FROM GESTURES TO WORDS: AMERICAN SIGN LANGUAGE END-TO-END DEEP LEARNING INTEGRATION WITH TRANSFORMERS AND MEDIAPIPE

by

Ngoc Hiep Nguyen

Submitted in partial fulfillment of the

requirements for Departmental Honors in

the Department of Computer Science

Texas Christian University

Fort Worth, Texas

May 6, 2024

FROM GESTURES TO WORDS: AMERICAN SIGN LANGUAGE END-TO-END DEEP LEARNING INTEGRATION WITH TRANSFORMERS AND MEDIAPIPE

Project Approved:

Supervising Professor: Bingyang Wei, PhD

Department of Computer Science

Bo Mei, PhD

Department of Computer Science

Drew Tomlin, PhD

Department of Mathematics

ABSTRACT

Speech impairment ranks among the world's most prevalent disabilities, affecting over 430 million adults [1]. Despite its widespread impact, many existing video-conferencing applications lack a comprehensive end-to-end solution for this challenge. In response, we present a holistic approach to translating American Sign Language (ASL) to subtitles in real-time by leveraging advancements in Google Mediapipe, Transformer models, and web technologies. In March 2024, Google[1] released the largest dataset for the problem domain with over 180 GB in size, containing ASL gesture sequences represented as Mediapipe numeric values. Our methodology begins with the implementation and training of a Transformer model using a preprocessed Google dataset, followed by the establishment of a back-end server that encapsulates the trained model for application integration. This server handles video input preprocessing and real-time inference, communicating with client services as a Representational State Transfer (REST) endpoint. To demonstrate the practicality of our approach, we developed a video conferencing application utilizing the AgoraRTC Software Development Kit (SDK), which communicates with our back-end server to transcribe user gestures to text and display the characters on the receiving end. Through this end-to-end system, we enable video calls enhanced by the real-time transcription of fingerspelled gestures with low latency and high accuracy, effectively bridging the communication gap for individuals with speech disabilities.

With a growing imperative for AI applications engineered for human well-being, our project seeks to promote the integration of AI in applications designed to enhance human wellness, thus bringing broader awareness and adoption of this endeavor.

[1] Google.,: “Google - American Sign Language Fingerspelling Recognition”, https://www.kaggle.com/competitions/asl-fingerspelling, last accessed 2023/11/21.

ACKNOWLEDGEMENTS

I extend my deepest appreciation to my advising professor, Dr. Bingyang Wei, whose unwavering patience and insightful feedback were instrumental throughout the journey of this honors thesis. Without his guidance, I wouldn't have navigated this problem domain with such confidence and determination.

Furthermore, I am deeply appreciative of Dr. Bo Mei and Dr. Drew Tomlin, whose unwavering support and guidance have been invaluable to me. I vividly recall standing outside Dr. Tomlin's office a year ago, nervously presenting my rough thesis idea, fully expecting a rejection due to my shaky presentation. To my astonishment and immense gratitude, Dr. Tomlin not only agreed to support my thesis but also offered her unwavering encouragement throughout the process.

Special gratitude goes to Dr. Michael Scherger, Chair of the Department of Computer Science at Texas Christian University, for encouraging senior students to pursue upper-division honors through thesis work. Additionally, I am thankful to the College of Science and Engineering for their generous funding, which enabled the realization of this project.

Finally, I want to give heartfelt thanks to my family and close friends for their unwavering support and encouragement during countless late nights of study and through challenging times.

TABLE OF CONTENTS

1 Introduction

According to the World Health Organization, there are over 430 million adults and 34 million children globally currently experiencing hearing loss, and it is estimated that in 30 years - there will be over 700 million people worldwide experiencing hearing loss [1]. Individuals who are deaf or nonverbal face significant challenges when it comes to communication within their community. Sign languages, such as American Sign Language (ASL), serve as a vital means of communication for those with hearing impairments, offering a natural and complete language for expression. However, despite its importance, many individuals, referred to as non-signers, are unfamiliar with sign language principles, hindering effective communication with the deaf and hard-of-hearing community. Therefore, as we transition to a hybrid working model characterized by distributed teams spanning the globe and the increased usage of video meetings in virtual workplaces, the need for accessibility features catering to the disabled rose significantly to ensure workplace equality and empower individuals of all abilities to participate fully in collaborative settings.

Despite the need for ASL-supported features for video conferencing tools, mature software solutions such as Zoom or Teams lack the capability of accurately translating ASL into real-time using Artificial Intelligence (AI), and require much logistics to set up a human interpreter. For Zoom, the software application only provides a feature to add a human interpreter in the conference call, instead of automatic AI sign language translation. An argument could be that one might use chat for communication, however, chat is ineffective and often be the least interactive means of communication, thus showing its inferiority in providing accessibility for disabled people in live discussions with other team members. With the fast-paced setting of the digital workplace, this gap limits the ability of nonverbal individuals to participate fully in digital communication spaces.

With the motivation for providing accessibility for hearing-impaired people in the workplace, in this paper, we propose an end-to-end Machine Learning solution to automatically interpret ASL to text with high accuracy and low latency, aimed at enhancing the well-being and convenience of nonverbal individuals by enabling more effective communication in digital spaces.

The paper is organized as follows: Section 2 provides a comprehensive review of existing literature on the development of automated tools for translating ASL to text, setting the context for our work. Section 3 outlines our methodology, detailing the implementation and training of a Transformer model, the development of an AI-integrated server for real-time inference, and the process of creating and integrating a video conferencing application with our AI system. In Section 4, we present the results of our trained model in predicting ASL characters, followed by a discussion on the effectiveness of our end-to-end solution. In addition, Section 4 also explores the implications of our findings. In Section 5, we conclude and propose potential avenues for future research, with a focus on further enhancing the accessibility and inclusivity of digital communication platforms for nonverbal individuals.

2 Literature Review

Addressing the challenges faced by hearing-disabled individuals using technology has garnered significant attention in research. Much of this work focuses on leveraging computer vision techniques to recognize various signs in American Sign Language. These implementations typically involve a combination of hardware and software or rely solely on deep learning models trained with extensive datasets.

Traditionally, Hidden Markov Models (HMMs) were widely used predictive models for sign language recognition. HMMs utilize labeled datasets of sign language sequences paired with corresponding textual representations [2]. However, modern approaches integrate motion sensors with machine learning software. For instance, Teak-Wei Chong and Boon-Giin Lee discussed the use of the Leap Motion Controller combined with machine learning techniques to recognize ASL letters and digits [3]. However, this approach's accuracy, limited by the use of a Support Vector Machine (SVM), reached only 80.30%.

