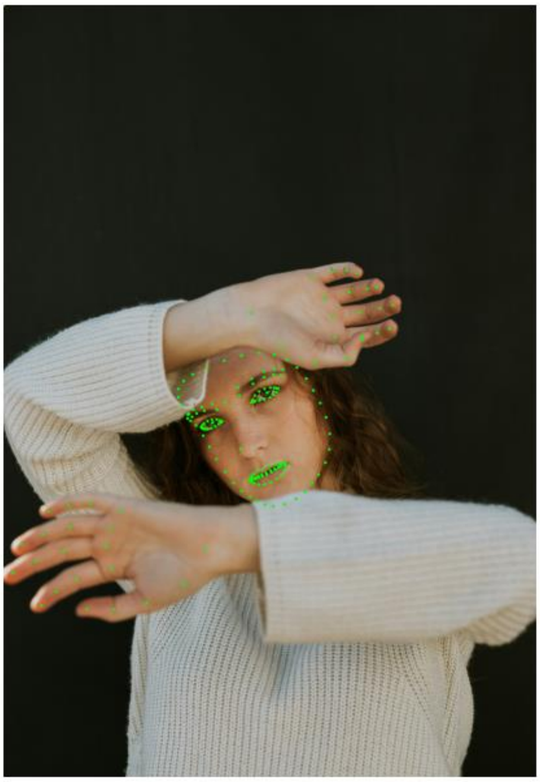
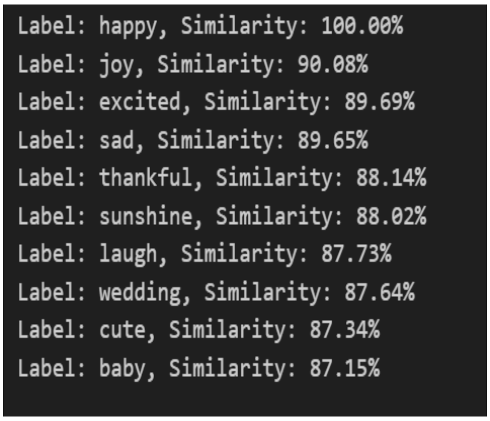


Figure 8: Visual representation of extracted features

The data is then encoded and saved in a npz file. Using a tokenizer (BERT proved to have the highest vocabulary variety), the nearest neighbours of certain words are calculated to a percentage through cosine similarities.



```
Label: happy, Similarity: 100.00%
Label: joy, Similarity: 90.08%
Label: excited, Similarity: 89.69%
Label: sad, Similarity: 89.65%
Label: thankful, Similarity: 88.14%
Label: sunshine, Similarity: 88.02%
Label: laugh, Similarity: 87.73%
Label: wedding, Similarity: 87.64%
Label: cute, Similarity: 87.34%
Label: baby, Similarity: 87.15%
```

Figure 10: BERT tokenizer word similarity

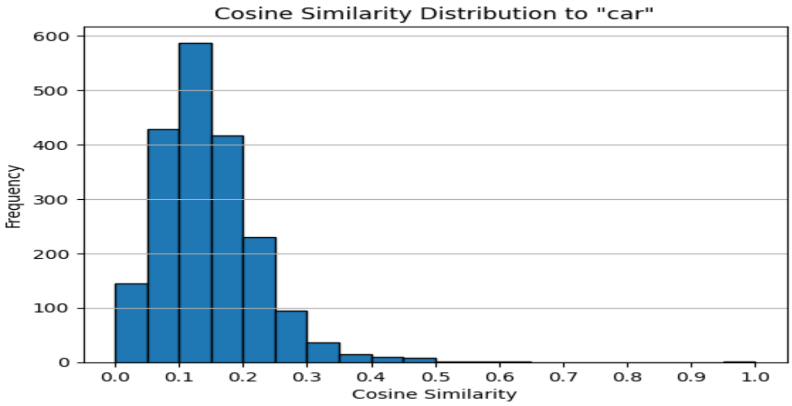
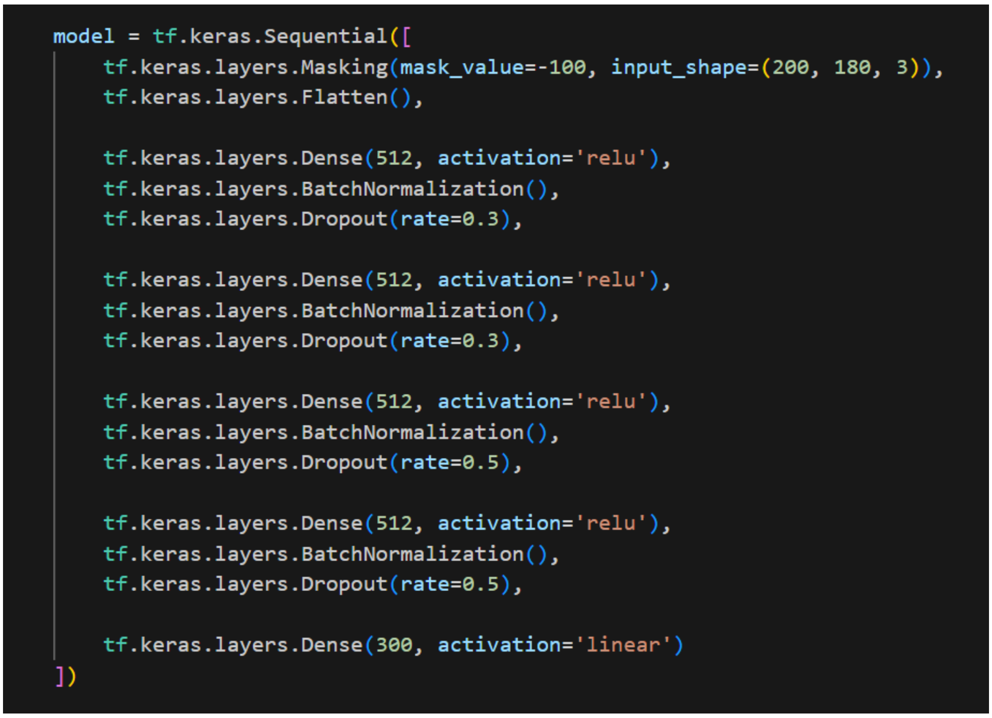


Figure 9: Work similarity through Cosine: Graph

Augmentations are then applied in order to ensure robustness of the dataset. These included scaling the landmarks randomly, moving landmarks horizontally and vertically, rotating around the x, y and z axes individually, masking random landmarks, mirroring landmarks left to right, and reducing frame count by skipping every second frame.

The data is then split into training, test and validation sets. As models work better with encoded variables, rather than alphabetical labels, the words are encoded into a numerical format.

A model is then created using deep learning techniques. Early stopping as validation accuracy plateaus, learning rate schedulers to modify the rate automatically to improve model working, and checkpoints are implemented in the code.



```python
model = tf.keras.Sequential([
    tf.keras.layers.Masking(mask_value=-100, input_shape=(200, 180, 3)),
    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(rate=0.3),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(rate=0.3),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(rate=0.5),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(rate=0.5),

    tf.keras.layers.Dense(300, activation='linear')
])
```

Figure 11: Sequential model used

The model working is then plotted. As seen below, the accuracy plateaus at around 46%.

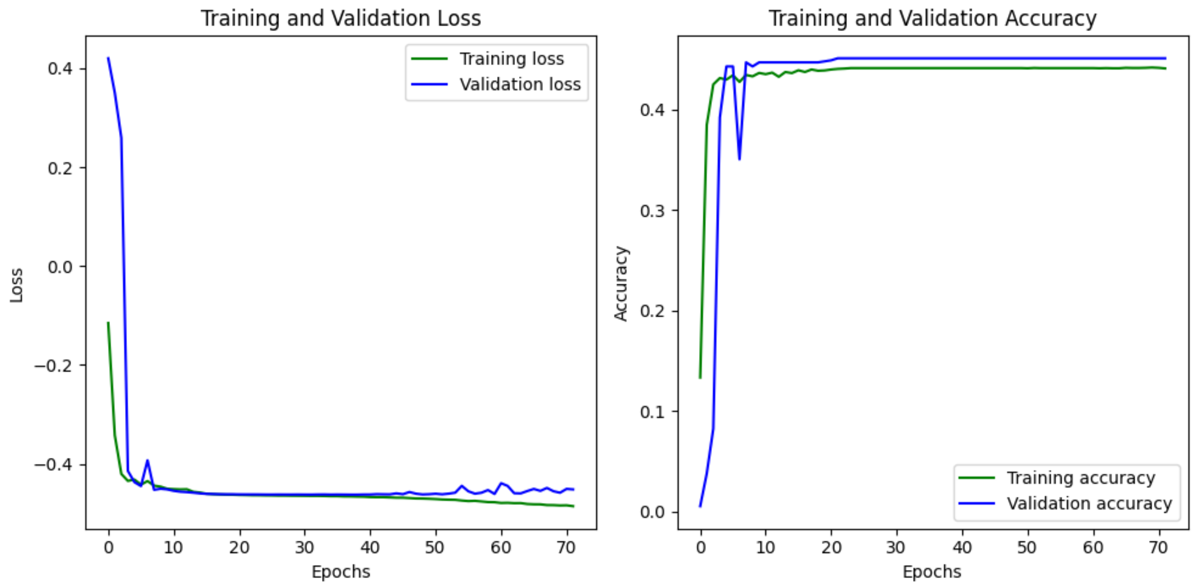
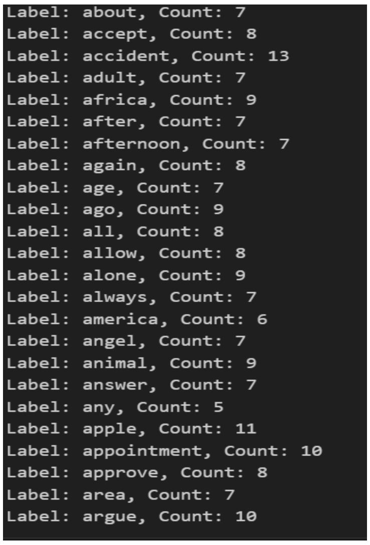


Figure 12: Graph of results

Considering the size of the dataset, such an accuracy is only to be expected. However, when a model is implemented onto a real-time system, it is expected to perform at a higher accuracy in order to cope with the difficulties of video-based detection, and an accuracy below 50% is considered inadequate for the project.

The accuracy cannot be blamed on the model that produced it; rather, it was the incapabilities of the dataset trained on that caused it. Namely, the inconsistency in the number of samples for each label as well as their subpar quality.



```
Label: about, Count: 7
Label: accept, Count: 8
Label: accident, Count: 13
Label: adult, Count: 7
Label: africa, Count: 9
Label: after, Count: 7
Label: afternoon, Count: 7
Label: again, Count: 8
Label: age, Count: 7
Label: ago, Count: 9
Label: all, Count: 8
Label: allow, Count: 8
Label: alone, Count: 9
Label: always, Count: 7
Label: america, Count: 6
Label: angel, Count: 7
Label: animal, Count: 9
Label: answer, Count: 7
Label: any, Count: 5
Label: apple, Count: 11
Label: appointment, Count: 10
Label: approve, Count: 8
Label: area, Count: 7
Label: argue, Count: 10
```

Figure 13: Dataset inconsistencies

The videos were grainy and required an excessive quantity of resources and time to be converted into a high-definition version (~ 4 days), which did not seem feasible for the project at hand and therefore was not undertaken.

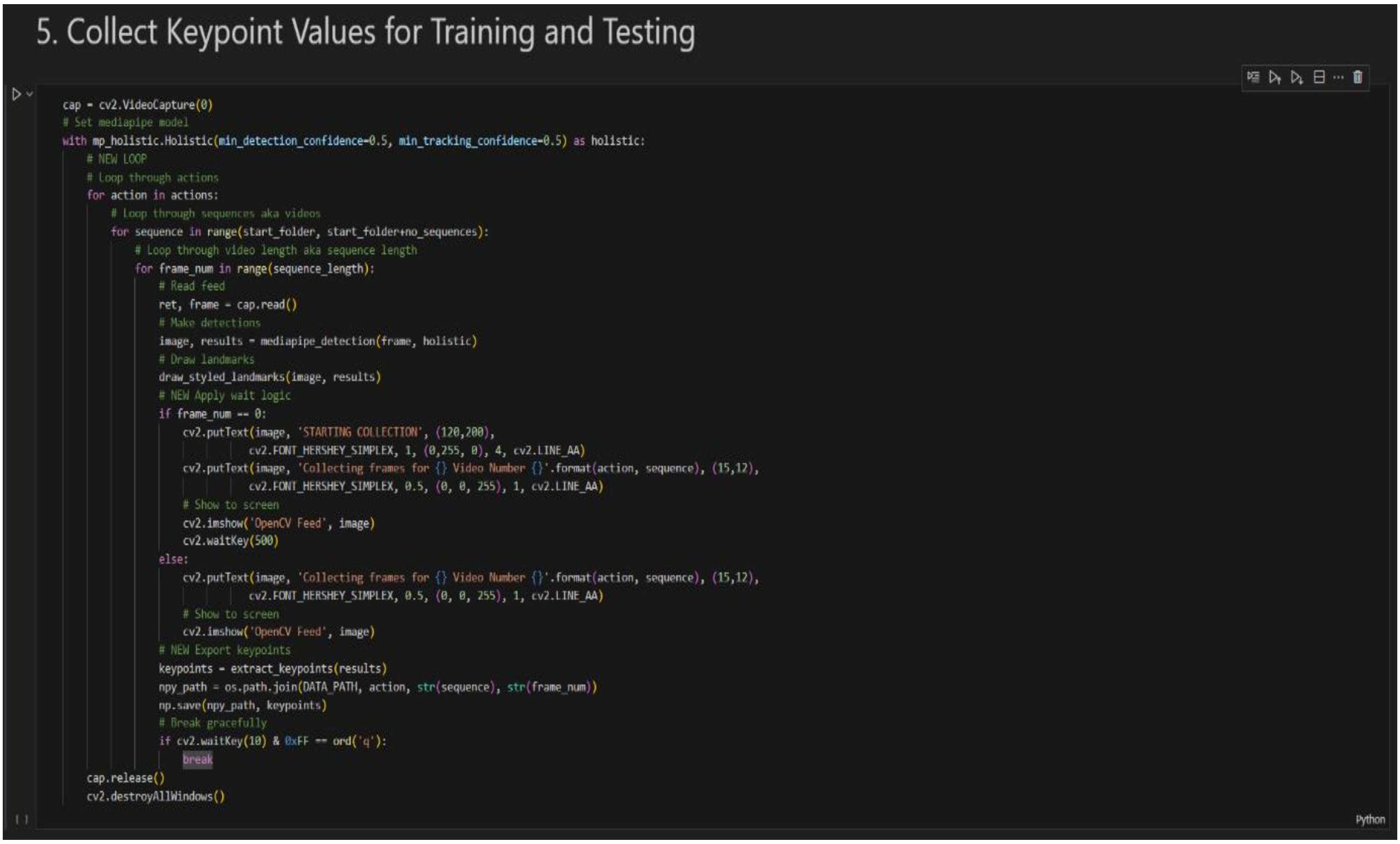
To conclude, while the WLASL dataset provides a solid foundation for computational research and analysis in American Sign Language, its inconsistencies in sample distribution and video quality significantly impacted model performance, making it unsuitable for the project at hand.

