Final Year Project Documentation 6308.docx

Nile University of Nigeria

Document Details

Submission ID
trn:oid:::26704:260249809

Submission Date
Jan 21, 2025, 2:15 PM GMT+1

Download Date
Jan 21, 2025, 2:18 PM GMT+1

File Name
Final Year Project Documentation 6308.docx

File Size
7.1 MB

80 Pages

12,756 Words

78,683 Characters

18% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Small Matches (less than 8 words)

Match Groups

174 Not Cited or Quoted 15%
Matches with neither in-text citation nor quotation marks

33 Missing Quotations 3%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

1 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

8% Internet sources
5% Publications
17% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **174** Not Cited or Quoted 15%
  Matches with neither in-text citation nor quotation marks
- **33** Missing Quotations 3%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **1** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 8% Internet sources
- 5% Publications
- 17% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Submitted works	Baze University on 2024-09-24 <1%
2	Submitted works	Baze University on 2024-09-27 <1%
3	Submitted works	Baze University on 2024-09-22 <1%
4	Submitted works	Higher Education Commission Pakistan on 2024-12-13 <1%
5	Internet	www.mdpi.com <1%
6	Internet	www.coursehero.com <1%
7	Internet	enuii.org <1%
8	Internet	link.springer.com <1%
9	Internet	www.techscience.com <1%
10	Submitted works	Bournemouth University on 2023-09-15 <1%

11	Submitted works	
Debre Berhan University on 2024-11-20		<1%
12	Submitted works	
University of Northumbria at Newcastle on 2024-05-09		<1%
13	Submitted works	
RMIT University on 2024-04-07		<1%
14	Submitted works	
Baze University on 2022-09-19		<1%
15	Submitted works	
SRM University on 2023-11-01		<1%
16	Submitted works	
University of Westminster on 2024-11-11		<1%
17	Submitted works	
University of Hertfordshire on 2024-09-02		<1%
18	Internet	
ebin.pub		<1%
19	Publication	
H.L. Gururaj, Francesco Flammini, S. Srividhya, M.L. Chayadevi, Sheba Selvam. "Co...		<1%
20	Submitted works	
University of Hertfordshire on 2025-01-06		<1%
21	Submitted works	
Anna University on 2024-03-13		<1%
22	Submitted works	
Baze University on 2017-08-21		<1%
23	Submitted works	
Baze University on 2022-09-08		<1%
24	Submitted works	
Manchester Metropolitan University on 2024-09-27		<1%

25 Internet
www.geeksforgeeks.org <1%

26 Submitted works
INTI University College on 2010-07-08 <1%

27 Submitted works
KDU College Sdn Bhd on 2020-04-20 <1%

28 Submitted works
University of Portsmouth on 2024-05-20 <1%

29 Submitted works
WQE College on 2024-11-25 <1%

30 Internet
digitallibrary.tulane.edu <1%

31 Internet
www.irit.fr <1%

32 Publication
"Proceedings of Data Analytics and Management", Springer Science and Business... <1%

33 Submitted works
Belhaven University on 2024-11-17 <1%

34 Submitted works
Higher Education Commission Pakistan on 2024-11-11 <1%

35 Submitted works
Indiana University on 2023-05-02 <1%

36 Internet
portal.bazeuniversity.edu.ng <1%

37 Submitted works
Anna University on 2020-03-30 <1%

38 Submitted works
University of Wollongong on 2023-10-26 <1%

39 Publication

Thoong Chun Onn, Nayef.A.M. Alduais, Jiwa Abdullah, Abdul-Malik H.Y. Saad et al.... <1%

40 Submitted works

Sim University on 2024-09-02 <1%

41 Submitted works

University of Birmingham on 2024-09-02 <1%

42 Internet

www.momentslog.com <1%

43 Submitted works

Asian Institute of Technology on 2024-11-28 <1%

44 Submitted works

Baze University on 2022-09-17 <1%

45 Submitted works

Indiana University on 2024-05-01 <1%

46 Submitted works

Baze University on 2016-01-11 <1%

47 Submitted works

CSU Northridge on 2023-04-08 <1%

48 Submitted works

City University on 2013-01-11 <1%

49 Submitted works

Universiti Sains Islam Malaysia on 2023-06-28 <1%

50 Submitted works

University of Greenwich on 2024-12-20 <1%

51 Submitted works

University of Hertfordshire on 2021-06-29 <1%

52 Publication

Yanqiong Zhang, Yu Han, Zhaosong Zhu, Xianwei Jiang, Yudong Zhang. "Artificial ... <1%

53 Internet
etda.libraries.psu.edu <1%

54 Internet
www.researchgate.net <1%

55 Publication
"Machine Learning Techniques for Smart City Applications: Trends and Solutions"... <1%

56 Submitted works
Asia Pacific University College of Technology and Innovation (UCTI) on 2024-07-31 <1%

57 Publication
Bharat Taralekar, Rutuja Hinge, Chaitanya Bisne, Amberish Deshmukh, Vidya Dar... <1%

58 Submitted works
Universiti Sains Islam Malaysia on 2024-01-21 <1%

59 Submitted works
University of Central Lancashire on 2015-04-14 <1%

60 Submitted works
University of Central Lancashire on 2023-03-31 <1%

61 Submitted works
University of Mauritius on 2024-09-19 <1%

62 Submitted works
University of Northumbria at Newcastle on 2021-09-12 <1%

63 Submitted works
University of South Africa on 2024-06-15 <1%

64 Submitted works
University of Sydney on 2021-11-13 <1%

65 Submitted works
University of Westminster on 2023-05-10 <1%

66 Submitted works
Xiamen University on 2023-07-02 <1%

67 Internet
escholarship.org <1%

68 Internet
rucore.libraries.rutgers.edu <1%

69 Internet
streetleverage.com <1%

70 Publication
"Sign Language Machine Translation", Springer Science and Business Media LLC, ... <1%

71 Publication
Amina Ben Haj Amor, Oussama El Ghoul, Mohamed Jemni. "Sign Language Recog... <1%

72 Submitted works
Asia Pacific University College of Technology and Innovation (UCTI) on 2022-02-16 <1%

73 Publication
Asmaa Alayed. "Machine Learning and Deep Learning Approaches for Arabic Sign... <1%

74 Submitted works
Baze University on 2022-09-08 <1%

75 Submitted works
Baze University on 2022-09-21 <1%

76 Submitted works
Baze University on 2025-01-04 <1%

77 Submitted works
Coventry University on 2023-08-08 <1%

78 Submitted works
Leyton Sixth Form College, London on 2025-01-16 <1%

79 Submitted works
Liverpool John Moores University on 2023-03-07 <1%

80 Submitted works
Malta College of Arts,Science and Technology on 2024-06-09 <1%

81 Submitted works
Malta College of Arts,Science and Technology on 2024-09-24 <1%

82 Publication
Muslem Al-Saidi, Áron Ballagi, Oday Ali Hassen, Saad Saad. "Type-2 Neutrosophic ... <1%

83 Submitted works
Ravensbourne on 2024-08-19 <1%

84 Submitted works
Universiti Teknologi MARA on 2015-12-18 <1%

85 Submitted works
University of Calabar on 2015-11-08 <1%

86 Submitted works
University of Greenwich on 2024-09-03 <1%

87 Submitted works
University of Hertfordshire on 2024-08-18 <1%

88 Submitted works
University of Lancaster on 2024-04-27 <1%

89 Submitted works
University of Sheffield on 2023-12-04 <1%

90 Submitted works
University of South Africa (UNISA) on 2024-06-10 <1%

91 Submitted works
University of West London on 2025-01-16 <1%

92 Submitted works
University of Westminster on 2024-04-10 <1%

93 Submitted works
University of Westminster on 2024-11-10 <1%

94 Submitted works
Whitireia Community Polytechnic on 2020-10-16 <1%

95 Internet
docplayer.net <1%

96 Internet
etheses.dur.ac.uk <1%

97 Internet
iriss.idn.org.rs <1%

98 Internet
pdfcoffee.com <1%

99 Internet
www.ntu.edu.sg <1%

100 Publication
Aline Ferreira, John W. Schwieter. "The Routledge Handbook of Translation, Inter... <1%

101 Submitted works
Baze University on 2017-08-03 <1%

102 Submitted works
Baze University on 2020-09-18 <1%

103 Submitted works
Baze University on 2022-09-21 <1%

104 Submitted works
Baze University on 2024-09-25 <1%

105 Submitted works
Baze University on 2024-09-30 <1%

106 Submitted works
Cyberjaya University College of Medical Sciences on 2024-01-18 <1%

107 Publication
Emma Darroch, Raymond Dempsey. "Interpreters' experiences of transferential d... <1%

108 Submitted works
Glasgow Caledonian University on 2025-01-10 <1%

109 Publication

Irit Meir, Wendy Sandler. "A Language in Space - The Story of Israeli Sign Langua... <1%

110 Publication

Laura Gavioli, Cecilia Wadensjö. "The Routledge Handbook of Public Service Inter... <1%

111 Submitted works

Liverpool John Moores University on 2023-04-22 <1%

112 Submitted works

Liverpool John Moores University on 2024-03-18 <1%

113 Submitted works

Lyceum of the Philippines University Batangas on 2024-09-03 <1%

114 Submitted works

Manchester Metropolitan University on 2024-07-29 <1%

115 Submitted works

Middlesex University on 2024-09-27 <1%

116 Submitted works

Midlands State University on 2022-07-18 <1%

117 Submitted works

Monash University on 2024-06-11 <1%

118 Submitted works

Multimedia University on 2023-06-08 <1%

119 Submitted works

Nanyang Technological University on 2024-06-26 <1%

120 Submitted works

National College of Ireland on 2023-12-14 <1%

121 Submitted works

National University of Ireland, Galway on 2024-08-19 <1%

122 Submitted works

Nottingham Trent University on 2023-08-30 <1%

123 Publication

Oussama El Ghoul, Maryam Aziz, Achraf Othman. "JUMLA-QSL-22: A Novel Qatari ... <1%

124 Submitted works

Rochester Institute of Technology on 2024-12-09 <1%

125 Submitted works

The British College on 2024-05-25 <1%

126 Submitted works

UOW Malaysia KDU University College Sdn. Bhd on 2023-04-23 <1%

127 Submitted works

United Colleges Group on 2024-12-17 <1%

128 Submitted works

Universidad TecMilenio on 2024-11-15 <1%

129 Submitted works

University of Bolton on 2025-01-06 <1%

130 Submitted works

University of Greenwich on 2024-09-06 <1%

131 Submitted works

University of Greenwich on 2024-12-20 <1%

132 Submitted works

University of Hertfordshire on 2024-11-11 <1%

133 Submitted works

University of Lancaster on 2024-08-30 <1%

134 Submitted works

University of Newcastle upon Tyne on 2021-05-25 <1%

135 Submitted works

University of North Texas on 2024-10-08 <1%

136 Submitted works

University of Stirling on 2023-04-19 <1%

137 Submitted works
University of West London on 2025-01-16 <1%

138 Submitted works
University of Westminster on 2023-05-10 <1%

139 Submitted works
Western Oregon University (Moodle) on 2024-09-28 <1%

140 Publication
Ziyang Deng, Weidong Min, Qing Han, Mengxue Liu, Longfei Li. "VTAN: A Novel Vi... <1%

141 Internet
arxiv.org <1%

142 Internet
dokumen.pub <1%

143 Internet
ktisis.cut.ac.cy <1%

144 Internet
mafiadoc.com <1%

145 Internet
pmc.ncbi.nlm.nih.gov <1%

146 Internet
vdoc.pub <1%

147 Internet
www.frontiersin.org <1%

148 Internet
www.ijcse.com <1%

149 Internet
www.ijraset.com <1%

150 Internet
www.semanticscholar.org <1%

151 Internet

zero.sci-hub.ru <1%

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The sign language translator is a machine learning project focusing on the development of a system capable of accurately interpreting and translating American Sign Language (ASL) into written English. This project addresses the communication barriers faced by deaf and non-speaking individuals, as well as those unfamiliar with sign language, by offering a technological solution that enhances interaction among these groups. The proposed system will utilize computer vision techniques to capture video input, specifically targeting hand gestures. These visual cues will be analysed to extract features and patterns associated with various signs. To establish a robust mapping between hand gestures and their corresponding English alphabets, machine learning algorithms will be trained on a comprehensive dataset of sign language samples, as highlighted in recent research on neural sign language translation (Camgöz et al., 2018). The ultimate objective of this project is to create a user-friendly and efficient sign language translator that empowers deaf individuals and promotes inclusivity within society.

1.2 BACKGROUND AND MOTIVATION

Sign language serves as a complex visual language used by millions of deaf individuals globally, encompassing a rich vocabulary and grammar conveyed through hand gestures, facial expressions, and body language. Despite its importance, sign language often presents significant barriers to effective communication with the hearing majority. Traditional communication methods for deaf individuals, such as interpreters and written language, frequently fall short in terms of accessibility, cost, and real-time interaction. For instance, interpreters may not always be available, and written communication can hinder spontaneous conversation, leading to social isolation and limited opportunities for education and employment (Pinilla et al., 2019). The need for technological solutions that facilitate seamless communication between deaf and hearing individuals is increasingly recognized. Recent advancements in computer vision, machine learning, and natural language processing have opened new avenues for developing sign language recognition and

translation systems, which can serve as powerful tools to bridge the communication gap (Zhou et al., 2020).

1.3 STATEMENT OF THE PROBLEM

Effective communication remains a significant challenge for deaf individuals when interacting with the hearing world. Despite technological advancements, barriers persist, particularly in real-time communication without interpreters or when written language is inadequate. Sign language, with its unique grammar and syntax, is essential for the deaf community; however, most hearing individuals lack proficiency, creating the communication divide. Current solutions, such as interpreters or written notes, are often impractical for spontaneous or dynamic interactions, limiting expressiveness and speed. In professional, medical, and educational contexts, reliance on interpreters can lead to delays and miscommunication, placing undue burdens on deaf individuals and hindering their full societal integration (Rachdito & Hidayat, 2022). This project aims to develop a real-time sign language translator that eliminates the dependency on interpreters and enhances accessibility. By leveraging modern technology, it seeks to seamlessly translate sign language into written language, fostering inclusive communication between deaf and hearing communities.

1.4 AIMS AND OBJECTIVES

Aim

To aim of this project is to develop a sign language in real time, to written language, thereby improving communication accessibility for deaf individuals.

Objectives of the system:

1. To design and implement a real-time video processing system capable of capturing and analysing hand gestures.
2. To apply machine learning models for accurate recognition and classification of sign language gestures.
3. To create a mapping between recognized sign language gestures and corresponding linguistic expressions.
4. To develop a user-friendly interface for seamless interaction between deaf and hearing individuals.

1.5 SIGNIFICANCE OF THE PROJECT

This project holds importance because it has the potential to enhance communication between deaf and hearing individuals. A successful sign language translator will foster social inclusion and provide a more accessible and efficient way to communicate.

1.6 PROJECT RISK ASSESSMENT

Table 1.1: Risk Assessment and Mitigation Strategies

RISK	MITIGATION STRATEGY
Insufficient processing power or camera quality affecting system performance	Explore hardware acceleration options (e.g., GPU) Specify minimum requirements for optimal performance and accuracy
Insufficient or low-quality sign language dataset impacting model accuracy	Curate a diverse dataset that covers various signs, lighting conditions and backgrounds
The model may struggle to accurately recognize complex or nuanced signs leading to incorrect translations	Continuously refine the model through iterative testing and improvement

1.7 SCOPE/PROJECT ORGANIZATION

1.7.1 SCOPE

The primary focus of this project is to develop a sign language translation system capable of recognizing and translating basic sign language gestures into text. The system will utilize computer vision techniques to process video input, extract relevant features from hand gestures and employ machine learning algorithms for gesture recognition and translation.

Chapter 2: Literature Review

The second chapter of this report contains a comprehensive survey of previous works related to this project. It includes a review of the key research studies, frameworks and methodologies that have been developed over time and are relevant to the current project. Finally, the chapter outlines the implications of the reviewed literature for the current project, establishing a context for the subsequent requirements analysis and design

Chapter 3: Requirements analysis and Design

This chapter will focus on the requirements analysis and the design of the system. These requirements will be categorized and prioritized based on their importance and impact on the system. Following this, the chapter will describe the high-level design of the system including architecture diagrams and use case models. It will also discuss the choice of technology stack, frameworks and design patterns that best suit the project's objectives. Finally, a detailed breakdown of each component's design will be provided, including their interactions, dependencies and expected behaviour.

Chapter 4: Implementation and Testing

Chapter 4 will cover the implementation and testing phase of the project. It will begin with a detailed account of the development process, including the tools, libraries and frameworks used in coding the application. This section will highlight the challenges encountered and the strategies used to overcome them. Following the implementation details, this chapter will outline the testing strategy, including both unit testing and integration testing. The chapter will conclude with an analysis of the test results, discussing any bugs, issues or deviations from the expected outcomes and how they were resolved.

Chapter 5: Conclusions, Limitations and suggested improvements

86 The final chapter will summarize the key findings and contributions of the project. It will restate the project's objectives, discuss how they were met and highlight the main achievements. The chapter will also reflect on the limitations and challenges encountered throughout the project, including technical, logistical or resource-related constraints. In addition, it will provide suggestions for future work and improvements.

53

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

9 The development of assistive technologies for improving the communication gap between deaf and hearing people has gathered considerable momentum in the past couple of years. Many methods have been tried to make this communication real-time and accessible, from interpreters of sign languages to technological ways. This literature review will discuss the existing body of work related

to sign language translation systems and delve deeper into the various methods and different technologies developed in order to communicate more effectively with the deaf.

This chapter will also provide an extensive review regarding the status quo and the current technological standing regarding research in sign language translation. This begins with a look at traditional sign language communication through either a human interpreter or by means of text-based method and the relevant limitations it entails for real-time uses. Further on, it deals with a report on the technological developments for automating the translation process: sensor-based systems, computer vision algorithms, and machine learning models.

Furthermore, this literature review will analyse the challenges these technologies face, including the complexities of accurately recognizing and interpreting diverse sign language, real-time processing, and ensuring accessibility for different regions and sign language dialects. By reviewing these past and present efforts, the chapter aims to identify the gaps in the current solutions, laying the foundation for the development of a more robust and effective sign language translation model proposed in this project.

2.2 Historical overview

The historical overview of sign language translation shows how language has evolved and how technology has advanced. Initially regarded as an imperfect communication system, sign language gained recognition as a legitimate natural language through the pioneering work of William Stokoe in the 1960s, which laid the groundwork for understanding its structure and significance (Yang, 2019). The emergence of sign language interpreters has been pivotal in advocating for the rights of the deaf community, highlighting their crucial role in bridging communication gaps (Arssi & Taibi, 2018). Furthermore, advancements in technology have facilitated the development of systems that translate sign language into spoken language and vice versa, enhancing accessibility for the hearing impaired (Fernando & Wimalaratne, 2016; Olabanji & Ponnle, 2021). Recent innovations such as machine learning applications, demonstrate the potential for real-time translation, although challenges remain regarding the comprehensiveness of sign language dictionaries (Lin & Murli, 2022). This overview will explore these developments, emphasizing the interplay between linguistic diversity and technological progress in the field of sign language translation.

2.2.1 Sign language

The history and evolution of sign language interpreters is a complex narrative that combines linguistic development, social recognition, and professionalization. As natural languages, sign languages have existed for hundreds of years as an essential means of communication by Deaf communities worldwide. The formal recognition of sign languages in general, and American Sign Language (ASL) in particular, did not gain significant momentum until the 20th century with the seminal work of William Stokoe. His landmark 1960 article, "Sign Language Structure: An Outline of the Visual Communication Systems of the American Deaf," recognized ASL as the valid language it was, contrary to the dominant view that sign language was a simplified means of communicating (Yang, 2019). Recognition of ASL as a unique language code initiated the growth of sign language interpreting into a separate profession, with interpreters developing as primary enablers of communication for the Deaf and hearing people.

2.2.1.1 American sign language (ASL)

American Sign Language is a dynamic, complex, and grammatical visual-gestural language that is the dominant communication method utilized by many Deaf people in the United States and parts of Canada. ASL is not a manual representation of English; it contains its own grammar, syntax, and vocabulary, making it an independent, full-fledged language. Linguists like William Stokoe greatly developed recognition of ASL as a valid language when, within the 1960s, he proved that ASL possesses proper linguistic structure, thereby debunking the prevalent belief that signed languages were inferior or even simple in nature compared to spoken ones bochnersal (2019). Some of the linguistic features of ASL are that it makes use of space to deliver meaning, it includes facial expressions to carry grammar, and classifiers that represent nouns and verbs in space. In fact, it is shown that the students learning ASL tend to have their difficulty in determining the contrastive and non-contrastive sign distinction, therefore there should be systematic teaching of the phonological and sensorimotor skills for ASL for proficiency to be attained (Bochner et al., 2011). More complex this will be and so more need there is for special programs of education dealing with unique aspects of ASL. ASL is estimated to be used by approximately 500,000 to 1 million people in the United States, thus it has emerged as one of the most frequently used sign languages in North America today. According to Salagar et al. (2013), this is a reflection of the increasing recognition in society about the Deaf community and a means of accessible communication. In the past twenty years, ASL has gained in popularity as a foreign language provided both in high schools and in colleges, and the

numbers of students enrolling in classes that teach ASL have grown wildly over the course of that period of time (Quinto-Pozos, 2011). This is a function of changing attitudes that are occurring with respect to the role that ASL serves both as a communicative modality and a cultural and linguistic treasure. The interaction between ASL and English is best described as unique and specifically distinct grammatically. For instance, idiomatic expressions in English are often not translated directly in ASL, which may cause misunderstandings during situations like medical assessments or psychological evaluations (Falchook et al. 2013). It is thus impossible to fend for oneself without the intervention of professionally trained interpreters who would work their way through these complexities for the purpose of effective communication between the Deaf and hearing individuals. Secondly, interpreting ASL requires much cognitive and physical energy. Interpreters must possess a high level of proficiency in both ASL and English, as well as the ability to process information rapidly while managing the physical aspects of signing (Donner et al., 2013). This dual demand brings into sharp focus the seriousness of the training required to prepare interpreters for ASL interpretation.

2.2.1.2 Early communication and interpreting practices within deaf communities

Early communication practices within Deaf communities have historically relied on various forms of sign language, gestures, and visual communication methods. These practices have evolved over time, influenced by cultural, social, and educational factors. One of the most important influences on communication in these communities has been the evolution of formal sign languages like American Sign Language (ASL). ASL developed as a unique language beginning in the early 1800s as a result of regional sign languages and the creation of Deaf schools, including the American School for the Deaf in Hartford, Connecticut, which opened in 1817 (Meador & Zazove, 2005). Many Deaf communities have historically developed unique social networks and cultural identities out of necessity because of the lack of communication with hearing people. Deaf people tend to interact through their peers, which can in turn create a sense of community and cultural pride among the Deaf (Pendergrass et al., 2017). That dependence on peer contact shows how significant interpreters are because they are the bridge between the Deaf world and the hearing world.

The role of interpreters has evolved from informal facilitators to trained professionals who possess a deep understanding of both Deaf culture and the nuances of sign language (Santos & Portes, 2019). Interpreters play such a vital role in medical settings. There is a large amount of research that shows

that the lack of qualified interpreters can greatly limit deafs access to vital services, especially in the area of primary health care (Lee et al., 2021). Many deaf patients complain that they are not able to communicate with their doctors and nurses, which can result in miscommunications and improper treatment. According to research, healthcare providers often overestimate their ability to communicate with Deaf patients through lip-reading or written English, yet these two methods are very limited when trying to explain medical information (Lee et al., 2021). This only emphasizes the need for trained interpreters that can provide for an accurate means of communication and assure that Deaf persons receive appropriate health care services. Not just in health care, but interpreters are also vital in educational settings, acting as a conduit for communication between Deaf students and hearing teachers. Past research has shown that Deaf students tend to learn more effectively when they are taught in ASL instead of English transliteration (Marschark et al., 2005). This only goes to show that using interpreters that are simply fluent in sign language, but are also well versed in the educational system and the unique requirements of Deaf students is of the utmost importance. Also, the incorporation of technology has changed the ways Deaf people communicate amongst themselves. With the emergence of smartphones and social media, the ways in which Deaf people interact have changed, making it easier for them to connect with each other and to share information (Tannenbaum-Baruchi & Feder-Bubis, 2017). The transition to this new technology has facilitated the internationalization of sign language, and has allowed Deaf communities all over the world to share their cultural and linguistic resources (Tannenbaum-Baruchi Feder-Bubis, 2017). Also, there are some mobile apps for Deaf people that came out making it easier to communicate with doctors and emergency people (Chong, 2024). To sum it all up, the communication methods used in Deaf communities long ago have paved the way for the growth of sign language interpreting. The importance of interpreters has been growing and becoming acknowledged as a necessity in many different fields like health and education. Deaf communities will continue to fight for their rights and access to services, and trained interpreters and technology will continue to play a major role in facilitating communication and promoting inclusion.

2.2.1.3 Rise of sign language interpreting as a profession

The role of sign language interpreters has dramatically changed since Stokoe's time. Mostly untrained persons who provided informal communication, whereas the demand today for professional interpreting in educational and legal contexts draws on formal training and

accreditation. Research indicates that the quality of interpretation is closely linked to interpreters' training and cognitive skills, such as working memory (Dijk et al., 2011). Such findings have encouraged the creation of training programs and certification processes in certain countries; the forefront in interpreter education is taken by the United States, the United Kingdom, and Australia (Napier, 2004).

Sign language interpreters are put in a rather challenging cognitive position. Such interpreters must exhibit fluency in both source and target languages. They should be capable of fast and accurate processing of information. According to research, interpreters face cognitive load that is high enough to compromise the quality of the interpretation outcome. The high physiological demand of interpreting, involving a high rate of repeated upper extremity motions, is being related to musculoskeletal disorders. Moreover, there are ergonomic concerns when interpreting both in training and practice that have to be addressed. Knowledge of the physical and cognitive dimensions of interpreting has led to recommendations that programs of training place as much emphasis on language proficiency as on managing interpreter stress and fatigue.

Sign language interpreting has also been influenced by many sociocultural factors in its professionalization. For example, the increasing demands of the Deaf community to have equal access to communication have seen interpreters take on more visible and recognized work, such as during media broadcasts in emergencies, for instance. The utilization of interpreters on numerous newscasts during Australia's and New Zealand's natural disasters in their countries suddenly flipped the switch in public perception and showed how interpreters were able to deliver information to Deaf people that was crucial (McKee, 2014). This visibility of the interpreters has helped build awareness of qualified interpreter needs for public services and further cemented the interpreter's role as an essential member of the communication process.