Another approach involves leveraging TensorFlow Object Detection API for static 2D sign language recognition. Sharvani Srivastava et al. proposed a system for recognizing Indian Sign Language using this API and Transfer Learning [4]. While achieving an 85.45% average confidence rate, this method falls short in capturing the dynamics of ASL, which includes motion-based signs and words, thus hindering the future extension of word-level ASL prediction.

For gesture recognition, recognizing ASL gestures as sequences of data, rather than static images, is crucial. Sundar B. and colleagues proposed a methodology utilizing Long Short-Term Memory (LSTM) and Mediapipe for ASL alphabet recognition [5]. Their approach, achieving 99% accuracy on a custom dataset, emphasizes the importance of viewing ASL gestures as dynamic sequences. However, their custom dataset is limited on certain background and distance conditions, thus the model is not performing well in modern application integration.

Despite these advancements, previous work often lacks consideration for end-to-end integration and experimentation with newer models like the Transformer model. Integrating ASL recognition models into functional products poses challenges, particularly regarding real-time model inference and network communication for low-latency performance in video conferencing settings. This paper aims to address these gaps by training a Transformer model using Google's latest ASL fingerspelling dataset and integrating it into a functional product, thereby contributing to the advancement of ASL recognition technology.

3 **Research Methodology**

This section provides an overview of the research methodology employed in this study to develop and integrate an end-to-end solution for American Sign Language (ASL) recognition. It outlines the exploration of Mediapipe and the Google Dataset, the implementation of ASL recognition using Transformer architecture, the development of real-time inference and API using Flask, and the integration of the model into a video conferencing application using AgoraRTC.

Referring to Fig. 1, the video input is first fed into a layer of Google Mediapipe Landmark Recognition API to digitize body movements required to perform the action as coordinates, which sequentially being used as data input for a Deep Learning model to predict text output.

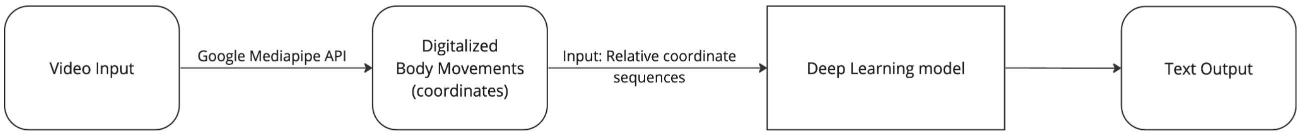

Video Input
Google Mediapipe API
Digitalized Body Movements (coordinates)
Input: Relative coordinate sequences
Deep Learning model
Text Output

Fig. 1. General flow diagram for translating video input of fingerspelling actions to text output.

3.1 Mediapipe and Google Dataset Exploration

This section provides an overview of the Google Mediapipe Landmarks Recognition API and the Google Fingerspelling Dataset, which are foundational components for the project's development.

a. Google Mediapipe Landmarks Recognition API

The project utilizes the Google Mediapipe Landmarks Recognition API to digitize face, body, left-hand, and right-hand landmarks into x, y, and z coordinates - giving a holistic approach to action recognition.

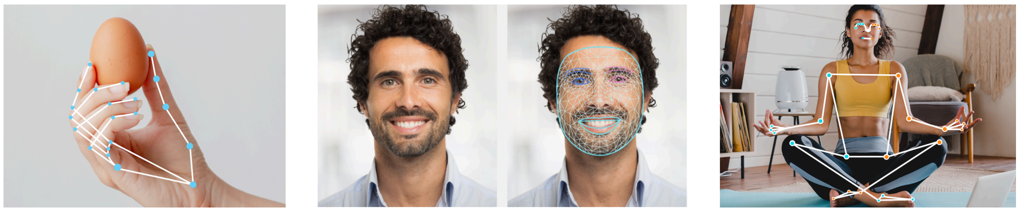


Fig. 2. Google Mediapipe's landmark detection for hand (left), face (center), and pose (right). [6]

For illustration, the hand landmark model bundle detects the keypoint localization of 21 hand-knuckle coordinates within the detected hand regions (see Fig. 3). There are 468 3D key points for face landmarks detection and 33 key points for pose landmarks detection.

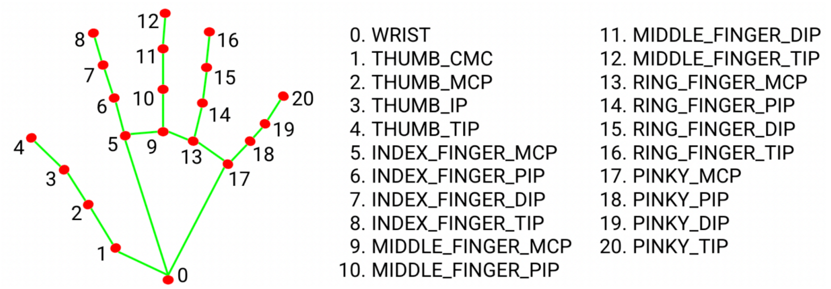


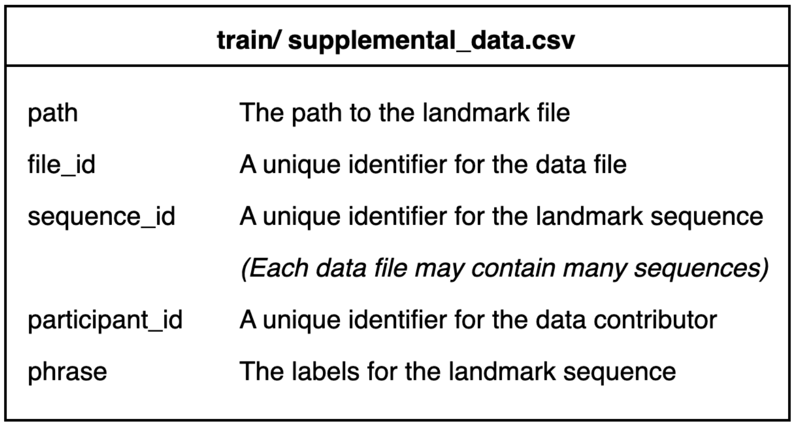
Fig. 3. Google Mediapipe's hand landmark model detects 21 different landmark coordinates. [7]

The Google Mediapipe API serves as a crucial preprocessing layer, generating landmark coordinates that serve as input for our trained model's inference. Furthermore, leveraging Mediapipe landmarks minimizes the necessity to train basic layers from scratch, streamlining the model development process. This approach also establishes a universal data standard for AI tasks related to human movements.