Such significant disadvantages also prevented the creation of a deep learning model with a large vocabulary, seeing as only 2 words in the entire dataset possessed over 30 samples. Therefore, in order to create a word recognition software, a custom dataset was created.

Custom Dataset

Words

Using the very same 180 landmarks through Mediapipe's Holistic model, each word is recorded in 30 sequences, each sequence containing 30 frames. The dataset is then structured and stored hierarchically, and landmarks from each frame are stored as individual NumPy files (.npy).



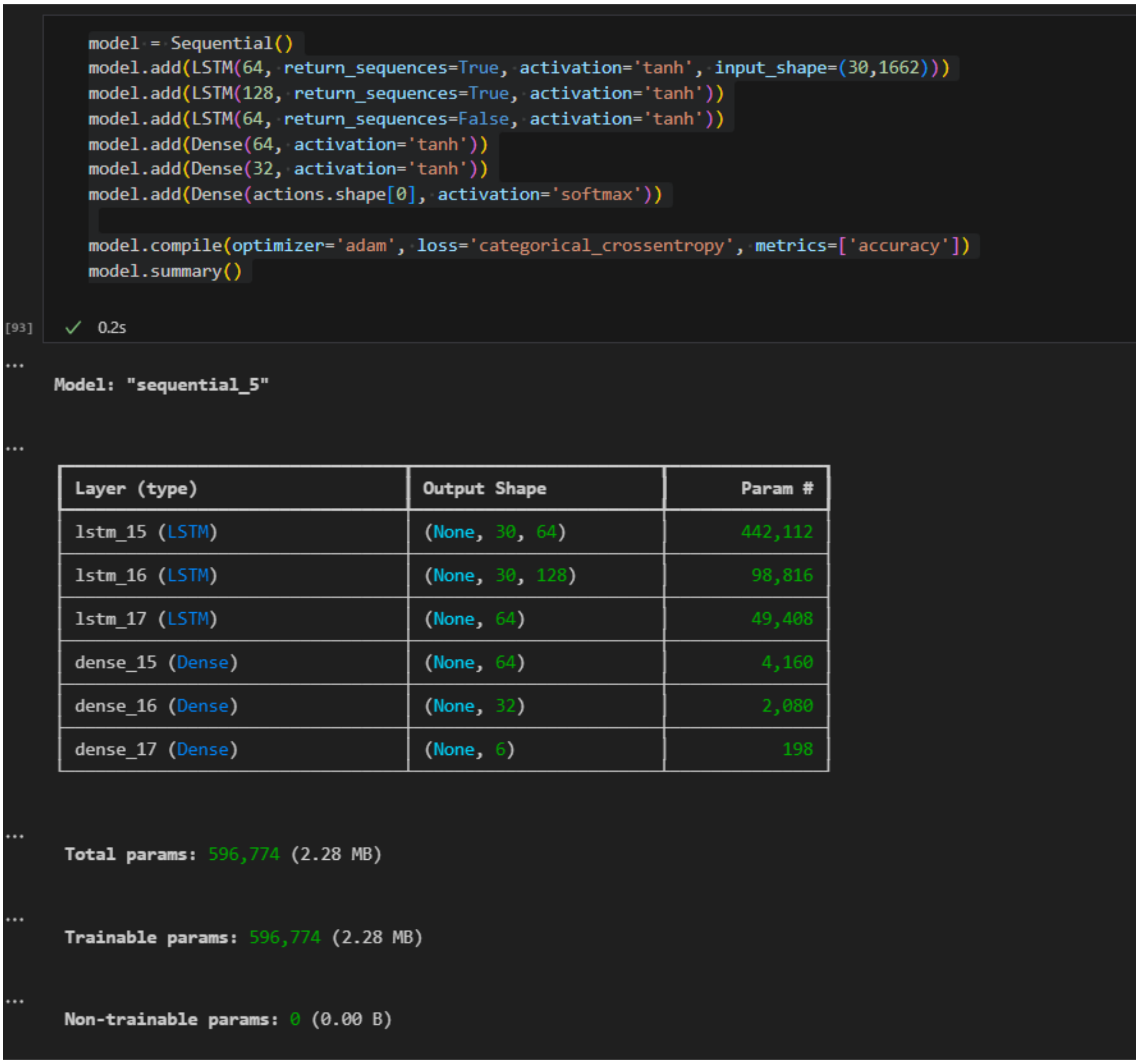
5. Collect Keypoint Values for Training and Testing

```python
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence in range(start_folder, start_folder+no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):
                # Read feed
                ret, frame = cap.read()
                # Make detections
                image, results = mediapipe_detection(frame, holistic)
                # Draw landmarks
                draw_styled_landmarks(image, results)
                # NEW Apply wait logic
                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(500)
                else:
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)
                # NEW Export keypoints
                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)
                # Break gracefully
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
    cap.release()
    cv2.destroyAllWindows()
```

Figure 14: Custom Dataset creation

Thereafter, the data is pre-processed without the excessive augmentation that the WLASL dataset required, the custom data being high-definition and consistent in sample quantity. Labels are converted into a one-hot encoded format, being easier to process, and data is divided into a train and test set.

A sequential model is then created using LSTM (Long Short-Term Network), a type of recurrent neural network (RNN) designed to handle sequential data by maintaining long-term dependencies. Unlike traditional RNNs, LSTMs use **gates (input, forget, output)** to retain or discard information over time, which is essential for recognizing sequential patterns in sign language. Further, the model was trained with early stopping to prevent overfitting and checkpointing to save the best weights.

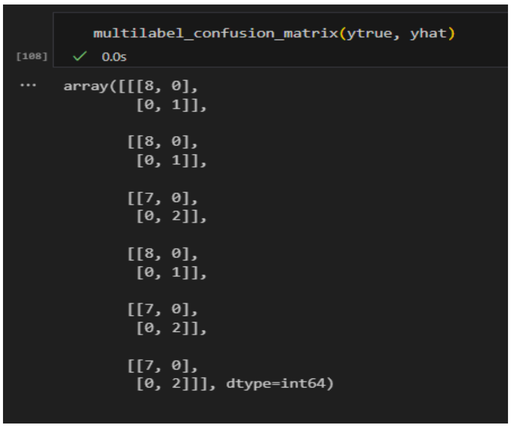


```
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='tanh', input_shape=(30,1662)))
model.add(LSTM(128, return_sequences=True, activation='tanh'))
model.add(LSTM(64, return_sequences=False, activation='tanh'))
model.add(Dense(64, activation='tanh'))
model.add(Dense(32, activation='tanh'))
model.add(Dense(actions.shape[0], activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
[93]  ✓ 0.2s

Model: "sequential_5"

┌──────────────────────────┬────────────────────┬────────────┐
│ Layer (type)             │ Output Shape       │    Param # │
├──────────────────────────┼────────────────────┼────────────┤
│ lstm_15 (LSTM)           │ (None, 30, 64)     │    442,112 │
├──────────────────────────┼────────────────────┼────────────┤
│ lstm_16 (LSTM)           │ (None, 30, 128)    │     98,816 │
├──────────────────────────┼────────────────────┼────────────┤
│ lstm_17 (LSTM)           │ (None, 64)         │     49,408 │
├──────────────────────────┼────────────────────┼────────────┤
│ dense_15 (Dense)         │ (None, 64)         │      4,160 │
├──────────────────────────┼────────────────────┼────────────┤
│ dense_16 (Dense)         │ (None, 32)         │      2,080 │
├──────────────────────────┼────────────────────┼────────────┤
│ dense_17 (Dense)         │ (None, 6)          │        198 │
└──────────────────────────┴────────────────────┴────────────┘

 Total params: 596,774 (2.28 MB)

 Trainable params: 596,774 (2.28 MB)

 Non-trainable params: 0 (0.00 B)
```

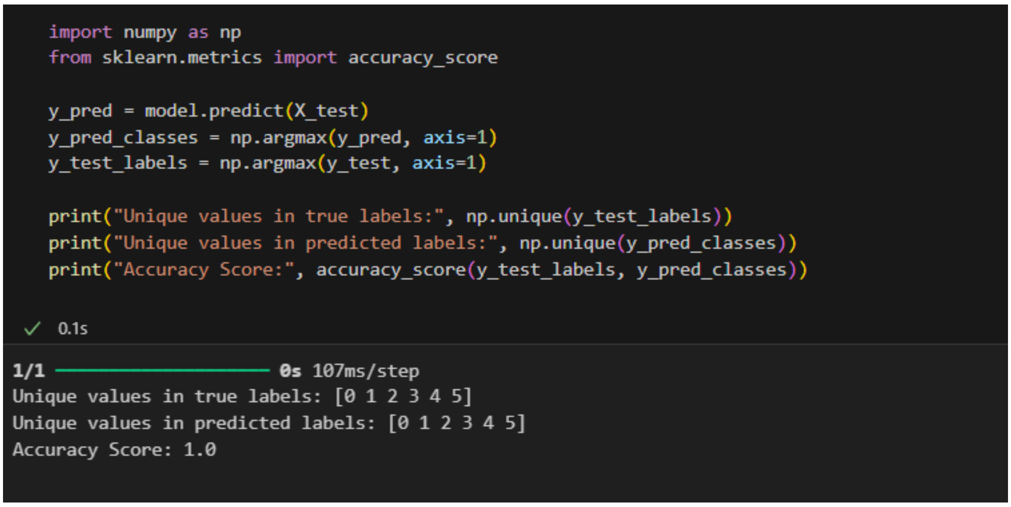
Figure 15: LSTM model utilized

The model is then duly compiled using a categorical cross entropy loss function and an Adam optimizer. After fitting, the final accuracy plateaus at 88.5%. Upon creating a confusion matrix, a suitable result is obtained, indicating an overall above-average performance of the model.



```
[108]  multilabel_confusion_matrix(ytrue, yhat)
       ✓ 0.0s

...  array([[[8, 0],
             [0, 1]],

            [[8, 0],
             [0, 1]],

            [[7, 0],
             [0, 2]],

            [[8, 0],
             [0, 1]],

            [[7, 0],
             [0, 2]],

            [[7, 0],
             [0, 2]]], dtype=int64)
```

Figure 16: Multiclass Confusion Matrix



```
import numpy as np
from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_labels = np.argmax(y_test, axis=1)

print("Unique values in true labels:", np.unique(y_test_labels))
print("Unique values in predicted labels:", np.unique(y_pred_classes))
print("Accuracy Score:", accuracy_score(y_test_labels, y_pred_classes))

✓ 0.1s

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step
Unique values in true labels: [0 1 2 3 4 5]
Unique values in predicted labels: [0 1 2 3 4 5]
Accuracy Score: 1.0
```