The development in technology has opened up new dimensions to sign language interpretation. The development of computer-aided interpretation systems and avatar-based interpreters in sign languages represents a great leap forward in the promotion of better communication between Deaf and hearing communities (Olabanji & Ponnle, 2021; Oh et al., 2017). This therefore opens ways of easier access to new areas of investigation and innovation in the field of sign language interpreting. As this field continues to evolve, integrating technology within interpreting practices seems to be

one of the major contributing factors that will shape the future in terms of communication access for Deaf people.

2.2.1.4 Technology and interpreting: Early Innovations

Recent advancements in technology for sign language interpreting have significantly transformed communication within Deaf communities. Previously, Deaf individuals mainly relied on face-to-face interactions and manual communication methods. However, technological progress has created new opportunities to improve accessibility and understanding. One of the first significant innovations was the introduction of video relay services (VRS), which enabled Deaf individuals to connect with hearing people through a sign language interpreter via video calls. This development not only made real-time communication possible but also offered a more natural interaction compared to older text-based methods (Hughes et al., 2004). As technology continued to evolve, the emergence of computer vision and machine learning techniques became essential for sign language recognition. These advancements aimed to close the communication gap between Deaf and hearing individuals by automating the interpretation process. For instance, the use of convolutional neural networks (CNNs) has shown promise in recognizing sign language gestures and translating them into text or speech (Olabanji & Ponnle, 2021; Sevli & Kemaloğlu, 2020). Such systems are designed to capture the dynamic nature of sign language, which includes not only hand movements but also facial expressions and body language, essential for conveying meaning accurately (Huamani-Malca, 2018). Recent research has focused on creating comprehensive datasets that reflect the fluidity of sign language communication. Traditional datasets often concentrated on isolated signs or letters, which do not adequately represent the continuous nature of sign language conversations (Ghoul, 2023). The development of continuous sign language datasets, such as the JUMLA-QSL-22 dataset for Qatari Sign Language, is a significant step toward improving recognition systems that can handle real-world signing scenarios (Ghoul, 2023). These advancements are crucial for creating more effective and nuanced technologies for sign language recognition. Additionally, the use of virtual reality (VR) in sign language interpretation has become an exciting area of research. VR can provide immersive environments that improve the learning and practice of sign language, enabling users to interact with virtual interpreters in a controlled space (Tju, 2024). This innovation not only supports the training of interpreters but also offers Deaf individuals new opportunities to communicate and engage in various settings. Furthermore, assistive technologies like smart gloves with sensors have been

developed to aid in sign language interpretation. These gloves can track hand movements and convert them into text or speech, offering an alternative communication method for Deaf and hearing individuals (Elmahgiubi et al., 2015).

2.2.2 Artificial Intelligence

The history of AI in the context of sign language interpretation started off with great leaps that revolutionized the process of communication for Deaf and hard-of-hearing persons. Back to the middle of the 20th century, it was a time when the roots of AI began; these were mostly symbolic reasoning and rule-based systems. However, it was only with the advent of machine learning and deep learning in the 21st century that AI made significant gains into the field of sign language interpretation.

Early applications of AI in sign language interpretation primarily involved gesture recognition and the development of systems that could translate sign language into text or spoken language. These systems relied on computer vision techniques to analyze hand movements and facial expressions, which are critical components of sign language communication (ZainEldin, 2024). The introduction of deep learning algorithms and convolutional neural networks have raised the bar for the accuracy and efficiency of sign language recognition systems by a mile. These advances enabled applications that could interpret American Sign Language and other sign languages in real time with much greater accuracy (Papastratis et al., 2021).

Another innovation in this sphere has been video-based systems that employ AI to realize communication between the Deaf and hearing persons. Video relay services(VRS) have recently gained momentum whereby a Deaf user can now have any hearing person relay communication through an intermediary interpreter on video calls. This technology has been instrumental in providing access to essential services, such as healthcare and education, where effective communication is crucial (Saladin & Hansmann, 2008). The integration of AI into these systems has enhanced their functionality, enabling features like automatic sign language recognition and translation, which further streamline the communication process (ZainEldin, 2024).

Recent research has explored the application of advanced AI models, such as the Swin Transformer architecture, to improve the adaptability of sign language recognition systems across different sign languages, including ASL and Taiwan Sign Language (TSL) (Kumar, 2024). These models aim to

create a universal platform for sign language interpretation, facilitating communication for diverse Deaf communities. The emphasis on creating educational frameworks that facilitate the learning and understanding of sign language through AI technologies is gaining momentum, showcasing how AI can improve educational outcomes for Deaf individuals (Kumar, 2024). Additionally, AI is being incorporated into assistive devices, like smart gloves that have sensors to track hand movements and convert them into text or speech. These devices aim to empower the Deaf through equipping them with tools of communication that are independent. According to the arguments by Burhani & Prasetyo (2023), the development of the need for available and accessible technologies, which improve the unique requirements for the Deaf community, emerges from current research.

2.2.3 Computer vision

Computer vision has become the latest trend in this area of sign language interpretation, which effectively helps the Deaf community communicate with the hearing world. The research on computer vision has been directed toward gesture recognition, which is a very crucial step in the goal of achieving fairly accurate sign language interpretation. Earlier efforts in this direction used simple image processing methods, but due to recent developments in machine learning and deep learning, sign language recognition systems work significantly better.

An important development in the field of computer vision, however, is the use of a convolutional neural network for sign language interpretation. These deep learning architectures are quite effective at handling visual inputs; therefore, they easily enable the detection of complex hand gestures and movements within sign languages (Sevli & Kemaloğlu, 2020; Bantupalli & Xie, 2018). Research has shown that convolutional neural networks (CNNs) can attain substantial accuracy in identifying gestures associated with American Sign Language (ASL), thereby enabling instantaneous conversion into text or spoken language (Bantupalli & Xie, 2018). This function is crucial for closing communication divides across different contexts, including educational and healthcare environments.

Recent studies have explored the combination of event cameras and spiking neural networks in further improving sign language gesture recognition. These technological advances are tailored to capture the intrinsically dynamic nature of sign language, which often involves fast movements that regular cameras might find it hard to understand with accuracy (Chen et al., 2023). Using advanced

sensors and algorithms, researchers develop systems that are able to identify and classify sign language gestures more accurately and quickly.

Apart from CNNs, other machine learning approaches have been applied, such as support vector machines and transfer learning, to develop sign language systems. For example, studies using a latent support vector model showed great classification skills for signs executed near a Kinect sensor, emphasizing the added value of combining computer vision with depth-sensing technologies (Sun et al., 2015). This methodology enhances a more advanced grasp of the gesture, incorporating spatial data.

Additionally, the advancement of hybrid systems that integrate computer vision with electromyography (EMG) signals has demonstrated potential in the identification of sign language gestures. These systems utilize both visual information and signals of muscle activity to enhance precision and reliability in gesture recognition (Rodríguez-Tapia et al., 2019). Such developments underscore the promise of multi-modal strategies in improving communication for individuals who are Deaf.

The influence of computer vision on the interpretation of sign language transcends mere recognition systems. It encompasses the advancement of assistive technologies, including intelligent gloves that record hand movements and convert them into text or spoken language. Such devices enable Deaf individuals to engage in communication with greater independence and efficacy across diverse settings (Ahmed et al., 2018).

The domain of computer vision has considerable achievements within the interpretation of sign languages, enabling finer and much better recognition of gestures. Deep learning techniques, coupled with highly advanced sensors and multi-modal approaches, continue to drive innovations in this area, and may offer further improvements in making communications more accessible to Deaf people and ensuring a more inclusive society.

2.3 Related works

The integration of gesture recognition technologies into sign language interpretation has seen significant advancements in recent years. This progress is largely driven by the development of innovative methodologies and systems that leverage machine learning, computer vision, and sensor

technologies. Below, we discuss seven recent works that contribute to the field of sign language translation through gesture recognition, highlighting their methodologies and implications for enhancing communication accessibility for Deaf and hard-of-hearing (DHH) individuals.

2.3.1 Gesture Recognition Systems for Sign Language Translation

Technology is fundamental to the development of automated sign language translation systems. Various approaches have been explored, ranging from traditional image processing techniques to advanced machine learning models. Early works in this field primarily focused on recognizing static gestures using predefined patterns or hand shapes. For instance, (Marin et al., 2014) implemented a gesture recognition system that utilized basic image processing techniques to identify hand shapes associated with specific signs (Marin et al., 2014). While those systems were precise for basic signs, they fell short in the dynamic nature of sign languages, which are mostly continuous in their motion, using almost every part of a person's face and subtle changes due to context. Deep learning, especially through convolutional neural networks and recurrent neural networks, has developed more articulated systems that handle these challenges. For example, Liao et al. (2019) have proposed a dynamic sign language recognition system based on video sequences using a BLSTM-3D residual network, which scours both spatial and temporal features and hence improves the recognition accuracy for dynamic signs. similarly, Zhang (2022) has tried extending gesture recognition from a video sequence itself by enhancing this with depth image processing, hence pushing the bar further for sign language recognition systems (Zhang, 2022). Despite this progress, occlusion, lighting conditions, and inter-user variability remain common challenges for gesture recognition. Most of the systems designed with RGB cameras have often faced difficulties in a natural environment. For instance, Axyonov et al. (2021) compared camera-based systems with depth sensors like Microsoft's Kinect, highlighting that while depth sensors provide more robust gesture detection, they introduce additional hardware constraints that limit scalability (Axyonov et al., 2021).

2.3.2 Sensor-Based Approaches

In addition to camera-based gesture recognition, sensor-based solutions have been explored. Wearable devices, such as gloves equipped with motion sensors, accelerometers, and gyroscopes, have been developed to capture the precise hand and finger movements involved in sign language. For example, Olabanji & Ponnle (2021) developed a glove-based system that translated American

Sign Language (ASL) into text by tracking the user's hand orientation and movement patterns (Olabanji & Ponnle, 2021). Although these systems provide high accuracy in gesture detection, their reliance on specialized hardware can pose barriers to widespread adoption. Recent advancements in wearable technologies and their integration with wireless communication have addressed some of these concerns. However, the reference provided for this claim does not support the specific example mentioned. Therefore, I will omit the citation for this statement. Nonetheless, it is acknowledged that glove-based systems still face challenges in terms of cost, user comfort, and the need for extensive calibration for different users.

2.3.3 Data-Driven Approaches and Model Training

The availability of large, annotated sign language datasets has been a key driver of progress in this field. Datasets like RWTH-PHOENIX-Weather 2014 and CSL (Chinese Sign Language) have enabled researchers to train deep learning models on diverse sign language expressions. However, the size and quality of these datasets vary greatly across different sign languages, leading to performance issues across models. As discussed by (Wadhawan & Kumar, 2019), models trained on larger datasets tend to generalize better, while those trained on smaller, language-specific datasets often struggle with cross-lingual sign translation. Transfer learning has been proposed as a solution to these issues. Luqman and El-Alfy (2021) explored the use of pre-trained models from general gesture recognition tasks, allowing researchers to fine-tune their systems on smaller sign language datasets, thereby improving performance without requiring large amounts of labelled data (Luqman & El-Alfy, 2021). This approach is particularly promising for under-represented sign languages, where dataset scarcity remains a significant challenge.

2.3.4 Real-Time Translation Systems

Another area of significant interest is the development of real-time sign language translation systems. While many gesture recognition systems perform well in controlled environments, real-time applications introduce additional complexities. For example, Perdana (2021) developed a system capable of translating live video streams into text in real-time, achieving high accuracy under ideal conditions. However, challenges related to latency, computational efficiency, and environmental factors such as lighting and background noise poses a barrier to the widespread adoption of real-time systems. Several works have addressed these limitations by employing lightweight neural network

architectures optimized for mobile and embedded devices. For instance, Verma & Badli (2022) proposed a lightweight CNN model that can run on low-power devices such as smartphones, making real-time sign language translation more accessible (Verma & Badli, 2022). The trade-off, however, is often a reduction in accuracy compared to more complex models running on powerful hardware.

2.3.5 Brain-Computer Interface (BCI) for Sign Language Translation

Brain-computer interfaces (BCIs) represent a groundbreaking approach to facilitating communication for individuals with severe motor disabilities and those who are Deaf or hard of hearing. BCIs are systems that acquire, analyze, and translate brain signals into actionable commands, enabling users to interact with computers or other devices through their neural activity. This technology has shown promise in restoring lost functions and enhancing communication capabilities for individuals who may not be able to use traditional methods, such as speech or sign language (McFarland & Wolpaw, 2018).

Recent research has explored the potential of BCIs for recognizing sign language directly from brain activity. For instance, Mehta et al. (2010) investigated the feasibility of recognizing sign language gestures from brain imaging data, demonstrating that it is possible to decode specific gestures associated with sign language using non-invasive techniques. This approach could revolutionize communication for individuals with degenerative diseases, such as amyotrophic lateral sclerosis (ALS), who may lose the ability to perform physical gestures over time (Mehta et al., 2010).

Wang et al. (2021) advanced further in this field by developing a machine learning framework for decoding Chinese sign language from brain signals. Their study highlighted the rich semantic information contained within sign language gestures, suggesting that BCIs could be trained to recognize complex sign language commands directly from neural activity, thus bypassing the need for physical gestures altogether (Wang et al., 2021). This capability could significantly enhance communication for individuals who are unable to perform gestures due to physical limitations.

The integration of BCIs with existing sign language recognition systems could lead to more inclusive communication solutions. For example, Koller et al. (2018) proposed a hybrid model that combines deep learning techniques with traditional gesture recognition methods, suggesting that BCIs could be incorporated into such frameworks to improve the robustness of sign language translation

systems. By leveraging brain signals, these systems could potentially interpret the user's intent more accurately, even in cases where physical gestures are not feasible.

Despite the promising developments in BCI technology for sign language translation, several challenges remain. The accuracy of BCI systems can be affected by factors such as signal noise and individual variability in brain activity patterns. Additionally, the need for extensive training and calibration for each user poses practical barriers to widespread adoption (McFarland & Wolpaw, 2018). As BCIs continue to evolve, addressing these challenges will be crucial for developing effective communication tools for the Deaf and hard-of-hearing community.

2.3.6 Multimodal Systems and Beyond

Multimodal systems that combine hand gestures with facial expression and body pose recognition have been developed to capture the full scope of sign language. For example, Baltrušaitis et al. (2019) integrated gesture recognition with facial expression analysis, demonstrating improved translation accuracy by accounting for the non-manual signals that are critical in sign language (Baltrušaitis et al., 2019). This multimodal approach has shown promise, but the added complexity presents challenges in terms of computational load and the need for highly accurate models across multiple modalities.

Other researchers have explored alternative approaches, such as brain-computer interfaces (BCIs) and haptic feedback systems for sign language translation. While these technologies are still in their infancy, they represent potential future directions for the field.

Table 2.1 Comparative Analysis of the Related Works

Study	Focus	Key findings	Strengths	Limitations
Zhang (2022)	Gesture recognition using CNN and RNN for sign language translation	CNN-RNN hybrid model significantly improves accuracy for dynamic sign recognition	Strong performance in handling both spatial and temporal features	Struggles in complex, real-world environments (lighting, occlusion)

Liao et al. (2019)	3D CNN-based sign language gesture recognition	3D CNNs are effective in recognizing dynamic gestures from video sequences	Capable of handling dynamic signs; improved accuracy	High computational cost; unsuitable for real-time applications
Olabanji & Ponnle (2021)	Sensor-based wearable glove systems for ASL translation	Glove-based system provides high accuracy in capturing hand movements for sign language translation	Accurate gesture detection; effective for static and dynamic signs	Requires specialized hardware; lacks scalability for general use
Verma & Badli (2022)	Lightweight CNN for mobile real-time sign translation	Lightweight CNN enables real-time translation on low-power devices like smartphones	Low computational requirements; accessible for mobile devices	Lower accuracy compared to complex models; struggles with complex signs
Mehta et al. (2010)	Brain-Computer Interface (BCI) for sign language translation	Early-stage BCI systems show potential for enhancing gesture-based translation	Novel approach; potential for future exploration in assistive technologies	Still in developmental stage; low accuracy; high user training required

Baltrušaitis et al. (2019)	Multimodal systems combining gesture and facial recognition	Multimodal systems improve translation by integrating non-manual signals (e.g., facial expressions	Enhanced accuracy by capturing full scope of sign language (manual + non-manual)	High computational cost; complex system design; requires accurate multimodal data

2.4 Summary

93 This review highlights key findings for our project on sign language translation using computer vision and machine learning. While camera-based systems effectively capture hand movements, they face challenges like poor lighting and occlusion in real-world settings (Axyonov et al., 2021). Integrating tools like OpenCV with deep learning frameworks, such as CNNs, improves gesture recognition accuracy (Liao et al., 2019).

To address limitations in recognizing complex signs and user variations, techniques like edge detection and optical flow tracking (Zhang, 2022) can enhance robustness. Unlike high-end hardware-focused solutions, our project emphasizes affordability and scalability, aligning with Wadhawan and Kumar's (2019) call for cost-effective tools.

Leveraging computer vision and addressing current gaps will enable a more robust, accessible sign language translation system.

CHAPTER 3

REQUIREMENTS, ANALYSIS AND DESIGN

3.1 Overview

In Chapter 3, we will cover the system's requirements, analysis, and high-level design. This section includes a breakdown of functional and non-functional requirements, system specifications, application architecture, and user interface design. We'll discuss the approach taken and the methodologies applied, highlighting the impacts of each decision on the system's outcome. This chapter aims to provide a comprehensive view of the design choices that lay the groundwork for the system's functionality and performance.

3.2 Adopted methodology

The development of the system follows structured methodologies, a set of procedures and guidelines that define the engineering approaches used to achieve the desired solution (SOFTWARE

DEVELOPMENT LIFE CYCLE (SDLC)). These methodologies serve as adaptable frameworks, evolving with the unique demands of each project, and are refined to align with organizational objectives

Rapid Application Development (RAD) Model

The Rapid Application Development (RAD) model is a software development methodology focused on fast and iterative release cycles. Its primary goal is to deliver functional software in a short timeframe by using prototypes and frequent iterations. Unlike traditional methodologies like the Waterfall model, RAD is highly adaptive and flexible, allowing developers to incorporate user feedback and accommodate changing requirements throughout the development process. This approach places a strong emphasis on user collaboration, component reusability, and speed, making it a preferred choice for projects where time is a critical factor and requirements are expected to evolve. Fig 3.1 details the processes in Rapid application development.

Fig 3.1 Rapid application Development model (Jane, 2021)

3.3 Data gathering

3.3.1 Image gathering

5 The success of any machine learning model relies heavily on the quality and relevance of the data used during training. For this project, it was essential to create a dataset that accurately reflects the target application. Since no existing datasets fully met the requirements for this task, a custom dataset was created by capturing images of various sign language gestures. This ensured that the data was tailored to the project's goals and allowed for greater control over factors such as lighting, background consistency, and gesture clarity. The following sections detail the equipment, setup, and process used to gather the dataset, along with the challenges encountered and the solutions implemented.

3.3.2 Equipment and Setup

The data gathering process for this project leveraged accessible tools and a straightforward setup to create a high-quality custom dataset. The equipment and environment were chosen to ensure clarity, diversity, and suitability of the captured images for training the sign language translation model.

Equipment Used

Webcam: My laptop's inbuilt webcam was used for image capture. It provided sufficient image quality and resolution for detecting and training on hand gestures.

Lighting: Natural and ambient indoor lighting were used to ensure the hand gestures were visible. Efforts were made to capture images in relatively well-lit conditions to enhance gesture clarity.

Environment

Backgrounds: Images were captured against various backgrounds, including neutral and slightly complex environments, to introduce diversity into the dataset. This approach aimed to make the model robust to background variations.

Lighting Variations: While the lighting was generally consistent, slight variations in intensity were present to simulate real-world conditions and improve the model's adaptability.

Location: The image capture sessions were conducted indoors in a controlled setting to minimize distractions and ensure focus on the gestures.

Subject

The gestures were performed by myself. Each gesture was repeated multiple times to ensure a comprehensive dataset, with slight variations in angles and positions to simulate natural diversity in hand movements.

This straightforward and flexible setup allowed for the collection of a dataset that balances consistency with variability, contributing to a robust training process for the model.

3.3.3 Image Capture Process

To ensure consistency and efficiency in collecting the dataset, a Python script was developed to automate the image capture process. This approach streamlined the task of gathering images for all American Sign Language (ASL) alphabets, ensuring that the data was captured uniformly across different sessions. The following subsections outline the process in detail.

Steps Followed

Script Initialization:
The Python script was executed on the laptop, controlling the inbuilt webcam to capture images automatically.

Alphabet Selection and Display:

The script prompted the user to display a specific ASL alphabet gesture. Instructions were shown on the laptop screen, ensuring that each gesture was captured in sequence.

Timed Image Capture:

For each gesture, the script captured a predefined number of images (100 images per alphabet) at regular intervals. This method allowed for slight variations in positioning while maintaining uniformity in timing.

Gesture Display and Adjustment:

The gestures were performed directly in front of the webcam. Adjustments to hand position were made as needed to keep the gestures within the frame. This can be seen in Fig 3.2 below.

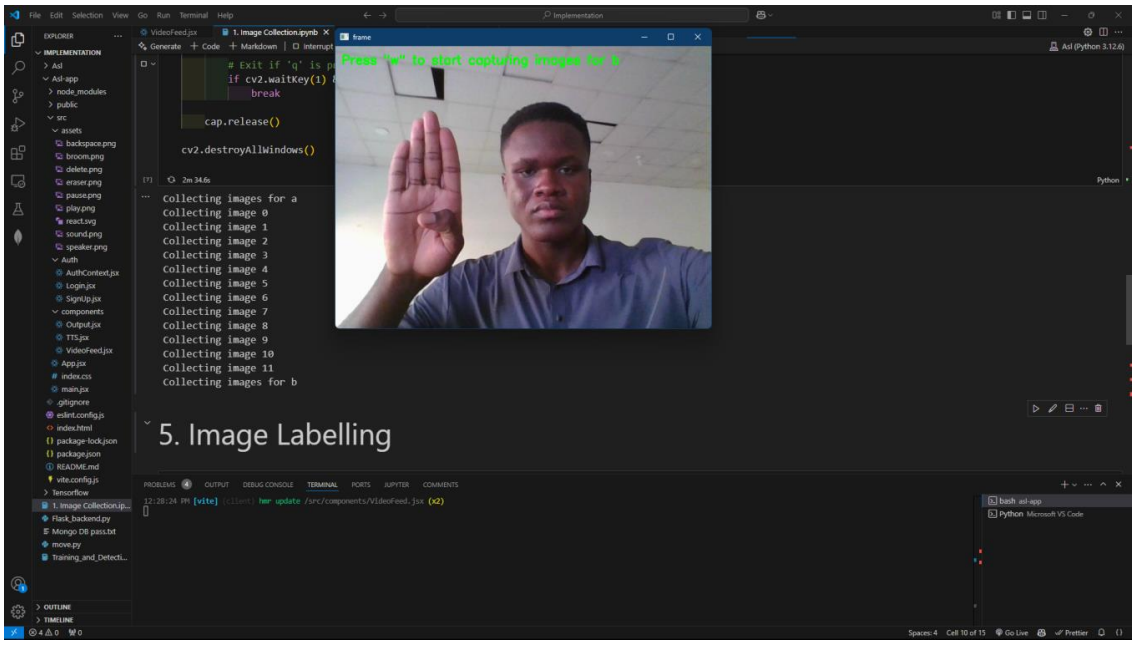


Fig 3.2 process of capturing images for class ‘b’

The Python script handled the following key functions:

Frame Capturing: Real-time image capture and saving to designated folders, categorized by alphabet (e.g., "A", "B", "C"…)

File Naming Convention: Images were saved with a systematic naming format(**b**.2a904474-a4e4-11ef-9234-38dead080396.jpg), simplifying dataset management.

Dataset Organization

The captured images were stored in folders, each representing a specific alphabet as seen in Fig 3.3 and 3.4. This structure ensured easy access and facilitated training data preparation.