b. Dataset Overview

This section provides a comprehensive explanation of the Google Fingerspelling Dataset, detailing each component and providing illustrative examples. The Google Fingerspelling Dataset is over 180GB in size and was created from the effort of over 100 signers. The dataset includes the following components

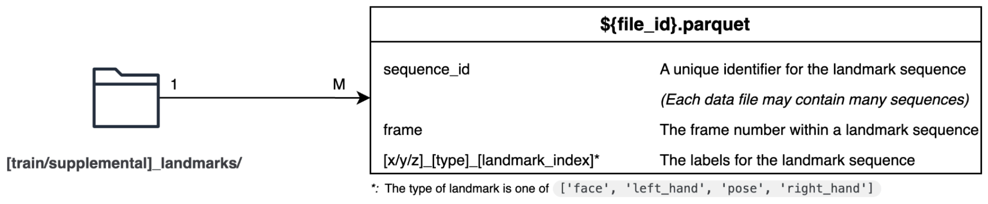
● *[train/supplemental]_metadata.csv*: provides metadata for each sequence of coordinates in the .parquet file (refer to Fig. 4).



train/ supplemental_data.csv	
path	The path to the landmark file
file_id	A unique identifier for the data file
sequence_id	A unique identifier for the landmark sequence *(Each data file may contain many sequences)*
participant_id	A unique identifier for the data contributor
phrase	The labels for the landmark sequence

Fig. 4. Content overview of train/supplemental_metadata.csv

● *[train/supplemental]_landmarks*: containing multiple .parquet files, each file comprising up to 1000 sequences of landmark coordinates representing phrases (refer to Fig. 5).



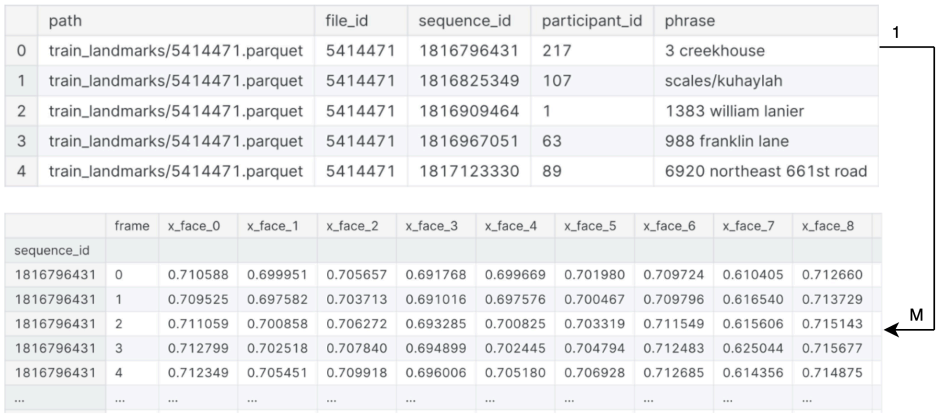
${file_id}.parquet	
sequence_id	A unique identifier for the landmark sequence *(Each data file may contain many sequences)*
frame	The frame number within a landmark sequence
[x/y/z]_[type]_[landmark_index]*	The labels for the landmark sequence

*: The type of landmark is one of `['face', 'left_hand', 'pose', 'right_hand']`

Fig. 5. Data is provided as parquet files, named using file_id. Each parquet file provides sequence_id, frame numbering, and the labels specifying each coordinate. One phrase is represented as one sequence specified by sequence_id, and a sequence contains many frames.

- *character_to_prediction_index.json*: This file provides a mapping between the characters for prediction and a numbered index.

To further explain the relationship between *[train/supplemental]_metadata.csv* and .parquet files of *[train/supplemental]_landmarks/,* refer to Fig. 6. as an example. Here, the phrase “3 creekhouse” has the file_id of “5414471” and the sequence_id of “1816796431”. Correspondingly, file 5414471.parquet contains the coordinate representation of the phrase “3 creekhouse” as multiple sequences, sharing the same sequence_id (“1816796431”).



	path	file_id	sequence_id	participant_id	phrase
0	train_landmarks/5414471.parquet	5414471	1816796431	217	3 creekhouse
1	train_landmarks/5414471.parquet	5414471	1816825349	107	scales/kuhaylah
2	train_landmarks/5414471.parquet	5414471	1816909464	1	1383 william lanier
3	train_landmarks/5414471.parquet	5414471	1816967051	63	988 franklin lane
4	train_landmarks/5414471.parquet	5414471	1817123330	89	6920 northeast 661st road

	frame	x_face_0	x_face_1	x_face_2	x_face_3	x_face_4	x_face_5	x_face_6	x_face_7	x_face_8
sequence_id										
1816796431	0	0.710588	0.699951	0.705657	0.691768	0.699669	0.701980	0.709724	0.610405	0.712660
1816796431	1	0.709525	0.697582	0.703713	0.691016	0.697576	0.700467	0.709796	0.616540	0.713729
1816796431	2	0.711059	0.700858	0.706272	0.693285	0.700825	0.703319	0.711549	0.615606	0.715143
1816796431	3	0.712799	0.702518	0.707840	0.694899	0.702445	0.704794	0.712483	0.625044	0.715677
1816796431	4	0.712349	0.705451	0.709918	0.696006	0.705180	0.706928	0.712685	0.614356	0.714875
...	...	...	...	...	...	...	...	...	...	...

Fig. 6. Example data representation of the phrase “3 creekhouse”, mapped from [train/supplemental]_metadata.csv to the located parquet file containing the data.

3.2 American Sign Language Recognition Using Transformer Architecture

In this section, we will discuss initial results using the Google Fingerspelling Dataset with Transformer architecture to predict ASL characters.

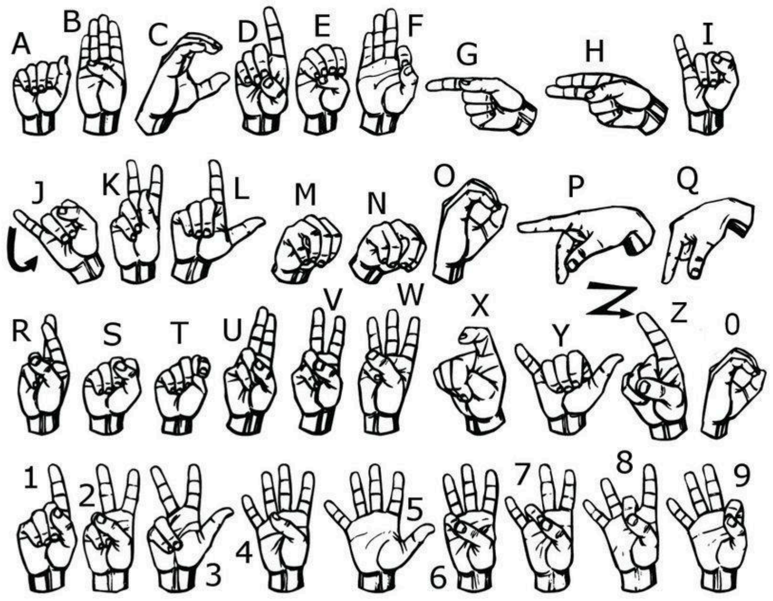


Fig. 7. American Sign Language Alphabet and Numbers.