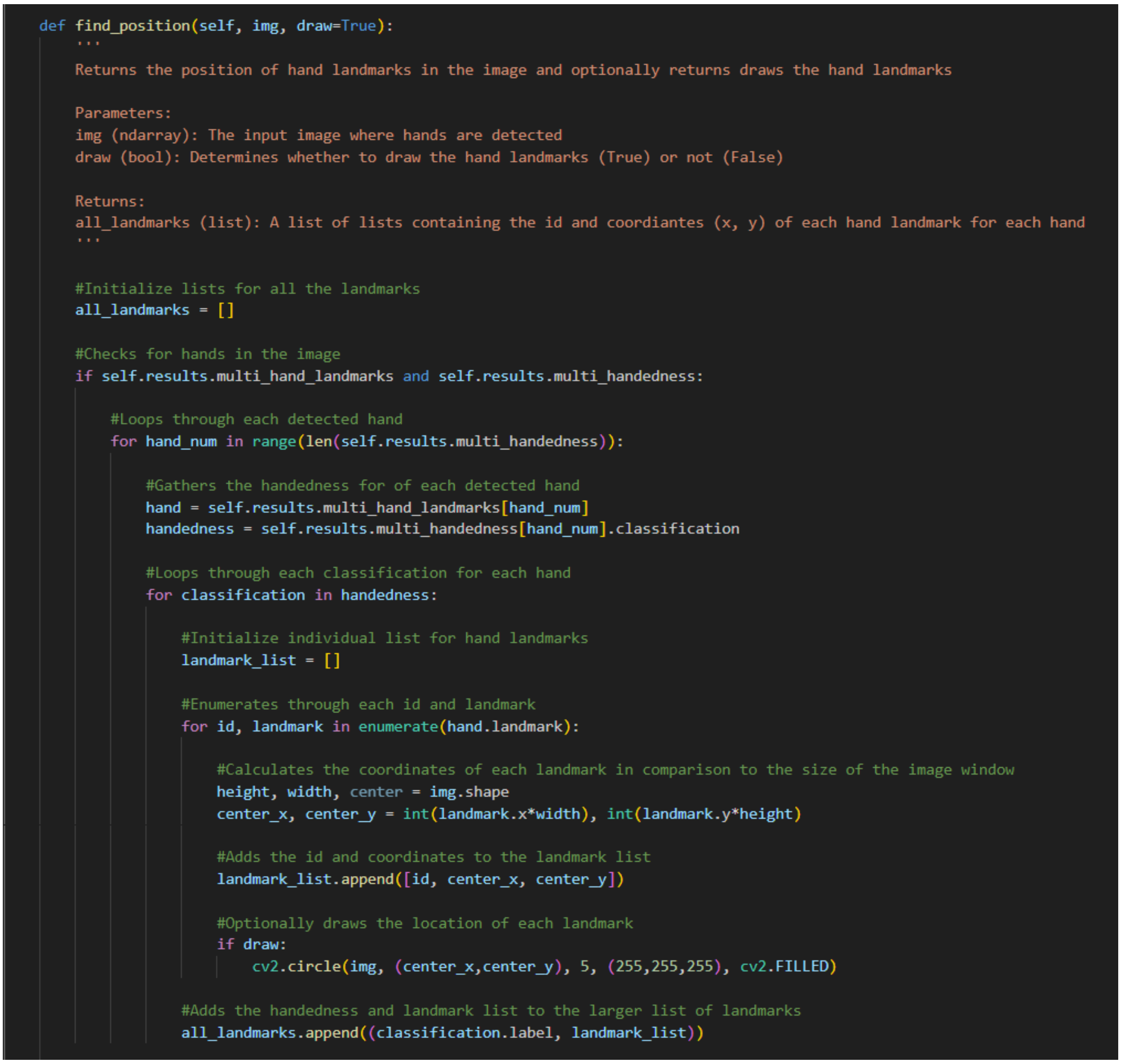
Figure 17: Accuracy Score of 100%

Evidently, the model's performance is optimal, displaying a high accuracy without overfitting. However, such behaviour comes at a price. The model currently functions with only a few words, and the RNN's outcome when the model is scaled to include 20+ words is questionable. Each word must be manually recorded, which allows for customizable inclusion of environmental factors and noise, which could help improve robustness of the model. Currently, the data includes a change in posture and position of hands every second frame, allowing for variability. Manually recording presents challenges to scalability and is extremely time-intensive, especially considering that there are over 200+ sign languages across the world, each with their own dynamically changing vocabulary. There is also the issue of latency that arises when discussing and implementing a real-time system. ASL movements are often rapid, mimicking spoken speech patterns; and latency could be crippling to communication were it to persist when deployed publicly. Another issue is the similarity between hand gestures of certain words, at times only distinguished through posture or facial movements, posing a source of confusion to the model and affecting the accuracy.

In order to expand the scope of the project, a collective effort must be expected from the DHH community, developers and researchers to create a system that recognizes and translates a variety of different sign languages with their respectively extensive dictionaries, fully considering the changes to word structure and semantics when interpreting.

Letters

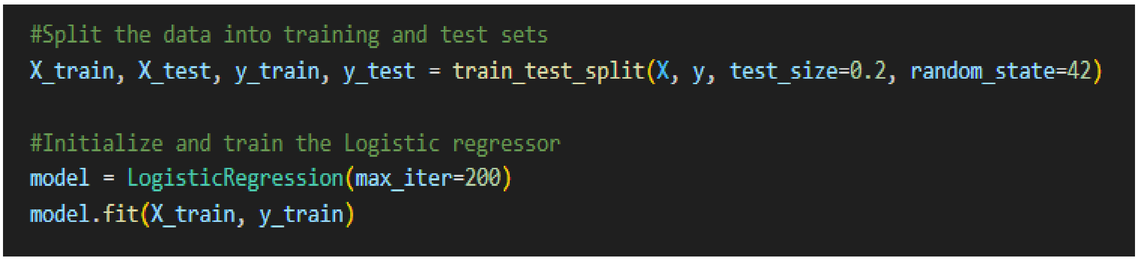
The system was further accommodated to include letter interpretation. Fingerspelling is an integral part of ASL communication, often used for proper nouns, uncommon words and teaching purposes. 21 landmarks for each hand were recorded using Mediapipe and OpenCV, a computer vision library. The landmarks are then stored in lists with their respective labels in an .csv file, which is later referred to when the model is fitted.



```
    def find_position(self, img, draw=True):
        '''
        Returns the position of hand landmarks in the image and optionally returns draws the hand landmarks

        Parameters:
        img (ndarray): The input image where hands are detected
        draw (bool): Determines whether to draw the hand landmarks (True) or not (False)

        Returns:
        all_landmarks (list): A list of lists containing the id and coordiantes (x, y) of each hand landmark for each hand
        '''

        #Initialize lists for all the landmarks
        all_landmarks = []

        #Checks for hands in the image
        if self.results.multi_hand_landmarks and self.results.multi_handedness:

            #Loops through each detected hand
            for hand_num in range(len(self.results.multi_handedness)):

                #Gathers the handedness for of each detected hand
                hand = self.results.multi_hand_landmarks[hand_num]
                handedness = self.results.multi_handedness[hand_num].classification

                #Loops through each classification for each hand
                for classification in handedness:

                    #Initialize individual list for hand landmarks
                    landmark_list = []

                    #Enumerates through each id and landmark
                    for id, landmark in enumerate(hand.landmark):

                        #Calculates the coordinates of each landmark in comparison to the size of the image window
                        height, width, center = img.shape
                        center_x, center_y = int(landmark.x*width), int(landmark.y*height)

                        #Adds the id and coordinates to the landmark list
                        landmark_list.append([id, center_x, center_y])

                        #Optionally draws the location of each landmark
                        if draw:
                            cv2.circle(img, (center_x,center_y), 5, (255,255,255), cv2.FILLED)

                    #Adds the handedness and landmark list to the larger list of landmarks
                    all_landmarks.append((classification.label, landmark_list))
```

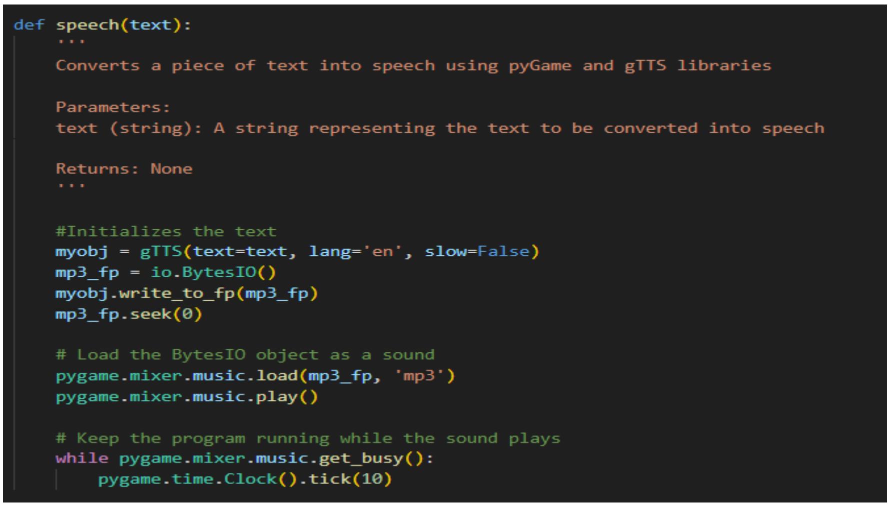
Figure 18: Letter Recognition system (Data extraction)

200 iterations of a basic Scikit-learn-based Logistic Regression model is implemented on the dataset, establishing the confidence threshold of the model for letter detection at 70%, along with a text-to-speech library that allows for the letters interpreted to be spoken aloud. The model is also highly scalable, as it allows for new letters to be added upon pressing the letter ‘c’ on the keyboard. This permits the expansion of the model’s vocabulary to other languages in given time.



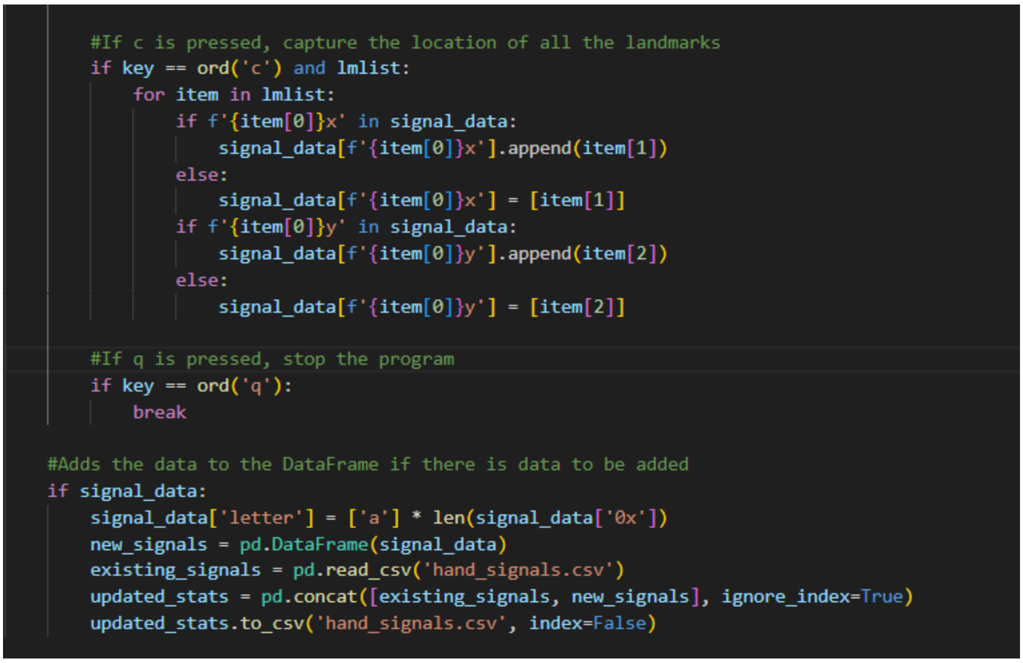
```
#Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Initialize and train the Logistic regressor
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
```

Figure 19: Usage of Logistic Regression model



```
def speech(text):
    '''
    Converts a piece of text into speech using pyGame and gTTS libraries

    Parameters:
    text (string): A string representing the text to be converted into speech

    Returns: None
    '''

    #Initializes the text
    myobj = gTTS(text=text, lang='en', slow=False)
    mp3_fp = io.BytesIO()
    myobj.write_to_fp(mp3_fp)
    mp3_fp.seek(0)

    # Load the BytesIO object as a sound
    pygame.mixer.music.load(mp3_fp, 'mp3')
    pygame.mixer.music.play()

    # Keep the program running while the sound plays
    while pygame.mixer.music.get_busy():
        pygame.time.Clock().tick(10)
```

Figure 20: Google Text-to-Speech (gTTs) software integration



```
    #If c is pressed, capture the location of all the landmarks
    if key == ord('c') and lmlist:
        for item in lmlist:
            if f'{item[0]}x' in signal_data:
                signal_data[f'{item[0]}x'].append(item[1])
            else:
                signal_data[f'{item[0]}x'] = [item[1]]
            if f'{item[0]}y' in signal_data:
                signal_data[f'{item[0]}y'].append(item[2])
            else:
                signal_data[f'{item[0]}y'] = [item[2]]

    #If q is pressed, stop the program
    if key == ord('q'):
        break

#Adds the data to the DataFrame if there is data to be added
if signal_data:
    signal_data['letter'] = ['a'] * len(signal_data['0x'])
    new_signals = pd.DataFrame(signal_data)
    existing_signals = pd.read_csv('hand_signals.csv')
    updated_stats = pd.concat([existing_signals, new_signals], ignore_index=True)
    updated_stats.to_csv('hand_signals.csv', index=False)
```

Figure 21: Flexibility to append to dataset in future

Logistic Regression was chosen for the fingerspelling model due to its simplicity, efficiency and effectiveness in multi-class classification of the English alphabet. Given that the dataset includes numerical landmark coordinates for each of the distinct letters, the model works well to learn the decision boundaries between each class accurately. Although logistic regression models are quite lightweight, upon thorough trials conducted using multiple other models (CNN, RNN, LSTM), logistic regression provided the shortest training period, as well as the highest accuracy without overfitting, making it highly practical for real-time applications like the ASL interpreter on hand. The low computational cost also allows the system's smooth integration into a web application without compromising performance.

Although scalability and manual data-collection remain a challenge, due to the nature of the data collected (letters, and therefore limited in quantity), the problem is not nearly as extensive as with words. The system effectively tracks various hand gestures, and can prove highly useful in teaching environments, as letters are often used to begin sign language education.