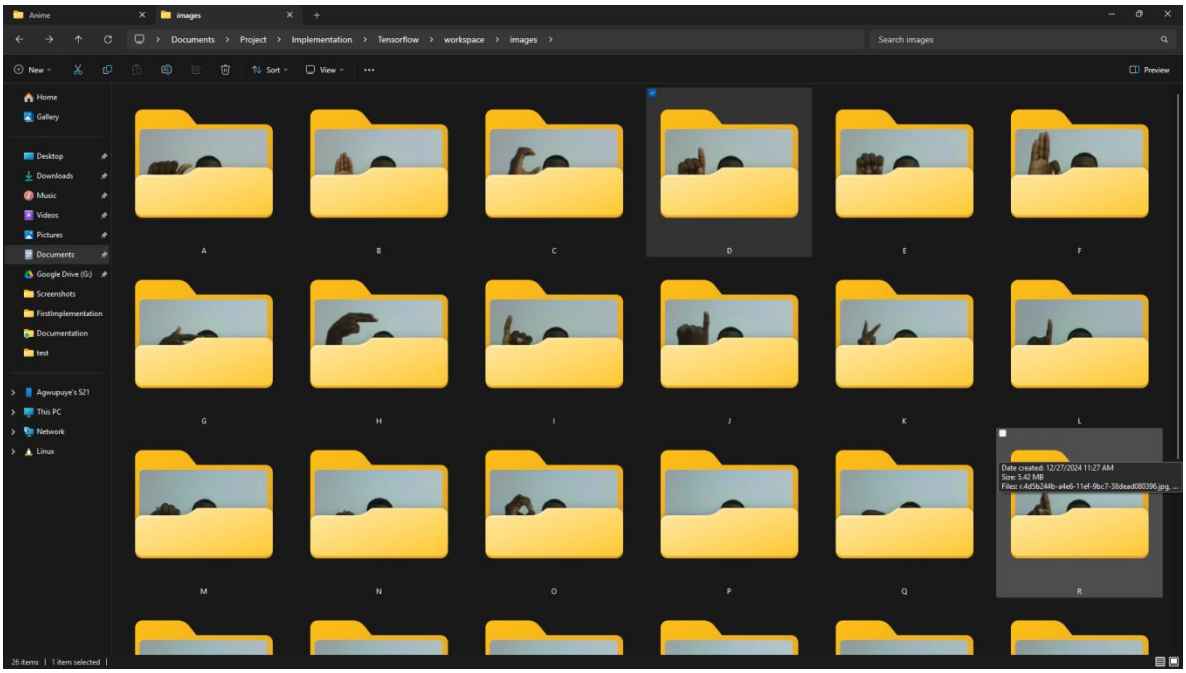


Fig 3.3 Folders containing images for various ASL alphabets

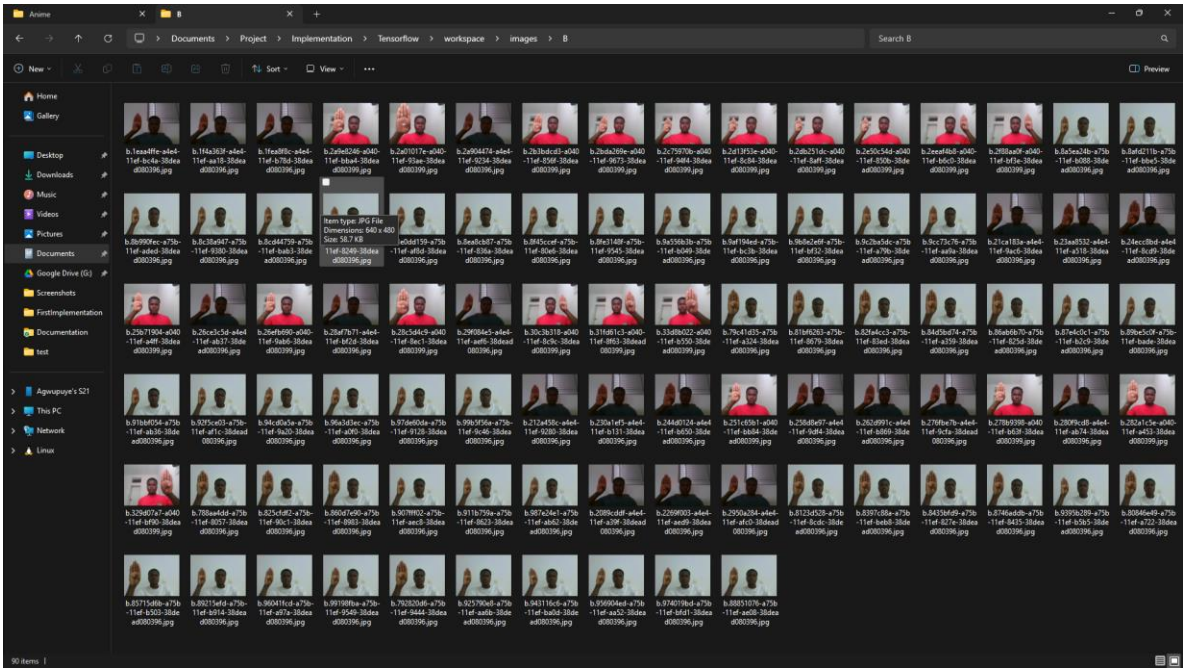


Fig 3.4 Images for class “b”

3.4 Tools and techniques

This project utilizes a combination of modern tools and technologies to create a robust and efficient sign language translation system. Each tool was chosen to meet specific needs of the project, from image processing and machine learning to front-end development and seamless integration. Below is an overview of the key tools and techniques employed:

OpenCV: OpenCV (Open Source Computer Vision Library) is a crucial component in this project for image and video processing. It is used to capture and process the video input of sign language gestures, enabling real-time detection and tracking of hand movements. OpenCV's powerful image analysis functions help pre-process the visual data, ensuring the input to the machine learning model is clean and accurate.

TensorFlow: TensorFlow is the primary machine learning framework used for building and training the sign language recognition model. It facilitates the development of a deep neural network capable of accurately interpreting hand gestures. TensorFlow's flexibility and comprehensive libraries allow for custom training and fine-tuning of the model, ensuring it performs well under various conditions, including different lighting scenarios.

React.js: React.js is employed to develop the front-end of the project. This JavaScript library is used to create a dynamic and responsive user interface that allows users to interact with the sign language translation system in real-time. React's component-based structure makes it easy to manage the application's state and seamlessly integrate the machine learning model once it's converted to TensorFlow.js.

Python: Python is the primary programming language used for the back-end development and model training. It is leveraged for data processing, handling machine learning workflows, and managing the training environment. Python's wide range of libraries and its compatibility with both OpenCV and TensorFlow make it an ideal choice for developing the machine learning aspects of the project.

3.5 Ethical consideration

Developing a sign language translation system involves several ethical considerations to ensure that the technology is inclusive, respectful, and used responsibly. These considerations have guided the design and implementation of the project to prioritize the well-being and dignity of users. Below are the primary ethical concerns addressed:

Privacy and Data Security: Given that this project involves capturing and processing video data of users' hand movements, privacy is a top priority. All video input data is processed locally to prevent unauthorized access or storage of sensitive information. Additionally, strong encryption standards and best practices for data security are applied to safeguard any potentially sensitive user information.

Inclusivity and Accessibility: The core purpose of this project is to improve accessibility for the Deaf and Hard of Hearing communities. The design of the system aims to enhance communication without replacing or devaluing the cultural significance of sign languages. Therefore, the project acknowledges that technology should complement human communication and not replace it. This system is built to respect the richness and diversity of sign languages while providing a helpful translation aid.

Transparency and Accountability: Transparency in how the machine learning model operates is crucial to build trust with users. The project's design and development process includes clear documentation on the model's capabilities, limitations, and the data sources used for training. Regular audits and updates are planned to refine the model and address any emerging ethical concerns, making the project accountable to its intended users.

Impact on the Community: The potential impact of this technology on the Deaf and Hard of Hearing communities is carefully considered. This project aims to support communication rather than replacing human translators, recognizing that human interpretation is often irreplaceable due to the nuances and cultural context inherent in sign languages. The project remains open to feedback from these communities to ensure the system is developed in a way that truly benefits its users without unintentionally causing harm or misinterpretation.

3.6 Requirement analysis

The success of this sign language translation project relies heavily on a clear understanding of the system's functional and non-functional requirements. A detailed analysis was conducted to identify the key requirements needed to achieve the project's objectives, ensuring that the final product is both effective and user-friendly. Below is an overview of the primary requirements:

3.6.1 Functional requirements

Table 3.1 Functional Requirements

Req. No	Description
R-101	The system shall be able to perform Real-Time Gesture recognition
R-102	The system shall Accurately detect hand signs
R-103	The system shall be able to form simple words and sentences from ASL alphabets
R-104	The user Interface shall be intuitive
R-105	The system shall be compatible with different device types
R-106	The system should allow users to register; giving only minimal information
R-107	The system should allow users to login with registered information
R-108	The system should be able to convert translated words and sentences into words

3.6.2 Non-Functional requirements

Table 3.2 Non-Functional Requirements specifications

Req. No	Description
R-101	The system shall maintain smooth and efficient performance, processing video input and generating translations with minimal latency.
R-102	The architecture shall be scale-able to accommodate future expansions, including adding new gestures or adapting to different sign languages.
R-103	The user interface shall be accessible, straightforward, and designed to accommodate users of all technical skill levels.

R-104	The system shall achieve recognition accuracy above 90% for common sign language gestures under standard conditions.
R-105	The code-base shall be well-documented and modular, allowing for easy maintenance, updates, and debugging.

3.7 System design

USER REGISTRATION: This feature allows new users to create an account by providing minimal details such as email and password. The system verifies unique credentials before completing the registration process.

LOGIN SYSTEM: Users with an existing account can log in securely using their email and password. The backend handles authentication, ensuring only authorized users gain access.

GESTURE DETECTION MODULE: This is the core functionality of the system, where gestures captured through the user's video feed are processed to predict American Sign Language (ASL) letters. Predictions are sent back as single characters for further use in building sentences.

FRONTEND:

Sentence Builder: This component appends and stores letters received from the backend. It maintains and displays the current sentence being formed by the user.

VIDEO INPUT MODULE: The frontend captures video input from the user's webcam. It pre-processes frames before sending them to the backend for gesture prediction.

BACKEND LOGIC:

Prediction Processor: Collects gesture predictions within a 1.5-second window, and sends it back to the frontend.

Space Logic: Monitors for a second hand appearing in the frame and triggers the addition of a space character.

TensorFlow Model Integration: The trained ASL model processes gestures and generates predictions based on the user's hand Gestures.

TEXT-TO-SPEECH (TTS): Converts the constructed sentence into speech using the Speechify API. Users can input text and receive audio output, enhancing accessibility.

USER INTERFACE:

Camera Selection: Enables users to select from available cameras for gesture detection.

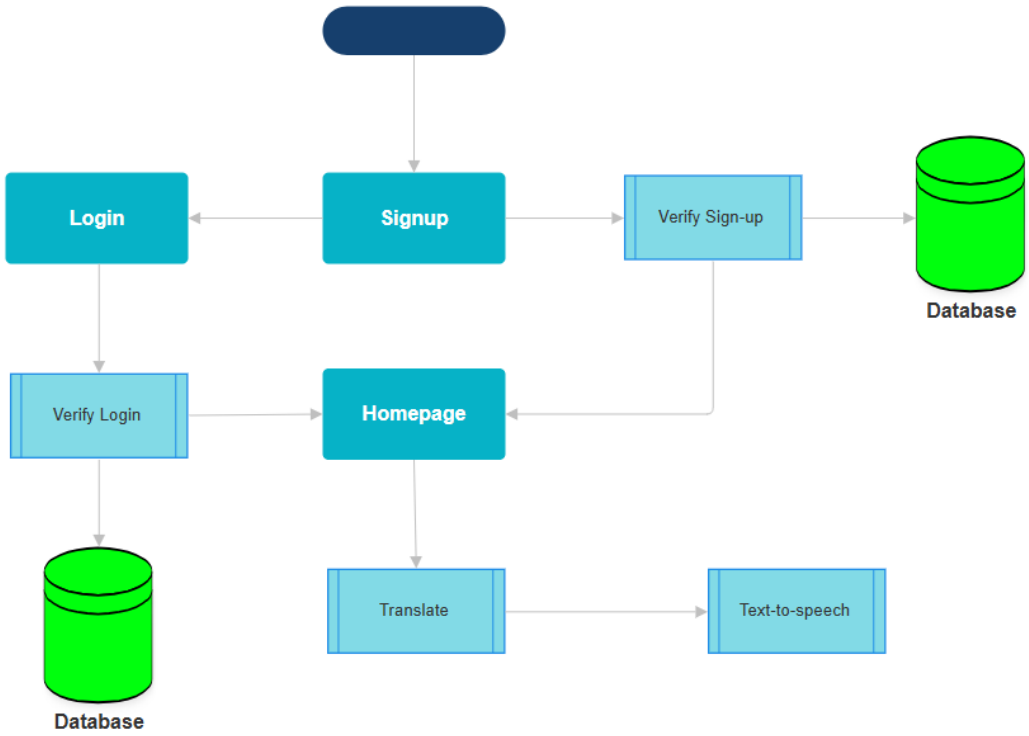
Controls: Provides buttons for starting/stopping gesture translation, clearing sentences, and navigating between features.

DATABASE: Stores user details (i.e., email, password) securely. MongoDB Atlas serves as the database backend.

ERROR HANDLING MODULE: Ensures graceful handling of issues such as model errors, API failures, or invalid user inputs, keeping the system robust and user-friendly.

3.7.1 System architecture

The application diagram illustrates the structure of an application and demonstrates how various system components interact to fulfill users' needs.

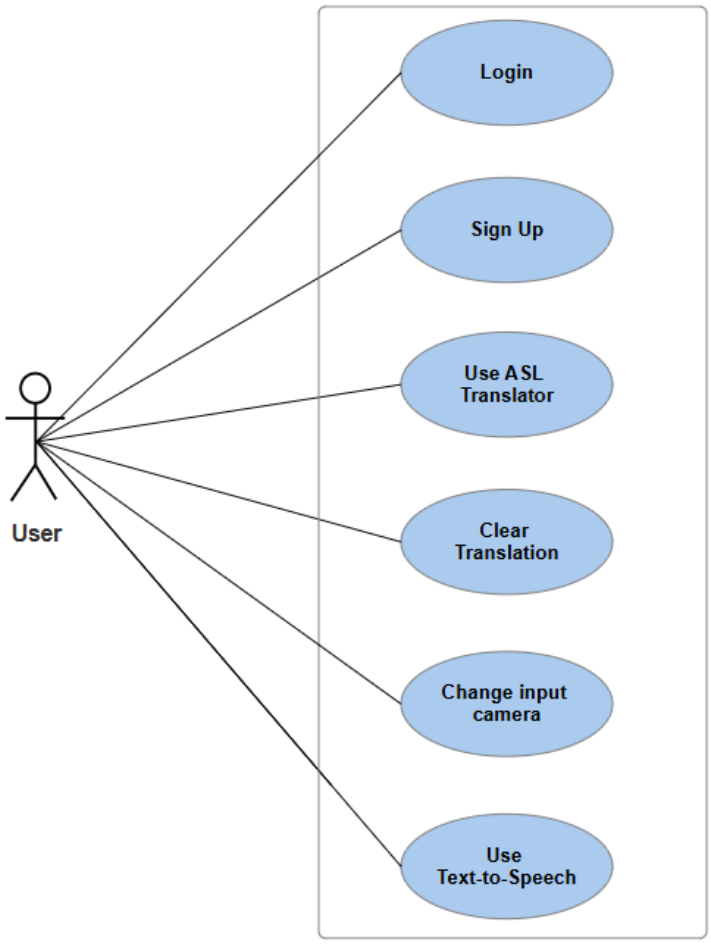


104

Fig 3.5 Application Architecture

3.7.2 Use case diagram

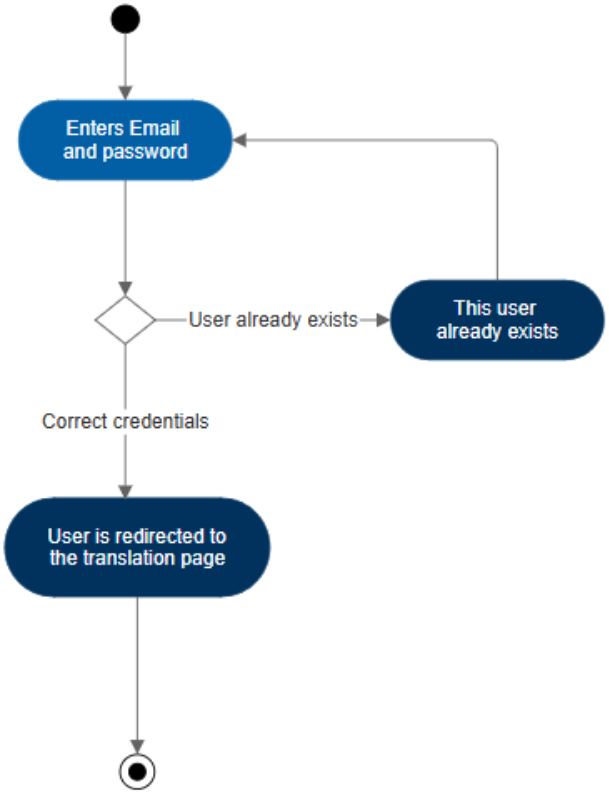
In Fig 3.6 the actor named “user” can perform various actions in the program



85

Fig 3.6 Use case diagram for User

3.7.4 Activity Diagrams



26

Fig 3.7 Activity diagram for user sign-up

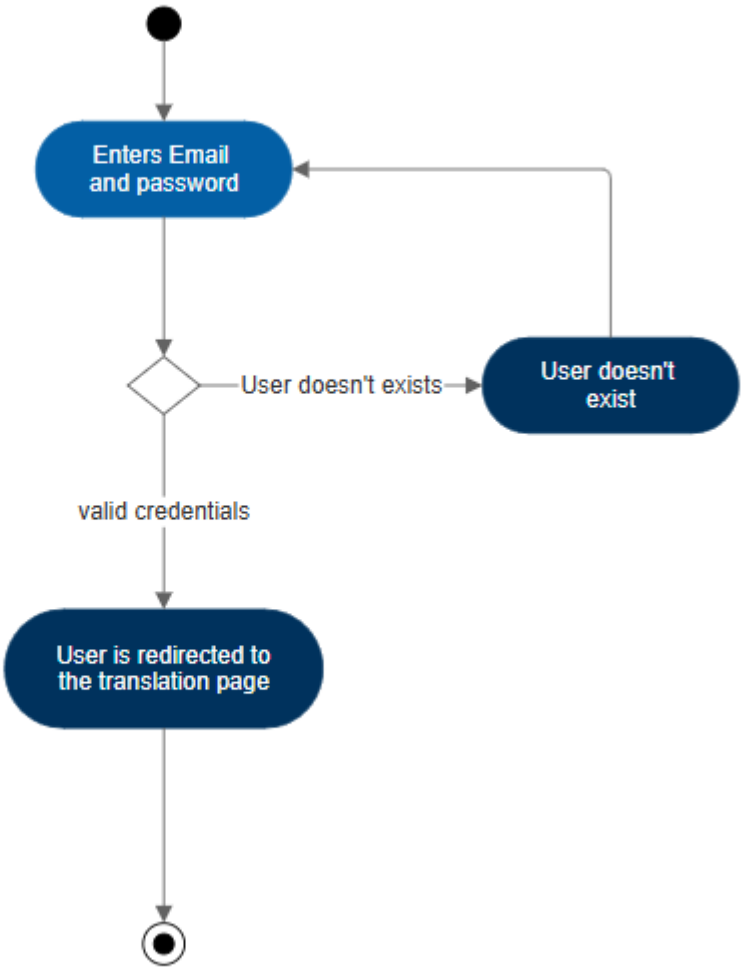


Fig 3.8 Activity diagram for user Login

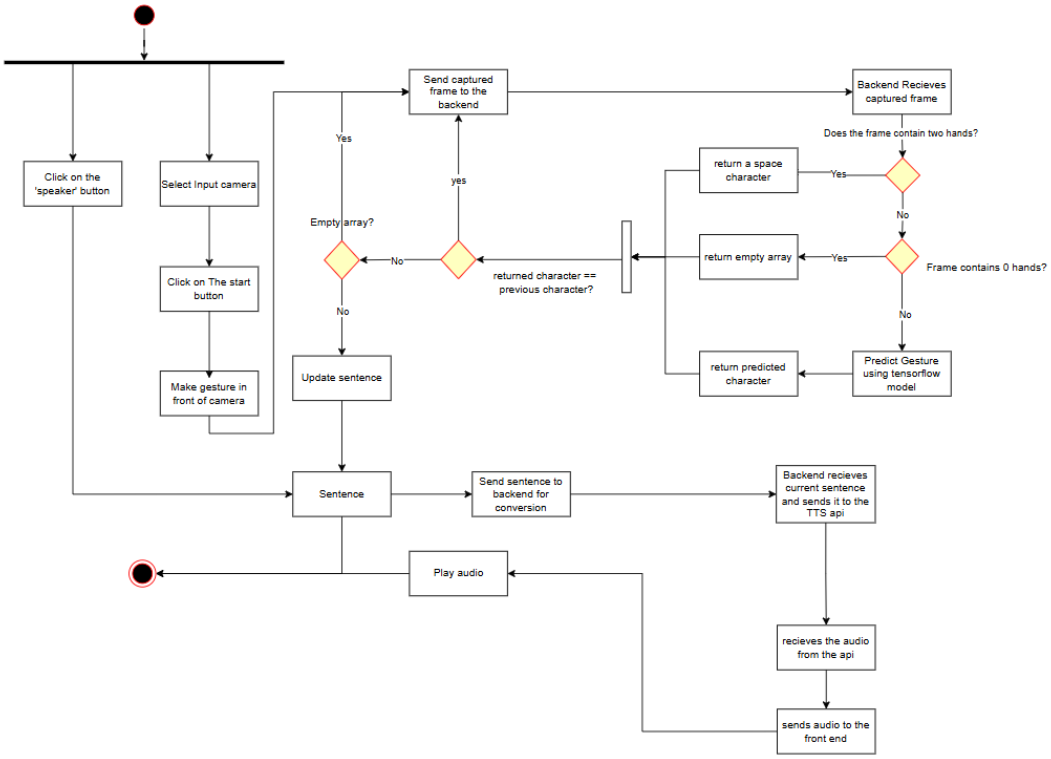


Fig 3.9 Activity diagram detailing the entire translation process

3.7.5 Data Flow Diagrams

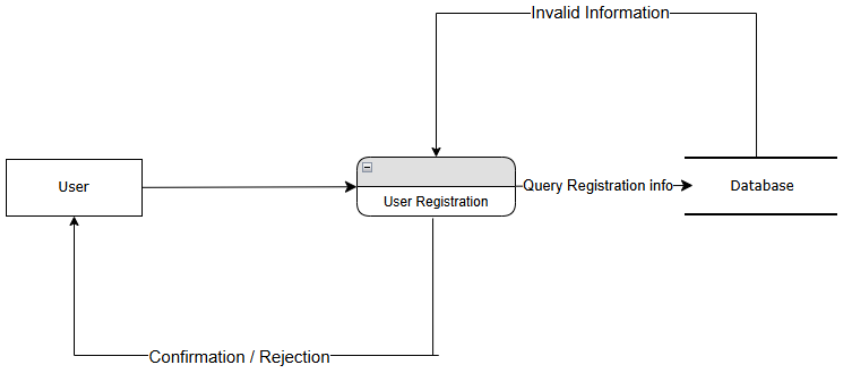


Fig 3.10 Data flow diagram for User sign-up

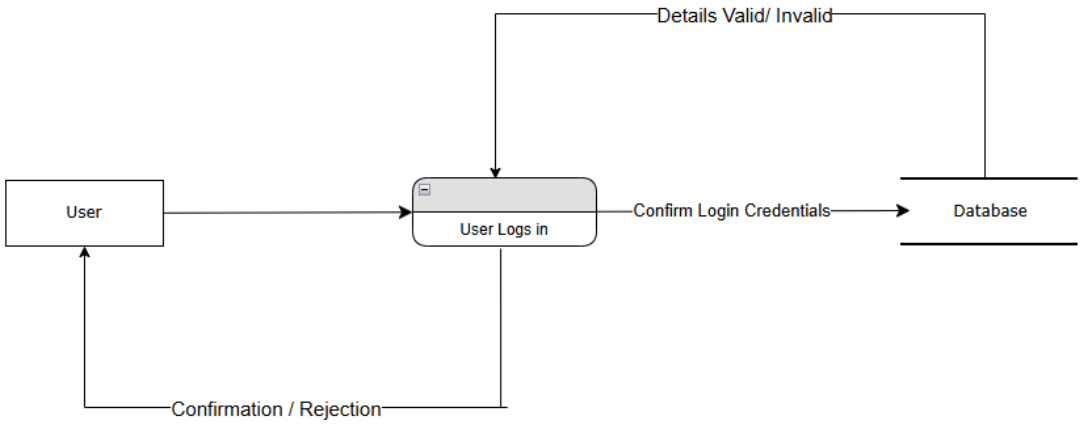


Fig 3.11 Data flow diagram for User Login

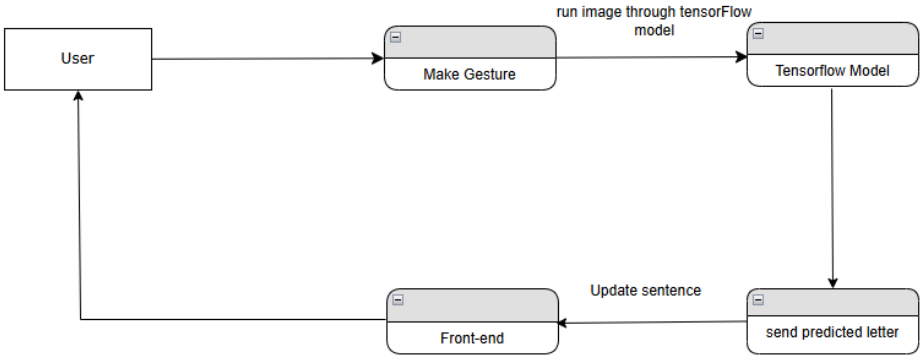
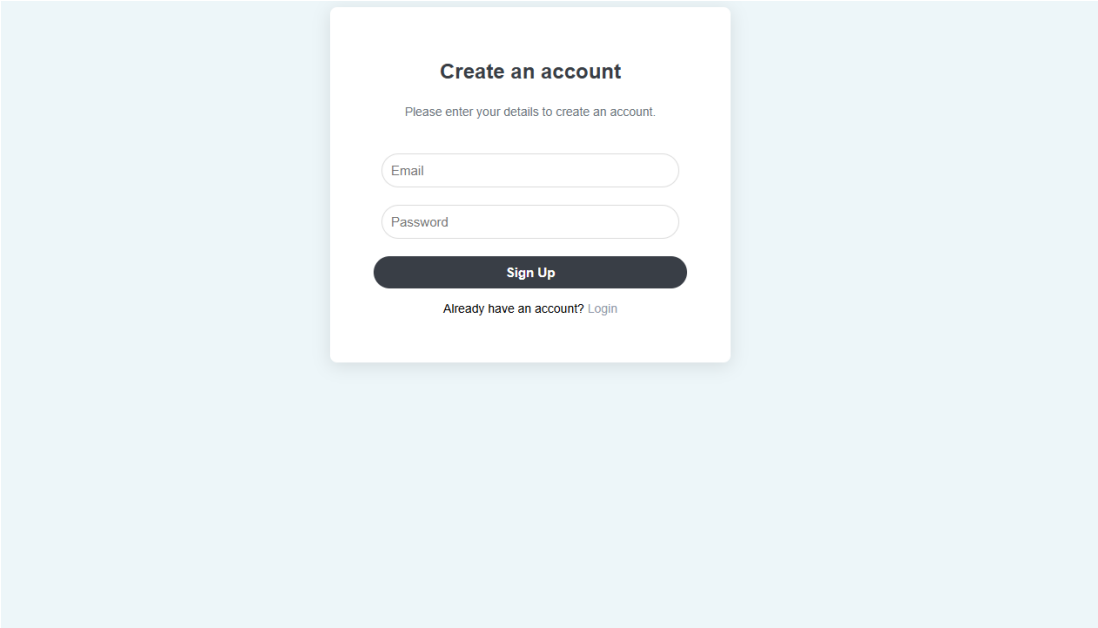


