

**EMONYAI : CONTEXTUAL CONVERSATION GUIDANCE LEVERAGING
MICROEXPRESSION AND BODY LANGUAGE INTERPRETATION**

BY
GOH CHUN SHING

A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2024

REPORT STATUS DECLARATION FORM

Title: EmonyAI : Contextual Conversation Guidance leveraging _____
Microexpression and Body Language Interpretation _____

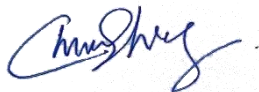
Academic Session: _JUNE 2024_

I _GOH CHUN SHING_
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,



(Author's signature)



(Supervisor's signature)

Address:

10, Lorong Sungai Bakap Permai 10,
Taman Sungai Bakap Permai _____
14200 Sungai Bakap _____
Pulau Pinang _____

Aun Yichiet
Supervisor's name

Date: 13 September 2024

Date: 13/09/2024

Universiti Tunku Abdul Rahman			
Form Title : Sample of Submission Sheet for FYP/Dissertation/Thesis			
Form Number: FM-IAD-004	Rev No.: 0	Effective Date: 21 JUNE 2011	Page No.: 1 of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN

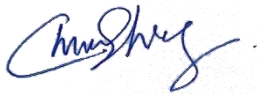
Date: 6 SEPTEMBER 2024

SUBMISSION OF FINAL YEAR PROJECT

It is hereby certified that *Goh Chun Shing* (ID No: 2004745) has completed this final year project entitled “**EmonyAI : Contextual Conversation Guidance leveraging Microexpression and Body Language Interpretation**” under the supervision of Dr. Aun Yi Chiet (Supervisor) from the Department of _____, Faculty of Information and Communication Technology, and _____ (Co-Supervisor)* from the Department of _____, Faculty/Institute* of _____.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

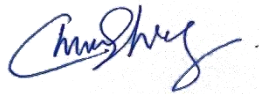


 (Student Name)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled “**EMONYAI : CONTEXTUAL CONVERSATION GUIDANCE LEVERAGING MICROEXPRESSION AND BODY LANGUAGE INTERPRETATION**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.



Signature : _____

Name : Goh Chun Shing

Date : 06 September 2024

ACKNOWLEDGEMENTS

I am deeply grateful to Dr. Aun Yi Chiet for providing me with the invaluable opportunity to work on EmonyAI, a state-of-the-art contextual chatbot. This project marks the beginning of my journey into the exciting world of AI technology, and I am sincerely thankful for Dr. Aun's guidance, expertise, and unwavering support throughout the entire process.

I would also like to express my heartfelt appreciation to my friend, Amanda Lean Rina, a master's student in psychology, for giving me invaluable advice on emotion classification. Her expertise in emotional patterns greatly enriched the conceptual framework of this project, and her support has been instrumental in the development of the emotion detection module.

I would also like to thank my university buddies, Goh Ken How, Soh Wen Kai, Ho Joe Ee, and Yeap Shevon, for serving as test subjects in evaluating the system's capabilities. Their unwavering support, particularly during challenging moments, has been a constant source of motivation, and I am deeply grateful for their presence in my life.

Besides, I would also like to thank all my surrounding friends, whose encouragement and companionship have made this journey more enjoyable. Their belief in my abilities and constant check-ins helped me stay focused and motivated throughout the project.

Most importantly, I must extend my deepest thanks to my parents and family. Their unwavering love, support, and encouragement have been pivotal in this journey, lifting me during moments of doubt and celebrating with me in times of success.

Together, their collective contributions have not only shaped the outcome of this project but have also profoundly influenced my personal and professional growth. I am forever grateful to each of them for their roles in bringing this vision to life.

ABSTRACT

Social anxiety and introversion can create barriers to effective communication, limiting both personal and professional interactions. In this project, we develop EmonyAI, a system designed to understand situational contexts such as user facial emotions and conversation content, to leverage that information and provide suggestions for more appropriate response in real time. EmonyAI is an advanced AI platform designed to enhance human interaction by integrating multiple functions, including facial recognition, speech-to-text processing, emotion detection, and text summarization, all aimed at improving communication skills. Rather than relying solely on traditional verbal cues, EmonyAI incorporates CNN for processing facial recognition and emotion detection, enabling the system to capture the subtleties of nonverbal communication. The contextual functionality is built based on the MetaGPT framework, which stores each individual as character and continuously evaluates character profiles, allowing for dynamic adaptation and improvement over time. With each interaction, the profiles grow richer, enabling the system to better assess the characteristics and preferences of the user, leading to more accurate predictions and suggestions. By leveraging this comprehensive approach, EmonyAI generates contextual conversation suggestions tailored to each user's emotional state and character, promising to enhance social experiences and foster more effective communication across various digital platforms.