Challenges and Future Improvements

1. Latency Issues

One of the major challenges while implementing ASLR systems is latency. ASL gestures are often executed in rapid succession, requiring the system to detect landmarks, process and interpret them in a very short span of time. Delays in recognition can disrupt communication, making the system impractical for real-world use. The computational burden of deep-learning models, namely LSTMs, contributes to this latency, as sequential processing, while suited to the task at hand, is inherently slower than parallel processing. Optimizing inference speed is possible through hardware acceleration (GPU/TPU optimization), model quantization and/or Edge AI deployments.

2. Uneven Data Distribution

The WLASL dataset, while ideal in terms of vocabulary variety, suffers from an imbalanced sample distribution as well as low-definition video collection. This creates a bias in model learning, where words with a higher number of samples are recognized better than those with fewer examples. The video quality also makes the system more computationally expensive to process and interpret. Addressing both issues is labour-intensive and time-consuming, thereby reducing the scope of availability of such a system to only a privileged few who may be able to afford the extensive resources required by such a system, thereby entirely defeating the purpose of a project that facilitates communication between all walks of life. The inabilities of the dataset may perhaps be conquered by data augmentation techniques which have been implemented above, but by no means replaces the need for an evenly distributed, high-definition dataset.

3. Expanding Vocabulary vs. Accuracy

Scalability remains an issue with the model implemented thus far to recognize words. As more labels are added, the per-word accuracy decreases due to the increasing complexity and overlap of certain gestures. Some ASL signage differs by subtle facial expressions or minor hand variations, making classification harder. Perhaps a solution could be the implementation of a hierarchical classification system, where basic hand shapes are recognized first before refining the interpretation. The workings of such a system considering environmental noise, semantics and word-structure, and latency remains to be explored.

4. Manual Dataset Cultivation

Due to the lack of satisfactory ASL datasets insofar, many words and gestures were required to be manually recorded, making the process of dataset cultivation incredibly time-intensive and resource-heavy, seeing as robustness mandates diversity in signers, positions, angles and lighting. Unlike spoken language datasets which may be sourced from text transcriptions or audio files, sign language datasets require video recordings, annotations and landmark extractions, making generalization and dataset creation a herculean task.

5. Transformer-Based Models for Future Research

While LSTMs perform well in sequence modelling, newer Transformer-based architectures could potentially be considered, such as **Vision Transformers (ViTs) and Temporal Convolutional Networks (TCNs)**, in order to improve accuracy. Unlike LSTMs, Transformers process entire sequences simultaneously rather than sequentially, allowing faster inference and potentially higher accuracy. Nonetheless, such a model remained unused due to the high computational power and large datasets required for its working. Their self-attention mechanism scales poorly with longer sequences like ASL features, making LSTMs a more feasible choice. Future work may explore efficient transformers with more optimized inference techniques.

USER INTERFACE

The system will be exhibited in two configurations, a word interpreter and a letter interpreter. Each requires a distinct GUI that facilitates efficient communication and allows for a smooth user experience, an integral part of any application.

There are multiple ways in which a deep learning model could be displayed publicly, and after attempting a majority of them, the below methods were chosen due to their simplicity and resource-effectiveness.

Word Interpreter

The word interpreter has been constructed using the aforementioned LSTM model, and functions with an accuracy rate of 88.5%. In order to display the model’s functioning in a neat, straightforward manner, a Tkinter GUI has been employed. Tkinter is Python's standard GUI toolkit, permitting the creation of simple interfaces using minimal code.

An essential component of a productive ASLR system is fluid recognition without excessive latency. In order to prevent such buffers, the system processes keyframes at strategic intervals rather than analysing every frame, thus reducing computational overhead while preserving overall accuracy. Additionally, a context-aware filtering approach is employed to prevent abrupt changes in predictions, ensuring that the detected words are displayed in a stable and readable manner. This implies that the code ensure that a word is only displayed if it were predicted continuously for 10 frames, and a word is only appended if it differs from the word before it, implicitly preventing model inference buffering. The function also calls itself every 10ms, ensuring continuous processing without rapidly overloading the computer vision program. The displayed word is determined through more than just frame-by-frame analysis of model accuracy. The probability of each word being the one signed is calculated individually and displayed upon dynamically adjusting bars to the UI's left. With the threshold set at 0.5, each word is assigned a colour of its own and the detected word is displayed at the bottom up to 5 words at a given time. Although this makes the model highly computationally ineffective, it is transparent and encourages user trust in the model's ability to predict words. It is also useful for debugging purposes, encouraging a solid understanding of the model's confidence in its interpretations. It is also helpful for learners, as it assists them in perfecting their hand motions and allows them to notice their source of error.

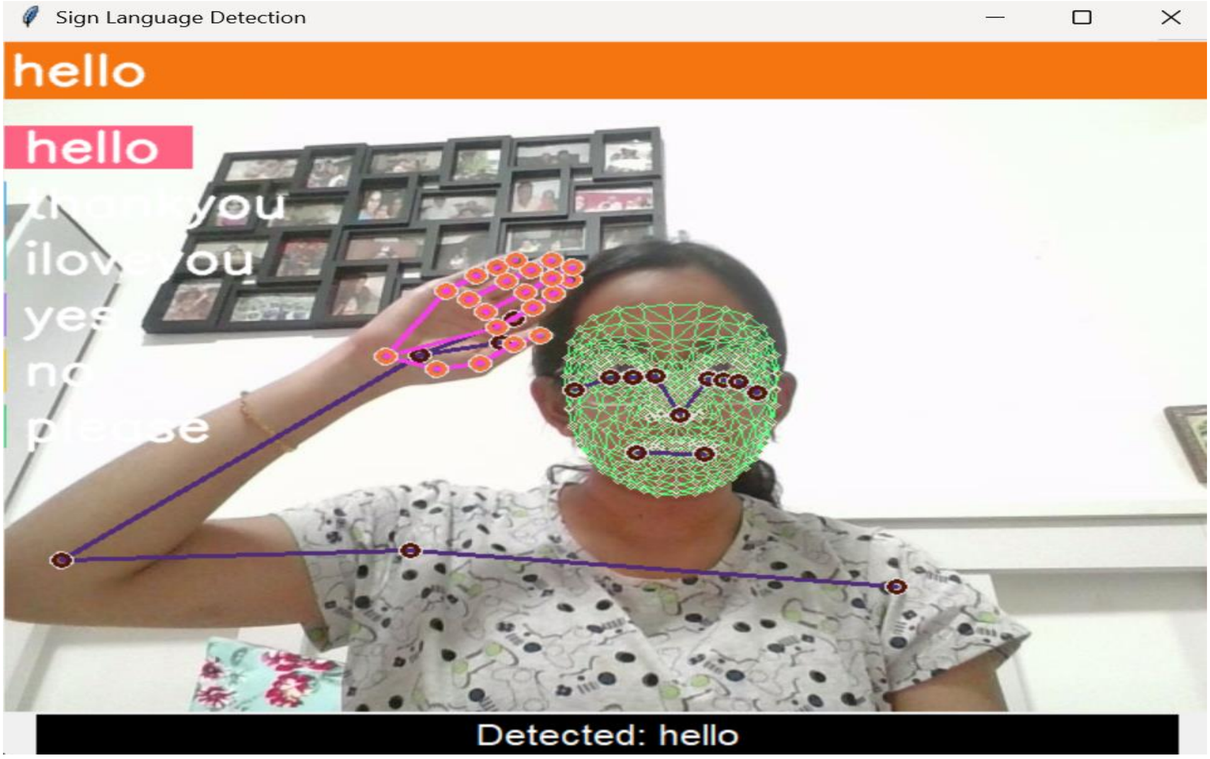
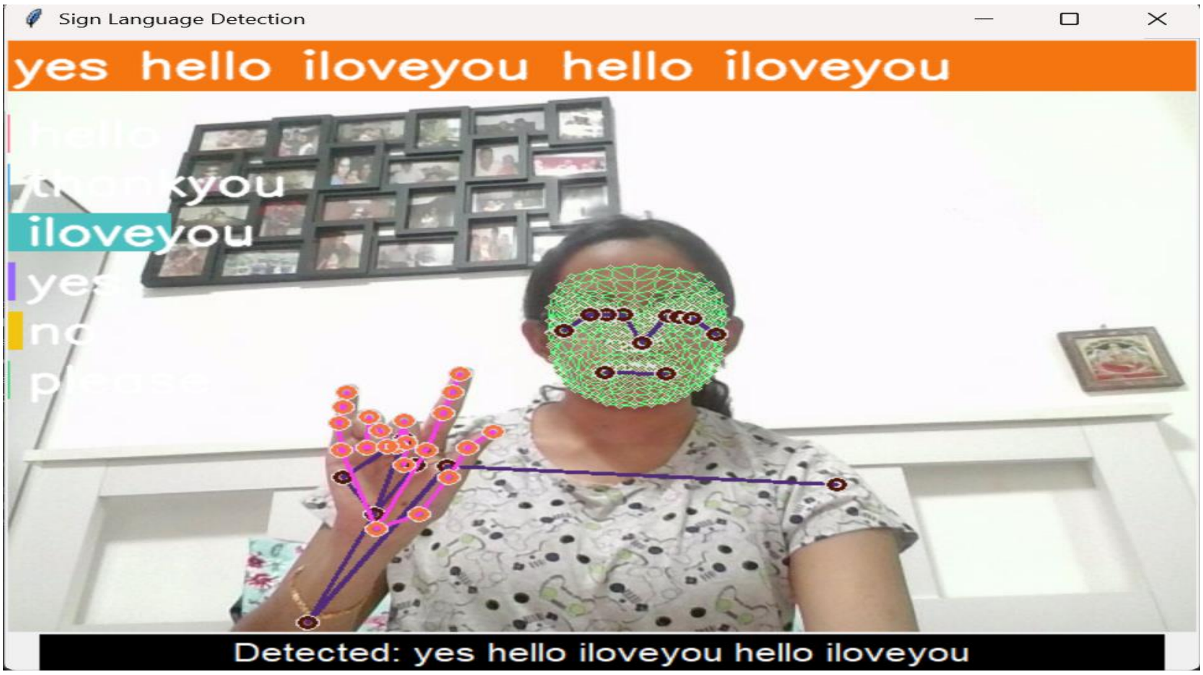


Figure 22: Word recognition of "Hello" in Tkinter GUI



yes hello iloveyou hello iloveyou

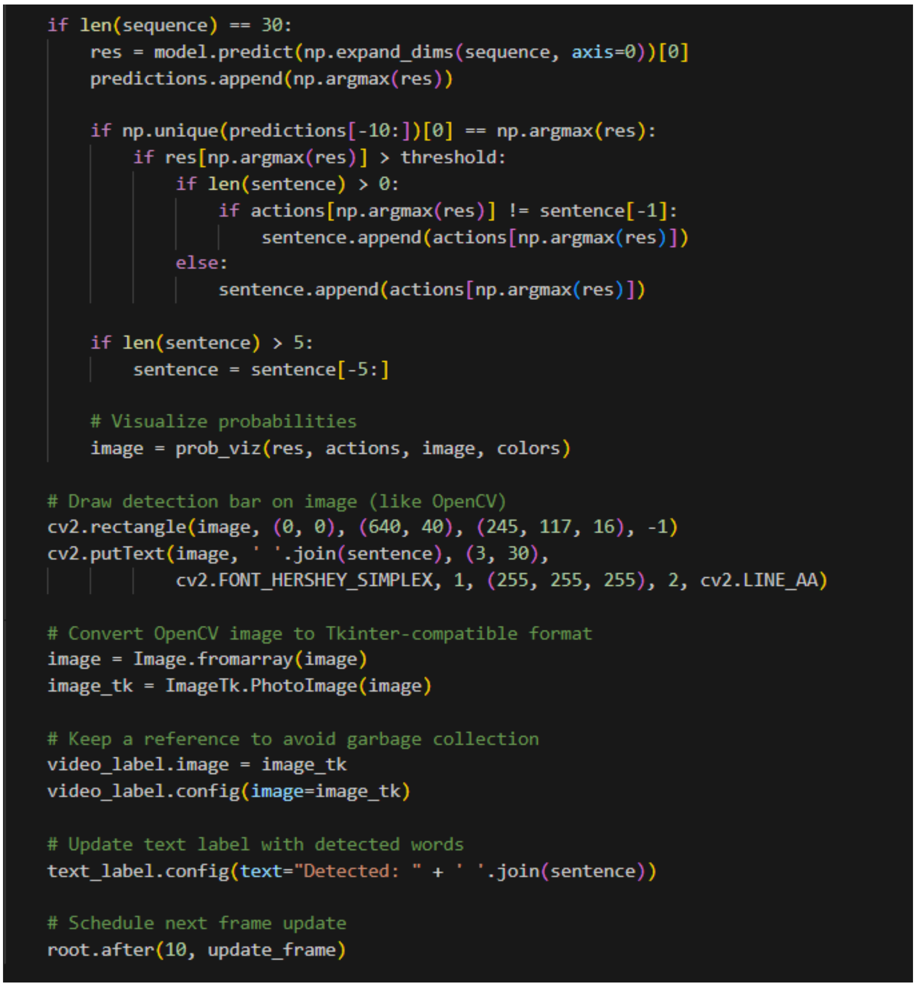
Detected: yes hello iloveyou hello iloveyou

Figure 23: Word recognition of "I Love You" in Tkinter GUI

For example:

Without probability bars: The system directly outputs "HELLO" even if it's only 51% confident.