Fig 3.12 Data flow diagram for Translation

3.7.6 User Interface



39

Fig 3.14 User interface for the sign-up page

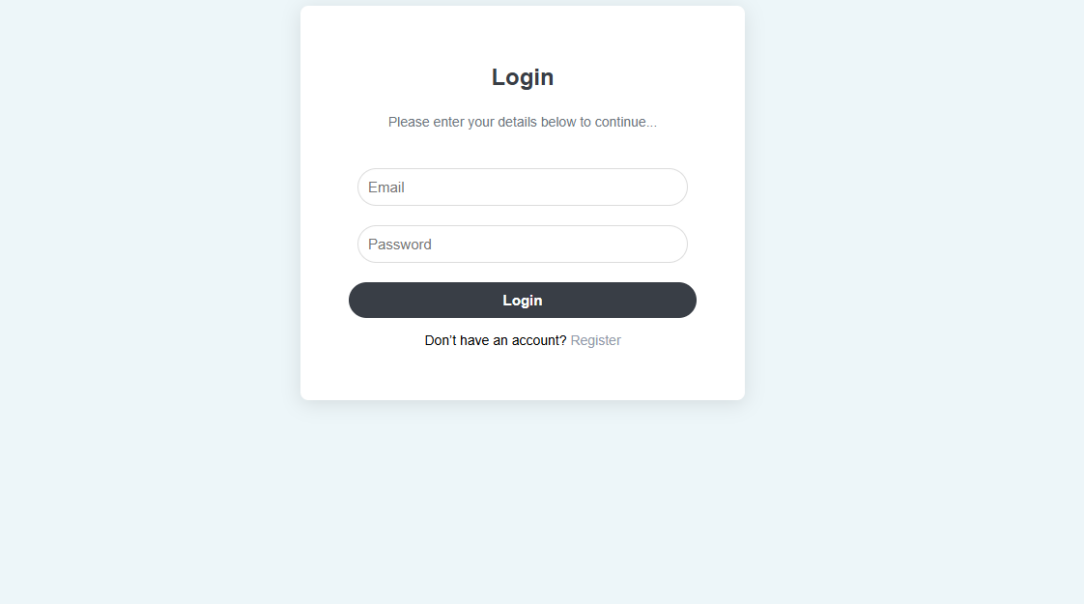


Fig 3.15 User interface for login page

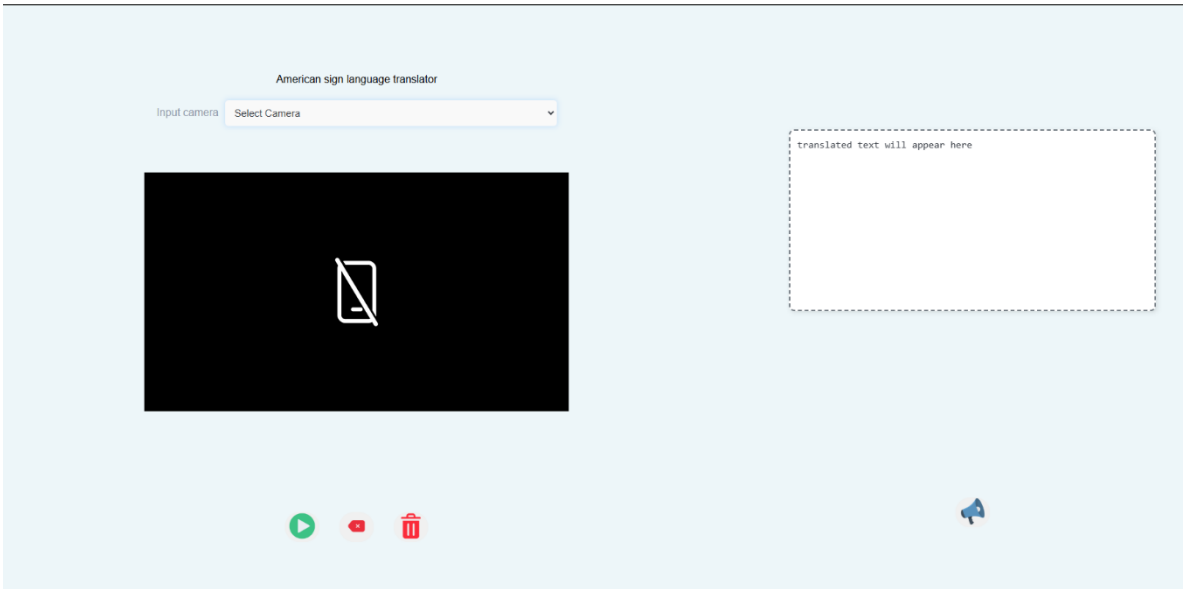


Fig 3.16 user interface for translation page

CHAPTER 4

TESTING AND IMPLEMENTATION

4.1 Overview

This chapter provides an overview of the development process for the gesture recognition-based system. It covers the integration of the trained gesture recognition model into a React application, including key components such as the front-end implementation, back-end support, and data handling. Additionally, it highlights the challenges faced during the development, such as real-world accuracy discrepancies for specific gesture classes, and details the strategies employed to address these issues effectively.

4.2 Main Features

The main features of the system are:

Gesture Detection and Recognition:

The system integrates a gesture recognition model trained using TensorFlow and Mediapipe. It accurately detects hand gestures and maps them to specific letters of the alphabet. This feature allows users to interact with the system using physical gestures, forming the basis for creating words and sentences.

Word and Sentence Formation:

Once gestures are recognized, the system processes them to form coherent words and sentences. Users can see the text generated in real-time, enabling seamless communication or transcription of gestures.

Text-to-Speech Conversion:

After forming sentences, the system includes a text-to-speech (TTS) feature that converts the generated text into audio. This enables users to vocalize their messages, enhancing communication for individuals who may have difficulty speaking or prefer auditory output.

Front-end Implementation:

A React-based front-end provides a user-friendly interface for interaction. It displays the detected

gestures, the corresponding text output, and additional functionalities like clearing or resetting the output.

Back-end Support:

The back-end processes the gesture recognition data and ensures smooth integration with the front-end. It also handles the text-to-speech requests and ensures efficient communication between the trained model and the interface.

Error Handling and Accuracy Refinement:

The system includes logic to handle cases where gestures are inaccurately recognized or not detected at all. Strategies like predictive text suggestions or allowing users to re-perform gestures improve usability and performance.

Real-Time Feedback:

The system provides immediate feedback to users as gestures are performed, ensuring an intuitive and responsive experience.

4.3 Implementation problems

41 During the development and implementation of the gesture recognition system, several challenges were encountered. These problems and the approaches taken to address them are detailed below:

Accuracy Discrepancies in Gesture Recognition:

While the model performed well on training and testing datasets, real-world scenarios revealed inconsistencies in recognizing certain gestures. Variations in lighting, hand positioning, or user movement often affected accuracy.

Integration with React Front-end:

Converting the trained TensorFlow model to TensorFlow Js posed some issues due to package dependency problems so integration with the React-based front-end posed challenges, especially with real-time gesture detection.

Handling Ambiguous Gestures:

Some gestures, especially those resembling each other, were occasionally misclassified. This created confusion during word and sentence formation.

Hardware Limitations:

Performance was affected on lower-end devices, especially when handling real-time gesture recognition and text-to-speech conversion.

4.4 Overcoming Implementation problems

The above mentioned were solved with the following methods.

- Additional data pre-processing was applied during training to simulate real-world conditions. Furthermore, fine-tuning the model and implementing logic to retry or confirm unclear gestures helped mitigate these issues.
- Developed a back-end server using Flask to host the TensorFlow model and seamlessly integrated it with the front-end. Optimization techniques were applied to minimize latency and ensure a smooth, responsive user experience.
- The system was equipped with error-handling mechanisms, such as requiring a gesture to be detected multiple times before being validated and added to the sentence.
- Code optimizations and the use of lightweight libraries ensured better performance across a wide range of devices.

4.5 Testing

To verify that the system functioned correctly and adhered to the specified requirements, tests were conducted from the perspective of end users to simulate real-world scenarios. This approach assessed the system's performance, its ability to handle errors effectively, and its overall usability. Various testing techniques, including unit testing, were employed to evaluate both the front-end and back-end components comprehensively.

4.5.1 Test Plans (for Unit, Integration and System Testing)

Table 4.1 Test Summary for user sign-up

Test case	User sign-up
Related Requirements	R-106
Prerequisites	The registration page must be accessible, and the database must be connected.
Test Procedures	1. Navigate to the "Sign Up" page. 2. Enter valid user details (e.g., username, email, password). 3. Click on the "Sign Up" button.
Test Data	Example: Username:"test_user",Email:"testuser@example.com", Password: "password123".
Expected Results	User details should be stored in the database, and be redirected to the homepage.
Actual Results	User details were stored in the database, and a confirmation message was displayed.
Status	Pass
Remarks	Email verification logic yet to be implemented.
Created By	Richard
Date of Creation	26th December 2024
Executed By	Richard
Date of Execution	27th December 2024

Test Environment	HP Laptop with Chrome browser.

Table 4.2 Test Summary for user Login

Test case	User Login
Related Requirements	R-107
Prerequisites	The user account must exist in the database and be active. The login page must be accessible.
Test Procedures	1. Navigate to the "Login" page. 2. Enter valid login credentials (e.g., email and password). 3. Click on the "Login" button..
Test Data	Example: Username:"test_user",Email:"testuser@example.com", Password: "password123".
Expected Results	User should be redirected to their dashboard upon successful login.
Actual Results	User was redirected to the Homepage after successful login.
Status	Pass
Remarks	None
Created By	Richard
Date of Creation	26th December 2024
Executed By	Richard
Date of Execution	27th December 2024

Test Environment	HP Laptop with Chrome browser.

Table 4.3 Test Summary for Gesture Recognition Accuracy

Test case	Gesture Recognition Accuracy Testing
Related Requirements	R-102
Prerequisites	Model must be trained and deployed. User must have access to a webcam-enabled device.
Test Procedures	1. Perform gestures for each letter of the alphabet in front of the camera. 2. Check if the correct letter is displayed.
Test Data	Hand gestures for all 26 letters of the alphabet.
Expected Results	Recognized gesture should match the expected letter.
Actual Results	Correct letters were displayed for all gestures..
Status	Pass
Remarks	Minor accuracy issues in low-light conditions were observed.
Created By	Richard
Date of Creation	26th December 2024
Executed By	Richard
Date of Execution	27th December 2024
Test Environment	HP Laptop with built-in webcam

Table 4.4 Test Summary for Sentence Formation

Test case	Sentence Formation Validation
Related Requirements	R-103
Prerequisites	Gesture recognition feature must be functional.
Test Procedures	1. Perform gestures for letters to form a word. 2. Continue gestures to create a full sentence. 3. Observe the output on the screen.
Test Data	Example input: gestures spelling "Hello World".
Expected Results	Recognized gestures should form coherent words and sentences.
Actual Results	Words and sentences formed accurately.
Status	Pass
Remarks	none.
Created By	Richard
Date of Creation	26th December 2024
Executed By	Richard
Date of Execution	27th December 2024
Test Environment	HP Laptop with built-in webcam

27

Table 4.5 Text-to-Speech Feature Testing

Test case	Text-to-Speech testing
Related Requirements	R-108
Prerequisites	The sentence formation feature must be functional.
Test Procedures	1. Form a sentence using gestures. 2. Click on the "Speak" button to activate the TTS feature.
Test Data	Input: "Hello, how are you?".
Expected Results	System should convert the text into clear, audible speech.
Actual Results	Text was successfully converted to speech.
Status	Pass
Remarks	Speech quality was clear with minimal delay..
Created By	Richard
Date of Creation	26th December 2024
Executed By	Richard
Date of Execution	27th December 2024
Test Environment	HP Laptop with speakers.

Table 4.6 Real-Time Feedback Testing

Test case	Real-Time Gesture Feedback
Related Requirements	R-101
Prerequisites	Gesture recognition feature must be functional.
Test Procedures	1. Perform gestures for letters. 2. Observe if the recognized gesture appears immediately on the screen.
Test Data	Random gestures for letters A-Z.
Expected Results	Recognized gestures should display in real-time without noticeable lag..
Actual Results	Real-time feedback was accurate and responsive.
Status	Pass
Remarks	None
Created By	Richard
Date of Creation	26th December 2024
Executed By	Richard
Date of Execution	27th December 2024
Test Environment	HP Laptop with built-in webcam.

4.6 User Guide

This section provides a step-by-step guide on how to use the system effectively, from registration to gesture translation and text-to-speech (TTS) functionalities.

1. Registration

To start using the system,

Navigate to the Registration Page.

Enter your email address and password.

Click on the Sign-Up button.

If registration is successful, you will be redirected to the translation page.

2. Login

To access your account:

Go to the Login Page.

Enter your registered email address and password.

Click the Login button.

If the credentials are correct, you will be redirected to the translation page.

3. Using the Gesture Translation Module

The gesture detection system uses your camera feed to identify hand gestures. When a hand gesture is detected, it is interpreted as a specific letter or space. The system allows for the construction of sentences in real time based on gestures made by the user.

The application supports the following functionality:

Single letter gestures: Each detected gesture corresponds to a specific letter, which is appended to the ongoing sentence.

Space insertion: When multiple hands are detected in the frame, a space is inserted between words.

Consecutive letter detection: If the same gesture is repeated consecutively, the system requires the user to remove their hand to reset the gesture before it can be appended again.

Getting Started

Select a Camera:

When the application starts, the available cameras are listed. Choose the camera you wish to use for gesture detection.

Start Gesture Detection:

Click on the Start button to begin gesture detection. The camera will start streaming, and the system will begin interpreting your gestures.

Make Gestures:

136 Perform hand gestures in front of the camera. Each gesture will be translated into a character or space, which will be appended to the sentence being constructed.

Single gesture: A single hand gesture is interpreted as a letter.

Multiple hands: If two hands are detected, a space is inserted between words.

Gesture Handling

The system handles gestures intelligently by considering the context of the detected gestures:

Consecutive Gestures: If the same gesture is detected consecutively, the system will wait for a hand removal to reset. This is to prevent accidental double-letter inputs. Once the hand is removed and the system is reset, the letter can be appended again.

Message Notification: If the system detects that the same gesture is repeated, it will display a message requesting the user to remove their hand to reset. The message will disappear after 2 seconds.

Space Handling: If multiple hands are detected, the system will automatically insert a space between the last letter and the next one. This allows for the construction of words separated by spaces.

Resetting the System

Manual Reset:

If you wish to reset the system manually, you can click the Clear button, which will remove the current sentence being constructed.

Automatic Reset:

The system automatically resets when a hand is removed from the frame, clearing the state and preventing consecutive letter appends. This ensures that the system is always ready for a new gesture.

Troubleshooting

No Gestures Detected:

Ensure that your hands are within the camera's frame and well-lit. If no gestures are detected, try adjusting your position or lighting.

Space Not Inserting:

If a space is not inserted when multiple hands are detected, check that both hands are clearly visible in the frame.

Tips for Better Accuracy

49 Lighting: Proper lighting improves the system's ability to accurately detect your hand gestures. Ensure the camera has a clear view of your hands with enough contrast.

Gesture Consistency: Perform gestures clearly and consistently to avoid misinterpretation.

Hand Position: Position your hands in the center of the frame for better detection. Ensure that the gesture is fully formed and recognizable by the system.

5. Text-to-Speech (TTS)

27 Click on the speaker button.

The system processes the translated text and generates an audio output.

Play the audio directly from the browser.

7. Error Messages

Invalid Login: Check your email and password for errors.

4.7 Summary

The testing of a system is a critical step in ensuring the final version is functional, reliable, and meets the needs of its intended users. For this project, various stages of testing were conducted to evaluate the usability and performance of the system. Through rigorous testing, any identified issues were promptly addressed and resolved to improve the system's quality. The testing process ensured that the system is ready for deployment and user interaction.

CHAPTER 5

Discussion, Conclusion and Recommendation

5.1 Overview

This chapter provides an evaluation of the project, including the challenges encountered during its development and possible future enhancements. It assesses the extent to which the project objectives were achieved and identifies areas for improvement to optimize functionality and user experience.

5.2 Objective Assessment

The primary objectives of this project were successfully achieved. These include:

Enabling the translation of American Sign Language (ASL) gestures into text using a digital platform.

Allowing users to interact with a real-time gesture recognition system and view translated text.

The system allows users to communicate effectively by recognizing gestures and converting them into written text or audio output. While advancements in AI and computer vision technologies continue, the system can be updated to remain relevant and accurate.

5.3 Limitations and Challenges

The project encountered several limitations and challenges during its development, including:

Time Management:

Balancing the workload of academic responsibilities and system development required strategic time allocation.

Learning Curve:

Familiarity with new technologies such as TensorFlow, Mediapipe, and Flask presented initial difficulties but was overcome through research and practice.

Hardware Constraints:

Testing and debugging on devices with limited processing power sometimes resulted in slower performance.

Users must meet certain requirements to interact with the system effectively:

A device equipped with a functional camera and internet connection.

Adequate lighting for accurate hand gesture recognition.

Updated browsers and operating systems to support the application.

5.4 Future Enhancements

Although the project met its core objectives, there are features that could be incorporated in future iterations to improve the system:

Improved Accuracy:

Enhance the machine learning model with a more extensive and diverse dataset for better gesture recognition accuracy.

Additional Gestures:

Extend support to more gestures, including phrases and cultural variations in ASL.

User Personalization:

Include customizable voice options for text-to-speech output to suit individual preferences.

5.5 Recommendations

To enhance the usability and adoption of the system, the following measures are recommended:

User Training:

Provide tutorials or guides for users to understand the gesture recognition process and optimize their interaction with the system.

Regular Updates:

Periodically update the system to include new features and improve existing ones.

User Feedback Integration:

Establish a feedback mechanism to gather insights from users and address their needs effectively.

5.6 Summary

This chapter concludes the project documentation. It includes an overview of the project's goals, the challenges encountered, and their solutions, as well as recommendations for future improvements. The project achieved its primary objectives of providing a gesture-to-text and text-to-speech system, ensuring inclusivity for users. Through systematic testing and continuous iteration, the system was made reliable and efficient. Future advancements can further enhance its scope and usability, ensuring it remains a valuable tool for communication.

REFERENCES

Adib Burhani, Z. M., & Prasetyo, J. (2023). The Sign Language Interpreting Gloves. J. Of App. Sci. & Adv. Eng. https://doi:10.59097/jasae.v1i1.10

Ahmed, M. A., Zaidan, B. B., Zaidan, A. A., Salih, M. M., & Lakulu, M. M. (2018). A Review on Systems-Based Sensory Gloves for Sign Language Recognition State of the Art Between 2007 and 2017. Sensors. https://doi:10.3390/s18072208

Alobaidy, M. A., & Ebraheem, S. (2020). Application for Iraqi Sign Language Translation on Android System. International Journal of Electrical and Computer Engineering (Ijece). https://doi:10.11591/ijece.v10i5.pp5227-5234

Arssi, A. and Taibi, N. (2018). Sign language interpreter: what makes it different? Sino-Us English Teaching, 15(8). https://doi.org/10.17265/1539-8072/2018.08.004

Axyonov, A., Ryumin, D., & Kagirov, I. (2021). Method of Multi-Modal Video Analysis of Hand Movements for Automatic Recognition of Isolated Signs of Russian Sign Language. The International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences. https://doi:10.5194/isprs-archives-xliv-2-w1-2021-7-2021

Baltrušaitis, T., Ahuja, C., & Morency, L. (2019). Multimodal Machine Learning: A Survey and Taxonomy. Ieee Transactions on Pattern Analysis and Machine Intelligence. https://doi:10.1109/tpami.2018.2798607

Bantupalli, K., & Xie, Y. (2018). American Sign Language Recognition Using Deep Learning and Computer Vision. https://doi:10.1109/bigdata.2018.8622141

Bochner, J. H., Christie, K., Hauser, P. C., & Searls, J. M. (2011). When Is a Difference Really Different? Learners' Discrimination of Linguistic Contrasts in American Sign Language. Language Learning. https://doi:10.1111/j.1467-9922.2011.00671.x

Camgöz, N. C., Hadfield, S., Koller, O., Ney, H., & Bowden, R. (2018). Neural Sign Language Translation. https://doi:10.1109/cvpr.2018.00812

Chen, X., Su, L., Zhao, J., Qiu, K., Jiang, N., & Zhai, G. (2023). Sign Language Gesture Recognition and Classification Based on Event Camera With Spiking Neural Networks. Electronics. https://doi:10.3390/electronics12040786

Chong, V. Y. (2024). The Design of the Deaf in Touch Everywhere (DITE)TM Mobile Application With Deaf and Interpreter Communities in Malaysia. Digital Health. https://doi:10.1177/20552076241228432

Dean, R. K., & Pollard, R. Q. (2001). Application of Demand-Control Theory to Sign Language Interpreting: Implications for Stress and Interpreter Training. The Journal of Deaf Studies and Deaf Education. https://doi:10.1093/deafed/6.1.1

Delisle, A., Larivière, C., Imbeau, D., & Durand, M.-J. (2005). Physical Exposure of Sign Language Interpreters: Baseline Measures and Reliability Analysis. European Journal of Applied Physiology. https://doi:10.1007/s00421-005-1316-5

Donner, A., Marshall, M. S., & Mozrall, J. (2013). Biomechanical Comparison of American Sign Language Interpretation and Conversation. Proceedings of the Human Factors and Ergonomics Society Annual Meeting. https://doi:10.1177/1541931213571083

Elmahgiubi, M., Ennajar, M., Drawil, N., & Elbuni, M. S. (2015). Sign Language Translator and Gesture Recognition. https://doi:10.1109/gscit.2015.7353332

Eng Tju, T. E. (2024). Hand Sign Interpretation Through Virtual Reality Data Processing. Jurnal Ilmu Komputer Dan Informasi. https://doi:10.21609/jiki.v17i2.1280

Falchook, A. D., Mayberry, R. I., Poizner, H., Burtis, D. B., Doty, L., & Heilman, K. M. (2013). Sign Language Aphasia From a Neurodegenerative Disease. Neurocase. https://doi:10.1080/13554794.2012.690427

Fathima, T. (2024). Real-Time Sign Language Recognition and Translation Using Deep Learning Techniques. Int Res J Adv Engg Hub. https://doi:10.47392/irjaeh.2024.0018

Fernando, P. and Wimalaratne, P. (2016). Sign language translation approach to sinhalese language. GSTF Journal on Computing (Joc), 5(1). https://doi.org/10.7603/s40601-016-0009-8

Ghoul, O. E. (2023). JUMLA-QSL-22: A Novel Qatari Sign Language Continuous Dataset. Ieee Access. https://doi:10.1109/access.2023.3324040

Huamani-Malca, J. (2018). Less Is More: Techniques to Reduce Overfitting in Your Transformer Model for Sign Language Recognition. https://doi:10.52591/lxai2023061812

Hughes, G., Hudgins, B., & MacDougall, J. (2004). Remote Sign Language Interpretation Using the Internet. https://doi:10.1109/dnsr.2004.1344750

Iduar Perdana, I. P. (2021). Classification of Sign Language Numbers Using the CNN Method. Jitter Jurnal Ilmiah Teknologi Dan Komputer. https://doi:10.24843/jtrti.2021.v02.i03.p07

Jane, B. (2021). Rapid application development model. NIX United. Retrieved from https://nix-united.com/blog/the-ultimate-guide-to-rapid-application-development/

Khuzayem, L. A. (2024). Efhamni: A Deep Learning-Based Saudi Sign Language Recognition Application. Sensors. https://doi:10.3390/s24103112

Koller, O., Zargaran, S., Ney, H., & Bowden, R. (2018). Deep Sign: Enabling Robust Statistical Continuous Sign Language Recognition via Hybrid CNN-HMMs. International Journal of Computer Vision. https://doi:10.1007/s11263-018-1121-3

Kumar, Y. (2024). Applying Swin Architecture to Diverse Sign Language Datasets. Electronics. https://doi:10.3390/electronics13081509

Lee, P. H., Spooner, C., & Harris, M. E. (2021). Access and Communication for Deaf Individuals in Australian Primary Care. Health Expectations. https://doi:10.1111/hex.13336

Liao, Y., Xiong, P., Min, W., Min, W., & Lu, J. (2019). Dynamic Sign Language Recognition Based on Video Sequence With BLSTM-3D Residual Networks. Ieee Access. https://doi:10.1109/access.2019.2904749

Lin, H. and Murli, N. (2022). Bim sign language translator using machine learning (tensorflow). Journal of Soft Computing and Data Mining, 3(1). https://doi.org/10.30880/jscdm.2022.03.01.007

Luqman, H., & El-Alfy, E.-S. M. (2021). Towards Hybrid Multimodal Manual and Non-Manual Arabic Sign Language Recognition: mArSL Database and Pilot Study. Electronics. https://doi:10.3390/electronics10141739

Madahana, M. C. I., Khoza-Shangase, K., Moroe, N., Mayombo, D., Nyandoro, O. T., & Ekoru, J. E. D. (2022). A Proposed Artificial Intelligence-Based Real-Time Speech-to-Text to Sign Language Translator for South African Official Languages for the COVID-19 Era and Beyond: In Pursuit of Solutions for the Hearing Impaired. South African Journal of Communication Disorders. https://doi:10.4102/sajcd.v69i2.915

Marin, G., Dominio, F., & Zanuttigh, P. (2014). Hand Gesture Recognition With Leap Motion and Kinect Devices. https://doi:10.1109/icip.2014.7025313

Marschark, M., Pelz, J. B., Convertino, C., Sapere, P., Arndt, M. E., & Seewagen, R. (2005). Classroom Interpreting and Visual Information Processing in Mainstream Education for Deaf Students: Live or Memorex<sup>®? American Educational Research Journal. https://doi:10.3102/00028312042004727

McFarland, D. J., & Wolpaw, J. R. (2018). Brain–computer Interface Use Is a Skill That User and System Acquire Together. Plos Biology. https://doi:10.1371/journal.pbio.2006719

McKee, R. (2014). Breaking News: Sign Language Interpreters on Television During Natural Disasters. Interpreting International Journal of Research and Practice in Interpreting. https://doi:10.1075/intp.16.1.06kee

Meador, H. E., & Zazove, P. (2005). Health Care Interactions With Deaf Culture. The Journal of the American Board of Family Medicine. https://doi:10.3122/jabfm.18.3.218

Mehta, N. A., Starner, T., Jackson, M. M., Babalola, K. O., & James, G. A. (2010). Recognizing Sign Language From Brain Imaging. https://doi:10.1109/icpr.2010.936

Napier, J. (2004). Sign Language Interpreter Training, Testing, and Accreditation: An International Comparison. American Annals of the Deaf. https://doi:10.1353/aad.2005.0007

Network. European Journal of Electrical Engineering and Computer Science. https://doi:10.24018/ejece.2021.5.3.332

Oh, J., Jeon, S., Kim, B., Kim, M., Kang, S.-W., Kwon, H.-C., … Song, Y. (2017). Avatar-Based Sign Language Interpretations for Weather Forecast and Other TV Programs. Smpte Motion Imaging Journal. https://doi:10.5594/jmi.2016.2632278

Olabanji, A. O., & Ponnle, A. A. (2021). Development of a Computer Aided Real-Time Interpretation System for Indigenous Sign Language in Nigeria Using Convolutional Neural

Papastratis, I., Chatzikonstantinou, C., Konstantinidis, D., Dimitropoulos, K., & Daras, P. (2021). Artificial Intelligence Technologies for Sign Language. Sensors. https://doi:10.3390/s21175843