The goal of the experiment is to create a Transformer encoder - decoder that takes the input of digitized body movements (coordinates), predicts the underlying characters that are spelled, and outputs a sequence of text (see Fig. 7). To get coordinates of body movements, we feed a video input to Google Mediapipe API that outputs the coordinates of pre-defined landmarks. An action is represented using a sequence of many frames, which are all digitized into pre-defined landmark coordinates.

For experimentation, we built upon a Transformer model by author Francois Chollet[2], and Mark Wijkhuizen[3] for his work on Kaggle. Despite the fact that the provided data from Google is holistic with 543 landmarks for face, pose, left hand, and right hand, in the context of real-time

[2] Francois Chollet.: "English-to-Spanish translation with a sequence-to-sequence Transformer", https://keras.io/examples/nlp/neural_machine_translation_with_transformer/, last accessed 2023/11/21.

[3] Mark Wijkhuizen, https://www.kaggle.com/code/markwijkhuizen/aslfr-transformer-training-inference, last accessed 2023/11/29.

inference over the network - that is too much data the system needs to process - given that the system will need to write and process the data as part of our API at the rate of multiple requests per second. Therefore, we preprocessed the data to only include lips landmarks and left and right hands, because most movements in each frame are recorded in those parts, by extracting the specific landmarks using their own indices. The specific landmarks will also be our inference argument.

The Transformer model employs an attention mechanism, which sets it apart from traditional Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs) that only process temporary sequential data dependencies. This mechanism enables the evaluation of relationships among all elements in a sequence, assigning weights to each based on their significance and relevance to the specific task.

A Transformer model consists of two main parts: the Encoder and the Decoder. The Encoder's function is to extract pertinent information from the input data, utilizing the attention mechanism to concentrate on the crucial segments of the input. This is achieved by layering the Encoder with multiple levels, incorporating multi-head attention layers and dense neural network layers to gather and analyze the contextual information from the input. Conversely, the decoder's job is to produce the required output, for example, converting the input into a different language. It too employs attention mechanisms, focusing on the original input using masked attention, as well as the output produced thus far. This enables the model to create output words or tokens that are contextually relevant and based on the words generated previously. Normalization and regularization strategies, like layer normalization and dropout, are implemented in every layer of both the encoder and decoder. These methods are designed to enhance the stability and consistency of the learning process, aiding in the model's ability to generalize and reducing the risk of overfitting. A table of key variables for our model is defined below.

Table 1. Important variable definition for Transformer architecture

Variable	Definition
LAYER_NORM_EPS	Epsilon value used in the layer normalization
UNITS_ENCODER/ UNITS_DECODER	Size of the final output and the embeddings of the encoder and decoder
NUM_BLOCKS_ENCODER/ NUM_BLOCKS_DECODER	Number of blocks (layers) in the encoder and decoder
NUM_HEADS	Number of attention heads in the multi-head attention mechanism
MLP_RATIO	Multiplication factor used to calculate the size of the feed-forward layer
EMBEDDING_DROPOUT, MLP_DROPOUT_RATIO, MHA_DROPOUT_RATIO, CLASSIFIER_DROPOUT_RATIO	The dropout rates used in different parts of the mode
GELU	GELU (Gaussian Error Linear Unit) activation function

The following model is used for training, inspired by Francoi Chollet in his work of Spanish translation with the Transformer model as mentioned, and Mark Wijkhuizen in his related work of American Sign Language recognition.