With probability bars: The user sees "HELLO" with **51%**, "THANK YOU" with **47%**, and adjusts their gesture to get a more confident prediction.



```
if len(sequence) == 30:
    res = model.predict(np.expand_dims(sequence, axis=0))[0]
    predictions.append(np.argmax(res))

    if np.unique(predictions[-10:])[0] == np.argmax(res):
        if res[np.argmax(res)] > threshold:
            if len(sentence) > 0:
                if actions[np.argmax(res)] != sentence[-1]:
                    sentence.append(actions[np.argmax(res)])
            else:
                sentence.append(actions[np.argmax(res)])

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

    # Visualize probabilities
    image = prob_viz(res, actions, image, colors)

# Draw detection bar on image (like OpenCV)
cv2.rectangle(image, (0, 0), (640, 40), (245, 117, 16), -1)
cv2.putText(image, ' '.join(sentence), (3, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

# Convert OpenCV image to Tkinter-compatible format
image = Image.fromarray(image)
image_tk = ImageTk.PhotoImage(image)

# Keep a reference to avoid garbage collection
video_label.image = image_tk
video_label.config(image=image_tk)

# Update text label with detected words
text_label.config(text="Detected: " + ' '.join(sentence))

# Schedule next frame update
root.after(10, update_frame)
```

Figure 24: Tkinter GUI implementation

Letter Interpreter

The UI for SignSpeak was carefully designed to ensure three essential criteria would be fulfilled: clarity, ease-of-use and responsiveness, all within a professional layout.

Vue.js was chosen as the frontend framework over Vanilla JS or React as its more lightweight and much easier to integrate. It also possesses a reactive state system and two-way binding, which allowed for a more seamless synchronization between the sign detection and display, reducing latency and improving maintainability. The styling was implemented using Tailwind CSS, a utility-first framework that offers a more refined control over the UI elements compared to Bootstrap or raw CSS. The system's functionality was maintained as the highest priority throughout the development of this project. Tailwind's utility classes allow for rapid iteration and greater precision while tuning the layout, spacing and colour schemes, which are essential for accessibility.

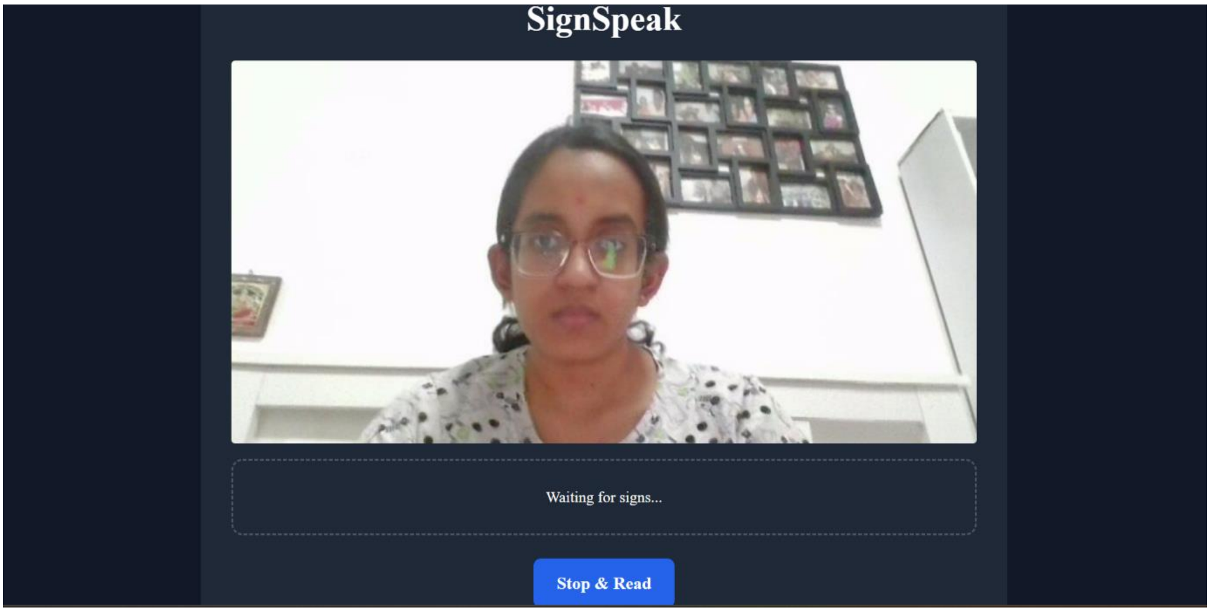


Figure 25: Web Interface for letter recognition (Without hands detected)

Another notable design choice was the per character box display, which vastly improves legibility and avoids misinterpretation during fast or partial detections and a clearer visual segmentation. Each letter appears in its own bounding box within a scrollable flex-wrapped container, preventing the overflow of letters past the screen width and maintaining a tidy layout. Free-flow layouts were considered but remain unused due to the unpredictability of word lengths and lack of boundary control on smaller viewports.

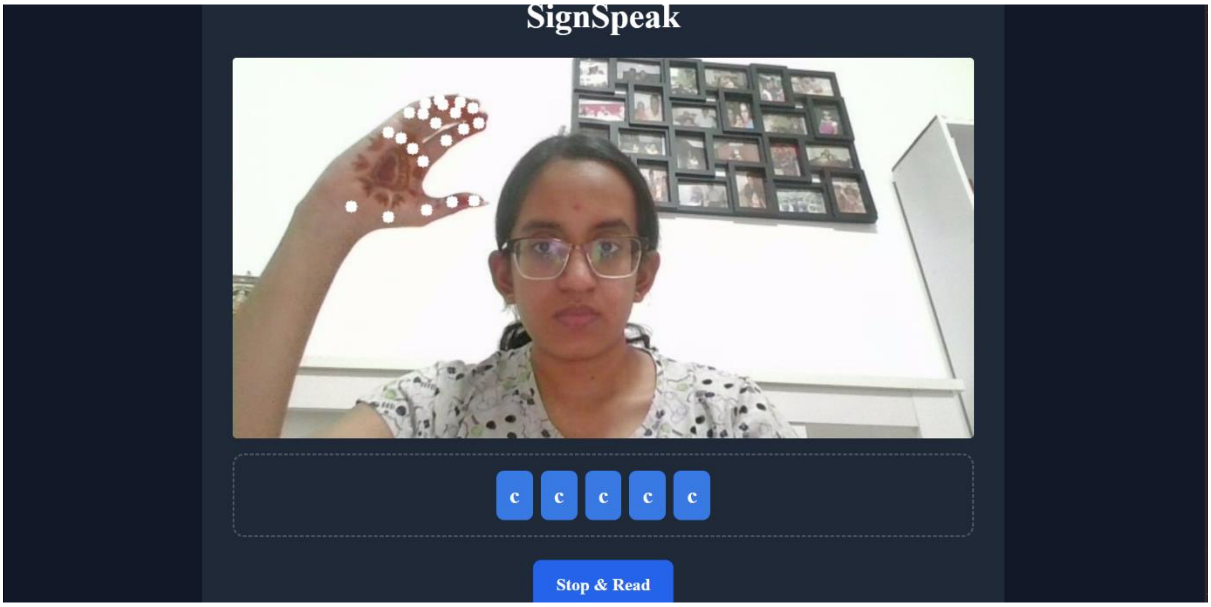
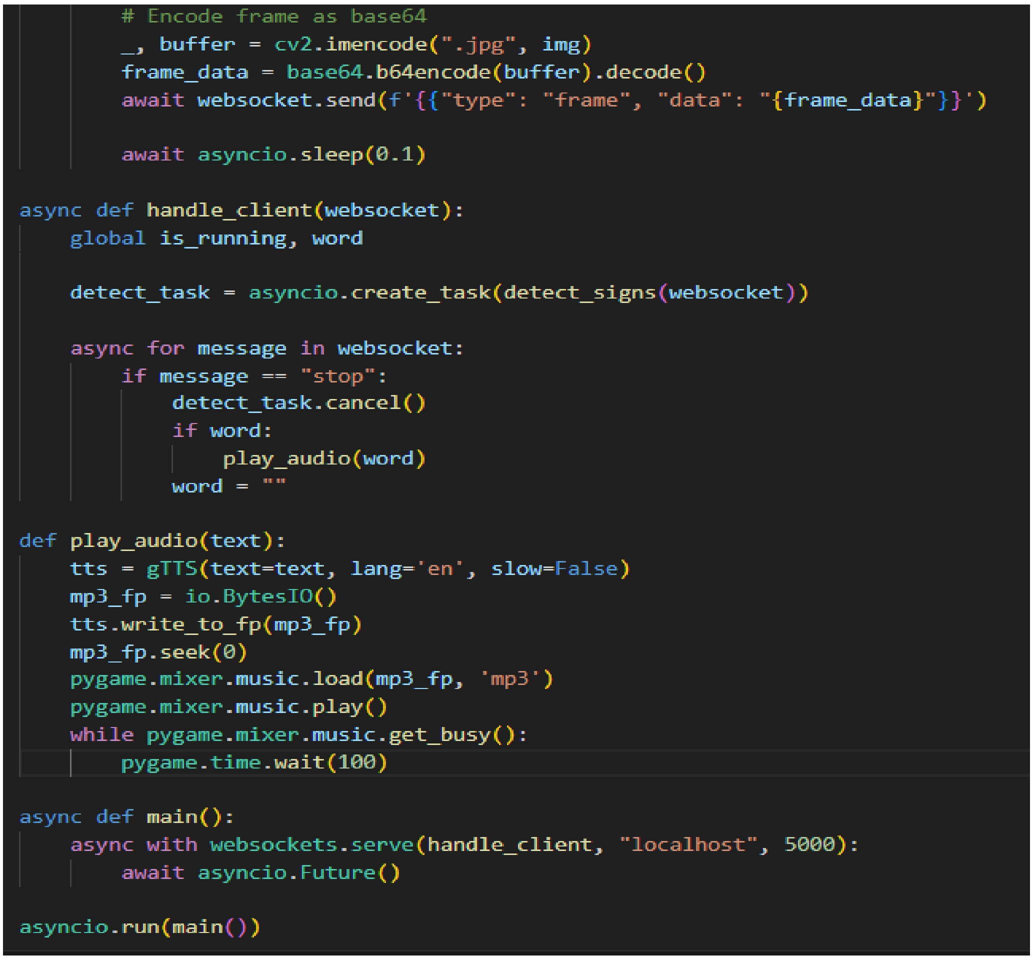


Figure 26: Web UI with hand detected for letter "C"

Communication between the front and backend relies upon a WebSocket connection, chosen over other traditional REST APIs due to its persistence, low-latency and bi-directional communication, which is vital for sending rapid updates for video frames and recognized characters. REST or other polling mechanisms introduced delays, excess network overhead and the interactivity was less fluid, which is an essential feature of a real-time communication system.



```
        # Encode frame as base64
        _, buffer = cv2.imencode(".jpg", img)
        frame_data = base64.b64encode(buffer).decode()
        await websocket.send(f'{{"type": "frame", "data": "{frame_data}"}}')

        await asyncio.sleep(0.1)

async def handle_client(websocket):
    global is_running, word

    detect_task = asyncio.create_task(detect_signs(websocket))

    async for message in websocket:
        if message == "stop":
            detect_task.cancel()
            if word:
                play_audio(word)
            word = ""

def play_audio(text):
    tts = gTTS(text=text, lang='en', slow=False)
    mp3_fp = io.BytesIO()
    tts.write_to_fp(mp3_fp)
    mp3_fp.seek(0)
    pygame.mixer.music.load(mp3_fp, 'mp3')
    pygame.mixer.music.play()
    while pygame.mixer.music.get_busy():
        pygame.time.wait(100)

async def main():
    async with websockets.serve(handle_client, "localhost", 5000):
        await asyncio.Future()

asyncio.run(main())
```

Figure 27: WebSocket implementation for server

The video stream itself is updated through base64-encoded frames received through WebSocket messages, which are then injected into the DOM without re-rendering the whole component, preserving efficiency and reducing time-delays. A “Stop & Read” button has been integrated to immediately terminate recognition and trigger speech synthesis using Google Text-to-Speech software, offering a clear point of interaction.

Overall, the interface balances professional simplicity with efficiency in real-time demands. Every component for the user interface was chosen with optimized performance, accessibility and maintainability in mind, thereby delivering a robust and intuitive platform that sets the first step in ASLR systems, leaving room for multi-sign phrases, words or even multi-language support, while retaining the core that defines the application today.

VISUAL REPRESENTATIONS

Here, we will represent the user experience in this project with UML, ER and flowchart diagrams. When the user positions themselves before their computer camera, the system detects the position of their hands, as well as the signs made, and classifies them accurately according to the American sign language.

Introduction to System Design Diagrams

System design diagrams are vital to the software development process as they provide a structured way to visualize and organize complex data. They serve as a blueprint for developers, designers and stakeholders, outlining how various components of the system work, ensuring a well-planned and effective project. By offering a clear representation of data flow, relationships and dependencies, these diagrams aid problem-solving and facilitates future modifications. In an AI-based application like the ASLR System being discussed, wherein multiple highly complex processes like video processing, feature extraction, machine learning, and user interaction are involved, system diagrams play a crucial role in breaking down the complexity of the tasks at hand.

The ASL Recognition System involves capturing real-time video input, processing frames to extract key points, and using trained AI models to interpret hand gestures into meaningful words or letters. The system also needs an efficient structure to handle big data storage, process rapid real-time predictions, and display results in an intuitive manner. To achieve this, various diagrams are used, each serving a distinct purpose in defining and refining the system.

The ER (Entity-Relationship) diagram provides one of many ways to structure and organize the data in a visually appealing manner. It defines key entities of the system and demonstrates how they are interconnected. By designing a clear ER diagram, viewers can understand the intricacies of each working component of the system and their dependencies on each other for optimum model functioning.

The UML (Unified Modelling Language) diagram, on the other hand, focuses on the object-oriented structure of the system, detailing the attributes of each component and their behaviours within the software. This organisation helps maintain modularity, allowing developers to understand the code in parts, update and/or extend individual functionalities without disrupting the entire system. A well-structured class diagram ensures that code is reusable, scalable, and easier to maintain.