TABLE OF CONTENTS

REPORT STATUS DECLARATION FORM.....	II
SUBMISSION OF FINAL YEAR PROJECT	III
DECLARATION OF ORIGINALITY	IV
ACKNOWLEDGEMENTS	V
ABSTRACT.....	VI
TABLE OF CONTENTS	VII
LIST OF FIGURES	XI
LIST OF TABLES.....	XIV
LIST OF SYMBOLS	XV
LIST OF ABBREVIATIONS	XVI
CHAPTER 1 INTRODUCTION.....	1
1.1 Problem Statement and Motivation	1
1.2 Research Objectives.....	2
1.3 Project Scope and Direction	3
1.4 Contributions	4
1.5 Report Organization.....	5
CHAPTER 2 LITERATURE REVIEW.....	7
2.1 Previous work on Micro Expression.....	7
2.2 Previous work on Body Language.....	13
2.3 Limitations of Previous Studies.....	19
2.4 Large Language Models	21
2.4.1 Comparison for different Large Language Models	25
2.4.2 Selection of Large Language Model.....	26

2.5 Multi-Agent Framework.....	27
CHAPTER 3 SYSTEM MODEL	29
3.1 System Methodology	29
3.2 System Design	29
3.2.1 Hardware.....	29
3.2.2 Software.....	30
3.2.3 System Architecture Diagram.....	32
3.3 Important modules	35
3.3.1 WebRTC - Video and Audio Processing.....	35
3.3.2 Emotion Detection	37
3.3.3 Google Speech-To-Text.....	39
3.3.4 Combining Emotion Detection and Speech-to-text into Context Aggregator	40
CHAPTER 4 SYSTEM DESIGN	43
4.1 System Block Diagram	43
4.2 System Components Specifications.....	44
4.3 Circuits and Components Design	45
4.4 Emotion Detection Module.....	46
4.4.1 Building of the Emotion Detection Model in Jupyter Notebook..	46
4.4.2 Converting Emotion Detection Notebook into Python script.....	63
4.4.3 Publish the Python Script through FastAPI.....	64
4.4.4 Model Evaluation.....	66
4.5 Webpage design.....	70
4.5.1 Technology Stacks	70
4.5.2 Frontend Components.....	71
4.5.3 Backend Components	78
4.5.4 Core Features	81
4.5.5 Deploying on Google Cloud Run	81
4.6 Multi-agent framework.....	85
4.7 Operation Logic	90
4.7.1 Before the Video Call Starts	90
4.7.2 During the Video Call.....	90
4.7.3 Pauses in Conversation	91
4.7.4 After the Video Call Ends.....	92
4.8 Special Situations.....	93

CHAPTER 5 EXPERIMENT/SIMULATION	95
5.1 Setup	95
5.1.1 Hardware Setup	95
5.1.2 Software Setup.....	95
5.2 System Operation.....	97
5.2.1 Interaction Involving Positive Feedback	97
5.2.2 Interaction Involving Negative Feedback.....	98
5.3 Multi-Agent Interaction	100
5.4 Application for Special Situations	101
5.5 Implementation Issues and Challenges.....	103
5.6 Concluding Remark	104
 CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION	 105
6.1 System Testing.....	105
6.1.1 Testing on Emotion Detection	105
6.1.2 Evaluation on Speech-To-Text	109
6.2 Evaluation of Conversation Suggestion Effectiveness	121
6.2.1 Case 1 (Test Subject 1: Goh Ken How).....	121
6.2.2 Case 2 (Test Subject 2: Amanda Lean Rina).....	123
6.2.3 Case 3 (Test Subject 3: Chu Joe Yen)	125
6.3 Performance Metrics.....	127
6.4 Project Challenges	129
6.5 Objectives Evaluation.....	131
6.6 Concluding Remark	134
 CHAPTER 7 CONCLUSION AND RECOMMENDATION	 136
7.1 Conclusion	136
7.2 Recommendation	136
 REFERENCES	 139
 FINAL YEAR PROJECT WEEKLY REPORT	 143
 POSTER	 149