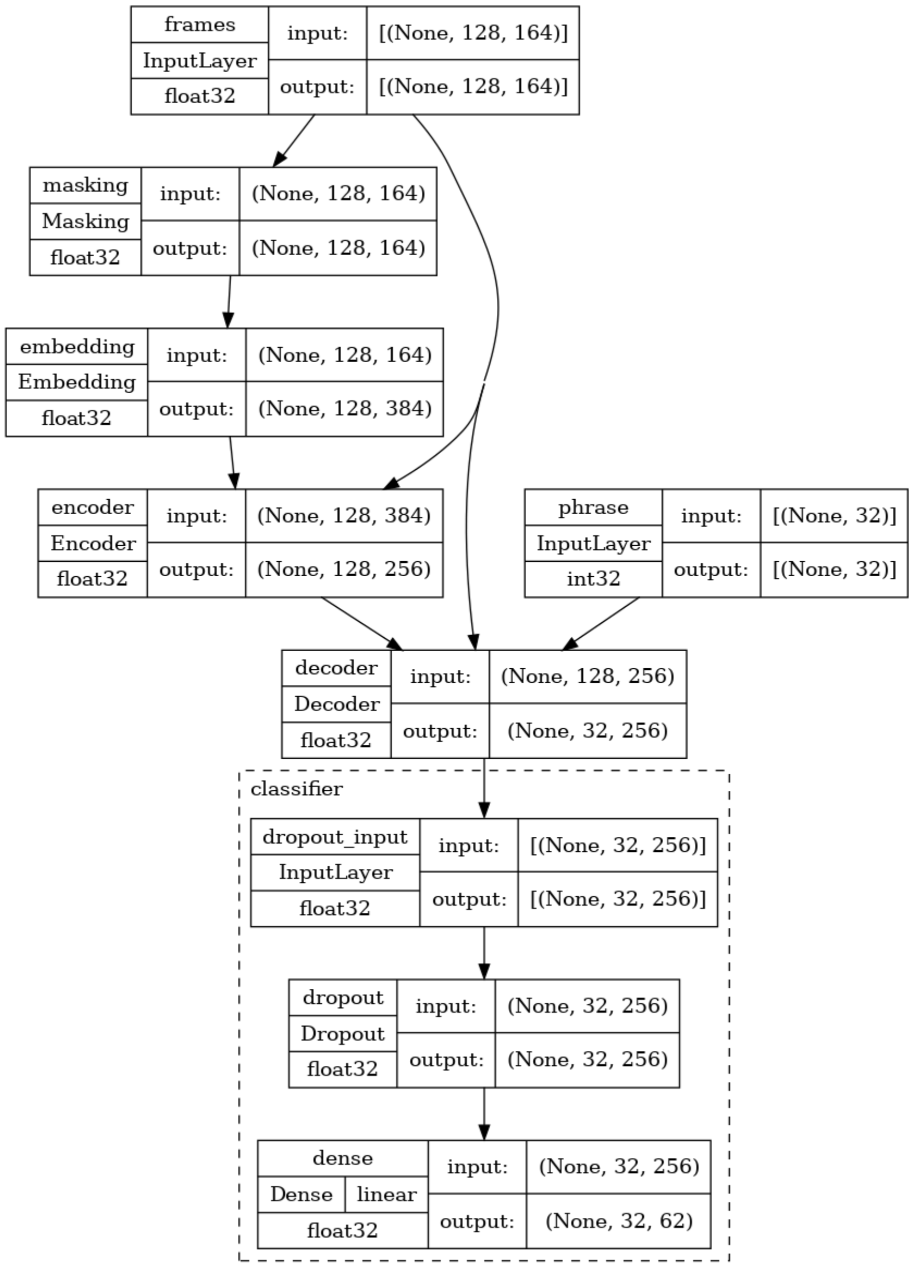


Fig. 8. Transformer model architecture with Embedding, Encoder, and Decoder classes.

Key classes of the model:

Frames: The first input layer of the model, containing the numerical sequence data taken from parquet files, the structure is specified in Section 3.1.

Phrase: The second input layer of the model, containing sequences of labels as specified from train_metadata.csv, the structure is specified in Section 3.1.

Masking: The layer is applied to ignore null values in the frame input layer.

Embedding: Class is used to create an embedded representation of each frame by normalizing input data and generating an embedded representation of landmarks with positional information. Positional information is used to specify the position of each character in relation to the sequence, which is used by the model to recognize patterns.

Encoder: takes an input sequence and transforms it into a vector representation of the input. Inside the Encoder class, there are attention blocks, each including two normalization layers, a multi-head attention layer, and multi-layer perceptrons. The output embeddings from the Encoder have weights, either negative or positive, to magnify or reduce the contribution of certain features of raw input.

Decoder: takes input embeddings from Encoder class to generate output sequence. Causal mask attention is applied to capture dependencies. Normalization and multi-layer perceptron techniques are applied to generate meaningful output based on the information of the Encoder.

We trained 90% of the dataset for 150 epochs and used 10% for validation. A validation set is used after training per each epoch. After training, the weighted model is exported into the TFLite module and in .h5 format for inference purposes. Model training results are discussed in a later section.

3.3 Real-time Inference and Application Programming Interface (API) Development

In this section, we discuss the development of a Flask-based backend API to enable real-time inference for American Sign Language (ASL) interpretation in video conferencing applications. The primary aim is to seamlessly integrate our trained model into production environments, facilitating live interpretation during video calls. Leveraging Flask's lightweight and Python-native framework, we implement an API architecture that enables communication between client devices and the server for efficient model inference.

a) Server Design

This section discusses the methodology behind server design for real-time preprocessing and model inference.

The end goal of the project is to be able to use the model in production for the context of video conferencing. To achieve this goal, model inference has to be done via API to leverage the request-response architecture of HTTP communication. Flask framework is an excellent choice for the development of our back-end API because it is native in Python, very lightweight, and has supported libraries for API development.

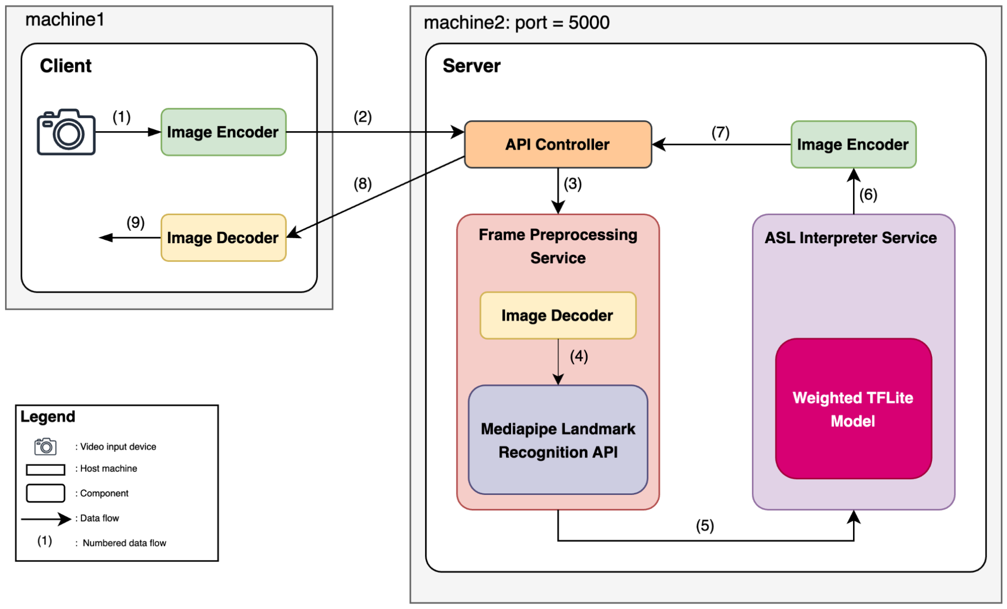


Fig. 9. Back-end Architecture for American Sign Language Real-time Inferencing Flask Server.

The workflow for live-inferencing from our client to the Flask Server is described below:

1. The client video input device captures a frame from the input frame, then the frame is encoded to base64 using an image encoder.

2. The client sends an HTTP POST request to the Flask server, payload contains a base64 encoded frame. The server receives client requests from the API Controller. Here, a buffer is implemented to capture a fixed number of frames before sending them to the next service. The buffer size is set to 128 frames.

3. Server API Controller sends base64 encoded frames to Frame Processing Service, which first decodes base64 image to RGB image.

4. RGB image is processed by Mediapipe Landmark Recognition API, outputting numerical landmark data, standardized as [x/y/z]_[type]_[landmark_index], similar to sequence input data explained in Section 3.2.

5. The sequence of landmark coordinates is transferred to ASL Interpreter Service, which inference to the Weighted TFLite Model occurs - outputting the predicted character.

6. The predicted character, combined with the output of Mediapipe Landmark Recognition API from step (4) constructs a full output image with a bounding box and predicted character. The output image is encoded to base64 by the Image Encoder, ready to be part of the response payload.