A flowchart illustrates the sequence of operations that occur within the system from beginning of usage to end. By providing a high-level overview of the entire process without delving into the intricacies, the diagram offers a simplified perspective that makes it easier to identify potential inefficiencies and optimize overall workflow execution.

Finally, the sequence diagram further refines the understanding of the system by depicting the communication between various segments of the system throughout its working. In contrast to the flowchart, the sequence diagram provides a detailed view on the interaction between the separate, interlinked components of the system. By visualizing these complexities, developers can gain an ants' eye view of the project and optimize the system performance accordingly.

Altogether, these diagrams create a comprehensive visual representation of the system. By providing valuable insights into data management, software architecture and process execution, they ensure that the system can be built with efficiency, scalability, and usability in mind. Seeing as visual learning is often attributed to a higher retention rate and understanding level of complex topics, it can be said that diagrams are essential to the creation of any project and this one is no exception.

UML DIAGRAM

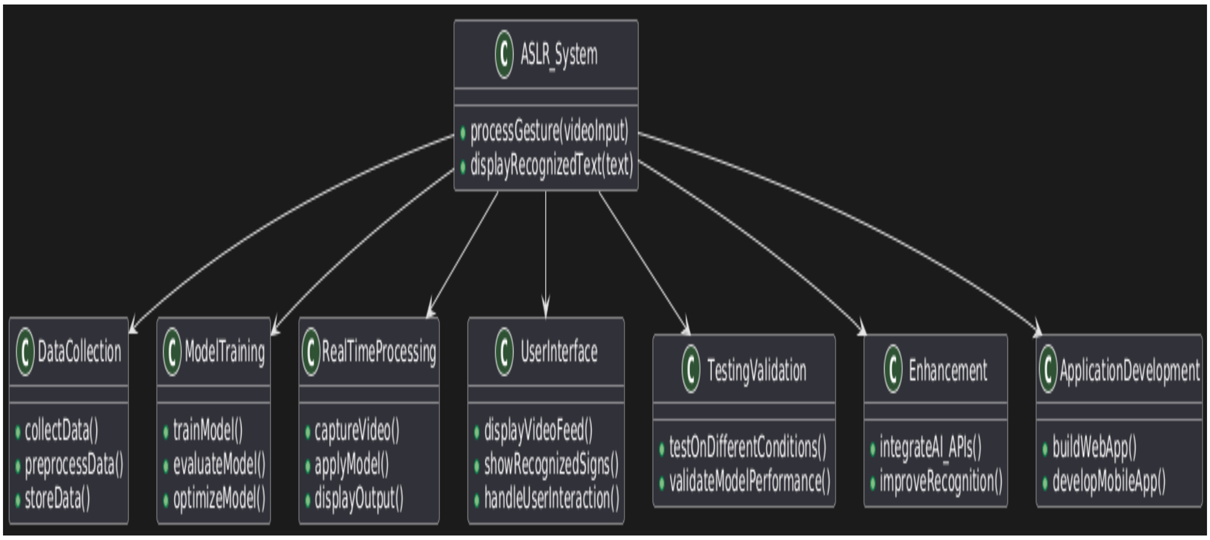


Figure 28: UML Diagram

SEQUENCE DIAGRAM

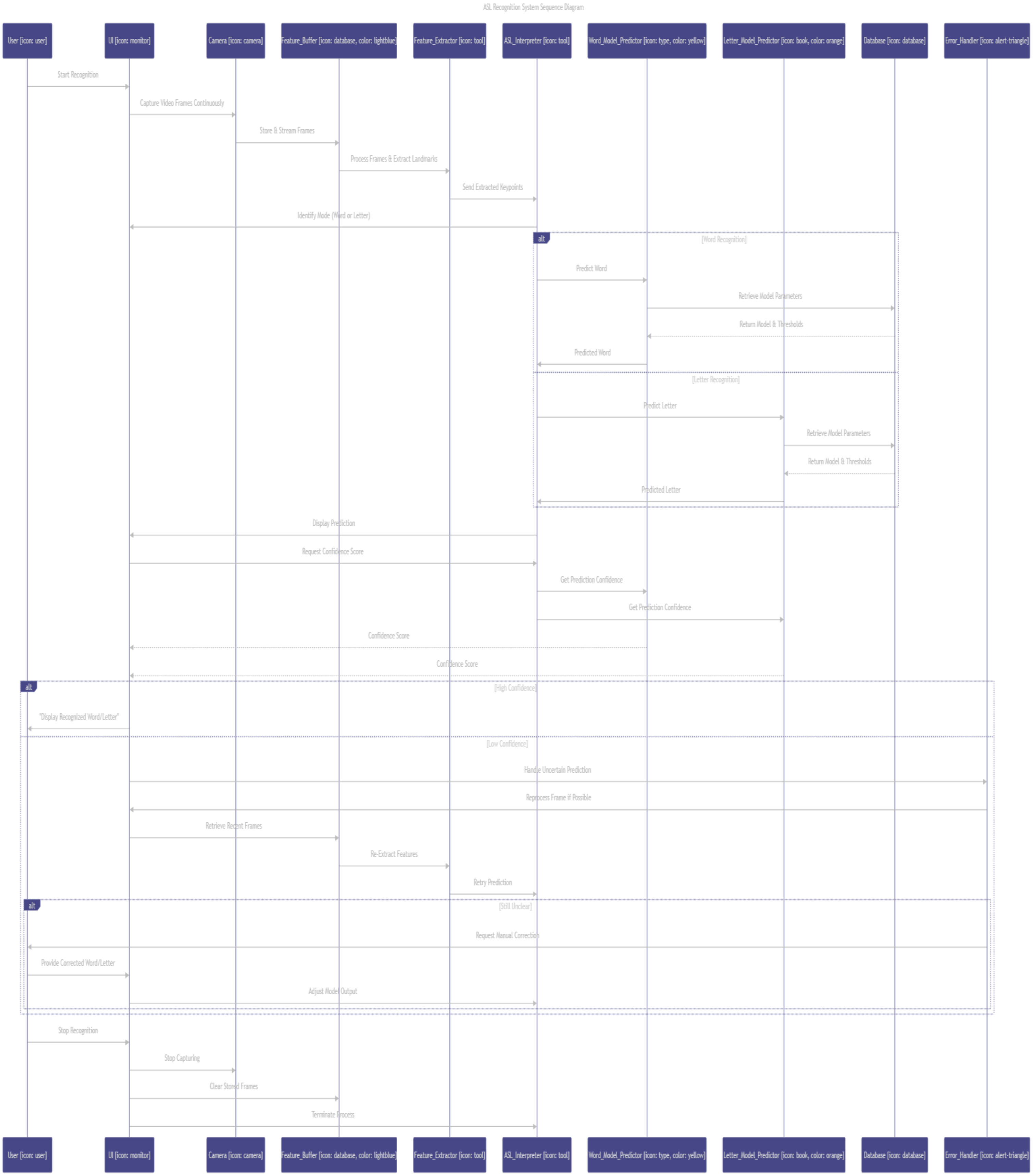

ASL Recognition System Sequence Diagram
User [icon: user]
UI [icon: monitor]
Camera [icon: camera]
Feature_Buffer [icon: database, color: lightblue]
Feature_Extractor [icon: tool]
ASL_Interpreter [icon: tool]
Word_Model_Predictor [icon: type, color: yellow]
Letter_Model_Predictor [icon: book, color: orange]
Database [icon: database]
Error_Handler [icon: alert-triangle]
Start Recognition
Capture Video Frames Continuously
Store & Stream Frames
Process Frames & Extract Landmarks
Send Extracted Keypoints
Identify Mode (Word or Letter)
alt
[Word Recognition]
Predict Word
Retrieve Model Parameters
Return Model & Thresholds
Predicted Word
[Letter Recognition]
Predict Letter
Retrieve Model Parameters
Return Model & Thresholds
Predicted Letter
Display Prediction
Request Confidence Score
Get Prediction Confidence
Get Prediction Confidence
Confidence Score
Confidence Score
alt
[High Confidence]
"Display Recognized Word/Letter"
[Low Confidence]
Handle Uncertain Prediction
Reprocess Frame if Possible
Retrieve Recent Frames
Re-Extract Features
Retry Prediction
alt
[Still Unclear]
Request Manual Correction
Provide Corrected Word/Letter
Adjust Model Output
Stop Recognition
Stop Capturing
Clear Stored Frames
Terminate Process
User [icon: user]
UI [icon: monitor]
Camera [icon: camera]
Feature_Buffer [icon: database, color: lightblue]
Feature_Extractor [icon: tool]
ASL_Interpreter [icon: tool]
Word_Model_Predictor [icon: type, color: yellow]
Letter_Model_Predictor [icon: book, color: orange]
Database [icon: database]
Error_Handler [icon: alert-triangle]