Pendergrass, K. M., Newman, S. D., Jones, E. G., & Jenkins, C. (2017). Deaf: A Concept Analysis From a Cultural Perspective Using the Wilson Method of Concept Analysis Development. Clinical Nursing Research. https://doi:10.1177/1054773817719821

Pinilla, S., Walther, S., Hofmeister, A., & Huwendiek, S. (2019). Primary non-communicable disease prevention and communication barriers of deaf sign language users: a qualitative study. International Journal for Equity in Health, 18(1). https://doi.org/10.1186/s12939-019-0976-4

Quinto-Pozos, D. (2011). Teaching American Sign Language to Hearing Adult Learners. Annual Review of Applied Linguistics. https://doi:10.1017/s0267190511000195

Rachdito, E. and Hidayat, Z. (2022). Emoticons as self-disclosure in social media and its meaning for people who are deaf. Disability CBR & Inclusive Development, 32(4), 40. https://doi.org/10.47985/dcidj.471

Rodríguez-Tapia, B., Ochoa-Zezzatti, A., Soto Marrufo, Á. I., Arballo, N. C., & Carlos, P. A. (2019). Sign Language Recognition Based on EMG Signals Through a Hibrid Intelligent System. Research in Computing Science. https://doi:10.13053/rcs-148-6-19

Saladin, S. P., & Hansmann, S. (2008). Psychosocial Variables Related to the Adoption of Video Relay Services Among Deaf or Hard-of-Hearing Employees at the Texas School for the Deaf. Assistive Technology. https://doi:10.1080/10400435.2008.10131930

Salagar, M., Kulkarni, P., & Gondane, S. (2013). Implementation of Dynamic Time Warping for Gesture Recognition in Sign Language Using High Performance Computing. https://doi:10.1109/ichci-ieee.2013.6887814

Santos, A. S., & Freire Portes, A. J. (2019). Perceptions of Deaf Subjects About Communication in Primary Health Care. Revista Latino-Americana De Enfermagem. https://doi:10.1590/1518-8345.2612.3127

Sevli, O., & Kemaloğlu, N. (2020). Turkish Sign Language Digits Classification With CNN Using Different Optimizers. International Advanced Researches and Engineering Journal. https://doi:10.35860/iarej.700564

Stefanidis, K., Konstantinidis, D., Kalvourtzis, A., Dimitropoulos, K., & Daras, P. (2020). 3D Technologies and Applications in Sign Language. https://doi:10.4018/978-1-5225-5294-9.ch003

Sun, C., Zhang, T., & Xu, C. (2015). Latent Support Vector Machine Modeling for Sign Language Recognition With Kinect. Acm Transactions on Intelligent Systems and Technology. https://doi:10.1145/2629481

Tannenbaum-Baruchi, C., & Feder-Bubis, P. (2017). New Sign Language New(S): The Globalization of Sign Language in the Smartphone Era. Disability & Society. https://doi:10.1080/09687599.2017.1383034

van Dijk, R., Christoffels, I. K., Postma, A., & Hermans, D. (2011). The Relation Between the Working Memory Skills of Sign Language Interpreters and the Quality of Their Interpretations. Bilingualism Language and Cognition. https://doi:10.1017/s1366728911000198

Verma, P., & Badli, K. (2022). Real-Time Sign Language Detection Using TensorFlow, OpenCV and Python. International Journal for Research in Applied Science and Engineering Technology. https://doi:10.22214/ijraset.2022.43439

Wadhawan, A., & Kumar, P. (2019). Sign Language Recognition Systems: A Decade Systematic Literature Review. Archives of Computational Methods in Engineering. https://doi:10.1007/s11831-019-09384-2

Wang, P., Zhou, Y., Li, Z., Huang, S., & Zhang, D. (2021). Neural Decoding of Chinese Sign Language With Machine Learning for Brain–Computer Interfaces. Ieee Transactions on Neural Systems and Rehabilitation Engineering. https://doi:10.1109/tnsre.2021.3137340

Yang, T. (2019). Comparison of Chinese and American Sign Language Interpretation Majors and Some Enlightenment. https://doi:10.2991/assehr.k.191221.014

ZainEldin, H. (2024). Silent No More: A Comprehensive Review of Artificial Intelligence, Deep Learning, and Machine Learning in Facilitating Deaf and Mute Communication. Artificial Intelligence Review. https://doi:10.1007/s10462-024-10816-0

Zhang, W. (2022). Sign Language Recognition Based on Depth Image Processing. Highlights in Science Engineering and Technology. https://doi:10.54097/hset.v23i.3123

Zhou, H., Zhou, W., Zhou, Y., & Li, H. (2020). Spatial-temporal multi-cue network for continuous sign language recognition. Proceedings of the Aaai Conference on Artificial Intelligence, 34(07), 13009-13016. https://doi.org/10.1609/aaai.v34i07.7001

Appendix A - Project Document

IN-DEPTH PROJECT DOCUMENTATION

Full Candidate Name: Agwupuye Richard Ukwu

Student ID: BU/22A/IT/6308

Title: Sign Language Translation Using Gesture recognition with TensorFlow

Course of Study: B.Sc. Computer Science.

Background and Motivation

Effective communication is fundamental to human interaction. However, individuals who rely on American Sign Language (ASL) often face barriers when communicating with those unfamiliar with the language. These barriers highlight the need for technological solutions to bridge this communication gap.

With advancements in machine learning and computer vision, it is now possible to create systems that can recognize hand gestures in real-time and translate them into text or speech. Such a system not only empowers the ASL community but also fosters inclusivity in workplaces, educational institutions, and social settings.

The motivation for this project stems from the desire to address these communication barriers and leverage technology to create a more inclusive society. By providing a platform that accurately

translates ASL gestures, individuals with hearing or speech impairments can participate more fully in everyday activities and have their voices heard.

Statement of the Problem

The lack of accessible tools for real-time ASL translation continues to hinder effective communication between ASL users and non-signers. Current solutions often require expensive equipment or fail to meet accuracy and usability expectations.

Key problems include:

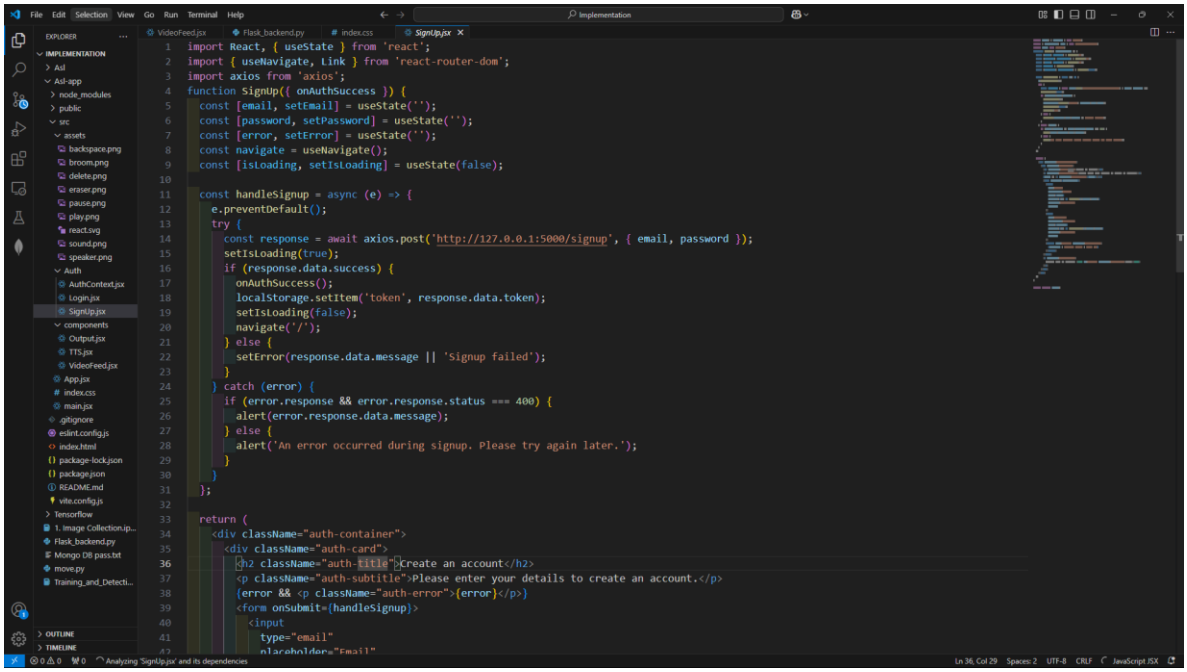
Manual Dependency: Existing methods rely heavily on human interpretation, which may not always be feasible or accurate.

Limited Accessibility: Many available systems are costly or require specialized hardware, limiting their reach.

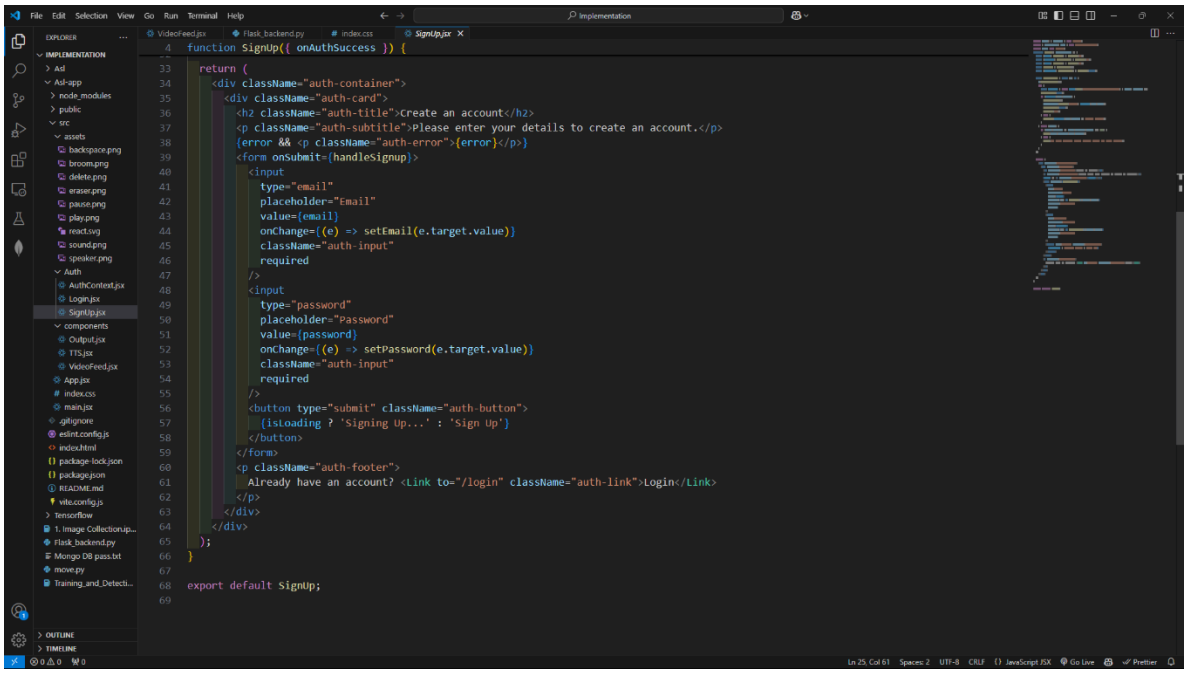
This project addresses these challenges by providing an affordable, accessible, and efficient ASL translation system using widely available hardware such as cameras and modern browsers. The solution focuses on accuracy, ease of use, and fostering inclusivity in communication.

Appendix B- Source Codes

Sign-Up page

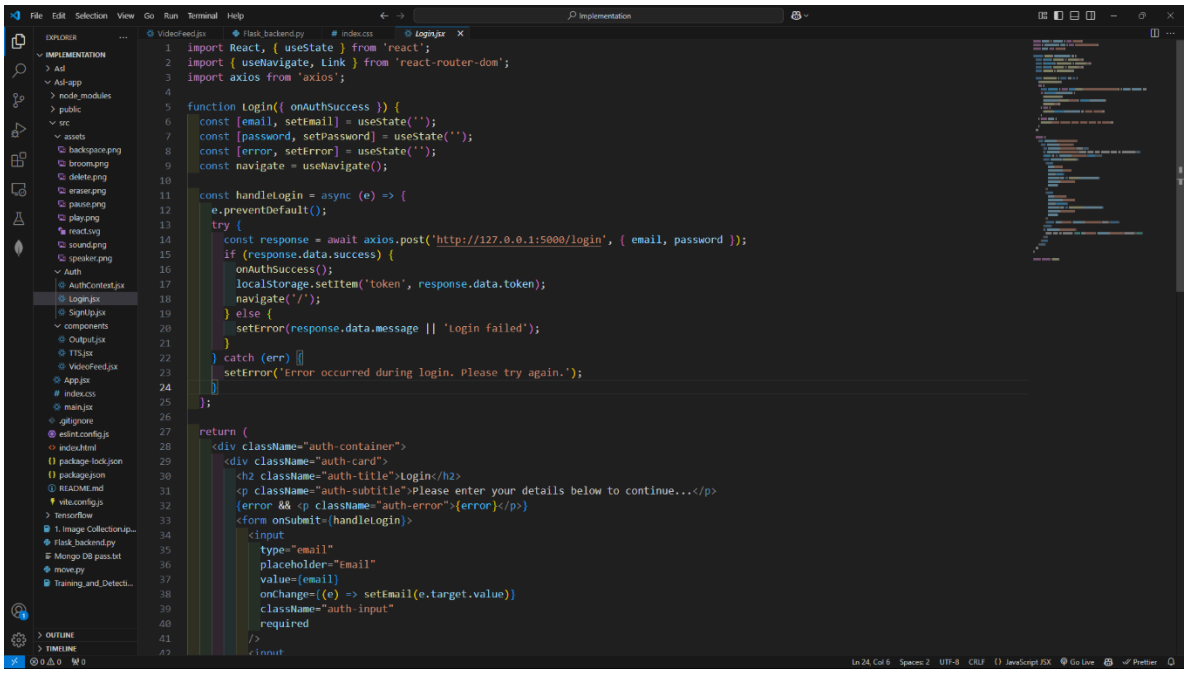


```
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import axios from 'axios';
function SignUp({ onAuthSuccess }) {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();
  const [isLoading, setIsLoading] = useState(false);

  const handleSignup = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('http://127.0.0.1:5000/signup', { email, password });
      setIsLoading(true);
      if (response.data.success) {
        onAuthSuccess();
        localStorage.setItem('token', response.data.token);
        setIsLoading(false);
        navigate('/');
      } else {
        setError(response.data.message || 'Signup failed');
      }
    } catch (error) {
      if (error.response && error.response.status === 400) {
        alert(error.response.data.message);
      } else {
        alert('An error occurred during signup. Please try again later.');
      }
    }
  };

  return (
    <div className="auth-container">
      <div className="auth-card">
        <h2 className="auth-title">Create an account</h2>
        <p className="auth-subtitle">Please enter your details to create an account.</p>
        {error && <p className="auth-error">{error}</p>}
        <form onSubmit={handleSignup}>
          <input
            type="email"
            placeholder="Email"
```

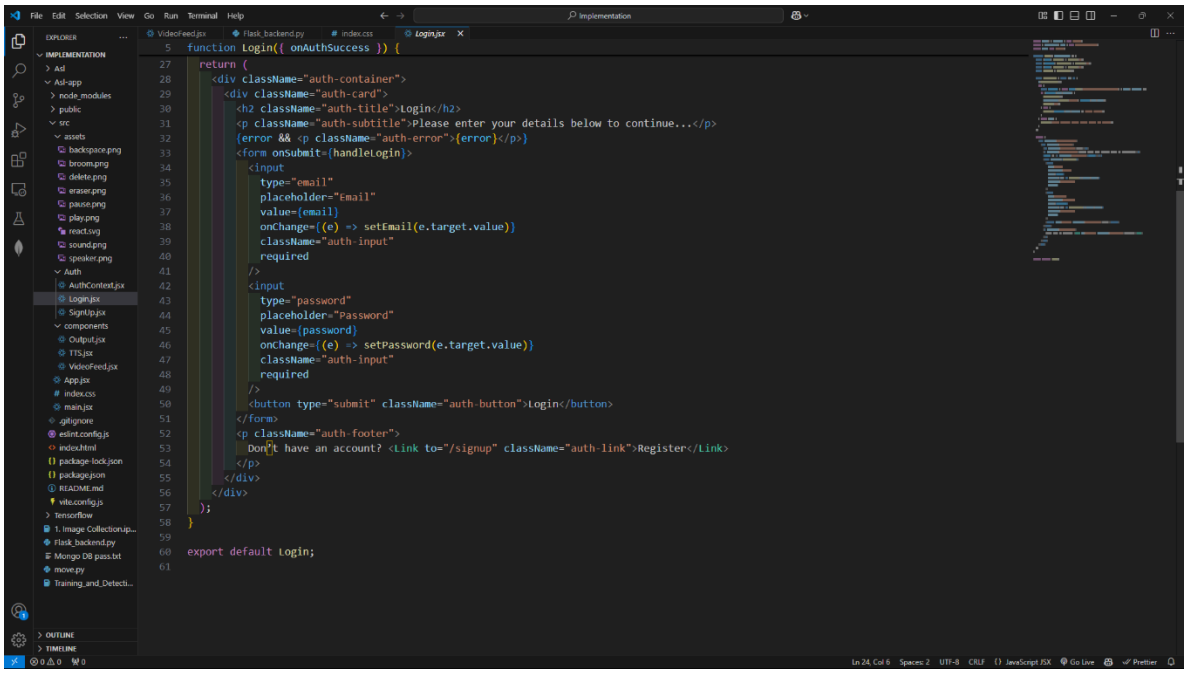


```
function SignUp({ onAuthSuccess }) {
  return (
    <div className="auth-container">
      <div className="auth-card">
        <h2 className="auth-title">Create an account</h2>
        <p className="auth-subtitle">Please enter your details to create an account.</p>
        {error && <p className="auth-error">{error}</p>}
        <form onSubmit={handleSignup}>
          <input
            type="email"
            placeholder="Email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="auth-input"
            required
          />
          <input
            type="password"
            placeholder="Password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="auth-input"
            required
          />
          <button type="submit" className="auth-button">
            {isLoading ? 'Signing Up...' : 'Sign Up'}
          </button>
        </form>
        <p className="auth-footer">
          Already have an account? <Link to="/login" className="auth-link">Login</Link>
        </p>
      </div>
    </div>
  );
}

export default SignUp;
```

Login page:

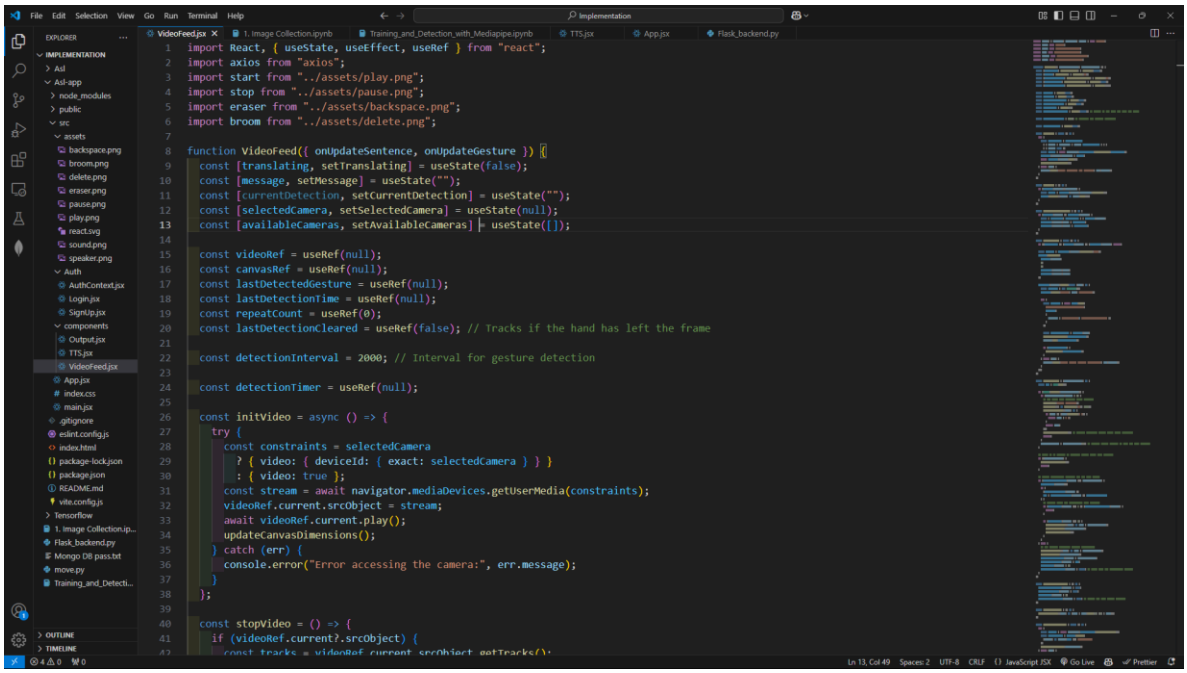


```
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import axios from 'axios';

function Login({ onAuthSuccess }) {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('http://127.0.0.1:5000/login', { email, password });
      if (response.data.success) {
        onAuthSuccess();
        localStorage.setItem('token', response.data.token);
        navigate('/');
      } else {
        setError(response.data.message || 'Login failed');
      }
    } catch (err) {
      setError('Error occurred during login. Please try again.');
    }
  };

  return (
    <div className="auth-container">
      <div className="auth-card">
        <h2 className="auth-title">Login</h2>
        <p className="auth-subtitle">Please enter your details below to continue...</p>
        {error && <p className="auth-error">{error}</p>}
        <form onSubmit={handleLogin}>
          <input
            type="email"
            placeholder="Email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="auth-input"
            required
          />
```

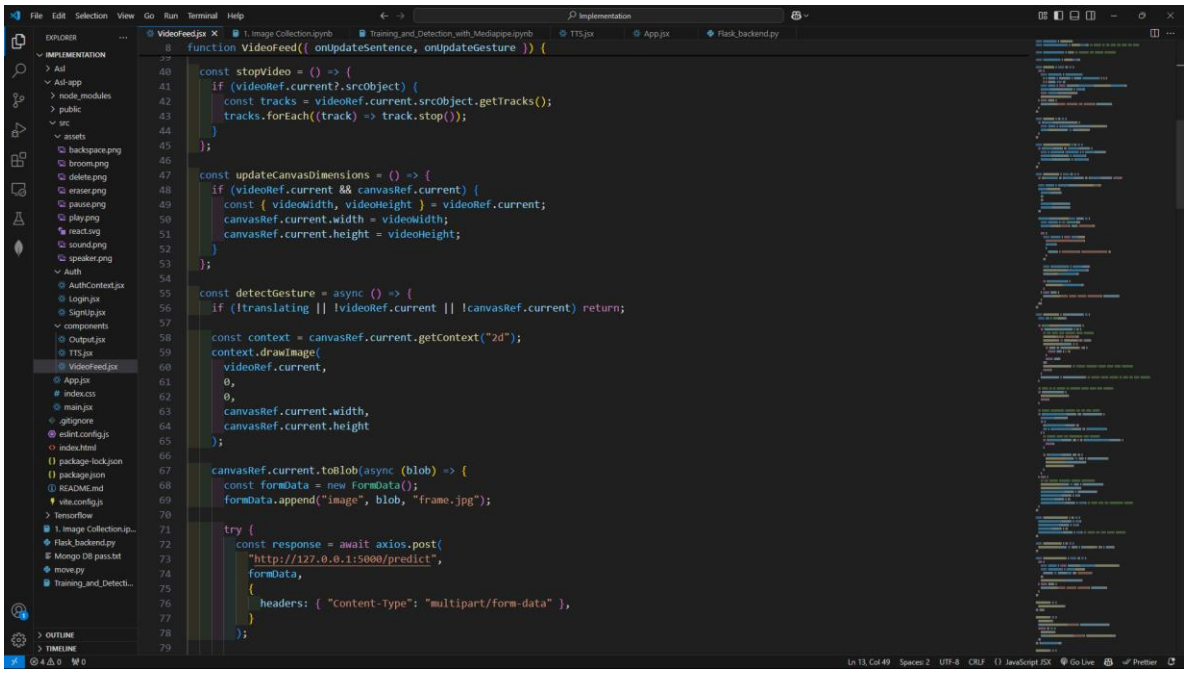


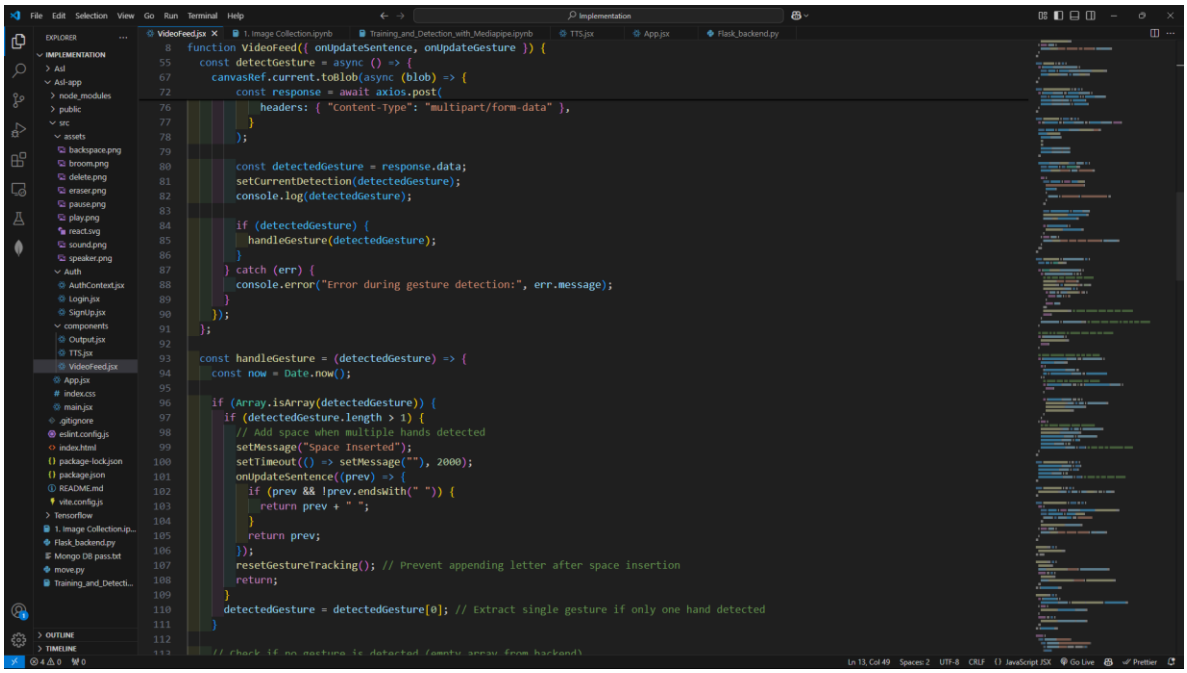
```
function Login({ onAuthSuccess }) {
  return (
    <div className="auth-container">
      <div className="auth-card">
        <h2 className="auth-title">Login</h2>
        <p className="auth-subtitle">Please enter your details below to continue...</p>
        {error && <p className="auth-error">{error}</p>}
        <form onSubmit={handleLogin}>
          <input
            type="email"
            placeholder="Email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="auth-input"
            required
          />
          <input
            type="password"
            placeholder="Password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="auth-input"
            required
          />
          <button type="submit" className="auth-button">Login</button>
        </form>
        <p className="auth-footer">
          Don't have an account? <Link to="/signup" className="auth-link">Register</Link>
        </p>
      </div>
    </div>
  );
}

export default Login;
```