7. API Controller receives base64 encoded image output, and predicted character.

8. API Controller responds to the client with payload, including base64 inference output image and predicted character

9. The client receives the server response, parses the predicted character, and displays it to the front-end. Optionally, users can reconstruct base64 inference output from server response to debug.

b) Server Real-time Inference Testing

This section discusses the methodology to independently test the implemented server. For local testing purposes, a client script called "live_test.py" was developed in Python, which captures and sends frames continuously from the camera stream and sends a POST request with a base64 encoded image to our Flask server.

Debugging images can be reconstructed at the client process as in Fig. 10 below. As we can see, the Flask server is working properly, and the predicted character and inference image can be retrieved on the client's side.

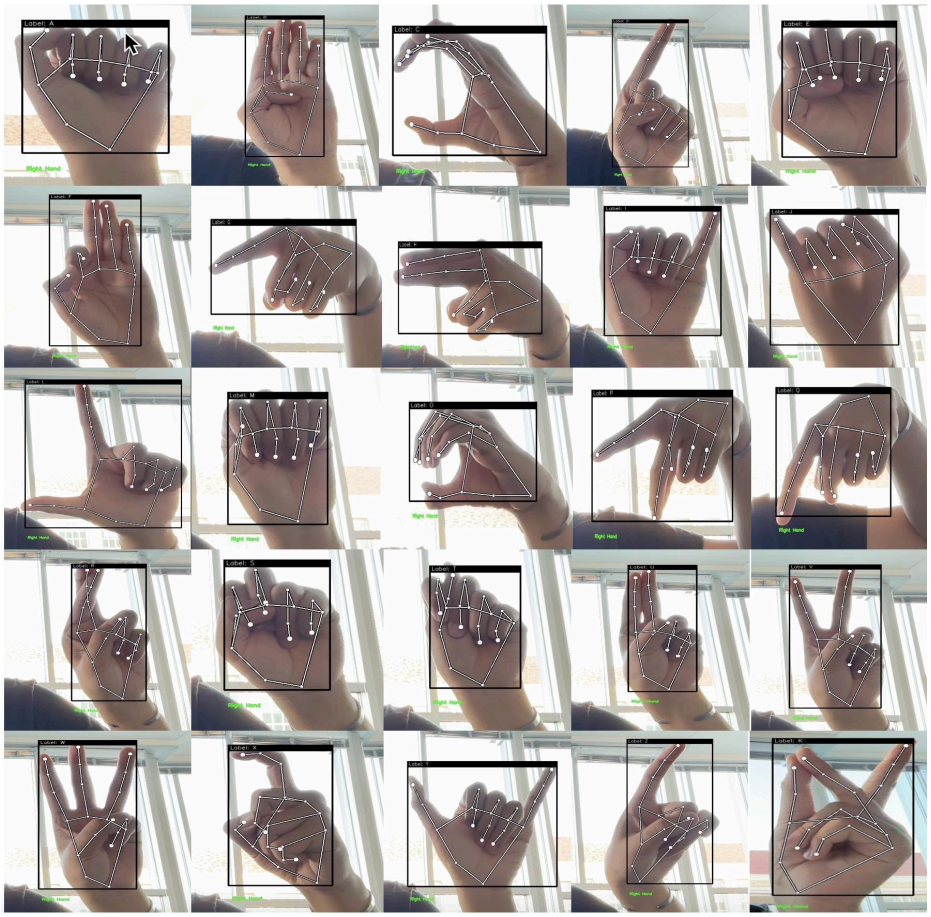


Fig. 10. Predicted ASL alphabet on reconstructed image from base64 at client's process.

3.4 Video Conferencing Application and Model Integration using AgoraRTC

This section discusses the implementation of a basic video conferencing application using AgoraRTC SDK, and server integration. The implemented video conferencing application allows users to perform meetings with real-time images and audio; chat functionality is also included. While video conferencing services also provide their own SDK, for example: Zoom SDK, the implementation is boxed and not easy to interfere with video layers. Instead, we decided to use AgoraRTC SDK, which is a software development kit that facilitates real-time communication (RTC), including video calls and chat. Along with HTML, JavaScript, and CSS, we were able to create a video conferencing application of our own, which includes all three basic functionalities that AgoraRTC SDK offers.

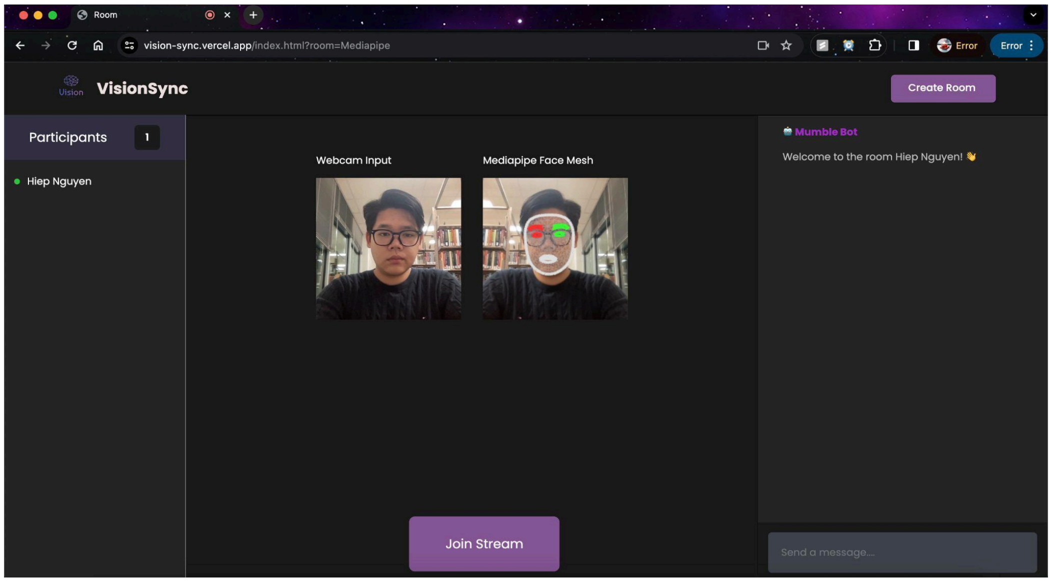


Fig. 11. Video conferencing web application using AgoraRTC.

We integrated the Flask server for real-time inference, and the workflow for end-to-end integration with the front application is described in the diagram below. Video input is captured in frames, and sent through a Base64 Encoder to serve as payload for HTTP POST requests to the Flask server mentioned above. Server responses with predicted text for gesture performed which can be placed directly onto the video stream, which is sent to the AgoraRTC socket to the other caller. The key important feature is that the process runs asynchronously with the workflow of the main video input to the AgoraRTC socket (refer to Fig. 12.), ensuring video image is still delivered if the server is corrupted, and also provides a simple context switch between normal and ASL assisted mode.