Figure 29: Sequence Diagram

FLOWCHART

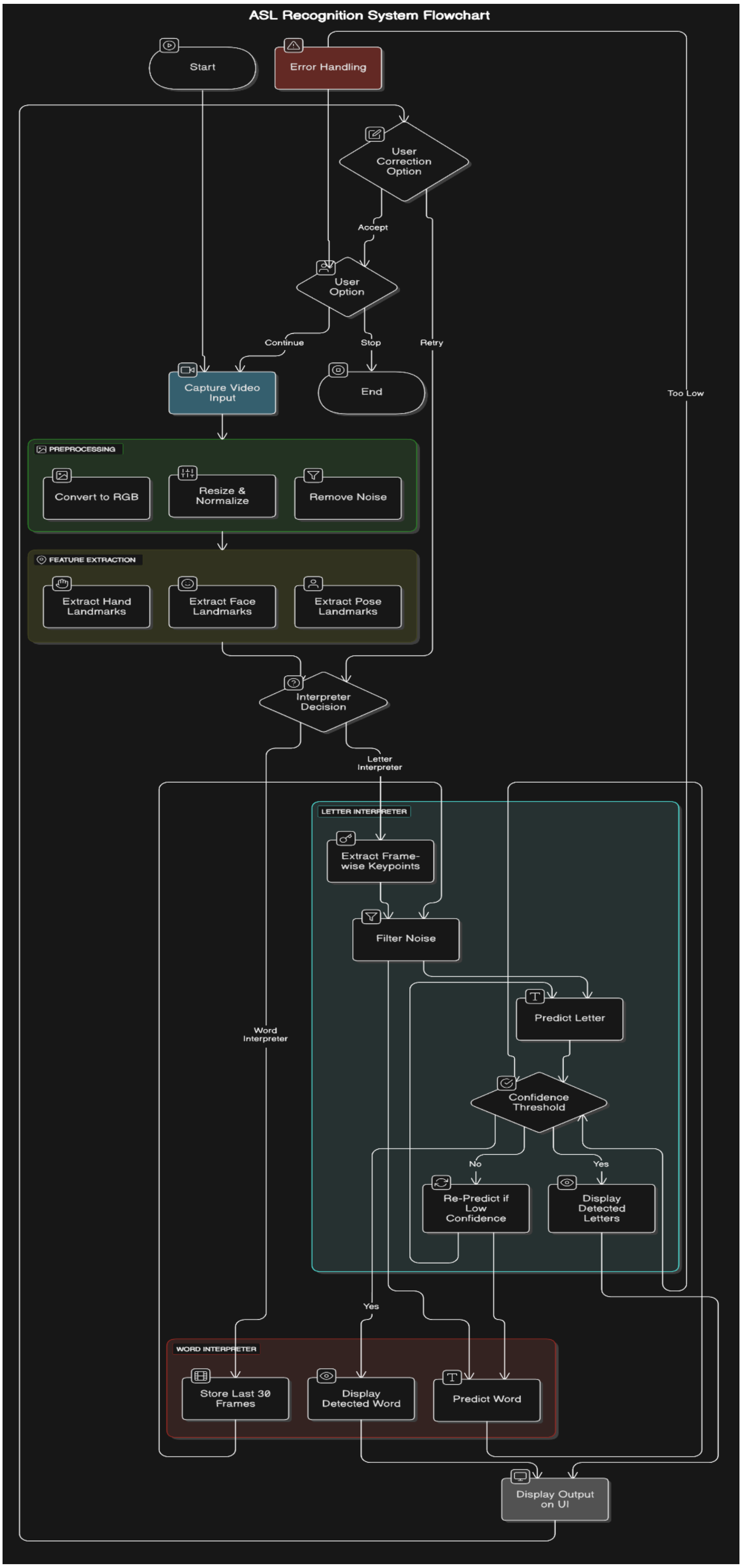


Figure 30: Flowchart diagram

ER DIAGRAM

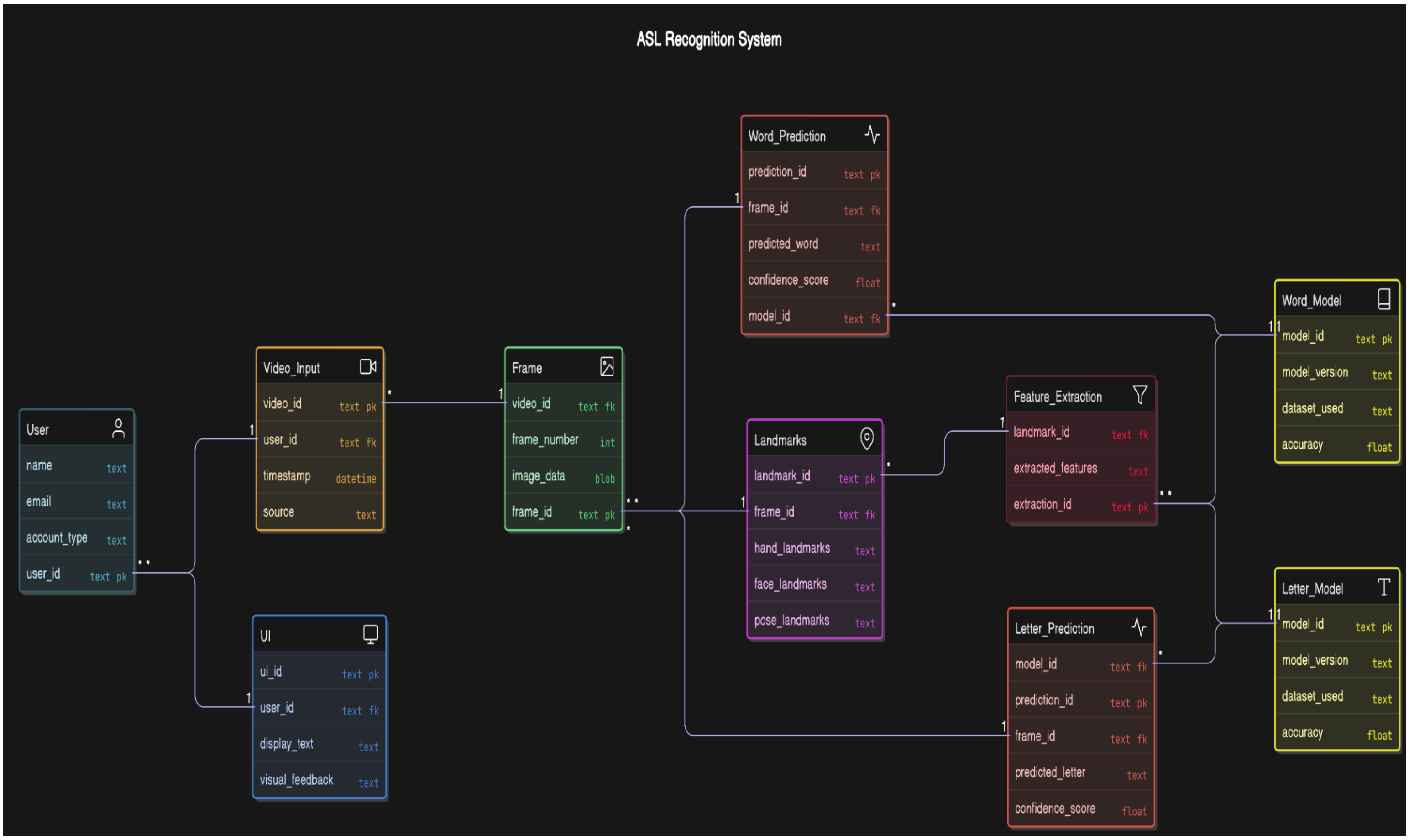


Figure 31: ER diagram

EVALUATION AND TESTING

The WLASL dataset obtained an overall accuracy of >40% with ~ 2000 words, however, no real-time detection was implemented.

The ASL custom datasets for words and letters received accuracies of 88.5% and >95% respectively. When implemented on real-time systems, they performed well, with low latency and high accuracy.

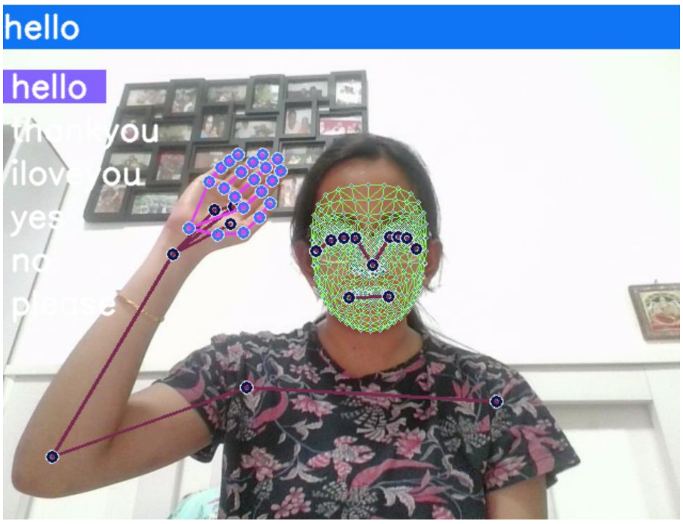


Figure 33: Testing phase of word "Hello"

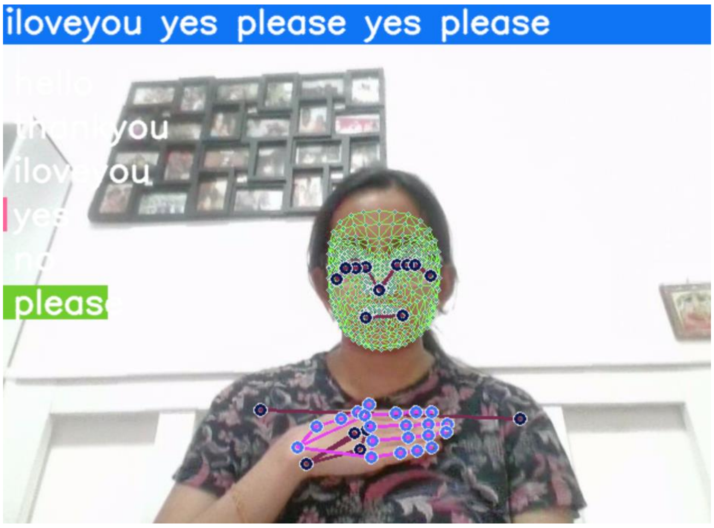


Figure 32: Testing phase of word "Please"

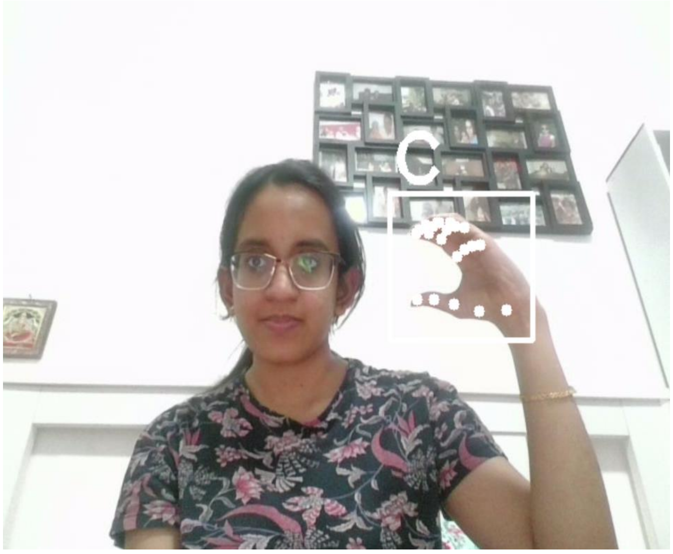


Figure 34: Testing of letter "C"

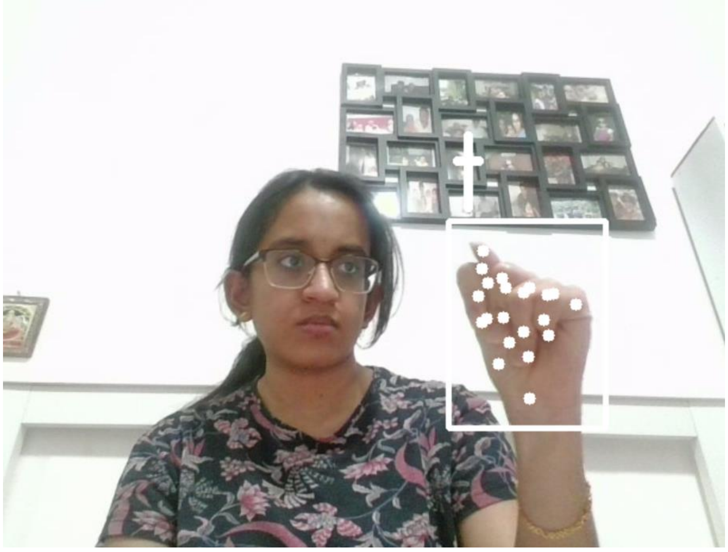


Figure 35: Testing of letter "T"

ERROR ANALYSIS

Despite the model's overall success rate, several challenges affect its performance and usability. One of the primary issues was the system's sensitivity to background conditions and lighting variations. Poor lighting and crowded backgrounds with excessive visual noise led to inconsistent feature extraction, resulting in misclassifications and false positives. Alternatively, the model also takes slightly longer to detect signs that are visibly similar, like S and T (bunched-up fist with thumb placed above other fingers and between them respectively), and dynamic signs like J and Z. To address this issue, Transformer-based sequence modelling could be considered for a better adjusted temporal context understanding, allowing the system to differentiate between gestures that only slightly differ in pattern.

The accuracy of gesture recognition was also negatively impacted by rapid hand movements. The faster users signed, motion blur often led to incomplete or inaccurate feature extraction, inconveniencing the model in accurately predicting gestures. A potential solution might possibly involve the integration of optical flow algorithms, which enhances movement tracking by capturing the direction and speed of hand gestures more effectively.

Finally, integrating the word interpreter into a website proved challenging due to significant latency issues. Unlike simpler web-based ASL recognition models (similar to the model created for the letter interpreter) that classify single frames, the word interpreter system required sequential frame analysis due to the dynamic nature of most words in the ASL, thereby introducing delays that disrupt real-time interaction fluidity. Running LSTM-based temporal modelling in the browser, which was constrained by limited computational resources led to the severe lag, which when coupled with high processing overhead, made the system impractical for web deployment. As a result, the word interpreter remains a standalone application within a GUI while optimizations for browser-based implementation are being explored.

FUTURE IMPROVEMENTS

Upon review of the shortcomings of the system implemented, a few enhancements that may potentially be implemented in the near future are listed below.

One of the most impactful improvements would be the integration of Transformer-based architectures, such as Vision Transformers, which offer better sequence learning capabilities. Unlike traditional LSTM models, Transformers analyse entire sequences in parallel, improving both speed and contextual accuracy. This advancement would allow the system to recognize gesture sequences with higher precision, reducing errors in similar sign classifications.

One of the most impactful implementations would be the integration of a ViTs (Vision Transformers) within the deep learning model, which would offer better sequence learning capabilities. Unlike traditional LSTMs, transformers analyse whole sequences in parallel, which would improve the speed and contextual accuracy, but would be highly resource intensive and require a great amount of computational power and memory. To incorporate such an architecture while managing the adverse effects will possibly be an ensuing challenge, but also a potential development towards the optimization of the ASLR system.

Another major upgrade would involve the expansion of the system's vocabulary. While the system currently performs on a limited dataset, incorporating a broader set of signs would significantly improve the scope of the project and its generalization. This could be achieved through alternate datasets like the RWTH-PHOENIX-Weather or MS-ASL, as well as data augmentation techniques to generate more samples. The interpretation of the number system will certainly be implemented in the near future.

An adaptive background removal system has also been considered to address the impact of environmental disturbances upon the accuracy of predictions. By utilising computer vision tactics like real-time segmentation models, the system could dynamically remove the background of the user's camera feed and adjust to various lighting conditions, making the system more robust for real-world deployment.

To further improve accessibility and usability, the system will be developed into a mobile application. A mobile-based ASL recognition system would allow users to interact with a portable system seamlessly without requiring high-performance, bulky computing hardware. Moreover, integrating the Google Translate API into the project structure would vastly increase the system's scope and purpose by providing real-time translation of recognized ASL gestures into multiple spoken and written languages, having a much greater influence on the DHH community globally, making the system a more practical and versatile tool in everyday scenarios.

Lastly, integrating optical flow-based hand tracking would enhance rapid motion detection and effectively mitigate the effects of motion blur. Optical flow techniques track pixel movements across video frames, which allows smoother and more precise recognition of fast-moving gestures.

CONCLUSION

This project serves as a testament to the bridges that could be built if technology and empathy worked hand in hand. Translating silent gestures into their meanings using Machine learning and artificial intelligence provides more than a tool to the DHH community, it gives them a voice. Leveraging the most efficient and accurate deep learning systems through much trial and error has enabled the creation of lightweight systems that remain accessible, responsive and highly multi-functional, with great scope for the future of communication. Beyond metrics and model however, this project must serve to remind the tech world that every letter, every word detected is a thought shared, and a barrier broken.

Looking to the future, the scope of this work extends far beyond its current framework. The principles that guide it—lightweight computation, real-time processing, and user-centric design—open doors to broader applications across education, assistive technologies, and human-computer interaction. With ongoing advancements in model optimization, edge computing, and sensor precision, systems like this will only become more efficient, more accurate, and more human in their ability to respond.

Looking to the future, the scope of this project extends far beyond its current framework. The fundamental ideas that have guided its creation; lightweight communication, real-time processing, user-centric design; open doors to broader applications in the real-world, spreading across the education, emergency, entertainment and e-commerce sectors, in assistive technologies and human-computer interaction (inclusion withing GenAI tech). With the ongoing advancements in AI, model optimization, edge computing and sensor precision, systems as such will become more efficient, promising more than a technical obstacle overcome, but a future of technologically aided, abundantly available communication between all walks of life.

This project is not the final step, but a hopeful stride toward a more connected and compassionate future.

CITATIONS

[1] Camgoz, N. C., Hadfield, S., Koller, O., & Bowden, R. (2018). **Neural sign language translation.** *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 7784–7793.

[2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). **You only look once: Unified, real-time object detection.** *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 779–788.

[3] Zhou, B., Wang, J., & Zhao, Y. (2021). **Towards real-time sign language recognition: Challenges and opportunities.** *IEEE Transactions on Pattern Analysis and Machine Intelligence, 43*(12), 1234–1248.