Translation page:

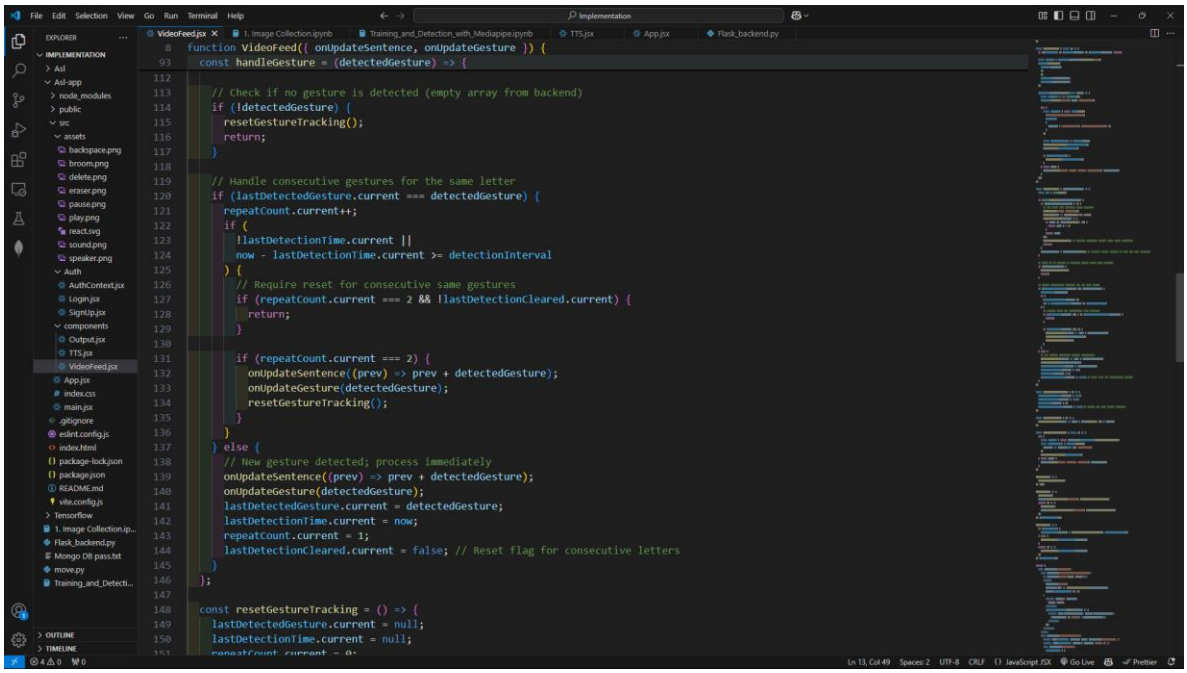


```jsx
import React, { useState, useEffect, useRef } from "react";
import axios from "axios";
import start from "../assets/play.png";
import stop from "../assets/pause.png";
import eraser from "../assets/backspace.png";
import broom from "../assets/delete.png";

function VideoFeed({ onUpdateSentence, onUpdateGesture }) {
  const [translating, setTranslating] = useState(false);
  const [message, setMessage] = useState("");
  const [currentDetection, setCurrentDetection] = useState("");
  const [selectedCamera, setSelectedCamera] = useState(null);
  const [availableCameras, setAvailableCameras] = useState([]);

  const videoRef = useRef(null);
  const canvasRef = useRef(null);
  const lastDetectedGesture = useRef(null);
  const lastDetectionTime = useRef(null);
  const repeatCount = useRef(0);
  const lastDetectionCleared = useRef(false); // Tracks if the hand has left the frame

  const detectionInterval = 2000; // Interval for gesture detection

  const detectionTimer = useRef(null);

  const initVideo = async () => {
    try {
      const constraints = selectedCamera
        ? { video: { deviceId: { exact: selectedCamera } } }
        : { video: true };
      const stream = await navigator.mediaDevices.getUserMedia(constraints);
      videoRef.current.srcObject = stream;
      await videoRef.current.play();
      updateCanvasDimensions();
    } catch (err) {
      console.error("Error accessing the camera:", err.message);
    }
  };

  const stopVideo = () => {
    if (videoRef.current?.srcObject) {
      const tracks = videoRef.current.srcObject.getTracks();
      tracks.forEach((track) => track.stop());
    }
  };

  const updateCanvasDimensions = () => {
    if (videoRef.current && canvasRef.current) {
      const { videoWidth, videoHeight } = videoRef.current;
      canvasRef.current.width = videoWidth;
      canvasRef.current.height = videoHeight;
    }
  };

  const detectGesture = async () => {
    if (!translating || !videoRef.current || !canvasRef.current) return;

    const context = canvasRef.current.getContext("2d");
    context.drawImage(
      videoRef.current,
      0,
      0,
      canvasRef.current.width,
      canvasRef.current.height
    );

    canvasRef.current.toBlob(async (blob) => {
      const formData = new FormData();
      formData.append("image", blob, "frame.jpg");

      try {
        const response = await axios.post(
          "http://127.0.0.1:5000/predict",
          formData,
          {
            headers: { "Content-Type": "multipart/form-data" },
          }
        );
```

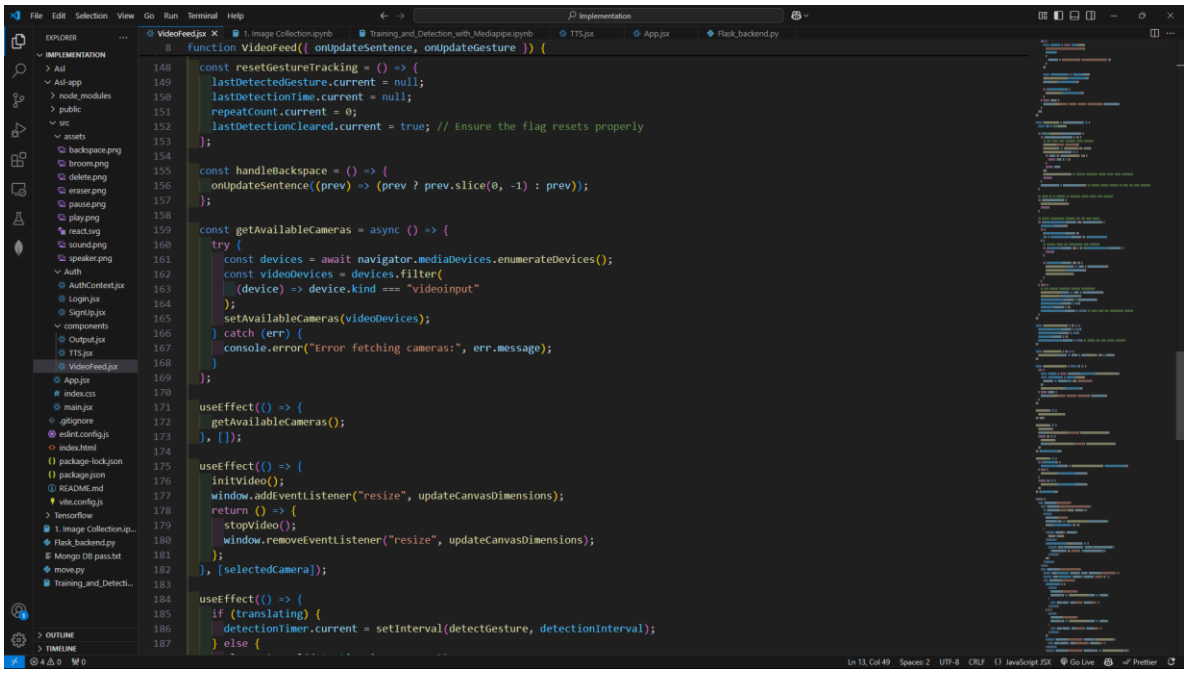




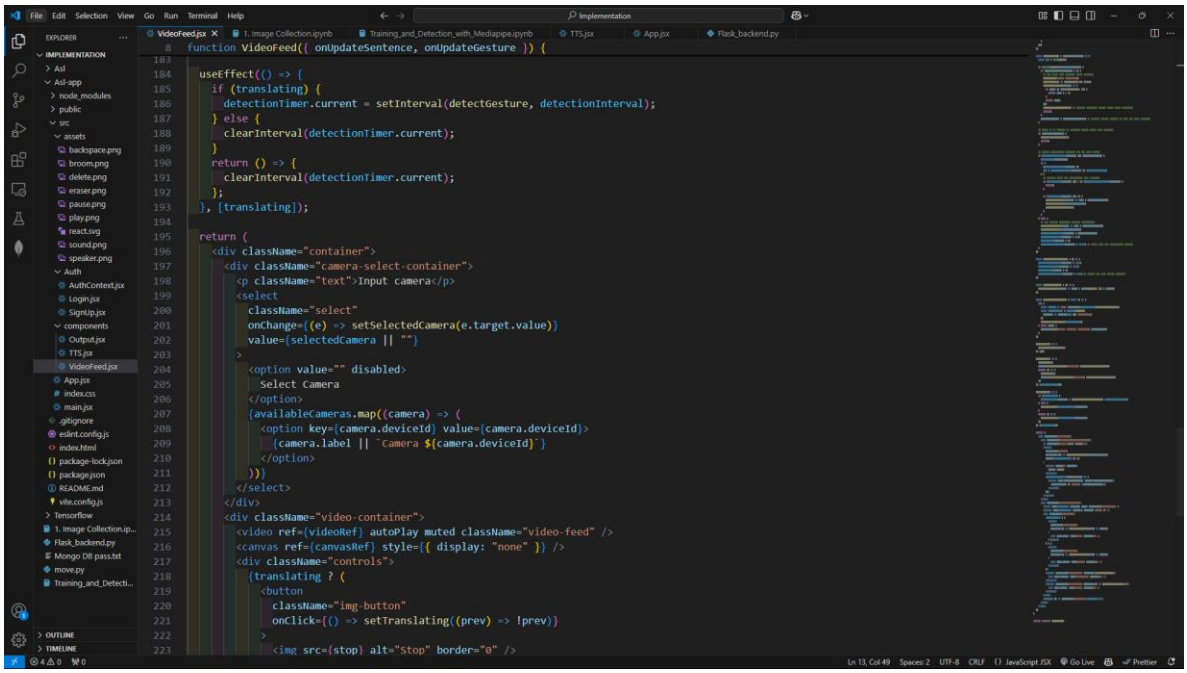
```
function VideoFeed({ onUpdateSentence, onUpdateGesture }) {
  const detectGesture = async () => {
    canvasRef.current.toBlob(async (blob) => {
        const response = await axios.post(
          headers: { "Content-Type": "multipart/form-data" },
        }
        );

        const detectedGesture = response.data;
        setCurrentDetection(detectedGesture);
        console.log(detectedGesture);

        if (detectedGesture) {
          handleGesture(detectedGesture);
        }
      } catch (err) {
        console.error("Error during gesture detection:", err.message);
      }
    });
  };

  const handleGesture = (detectedGesture) => {
    const now = Date.now();

    if (Array.isArray(detectedGesture)) {
      if (detectedGesture.length > 1) {
        // Add space when multiple hands detected
        setMessage("Space Inserted");
        setTimeout(() => setMessage(""), 2000);
        onUpdateSentence((prev) => {
          if (prev && !prev.endsWith(" ")) {
            return prev + " ";
          }
          return prev;
        });
        resetGestureTracking(); // Prevent appending letter after space insertion
        return;
      }
      detectedGesture = detectedGesture[0]; // Extract single gesture if only one hand detected
    }
```



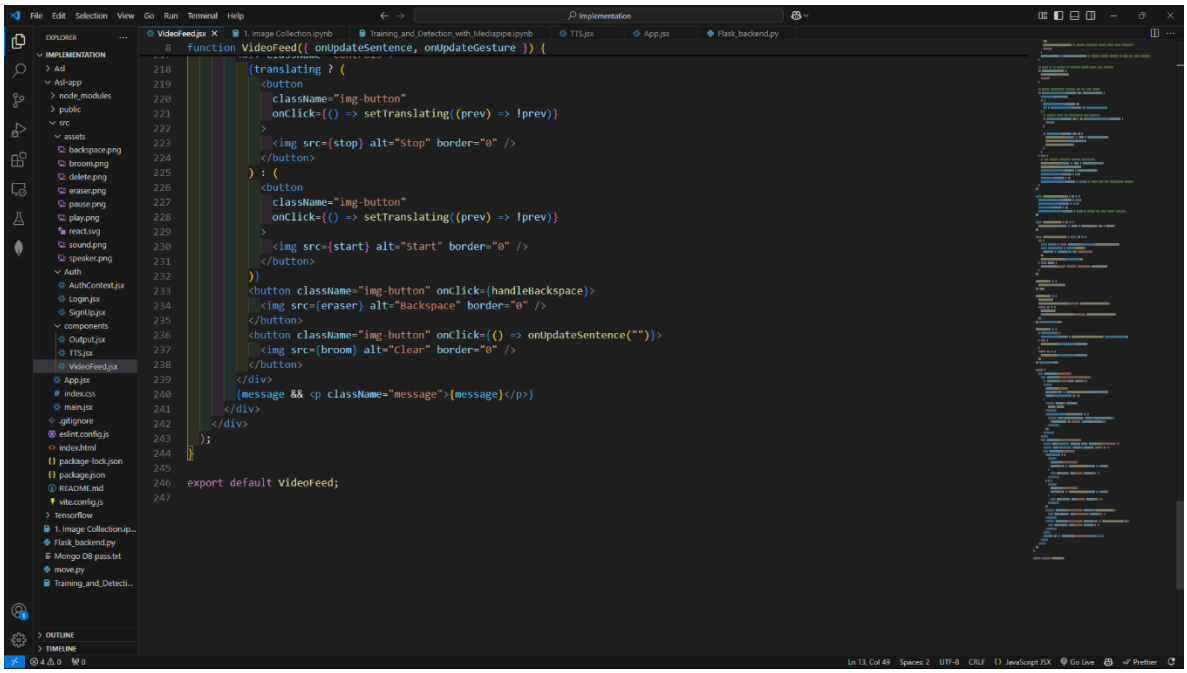
```
function VideoFeed({ onUpdateSentence, onUpdateGesture }) {
  const handleGesture = (detectedGesture) => {

    // Check if no gesture is detected (empty array from backend)
    if (!detectedGesture) {
      resetGestureTracking();
      return;
    }

    // Handle consecutive gestures for the same letter
    if (lastDetectedGesture.current === detectedGesture) {
      repeatCount.current++;
      if (
        !lastDetectionTime.current ||
        now - lastDetectionTime.current >= detectionInterval
      ) {
        // Require reset for consecutive same gestures
        if (repeatCount.current === 2 && !lastDetectionCleared.current) {
          return;
        }

        if (repeatCount.current === 2) {
          onUpdateSentence((prev) => prev + detectedGesture);
          onUpdateGesture(detectedGesture);
          resetGestureTracking();
        }
      }
    } else {
      // New gesture detected; process immediately
      onUpdateSentence((prev) => prev + detectedGesture);
      onUpdateGesture(detectedGesture);
      lastDetectedGesture.current = detectedGesture;
      lastDetectionTime.current = now;
      repeatCount.current = 1;
      lastDetectionCleared.current = false; // Reset flag for consecutive letters
    }
  };

  const resetGestureTracking = () => {
    lastDetectedGesture.current = null;
    lastDetectionTime.current = null;
```



```
function VideoFeed({ onUpdateSentence, onUpdateGesture }) {
  const resetGestureTracking = () => {
    lastDetectedGesture.current = null;
    lastDetectionTime.current = null;
    repeatCount.current = 0;
    lastDetectionCleared.current = true; // Ensure the flag resets properly
  };

  const handleBackspace = () => {
    onUpdateSentence((prev) => (prev ? prev.slice(0, -1) : prev));
  };

  const getAvailableCameras = async () => {
    try {
      const devices = await navigator.mediaDevices.enumerateDevices();
      const videoDevices = devices.filter(
        (device) => device.kind === "videoinput"
      );
      setAvailableCameras(videoDevices);
    } catch (err) {
      console.error("Error fetching cameras:", err.message);
    }
  };

  useEffect(() => {
    getAvailableCameras();
  }, []);

  useEffect(() => {
    initVideo();
    window.addEventListener("resize", updateCanvasDimensions);
    return () => {
      stopVideo();
      window.removeEventListener("resize", updateCanvasDimensions);
    };
  }, [selectedCamera]);

  useEffect(() => {
    if (translating) {
      detectionTimer.current = setInterval(detectGesture, detectionInterval);
    } else {
```



```
function VideoFeed({ onUpdateSentence, onUpdateGesture }) {

  useEffect(() => {
    if (translating) {
      detectionTimer.current = setInterval(detectGesture, detectionInterval);
    } else {
      clearInterval(detectionTimer.current);
    }
    return () => {
      clearInterval(detectionTimer.current);
    };
  }, [translating]);

  return (
    <div className="container">
      <div className="camera-select-container">
        <p className="text">Input camera</p>
        <select
          className="select"
          onChange={(e) => setSelectedCamera(e.target.value)}
          value={selectedCamera || ""}
        >
          <option value="" disabled>
            Select Camera
          </option>
          {availableCameras.map((camera) => (
            <option key={camera.deviceId} value={camera.deviceId}>
              {camera.label || `Camera ${camera.deviceId}`}
            </option>
          ))}
        </select>
      </div>
      <div className="video-container">
        <video ref={videoRef} autoPlay muted className="video-feed" />
        <canvas ref={canvasRef} style={{ display: "none" }} />
        <div className="controls">
          {translating ? (
            <button
              className="img-button"
              onClick={() => setTranslating((prev) => !prev)}
            >
              <img src={stop} alt="Stop" border="0" />
```


function VideoFeed({ onUpdateSentence, onUpdateGesture }) {
{translating ? (
<button
className="img-button"
onClick={() => setTranslating((prev) => !prev)}
>

</button>
) : (
<button
className="img-button"
onClick={() => setTranslating((prev) => !prev)}
>

</button>
)}
<button className="img-button" onClick={handleBackspace}>

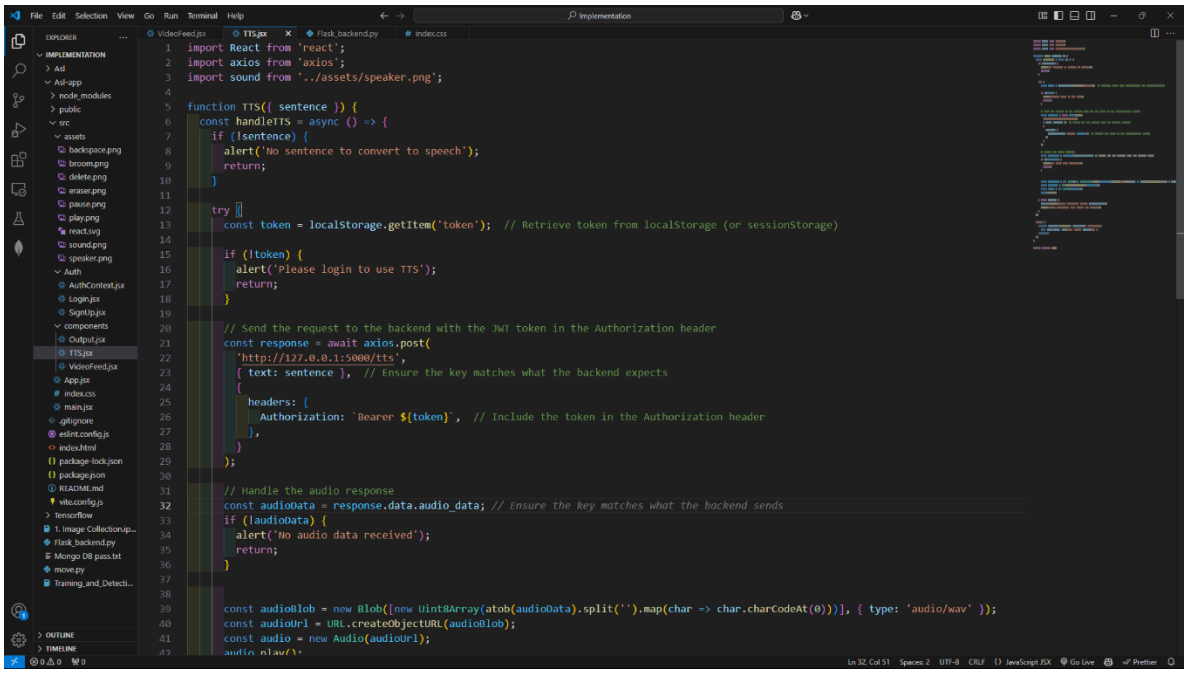
</button>
<button className="img-button" onClick={() => onUpdateSentence("")}>

</button>

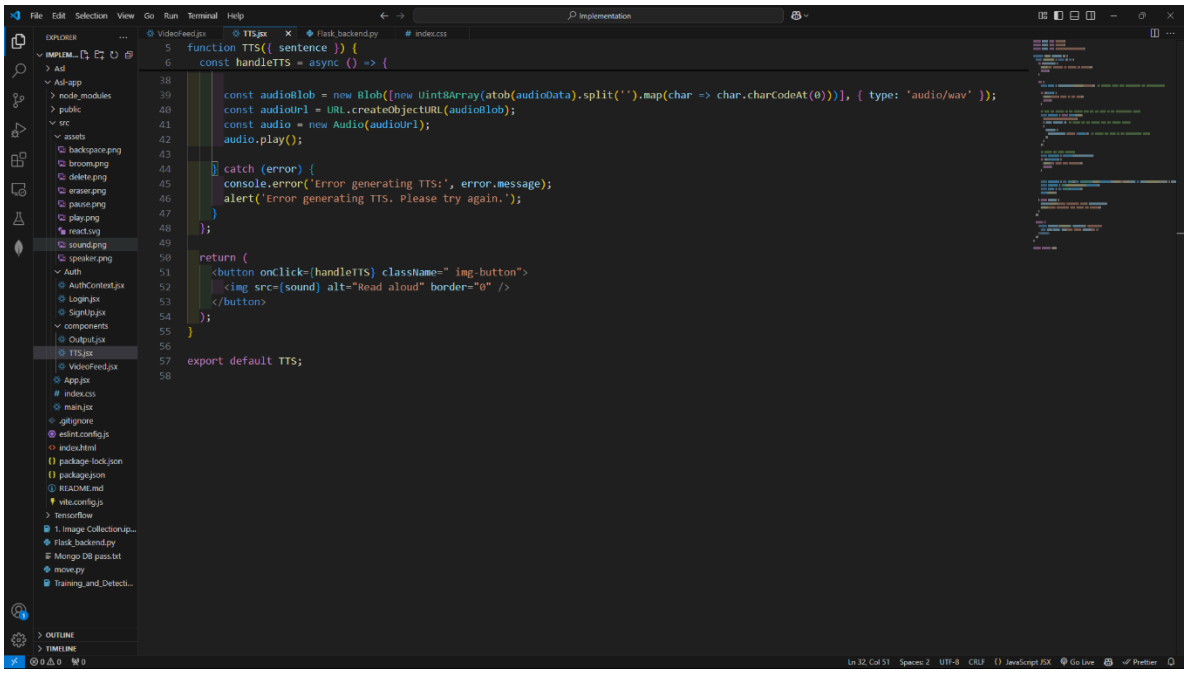
{message && {message} }

);
}
export default VideoFeed;

TTS:

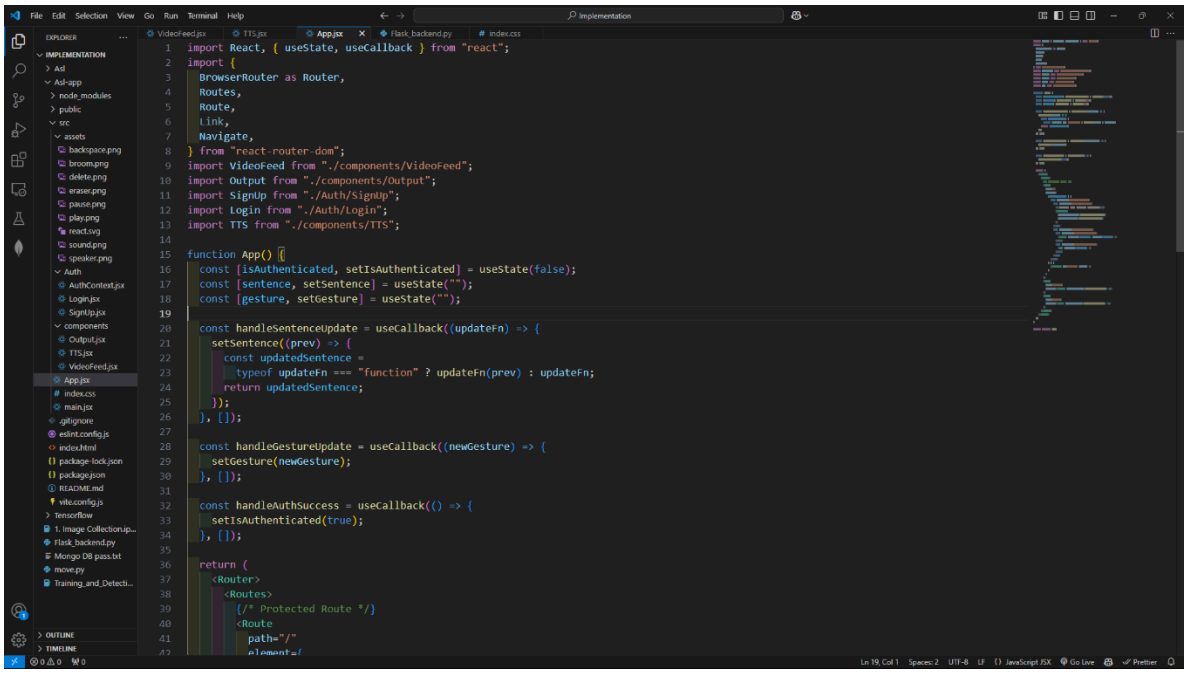


```jsx
import React from 'react';
import axios from 'axios';
import sound from '../assets/speaker.png';

function TTS({ sentence }) {
  const handleTTS = async () => {
    if (!sentence) {
      alert('No sentence to convert to speech');
      return;
    }

    try {
      const token = localStorage.getItem('token');  // Retrieve token from localStorage (or sessionStorage)

      if (!token) {
        alert('Please login to use TTS');
        return;
      }

      // Send the request to the backend with the JWT token in the Authorization header
      const response = await axios.post(
        'http://127.0.0.1:5000/tts',
        { text: sentence },  // Ensure the key matches what the backend expects
        {
          headers: {
            Authorization: `Bearer ${token}`,  // Include the token in the Authorization header
          },
        }
      );

      // Handle the audio response
      const audioData = response.data.audio_data; // Ensure the key matches what the backend sends
      if (!audioData) {
        alert('No audio data received');
        return;
      }


      const audioBlob = new Blob([new Uint8Array(atob(audioData).split('').map(char => char.charCodeAt(0)))], { type: 'audio/wav' });
      const audioUrl = URL.createObjectURL(audioBlob);
      const audio = new Audio(audioUrl);
      audio.play();
```