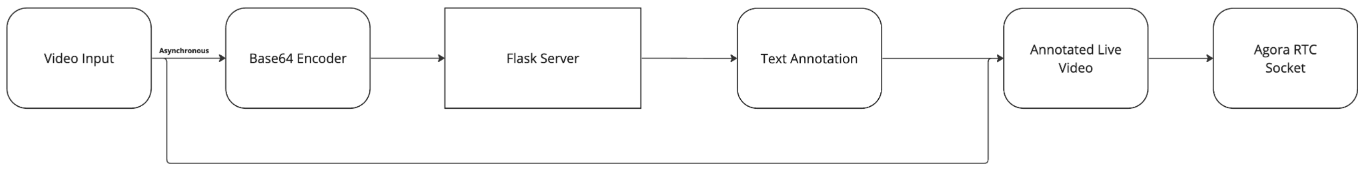


Fig. 12. End-to-end ASL Recognition Model Integration with Video Conferencing Application.

4 Results and Discussion

In this section, we present the outcomes and insights derived from our innovative approach to integrating ASL recognition into video conferencing applications. First, we will discuss the results of model performance as an isolated module, and then we will discuss the performance of the system as a whole.

4.1 Transformer Model Training Results

This section provides an overview of our model's training process, focusing on its performance metrics such as model loss, top-one accuracy, and top-five accuracy across 150 epochs.

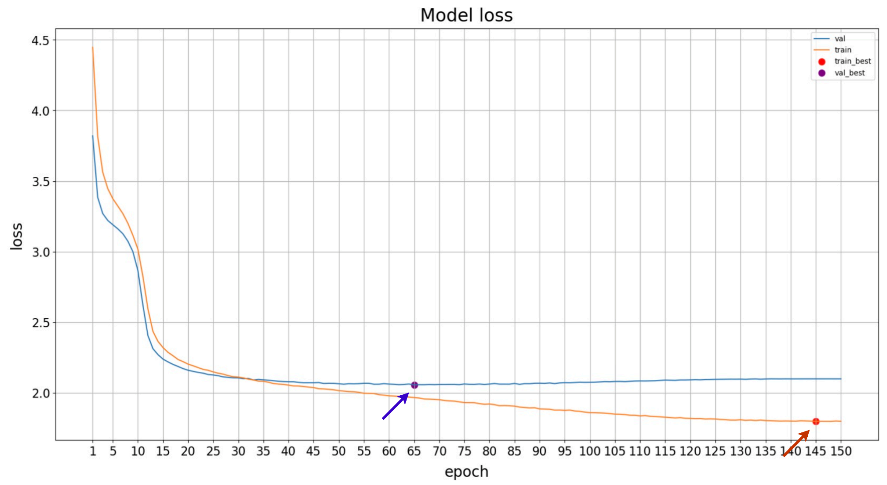


Fig. 13. Model loss visualized graph on the training dataset and validation dataset over 150 epochs.

Refer to Fig. 13., we can see that model loss decreases over each training epoch. After each epoch, the validation set is used to test model loss. Even though that model loss for training kept on decreasing until epoch 150, its validation loss shows a plateau around epoch 65 after reaching its peak. This is a limitation to be discussed in Section 5.

To assess the accuracy of the model across both training and validation datasets, we utilize two key metrics: top one and top five accuracy. The top one accuracy is the frequency with which the model's prediction exactly matches the expected label. On the other hand, the top five accuracy measures how often the expected label aligns with one of the five highest probability classes predicted by the model. In terms of top-one accuracy (see left image of Fig. 14), the model reached its peak on the training dataset, achieving an 88% accuracy rate by epoch 145. However, on the validation dataset, the peak was observed at 78% by epoch 80, then plateaued after that. This pattern mirrors that of the top five accuracy metrics (see right image of Fig. 14). The training dataset saw its highest accuracy of 96% at epoch 150, while the validation dataset peaked at 92% by epoch 70 before stabilizing.

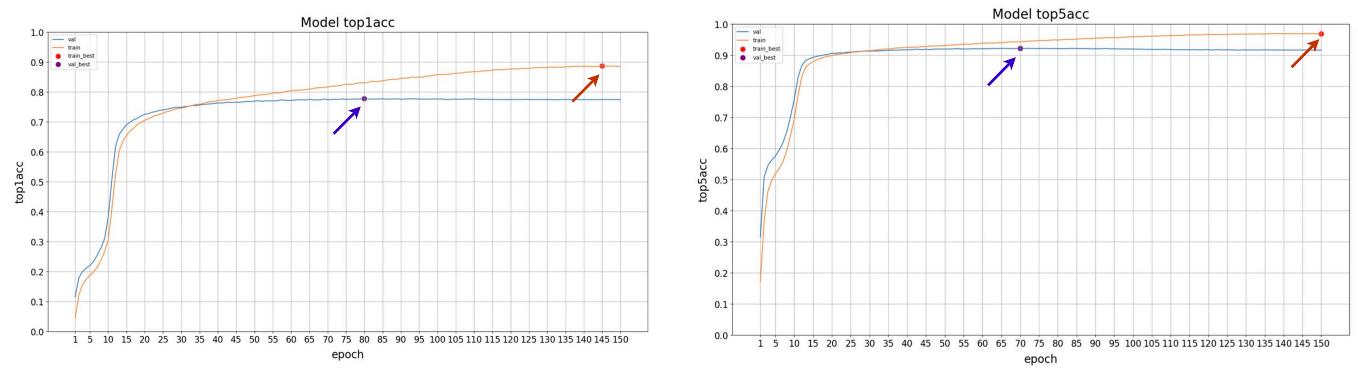


Fig. 14. Model accuracy over 150 epochs. Top one accuracy (left), and top five accuracy (right) are displayed for both the training and validation datasets, as well as the peak of both.

Levenstein distance is used to measure the similarity between the full predicted phrase and the label. Referring to Fig. 15, the Levenstein distance distribution for the train dataset achieves a mean of 3.6494, with more than 380 correctly predicted phrases.