[4] Mehdi, Syed Atif & Khan, Yasir Niaz. (2002). Sign language recognition using sensor gloves. 2204 - 2206 vol.5. 10.1109/ICONIP.2002.1201884.

[5] Pigou, L., Dieleman, S., Kindermans, P.-J., & Schrauwen, B. (2015). **Sign language recognition using convolutional neural networks.** *European Conference on Computer Vision (ECCV),* 572–578.

[6] Yin, B., & Read, J. (2021). **Sign language translation with transformer-based architectures: Progress and challenges.** *IEEE Transactions on Neural Networks and Learning Systems, 32*(5), 1897–1912.

[7] Hu, J., Zhang, Y., Li, L., & Wang, M. (2021). **SignBERT: Pre-training for sign language recognition with transformer-based architectures.** *Advances in Neural Information Processing Systems (NeurIPS),* 1–11.

[8] "Forster, Jens & Schmidt, Christoph & Hoyoux, Thomas & Koller, Oscar & Zelle, Uwe & Piater, Justus & Ney, Hermann." (“Towards Indian Sign Language Sentence Recognition using INSIGNVID ...”) (2012). RWTH-PHOENIX-Weather: A Large Vocabulary Sign Language Recognition and Translation Corpus.

[9] Ronchetti, F., Quiroga, F. M., Estrebou, C., Lanzarini, L., & Rosete, A. (2016). LSA64: A Dataset of Argentinian Sign Language. *XXII Congreso Argentino de Ciencias de la Computación (CACIC)*.

[10] S. A. E. El-Din and M. A. A. El-Ghany, "Sign Language Interpreter System: An alternative system for machine learning," 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), Giza, Egypt, 2020, pp. 332-337, doi: 10.1109/NILES50944.2020.9257958. keywords: {Sensors; Gesture recognition; Assistive technology; Sensor systems; Resistance; Bending;Pins;American Sign Language (ASL);Arabic Sign Language (ArSL);Sign Language Recognition (SLR);sensor glove},

[11] Jeyasheeli, P. G., & Indumathi, N. (2020). "Sentence Generation for Indian Sign Language Using NLP." *WEBology*, vol. 18, Special Issue 1, pp. 1-10, DOI: 10.14704/WEB/V18SI01/WEB18054.

[12] Brown, T. B., et al. (2020). *Language Models are Few-Shot Learners*. OpenAI. https://arxiv.org/abs/2005.14165

[13] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. NAACL. https://arxiv.org/abs/1810.04805

[14] Rakhmadi, A., Yudhana, A., & Sunardi, S. (2024). Virtual Reality and Augmented Reality in Sign Language Recognition: A Review of Current Approaches. *International Journal of Informatics and Computation*, 6(2), 64-83. https://doi.org/10.35842/ijicom.v6i2.94

[15] Li, D., Guo, Y., & Xu, Z. (2019). "Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison." ("Word-level Deep Sign Language Recognition from Video - S-Logix") *arXiv preprint arXiv:1910.11006*. Retrieved from https://arxiv.org/abs/1910.11006

[16] Alsharif, Bader, et al. "Deep Learning Technology to Recognize American Sign Language Alphabet." *Sensors*, vol. 23, no. 18, 1 Jan. 2023, p. 7970, www.mdpi.com/1424-8220/23/18/7970, https://doi.org/10.3390/s23187970.

[17] "ASL Alphabet." *Www.kaggle.com*, www.kaggle.com/datasets/grassknoted/asl-alphabet.

[18] Bantupalli, Kshitij, and Ying Xie. "American Sign Language Recognition Using Deep Learning and Computer Vision." *2018 IEEE International Conference on Big Data (Big Data)*, Dec. 2018, https://doi.org/10.1109/bigdata.2018.8622141.

[19] "Eraser – AI Co-Pilot for Technical Design." *Eraser.io*, 2025, app.eraser.io/all.

[20] Kasukurthi, Nikhil, et al. *American Sign Language Alphabet Recognition Using Deep Learning*.

[21] R.S, Sabeenian, et al. "Sign Language Recognition Using Deep Learning and Computer Vision." *Journal of Advanced Research in Dynamical and Control Systems*, vol. 12, no. 05-SPECIAL ISSUE, 30 May 2020, pp. 964–968, https://doi.org/10.5373/jardcs/v12sp5/20201842.

[22] Renotte, Nicholas. "Sign Language Detection Using ACTION RECOGNITION with Python | LSTM Deep Learning Model." *YouTube*, 19 June 2021, www.youtube.com/watch?v=doDUihpj6ro

[23] Sharma, Ayush. "American Sign Language (ASL) Recognition System Using Deep Learning." *Medium*, 3 Jan. 2024,

medium.com/@ayushjudesharp/american-sign-language-asl-recognition-system-using-deep-learning-e0b937a9378f.

[24] Zhang, Yanqiong, and Xianwei Jiang. “Recent Advances on Deep Learning for Sign Language Recognition.” *Computer Modeling in Engineering & Sciences*, vol. 139, no. 3, 2024, pp. 2399–2450, www.techscience.com/CMES/v139n3/55626/html, https://doi.org/10.32604/cmes.2023.045731.

[25] K. Dabre and S. Dholay, "Machine learning model for sign language interpretation using webcam images," 2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA), Mumbai, India, 2014, pp. 317-321, doi: 10.1109/CSCITA.2014.6839279. keywords: {Assistive technology; Gesture recognition; Training; Feature extraction; Cameras; Shape;Indian Sign Language (ISL);Computer Vision(CV)},

[26] *A Review on Systems-Based Sensory Gloves for Sign Language Recognition ...*, https://pmc.ncbi.nlm.nih.gov/articles/PMC6069389/.

[27]

[28] Vedak, O., Zavre, P., Todkar, A., & Patil, M. (2019). Sign language interpreter using image processing and machine learning. *International Research Journal of Engineering and Technology (IRJET)*, *6*(4), 1907. https://www.irjet.net

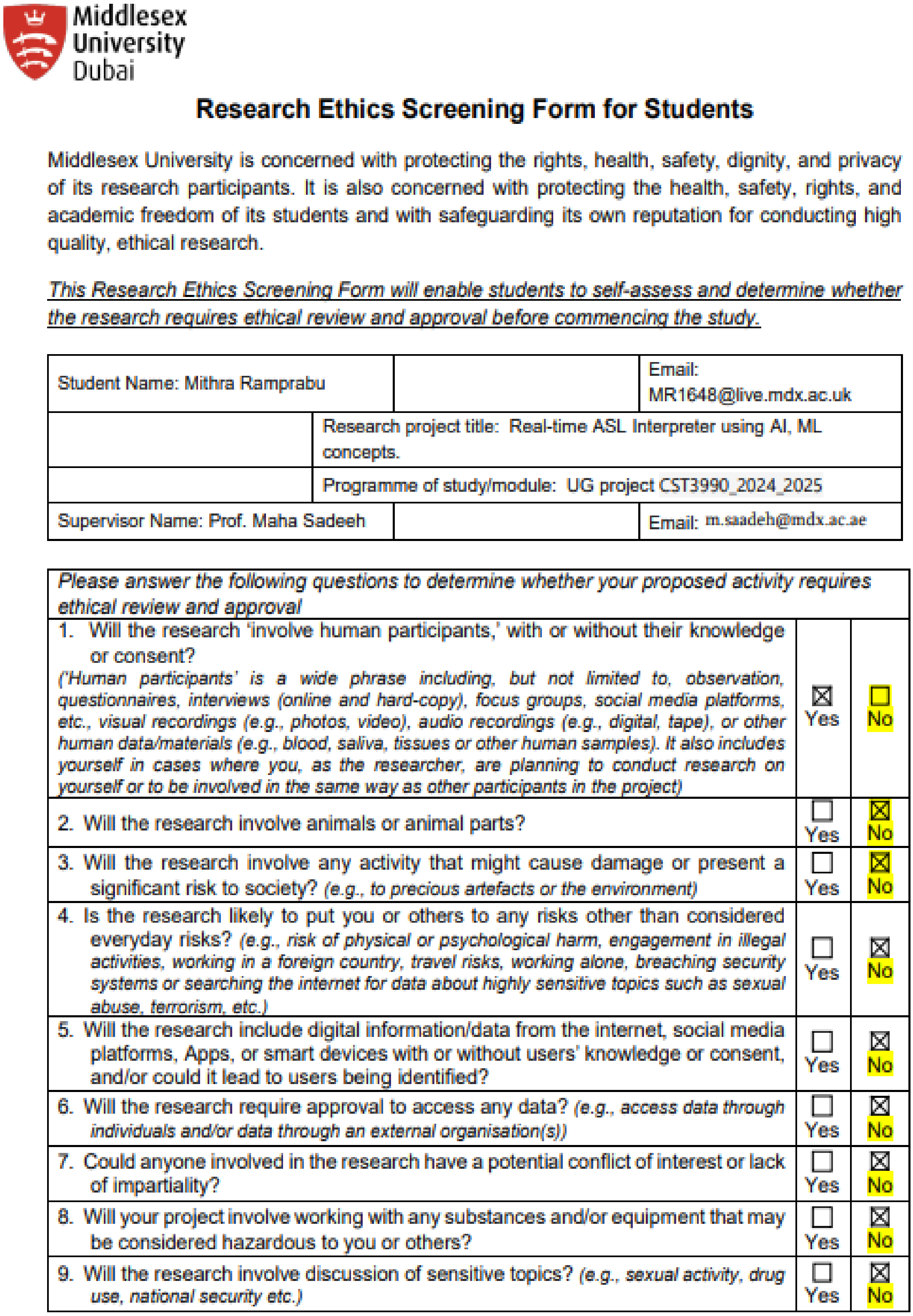
[29] Renotte, Nicholas. “Real Time Sign Language Detection with Tensorflow Object Detection and Python | Deep Learning SSD.” *YouTube*, 5 Nov. 2020, www.youtube.com/watch?v=pDXdlXlaCco.

[30] Rob Mulla. “ASL Fingerspelling Demo - #Kaggle Competition Solution.” *YouTube*, 29 Aug. 2023, www.youtube.com/watch?v=D89nObYBKTI.

[31] stas444. “WLASL-2000 Resized.” *Kaggle.com*, 2022, www.kaggle.com/datasets/sttaseen/wlasl2000-resized.

[32] “Learn How to Sign - YouTube.” *Www.youtube.com*, www.youtube.com/@LearnHowtoSign.

ETHICS SCREENING FORM



Middlesex University Dubai

Research Ethics Screening Form for Students

Middlesex University is concerned with protecting the rights, health, safety, dignity, and privacy of its research participants. It is also concerned with protecting the health, safety, rights, and academic freedom of its students and with safeguarding its own reputation for conducting high quality, ethical research.

This Research Ethics Screening Form will enable students to self-assess and determine whether the research requires ethical review and approval before commencing the study.

Student Name: Mithra Ramprabu		Email: MR1648@live.mdx.ac.uk
	Research project title: Real-time ASL Interpreter using AI, ML concepts.	
	Programme of study/module: UG project CST3990_2024_2025	
Supervisor Name: Prof. Maha Sadeeh		Email: m.saadeh@mdx.ac.ae

Please answer the following questions to determine whether your proposed activity requires ethical review and approval		
1. Will the research 'involve human participants,' with or without their knowledge or consent? *('Human participants' is a wide phrase including, but not limited to, observation, questionnaires, interviews (online and hard-copy), focus groups, social media platforms, etc., visual recordings (e.g., photos, video), audio recordings (e.g., digital, tape), or other human data/materials (e.g., blood, saliva, tissues or other human samples). It also includes yourself in cases where you, as the researcher, are planning to conduct research on yourself or to be involved in the same way as other participants in the project)*	☒ Yes	□ No
2. Will the research involve animals or animal parts?	□ Yes	☒ No
3. Will the research involve any activity that might cause damage or present a significant risk to society? *(e.g., to precious artefacts or the environment)*	□ Yes	☒ No
4. Is the research likely to put you or others to any risks other than considered everyday risks? *(e.g., risk of physical or psychological harm, engagement in illegal activities, working in a foreign country, travel risks, working alone, breaching security systems or searching the internet for data about highly sensitive topics such as sexual abuse, terrorism, etc.)*	□ Yes	☒ No
5. Will the research include digital information/data from the internet, social media platforms, Apps, or smart devices with or without users' knowledge or consent, and/or could it lead to users being identified?	□ Yes	☒ No
6. Will the research require approval to access any data? *(e.g., access data through individuals and/or data through an external organisation(s))*	□ Yes	☒ No
7. Could anyone involved in the research have a potential conflict of interest or lack of impartiality?	□ Yes	☒ No
8. Will your project involve working with any substances and/or equipment that may be considered hazardous to you or others?	□ Yes	☒ No
9. Will the research involve discussion of sensitive topics? *(e.g., sexual activity, drug use, national security etc.)*	□ Yes	☒ No

Middlesex
University
Dubai

10. Will the outputs from your research (e.g., products, reports, publications, etc.) likely cause any harm to you, others, or to society; or have legal issues?	□ Yes	☒ No

If you have answered 'Yes' to ANY of the above questions, your application requires ethical review and approval prior to commencing your research. Please complete the 'Application for Ethical Approval for Research Projects for Students' form

If you have answered 'No' to ALL of the above questions, your application may not require ethical review and approval before commencing your research. Your research supervisor will confirm this below.

Student Signature: Date: 01/02/2025

To be completed by the supervisor:

Based on the details provided in the self-assesment form, I confirm that:	
The study does not require ethical review and approval	☒ No
The study requires ethical review and approval	Yes □

Superivsor Signature: Dr Maha Saadeh Date:28/2/20205