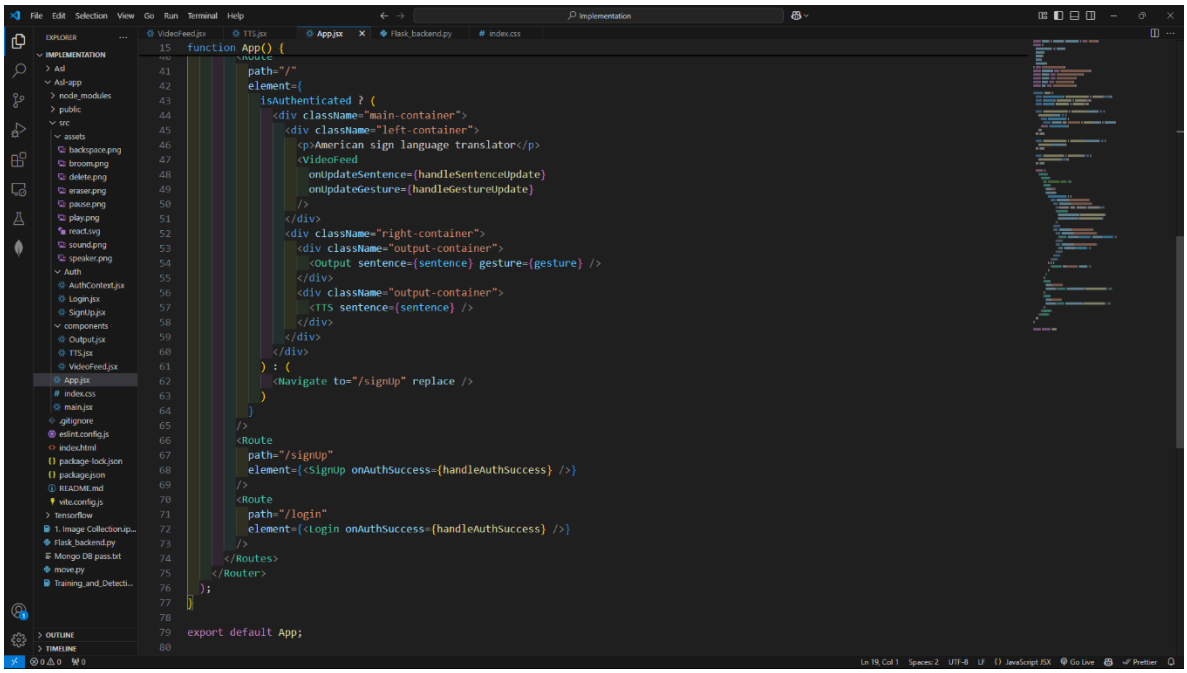


```jsx
function TTS({ sentence }) {
  const handleTTS = async () => {


      const audioBlob = new Blob([new Uint8Array(atob(audioData).split('').map(char => char.charCodeAt(0)))], { type: 'audio/wav' });
      const audioUrl = URL.createObjectURL(audioBlob);
      const audio = new Audio(audioUrl);
      audio.play();

    } catch (error) {
      console.error('Error generating TTS:', error.message);
      alert('Error generating TTS. Please try again.');
    }
  };

  return (
    <button onClick={handleTTS} className=" img-button">
      <img src={sound} alt="Read aloud" border="0" />
    </button>
  );
}

export default TTS;
```

Front-End

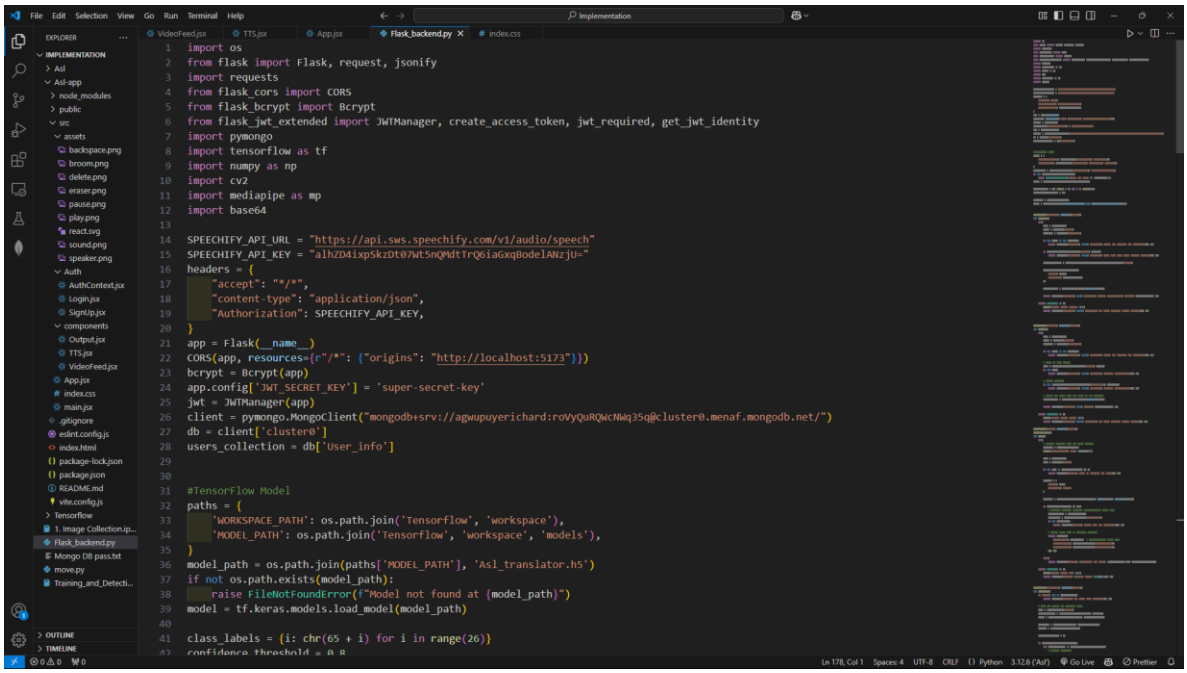


```jsx
import React, { useState, useCallback } from "react";
import {
  BrowserRouter as Router,
  Routes,
  Route,
  Link,
  Navigate,
} from "react-router-dom";
import VideoFeed from "./components/VideoFeed";
import Output from "./components/Output";
import SignUp from "./Auth/SignUp";
import Login from "./Auth/Login";
import TTS from "./components/TTS";

function App() {
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [sentence, setSentence] = useState("");
  const [gesture, setGesture] = useState("");

  const handleSentenceUpdate = useCallback((updateFn) => {
    setSentence((prev) => {
      const updatedSentence =
        typeof updateFn === "function" ? updateFn(prev) : updateFn;
      return updatedSentence;
    });
  }, []);

  const handleGestureUpdate = useCallback((newGesture) => {
    setGesture(newGesture);
  }, []);

  const handleAuthSuccess = useCallback(() => {
    setIsAuthenticated(true);
  }, []);

  return (
    <Router>
      <Routes>
        {/* Protected Route */}
        <Route
          path="/"
```

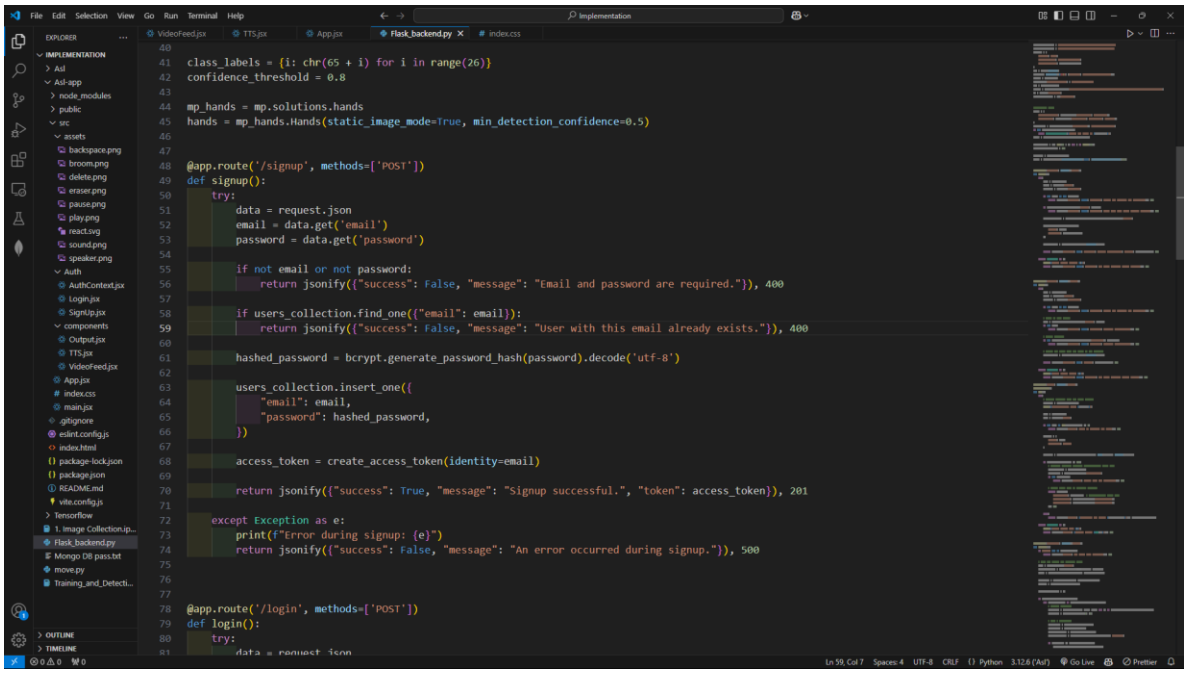


```jsx
function App() {
          path="/"
          element={
            isAuthenticated ? (
              <div className="main-container">
                <div className="left-container">
                  <p>American sign language translator</p>
                  <VideoFeed
                    onUpdateSentence={handleSentenceUpdate}
                    onUpdateGesture={handleGestureUpdate}
                  />
                </div>
                <div className="right-container">
                  <div className="output-container">
                    <Output sentence={sentence} gesture={gesture} />
                  </div>
                  <div className="output-container">
                    <TTS sentence={sentence} />
                  </div>
                </div>
              </div>
            ) : (
              <Navigate to="/signUp" replace />
            )
          }
        />
        <Route
          path="/signUp"
          element={<SignUp onAuthSuccess={handleAuthSuccess} />}
        />
        <Route
          path="/login"
          element={<Login onAuthSuccess={handleAuthSuccess} />}
        />
      </Routes>
    </Router>
  );
}

export default App;
```

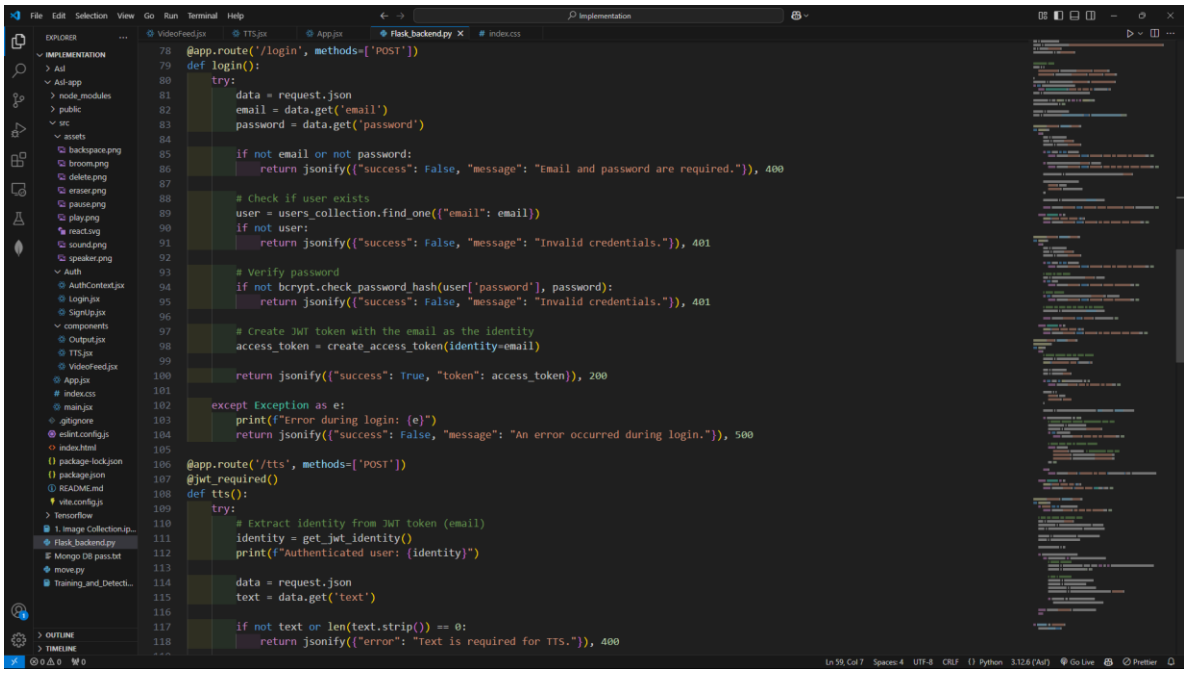
Back-End



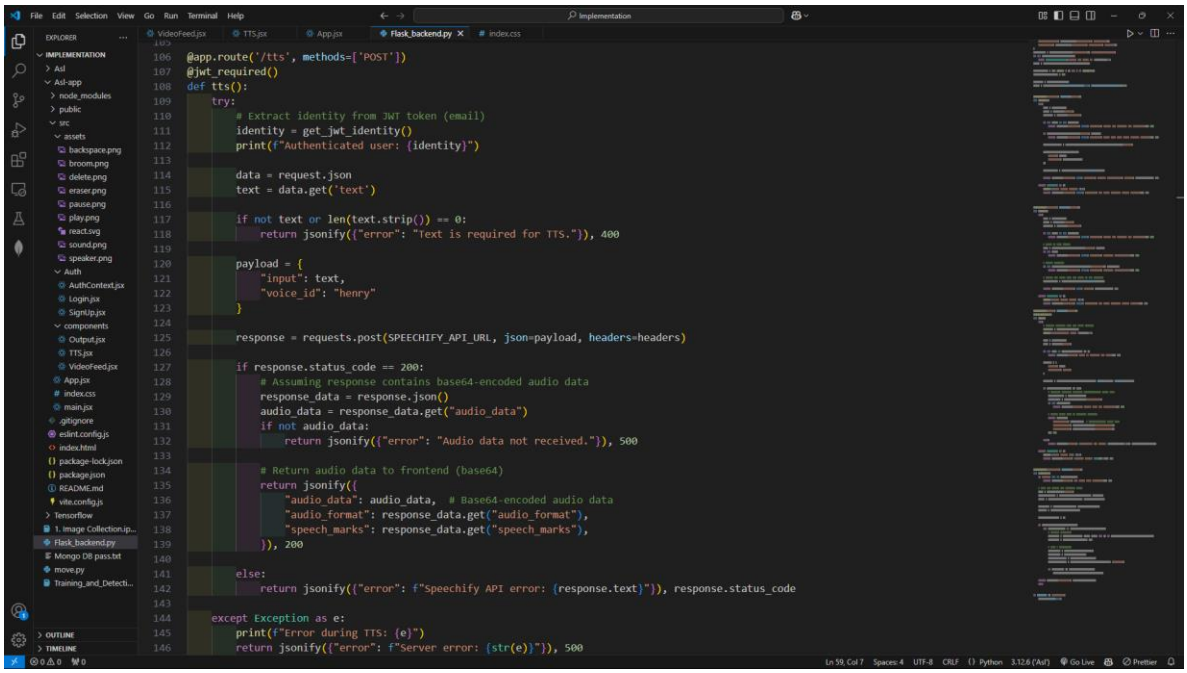
```python
import os
from flask import Flask, request, jsonify
import requests
from flask_cors import CORS
from flask_bcrypt import Bcrypt
from flask_jwt_extended import JWTManager, create_access_token, jwt_required, get_jwt_identity
import pymongo
import tensorflow as tf
import numpy as np
import cv2
import mediapipe as mp
import base64

SPEECHIFY_API_URL = "https://api.sws.speechify.com/v1/audio/speech"
SPEECHIFY_API_KEY = "alhZD4ixpSkzDt07Wt5nQMdtTrQ6iaGxqBodelANzjU="
headers = {
    "accept": "*/*",
    "content-type": "application/json",
    "Authorization": SPEECHIFY_API_KEY,
}
app = Flask(__name__)
CORS(app, resources={r"/*": {"origins": "http://localhost:5173"}})
bcrypt = Bcrypt(app)
app.config['JWT_SECRET_KEY'] = 'super-secret-key'
jwt = JWTManager(app)
client = pymongo.MongoClient("mongodb+srv://agwupuyerichard:roVyQuRQWcNWq35q@cluster0.menaf.mongodb.net/")
db = client['cluster0']
users_collection = db['User_info']


#TensorFlow Model
paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace', 'models'),
}
model_path = os.path.join(paths['MODEL_PATH'], 'Asl_translator.h5')
if not os.path.exists(model_path):
    raise FileNotFoundError(f"Model not found at {model_path}")
model = tf.keras.models.load_model(model_path)

class_labels = {i: chr(65 + i) for i in range(26)}
confidence_threshold = 0.8
```



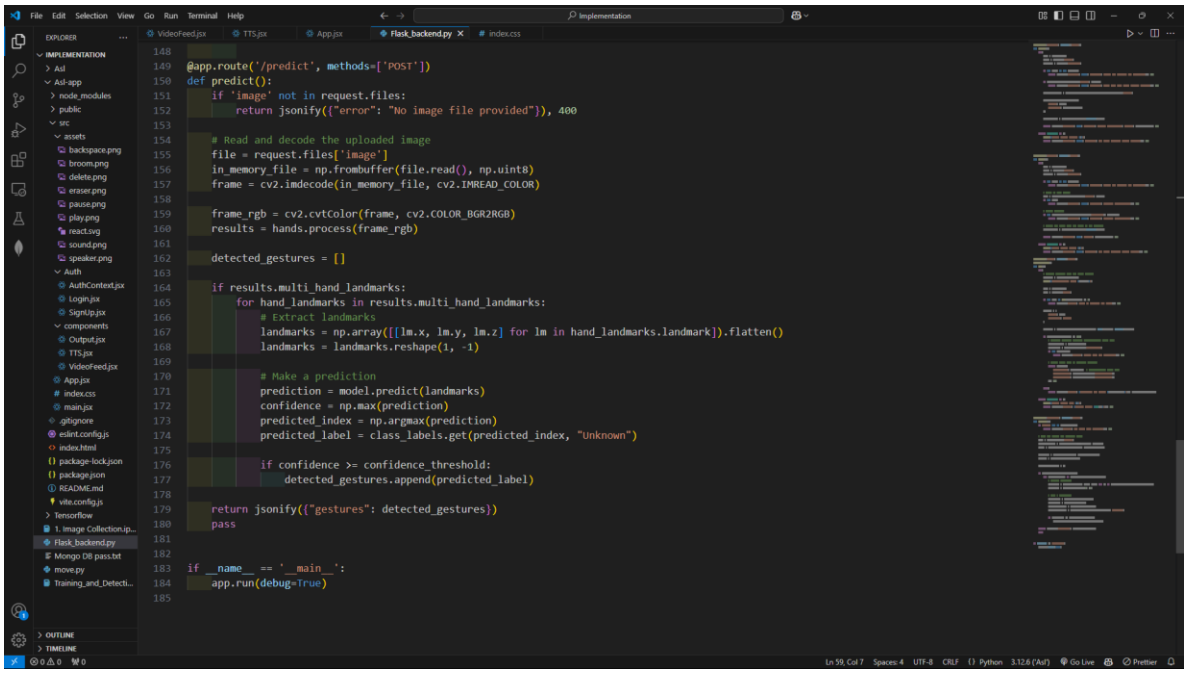
```python
class_labels = {i: chr(65 + i) for i in range(26)}
confidence_threshold = 0.8

mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.5)


@app.route('/signup', methods=['POST'])
def signup():
    try:
        data = request.json
        email = data.get('email')
        password = data.get('password')

        if not email or not password:
            return jsonify({"success": False, "message": "Email and password are required."}), 400

        if users_collection.find_one({"email": email}):
            return jsonify({"success": False, "message": "User with this email already exists."}), 400

        hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')

        users_collection.insert_one({
            "email": email,
            "password": hashed_password,
        })

        access_token = create_access_token(identity=email)

        return jsonify({"success": True, "message": "Signup successful.", "token": access_token}), 201

    except Exception as e:
        print(f"Error during signup: {e}")
        return jsonify({"success": False, "message": "An error occurred during signup."}), 500


@app.route('/login', methods=['POST'])
def login():
    try:
        data = request.json
```



```python
@app.route('/login', methods=['POST'])
def login():
    try:
        data = request.json
        email = data.get('email')
        password = data.get('password')

        if not email or not password:
            return jsonify({"success": False, "message": "Email and password are required."}), 400

        # Check if user exists
        user = users_collection.find_one({"email": email})
        if not user:
            return jsonify({"success": False, "message": "Invalid credentials."}), 401

        # Verify password
        if not bcrypt.check_password_hash(user['password'], password):
            return jsonify({"success": False, "message": "Invalid credentials."}), 401

        # Create JWT token with the email as the identity
        access_token = create_access_token(identity=email)

        return jsonify({"success": True, "token": access_token}), 200

    except Exception as e:
        print(f"Error during login: {e}")
        return jsonify({"success": False, "message": "An error occurred during login."}), 500

@app.route('/tts', methods=['POST'])
@jwt_required()
def tts():
    try:
        # Extract identity from JWT token (email)
        identity = get_jwt_identity()
        print(f"Authenticated user: {identity}")

        data = request.json
        text = data.get('text')

        if not text or len(text.strip()) == 0:
            return jsonify({"error": "Text is required for TTS."}), 400
```

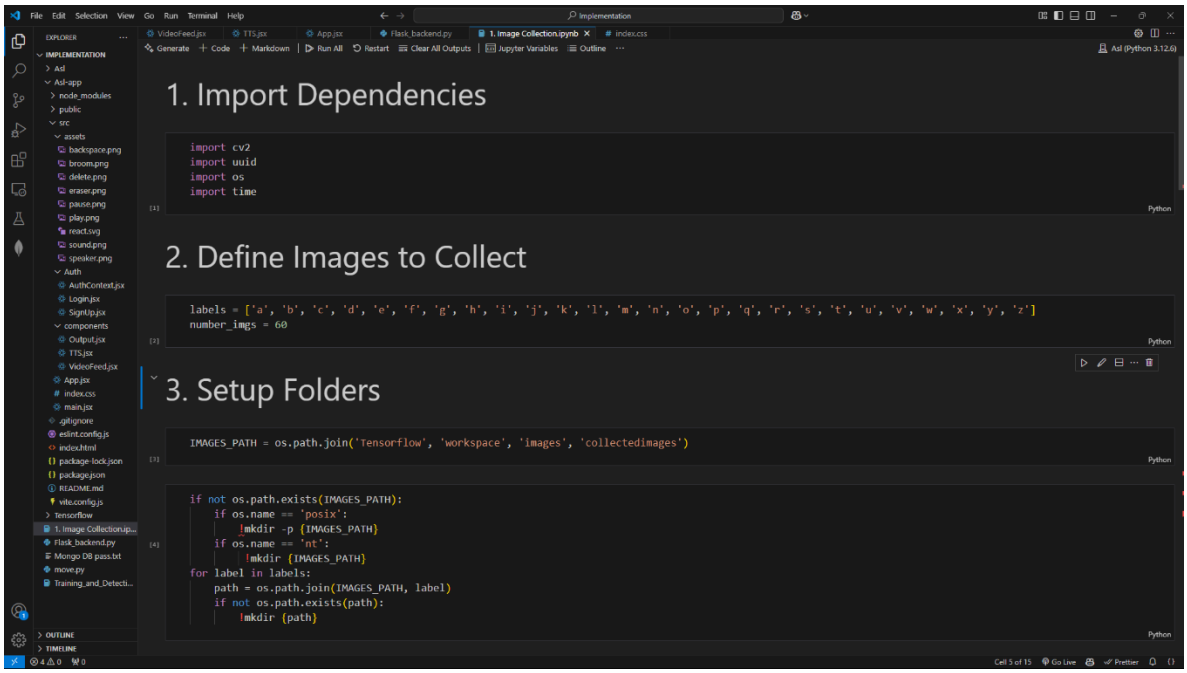


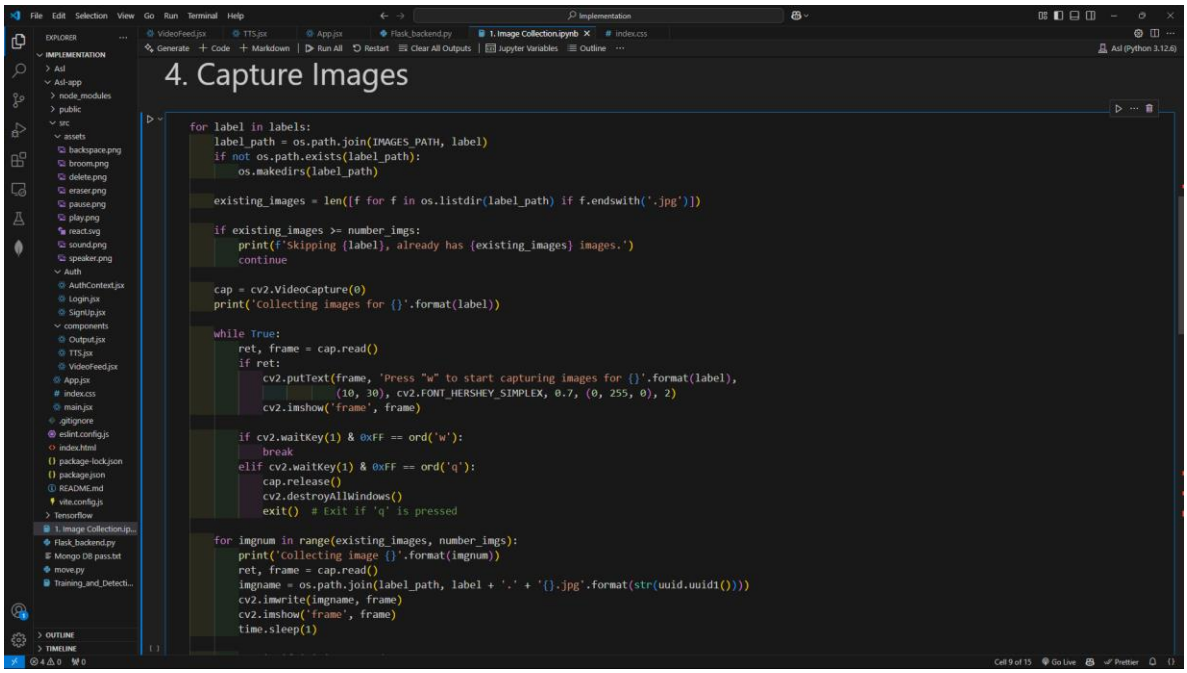
```python
@app.route('/tts', methods=['POST'])
@jwt_required()
def tts():
    try:
        # Extract identity from JWT token (email)
        identity = get_jwt_identity()
        print(f"Authenticated user: {identity}")

        data = request.json
        text = data.get('text')

        if not text or len(text.strip()) == 0:
            return jsonify({"error": "Text is required for TTS."}), 400

        payload = {
            "input": text,
            "voice_id": "henry"
        }

        response = requests.post(SPEECHIFY_API_URL, json=payload, headers=headers)

        if response.status_code == 200:
            # Assuming response contains base64-encoded audio data
            response_data = response.json()
            audio_data = response_data.get("audio_data")
            if not audio_data:
                return jsonify({"error": "Audio data not received."}), 500

            # Return audio data to frontend (base64)
            return jsonify({
                "audio_data": audio_data,  # Base64-encoded audio data
                "audio_format": response_data.get("audio_format"),
                "speech_marks": response_data.get("speech_marks"),
            }), 200

        else:
            return jsonify({"error": f"Speechify API error: {response.text}"}), response.status_code

    except Exception as e:
        print(f"Error during TTS: {e}")
        return jsonify({"error": f"Server error: {str(e)}"}), 500
```