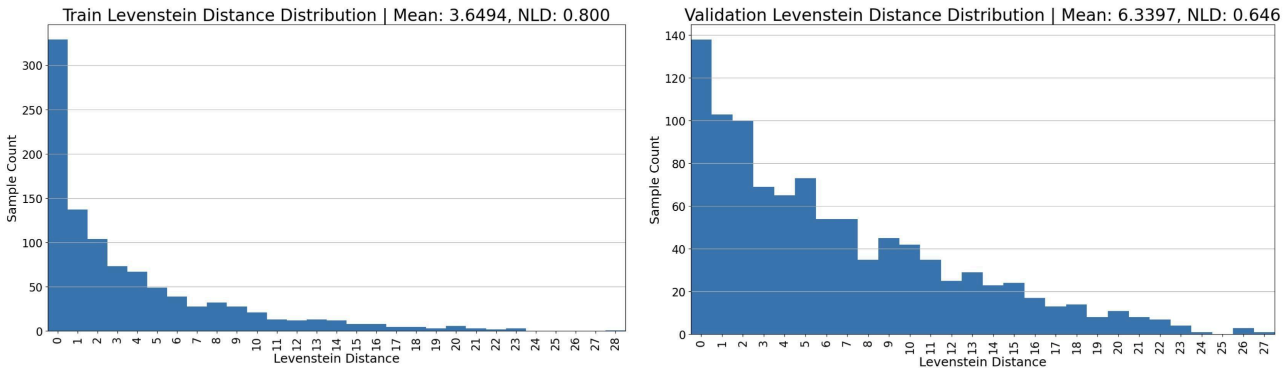


Fig. 15. Levenstein Distance Distribution for predicted phrase on train dataset (left) and validation dataset (right).

For the validation dataset, we achieved almost 140 correctly predicted phrases, with a mean of 6.3397 for the metric. This is an optimistic result, given that we are measuring the full phrase, composed of many characters, and comparing it with the given label phrase.

4.2 End-to-end System Result Discussion

In this section, we present a thorough analysis of the outcomes yielded by our end-to-end system. Through the development and integration efforts, we have successfully implemented a robust end-to-end integrated system that incorporates our Transformer model, enabling the real-time transcription and display of American Sign Language (ASL) fingerspelling gestures. The product is showcased in the resulting video, where viewers can witness the seamless operation of our system. Notably, our system demonstrates its efficiency by streaming transcribed ASL fingerspelling data onto the client side in real-time with almost no latency, paired with asynchronous fall-back described in Fig. 12, thereby enhancing the communication experience within a video conference setting.

Furthermore, our achievement signifies the practical applicability and effectiveness of our integrated approach. By facilitating the seamless integration of ASL fingerspelling transcription into digital communication platforms, we address critical accessibility needs for individuals relying on sign language. Through our system's capabilities, we aspire to bridge communication gaps and empower users to engage more inclusively in virtual interactions. The successful implementation of our end-to-end solution not only showcases the potential of advanced technologies like the Transformer model but also signifies a significant step forward in leveraging AI for enhancing accessibility and fostering inclusive communication environments.

4.3 Limitations and Discussion

In this section, we discuss the current limitation of the implemented Transformer model and our end-to-end approach to integrating ASL gesture prediction to production.

During implementation and training, our first approach was to use the entirety of the provided landmark coordinates, only omitting null values using masking. However, with inference arguments being 543. numerical values, in addition to many unnecessary features, we decided to omit most facial features (only leaving lips coordinates), and pose features since most gesture movements are from the hands.

While our trained Transformer model exhibited a plateau in model loss for the validation dataset around epoch 65, contrasting with the consistent decrease observed in the training dataset, it's worth noting that our model achieved significant milestones. Despite attaining a peak top-one score of 78% and a top-five score of 92% for the validation dataset, these results are particularly promising considering the diversity and scale of the dataset, sourced from over 150 signers across various contexts. Hence, when integrated into our end-to-end system, the model demonstrates robust performance and functionality.

For system end-to-end integration, the most notable limitation is API optimization for memory. Our first approach was to design our API in a cascading style, meaning objects will be created as data travels from the controller to individual components (refer to Fig. 9). However, once we tested our server with a client process, we quickly realized that the approach would lead to the memory stack being full, thus stopping our server. We later had to optimize by invoking Mediapipe globally to avoid stack overflow by recreating process threads.

5 Conclusion

In this section, we conclude our experiment with end-to-end integration for ASL in the context of video conferencing and discuss future work to extend the current approach to word-level ASL considering the scalability and usability of the system in production.

In this paper, we have presented an innovative end-to-end solution for bridging communication gaps faced by individuals with speech disabilities, particularly focusing on American Sign Language (ASL) translation in real-time video conferencing settings. Our approach integrates advancements in Google Mediapipe, Transformer models, and web technologies to create a comprehensive solution that enables the translation of ASL gestures into text subtitles with high accuracy and low latency.

Furthermore, our research contributes to the broader imperative of leveraging AI applications for human well-being, particularly in enhancing accessibility and inclusivity for individuals with disabilities. By showcasing the potential of advanced technologies like Transformer models in addressing real-world challenges, we aim to promote the integration of AI-driven solutions in applications designed to enhance human wellness.

However, our work also highlights certain limitations, particularly in the optimization of the API for memory efficiency. Future research could focus on improving the model's accuracy, potentially extending it to word-level ASL recognition, and optimizing the system's architecture for scalability and usability in production environments.

In conclusion, our project represents a significant step forward in leveraging AI technologies to address accessibility challenges, and we hope it will inspire further innovation and research in this important area.

References

1. World Health Organization. “Deafness and Hearing Loss.” World Health Organization, 28 Feb. 2024, http://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss, last accessed 2024/02/28.

2. Starner, Thad, and Alex Pentland.: "Real-time American sign language recognition from video using hidden Markov models." In: Proceedings of International Symposium on Computer Vision-ISCV. IEEE, 1995. https://doi.org/10.1109/ISCV.1995.477012.

3. Chong, Teak-Wei, and Boon-Giin Lee.: "American sign language recognition using leap motion controller with machine learning approach." Sensors 18(10), 3554 (2018). https://doi.org/10.3390/s18103554.

4. Srivastava, Sharvani, et al.: "Sign language recognition system using TensorFlow object detection API." In: Proceedings of the International Conference on Advanced Network Technologies and Intelligent Computing. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-96040-7_48.

5. Sundar, B., and T. Bagyammal.: "American Sign Language Recognition for Alphabets Using MediaPipe and LSTM." In: Procedia Computer Science 215, pp. 642-651 (2022). https://doi.org/10.1016/j.procs.2022.12.066.

6. MediaPipe Solutions guide, https://developers.google.com/mediapipe/solutions/guide, last accessed 2024/02/21.

7. Hand landmarks detection guide, https://developers.google.com/mediapipe/solutions/vision/hand_landmarker, last accessed 2024/02/21.