PLAGIARISM CHECK RESULT.....150

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1.1 :	Sparse Representation Classification	7
Figure 2.1.2	Modified Local Directional Patterns	8
Figure 2.1.3	Generalized Discriminant Analysis	8
Figure 2.1.4	3D face recognition approach	9
Figure 2.1.5	Support Vector Machines (SVMs)	9
Figure 2.1.6	Convolutional Neural Networks (CNNs)	10
Figure 2.1.7	Long Short-Term Memory (LSTM)	10
Figure 2.1.8	LBP-TOP feature extraction	11
Figure 2.1.9	Conditional Random Fields explained	14
Figure 2.1.10	Parallel Pyramid Network (PPNet)	15
Figure 2.1.11	Large Kernel Convolutional Layers	15
Figure 2.1.12	Spatio-Temporal Interest Point (STIP)	16
Figure 2.1.13	Deep Convolutional Neural Network (DCNN)	17
Figure 2.4.1 :	User Interface for GPT	22
Figure 2.4.2	User Interface for GPT4o	23
Figure 2.4.3	User Interface for Google Gemini	24
Figure 3.1	Process of Prototyping RAD methodology	29
Figure 3.2.1	System Architecture Diagram	32
Figure 3.3.1	WebRTC Architecture	36
Figure 3.3.3	Architecture for Emotion Detection Module	39
Figure 3.3.4	System architecture for Google Speech-to-Text API	40
Figure 3.3.5	System Architecture for Context Aggregator	42
Figure 4.1.1	System Block Diagram	43
Figure 4.4.1	Code snippet for importing libraries	47
Figure 4.4.2	Code snippet for setting hyperparameter and directories	48
Figure 4.4.3	Code snippet for data loading and Pre-processing	50
Figure 4.4.4	Code snippet for displaying images with different emotions	53
Figure 4.4.5	Checking data distribution among different emotions	55
Figure 4.4.6	Code Snippet for DenseNet169 Transfer Learning	57
Figure 4.4.7	Code Snippet for Summary of Model	59

Figure 4.4.8	Code snippet for training model with freeze layers	60
Figure 4.4.9	Code snippet for Fine Tuning the model	61
Figure 4.4.10	Loss vs Number of Epochs	66
Figure 4.4.11	Confusion Matrix	67
Figure 4.1.12	Classification Report	68
Figure 4.1.13	ROC-AUC Curve	69
Figure 4.5.1	User interface of EmonyAI website	70
Figure 4.5.2	Code snippet for index.html	71
Figure 4.5.3	Code snippet for main.jsx	72
Figure 4.5.4	Code snippet for App.jsx	72
Figure 4.5.5	Code snippet for MyCam.jsx	73
Figure 4.5.6	Code snippet for FriendCam.jsx	73
Figure 4.5.7	Code snippet for CallSetting.jsx	74
Figure 4.5.8	Code snippet for chatbot.jsx	75
Figure 4.5.9	Code snippet for style.css	76
Figure 4.5.10	Code snippet for chatbot.css	77
Figure 4.5.11	Code snippet for server.js	79
Figure 4.5.13	emony_backend_service_key.json	80
Figure 4.5.14	emonyai-6103e-8e4d94f3f5e2.json	80
Figure 4.5.15	Code snippet for Dockerfile	82
Figure 4.6.1	Template for agent profile	86
Figure 4.6.2	Agent Profile from CaiPin.py	86
Figure 4.6.3	Agent Profile from JoeYen.py	87
Figure 4.6.4	Agent Profile from KenHow.py	87
Figure 4.6.5	Agent Profile from KhaiShen.py	88
Figure 4.6.6	Agent Profile from Rina.py	88
Figure 4.6.7	Agent Profile from Wendy.py	89
Figure 5.2.1	Screenshot of interaction between me and Amanda Lean (Rina.py)	97
Figure 5.2.2	Screenshot of interaction between me and Chu Joe Yen (JoeYen.py)	98
Figure 5.2.3	Screenshot of interaction between me and Ng Cai Pin (CaiPin.py)	99

Figure 5.2.4	Screenshot of EmonyAI Multi-Agent prompting	100
Figure 5.2.5	Screenshot of interaction between me and Goh Ken How (KenHow.py)	101
Figure 5.2.6	Screenshot of interaction between me and Amanda Lean (Rina.py)	102
Figure 6.1.1	Code snippet from agent profile : JoeYen.py	111
Figure 6.1.2	Code snippet from agent profile : Wendy.py	115
Figure 6.1.3	Code snippet from agent profile : Rina.py	118
Figure 6.1.4	Google Form feedback from Test Subject 1	121
Figure 6.1.5	Google Form feedback from Test Subject 1	122
Figure 6.1.6	Google Form feedback from Test Subject 2	123
Figure 6.1.7	Google Form feedback from Test Subject 2	124
Figure 6.1.8	Google Form feedback from Test Subject 3	125
Figure 6.1.9	Google Form feedback from Test Subject 3	126