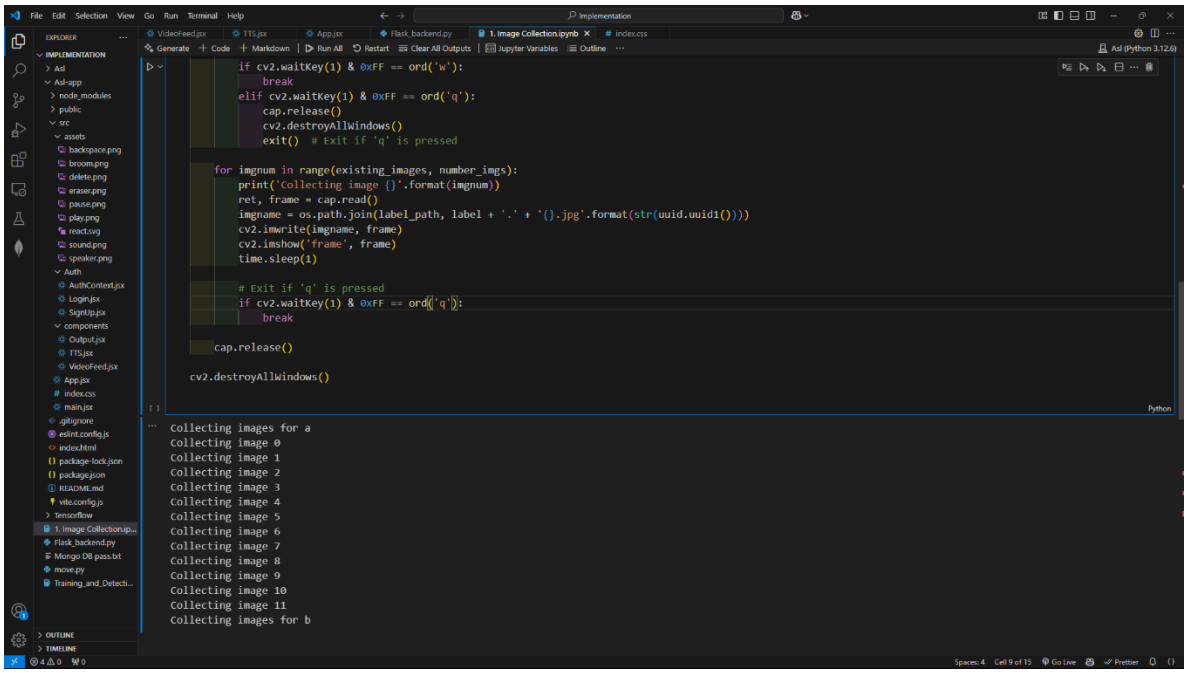


```python
@app.route('/predict', methods=['POST'])
def predict():
    if 'image' not in request.files:
        return jsonify({"error": "No image file provided"}), 400

    # Read and decode the uploaded image
    file = request.files['image']
    in_memory_file = np.frombuffer(file.read(), np.uint8)
    frame = cv2.imdecode(in_memory_file, cv2.IMREAD_COLOR)

    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(frame_rgb)

    detected_gestures = []

    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            # Extract landmarks
            landmarks = np.array([[lm.x, lm.y, lm.z] for lm in hand_landmarks.landmark]).flatten()
            landmarks = landmarks.reshape(1, -1)

            # Make a prediction
            prediction = model.predict(landmarks)
            confidence = np.max(prediction)
            predicted_index = np.argmax(prediction)
            predicted_label = class_labels.get(predicted_index, "Unknown")

            if confidence >= confidence_threshold:
                detected_gestures.append(predicted_label)

    return jsonify({"gestures": detected_gestures})
    pass


if __name__ == '__main__':
    app.run(debug=True)
```

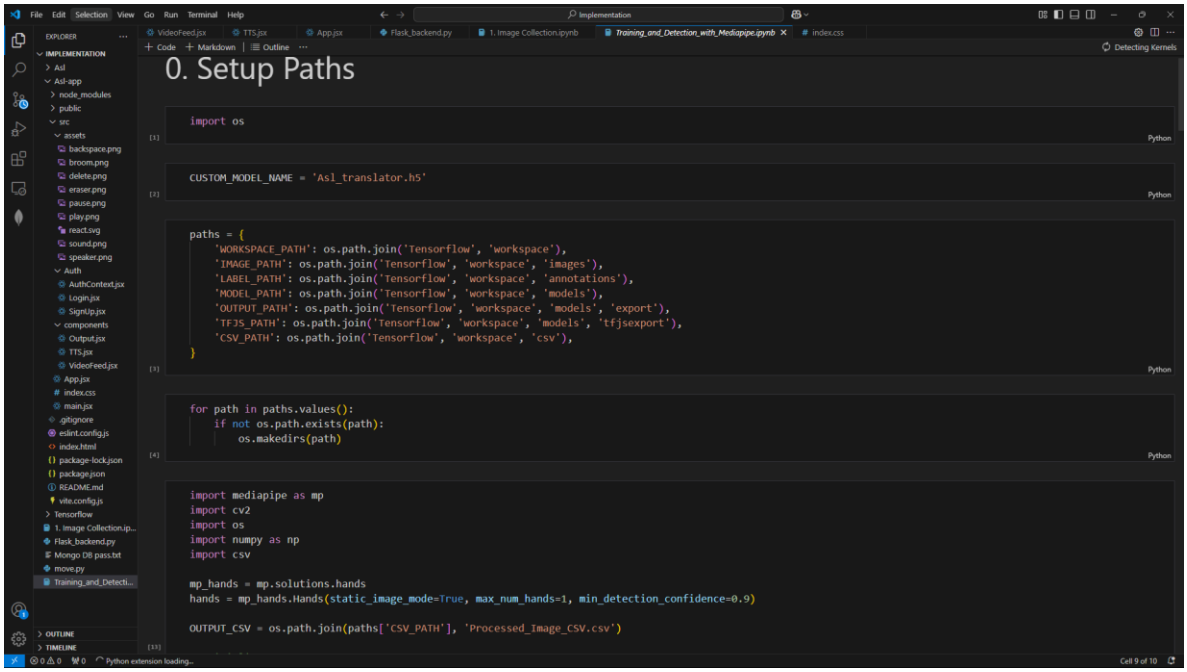
Image Collection:


1. Import Dependencies
import cv2
import uuid
import os
import time
2. Define Images to Collect
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
number_imgs = 60
3. Setup Folders
IMAGES_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'collectedimages')
if not os.path.exists(IMAGES_PATH):
if os.name == 'posix':
!mkdir -p {IMAGES_PATH}
if os.name == 'nt':
!mkdir {IMAGES_PATH}
for label in labels:
path = os.path.join(IMAGES_PATH, label)
if not os.path.exists(path):
!mkdir {path}


4. Capture Images
for label in labels:
label_path = os.path.join(IMAGES_PATH, label)
if not os.path.exists(label_path):
os.makedirs(label_path)
existing_images = len([f for f in os.listdir(label_path) if f.endswith('.jpg')])
if existing_images >= number_imgs:
print(f'Skipping {label}, already has {existing_images} images.')
continue
cap = cv2.VideoCapture(0)
print('Collecting images for {}'.format(label))
while True:
ret, frame = cap.read()
if ret:
cv2.putText(frame, 'Press "w" to start capturing images for {}'.format(label),
(10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
cv2.imshow('frame', frame)
if cv2.waitKey(1) & 0xFF == ord('w'):
break
elif cv2.waitKey(1) & 0xFF == ord('q'):
cap.release()
cv2.destroyAllWindows()
exit() # Exit if 'q' is pressed
for imgnum in range(existing_images, number_imgs):
print('Collecting image {}'.format(imgnum))
ret, frame = cap.read()
imgname = os.path.join(label_path, label + '.' + '{}.jpg'.format(str(uuid.uuid1())))
cv2.imwrite(imgname, frame)
cv2.imshow('frame', frame)
time.sleep(1)


if cv2.waitKey(1) & 0xFF == ord('w'):
break
elif cv2.waitKey(1) & 0xFF == ord('q'):
cap.release()
cv2.destroyAllWindows()
exit() # Exit if 'q' is pressed
for imgnum in range(existing_images, number_imgs):
print('Collecting image {}'.format(imgnum))
ret, frame = cap.read()
imgname = os.path.join(label_path, label + '.' + '{}.jpg'.format(str(uuid.uuid1())))
cv2.imwrite(imgname, frame)
cv2.imshow('frame', frame)
time.sleep(1)
# Exit if 'q' is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
break
cap.release()
cv2.destroyAllWindows()
Collecting images for a
Collecting image 0
Collecting image 1
Collecting image 2
Collecting image 3
Collecting image 4
Collecting image 5
Collecting image 6
Collecting image 7
Collecting image 8
Collecting image 9
Collecting image 10
Collecting image 11
Collecting images for b

Model Training and testing:



0. Setup Paths

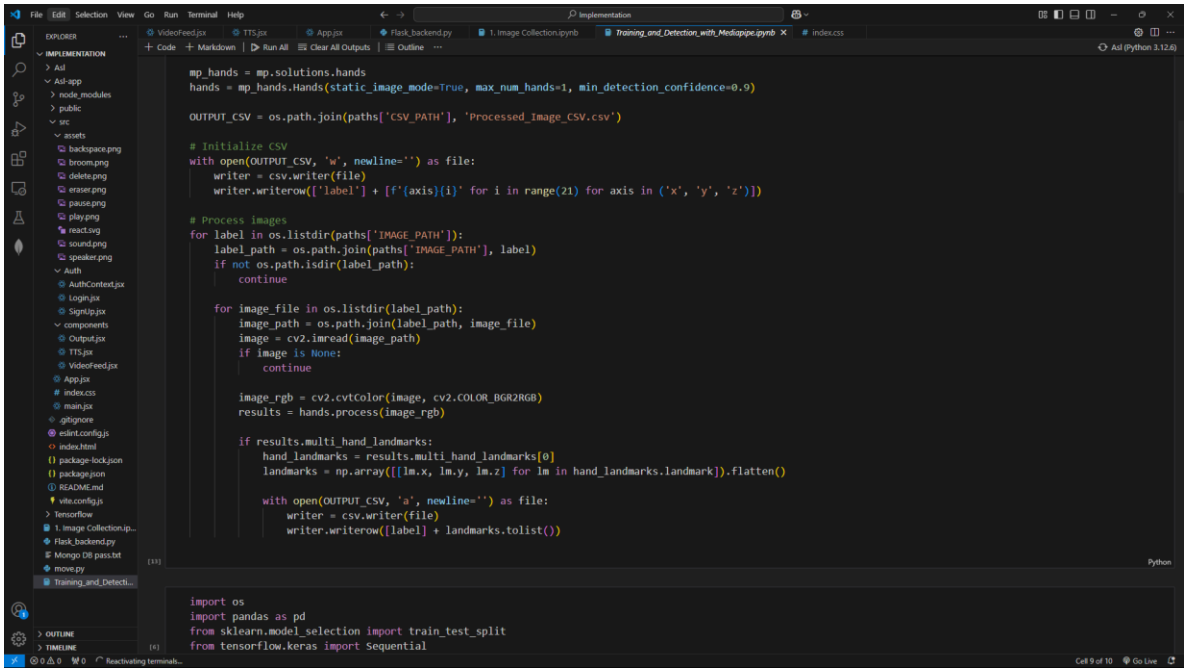
```
import os
```

```
CUSTOM_MODEL_NAME = 'Asl_translator.h5'
```

```
paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'Workspace'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace', 'images'),
    'LABEL_PATH': os.path.join('Tensorflow', 'workspace', 'annotations'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace', 'models'),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace', 'models', 'export'),
    'TFJS_PATH': os.path.join('Tensorflow', 'workspace', 'models', 'tfjsexport'),
    'CSV_PATH': os.path.join('Tensorflow', 'workspace', 'csv'),
}
```

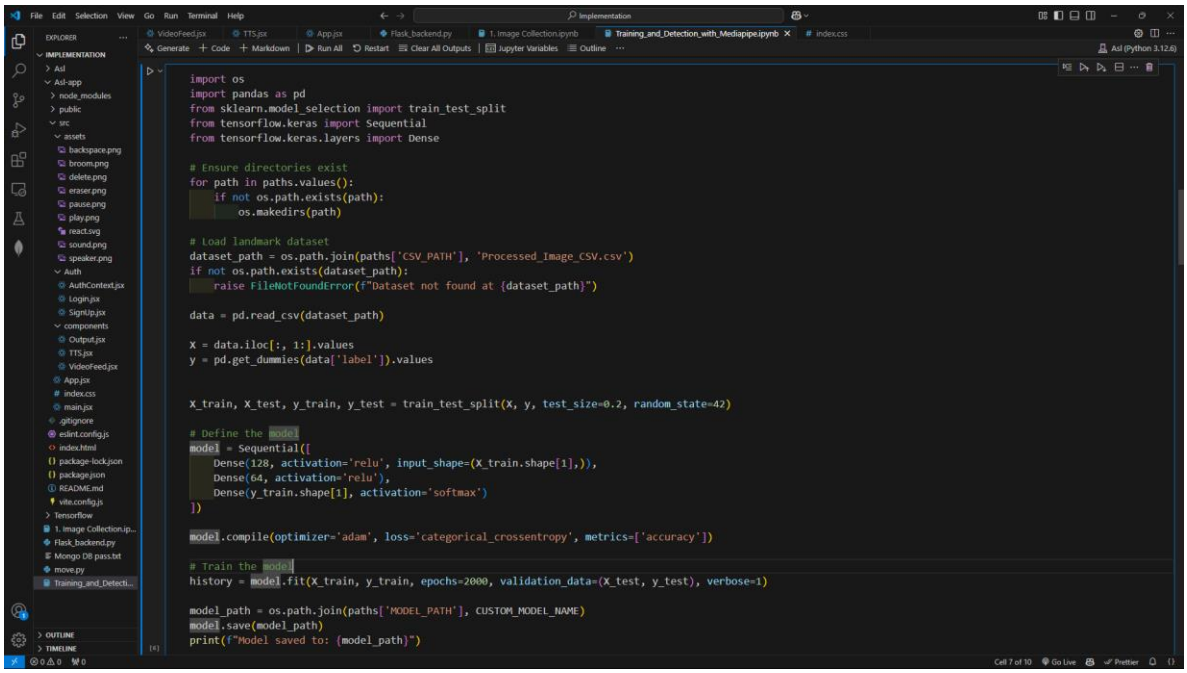
```
for path in paths.values():
    if not os.path.exists(path):
        os.makedirs(path)
```

```
import mediapipe as mp
import cv2
import os
import numpy as np
import csv

mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=True, max_num_hands=1, min_detection_confidence=0.9)

OUTPUT_CSV = os.path.join(paths['CSV_PATH'], 'Processed_Image_CSV.csv')
```

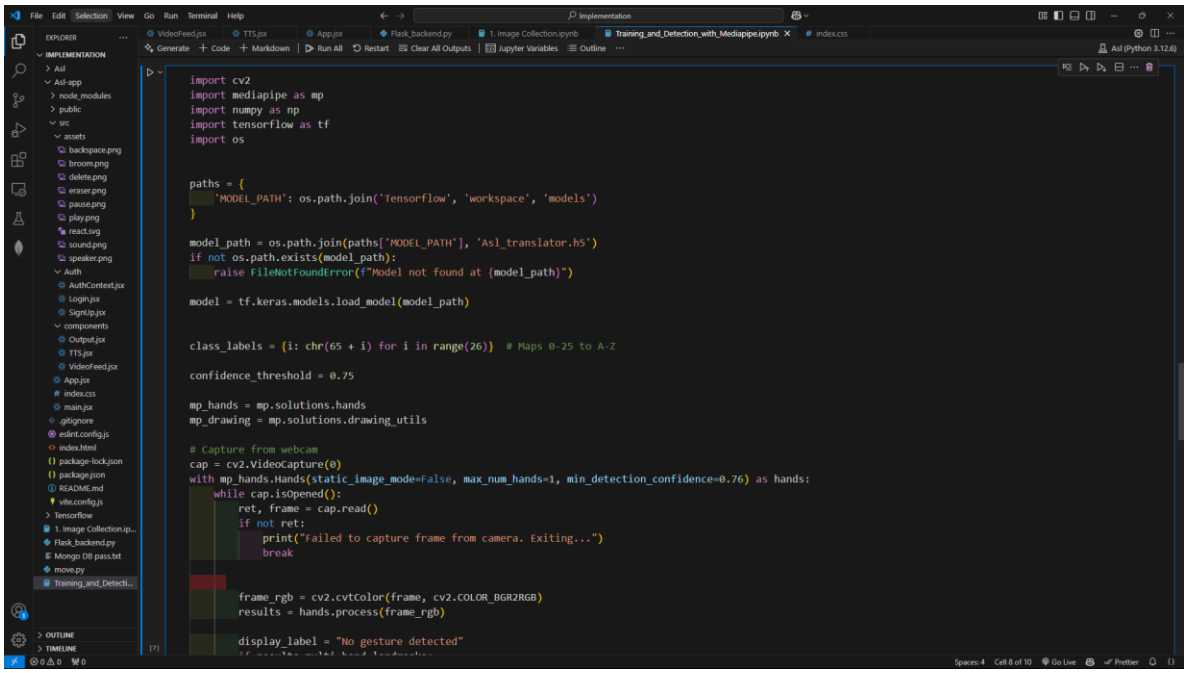


```
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=True, max_num_hands=1, min_detection_confidence=0.9)

OUTPUT_CSV = os.path.join(paths['CSV_PATH'], 'Processed_Image_CSV.csv')

# Initialize CSV
with open(OUTPUT_CSV, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['label'] + [f'{axis}{i}' for i in range(21) for axis in ('x', 'y', 'z')])

# Process images
for label in os.listdir(paths['IMAGE_PATH']):
    label_path = os.path.join(paths['IMAGE_PATH'], label)
    if not os.path.isdir(label_path):
        continue

    for image_file in os.listdir(label_path):
        image_path = os.path.join(label_path, image_file)
        image = cv2.imread(image_path)
        if image is None:
            continue

        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        results = hands.process(image_rgb)

        if results.multi_hand_landmarks:
            hand_landmarks = results.multi_hand_landmarks[0]
            landmarks = np.array([[lm.x, lm.y, lm.z] for lm in hand_landmarks.landmark]).flatten()

            with open(OUTPUT_CSV, 'a', newline='') as file:
                writer = csv.writer(file)
                writer.writerow([label] + landmarks.tolist())
```

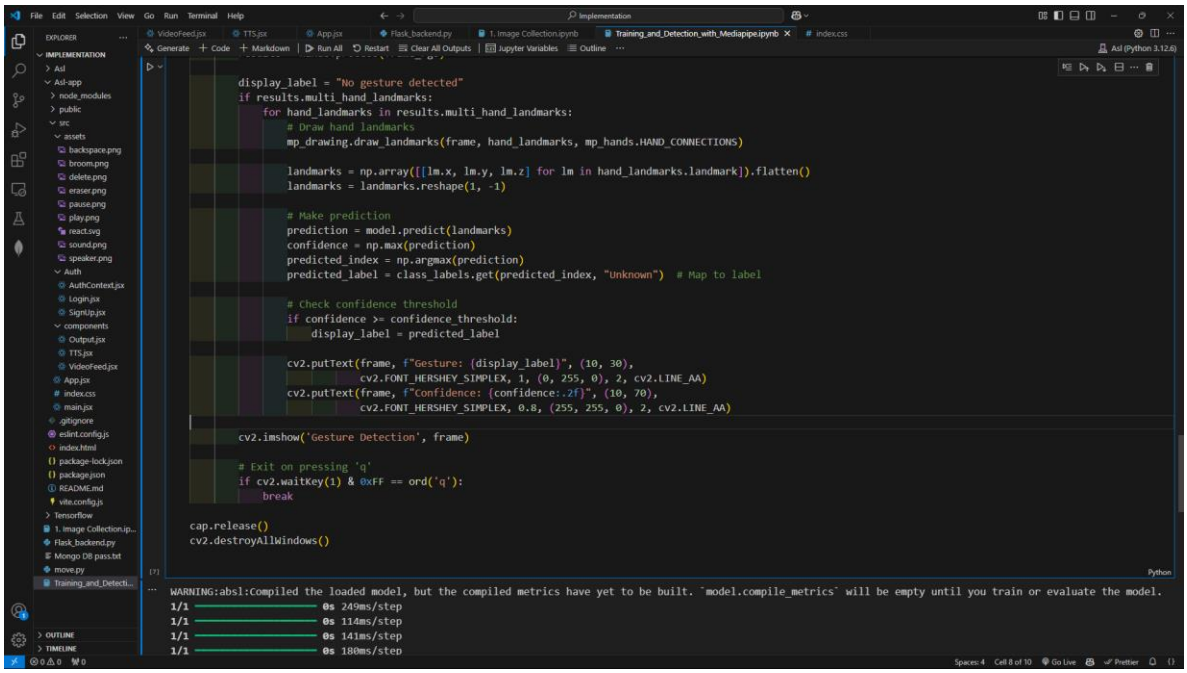
```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras import Sequential
```



```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# Ensure directories exist
for path in paths.values():
    if not os.path.exists(path):
        os.makedirs(path)

# Load landmark dataset
dataset_path = os.path.join(paths['CSV_PATH'], 'Processed_Image_CSV.csv')
if not os.path.exists(dataset_path):
    raise FileNotFoundError(f"Dataset not found at {dataset_path}")

data = pd.read_csv(dataset_path)

X = data.iloc[:, 1:].values
y = pd.get_dummies(data['label']).values


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(y_train.shape[1], activation='softmax')
])

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

# Train the model
history = model.fit(X_train, y_train, epochs=2000, validation_data=(X_test, y_test), verbose=1)

model_path = os.path.join(paths['MODEL_PATH'], CUSTOM_MODEL_NAME)
model.save(model_path)
print(f"Model saved to: {model_path}")
```



```
import cv2
import mediapipe as mp
import numpy as np
import tensorflow as tf
import os


paths = {
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace', 'models')
}

model_path = os.path.join(paths['MODEL_PATH'], 'Asl_translator.h5')
if not os.path.exists(model_path):
    raise FileNotFoundError(f"Model not found at {model_path}")

model = tf.keras.models.load_model(model_path)


class_labels = {i: chr(65 + i) for i in range(26)}  # Maps 0-25 to A-Z

confidence_threshold = 0.75

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils

# Capture from webcam
cap = cv2.VideoCapture(0)
with mp_hands.Hands(static_image_mode=False, max_num_hands=1, min_detection_confidence=0.76) as hands:
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            print("Failed to capture frame from camera. Exiting...")
            break


        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = hands.process(frame_rgb)

        display_label = "No gesture detected"
```


display_label = "No gesture detected"
if results.multi_hand_landmarks:
for hand_landmarks in results.multi_hand_landmarks:
# Draw hand landmarks
mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
landmarks = np.array([[lm.x, lm.y, lm.z] for lm in hand_landmarks.landmark]).flatten()
landmarks = landmarks.reshape(1, -1)
# Make prediction
prediction = model.predict(landmarks)
confidence = np.max(prediction)
predicted_index = np.argmax(prediction)
predicted_label = class_labels.get(predicted_index, "Unknown") # Map to label
# Check confidence threshold
if confidence >= confidence_threshold:
display_label = predicted_label
cv2.putText(frame, f"Gesture: {display_label}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
cv2.putText(frame, f"Confidence: {confidence:.2f}", (10, 70),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 0), 2, cv2.LINE_AA)
cv2.imshow('Gesture Detection', frame)
# Exit on pressing 'q'
if cv2.waitKey(1) & 0xFF == ord('q'):
break
cap.release()
cv2.destroyAllWindows()
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 0s 249ms/step
1/1 0s 114ms/step
1/1 0s 141ms/step
1/1 0s 180ms/step