LIST OF TABLES

Table Number	Title	Page
Table 2.1.1	Comparison of existing facial emotion detection	11
Table 2.1.2	Comparison of existing body language detection	17
Table 2.4.1	Comparison of Large Language Models	26
Table 3.1.1	Specifications of laptop	30
Table 3.1.2	Software used for the emotion detection module	30
Table 3.1.3	Software used for the user interface	30
Table 3.1.4	Software used for the profile management	31
Table 3.1.5	Software used for the website deployment	31
Table 3.1.6	Other software used and their functionality	31
Table 6.1.1	Testing Result for Emotion Detection	107
Table 6.1.2	Text Recorded and Evaluation for JoeYen.py	114
Table 6.1.3	Text Summarized and Evaluation for JoeYen.py	114
Table 6.1.4	Text Recorded and Evaluation for Wendy.py	117
Table 6.1.5	Text Summarized and Evaluation for Wendy.py	117
Table 6.1.6	Text Recorded and Evaluation for Rina.py	120
Table 6.1.7	Text Summarized and Evaluation for Rina.py	120

LIST OF SYMBOLS

LIST OF ABBREVIATIONS

<i>SRC</i>	Sparse Representation Classification
<i>MLDP</i>	Modified Local Directional Patterns
<i>SVM</i>	Support Vector Machines
<i>CNN</i>	Convolutional Neural Networks
<i>LSTM</i>	Long Short-Term Memory
<i>DTSA</i>	Discriminant Tensor Subspace Analysis
<i>ELM</i>	Extreme Learning Machine
<i>CBP-TOP</i>	centralized binary pattern on three orthogonal planes
<i>EVM</i>	Eulerian Motion Magnification
<i>SRF</i>	Conditional Random Fields
<i>CLD</i>	Convolutional Local Detectors
<i>PPNET</i>	Parallel Pyramid Network
<i>FDH</i>	Fusion Deconv Head
<i>RNN</i>	Recurrent Neural Networks
<i>DCNN</i>	Deep Convolutional Neural Networks
<i>HCI</i>	Human-computer interaction

CHAPTER 1 Introduction

In this chapter, we present the background and motivation of our research, our contributions to the field of Micro expression and conversation generator, and the outline of the thesis.

1.1 Problem Statement and Motivation

In today's rapidly evolving artificial intelligence capabilities, human-computer interaction is reaching new frontiers. The fusion of artificial intelligence model and conversation generation technologies has opened doors to innovative applications, ranging from chatbots to virtual assistants. However, one critical aspect remains a challenge: the nuanced understanding of nonverbal clues such as human emotions and body language during conversations. This project sets out to address this vital aspect by developing a robust nonverbal clues detection model for conversation generators.

Human communication is a complex interplay of words, tone, and facial expressions. Micro expressions, fleeting facial expressions that reveal genuine emotions, are often overlooked in digital conversations. Yet, they hold the key to truly empathetic and emotionally intelligent artificial agents. Our project aims to bridge this gap by equipping conversation generators with the ability to not only understand the words spoken but also perceive the subtle emotional cues conveyed through micro expressions and body language.

This project's significance lies in its potential to revolutionize the way we interact with AI-driven conversation generators. By enhancing their capacity to detect and respond to users' emotions, we can create more authentic, engaging, and context-aware conversations. This, in turn, can have profound implications for fields in human interaction such as customer service, mental health support, and educational platforms.

Despite the importance of micro expressions and body language in human interaction, existing conversation generators often overlook these vital nonverbal signals. This omission results in conversations that lack depth and authenticity, hindering their ability to establish meaningful connections and rapport with users. The absence of micro expression and body language awareness in these systems is a significant limitation in conversation generation that my research aims to address.

My research seeks to bridge the gap between micro expression detection, body language detection and conversation generation. By integrating a reliable micro expression detection model combine with body language recognition into conversation generators, we can enhance their ability to respond empathetically and adaptively to users' emotional states. This approach represents a superior solution to alternative methods, as it leverages latest large language models to offer users a more authentic and emotionally resonant conversational social experience.

1.2 Research Objectives

The objective of the EmonyAI project is to develop a context-aware and humanistic AI assistant to generate contextually relevant and appropriate responses in real time. This system aims to seamlessly integrate context aggregation with real-time detection of micro-expressions and body languages.

To better manage diverse user profiles for accurate response, EmonyAI also employ the Multi-Agent Framework, each tailored based on different user interactions, ensuring dynamic and accurate response generation.

EmonyAI aim to deploy a context aggregator to analyze verbal and non-verbal cues for insightful interactions. To support this capability, EmonyAI integrates cutting-edge micro-expression and body language models into a unified system that analyzes these cues simultaneously. This integration enables the real-time detection of subtle non-verbal signals, enhancing the system's ability to understand and respond to unspoken emotions and intentions.

Additionally, the project also focuses on connect with OpenAI's latest large language model to generate empathetic responses using real-time emotional and contextual data. This integration promises to revolutionize how this humanistic AI systems interact, making EmonyAI more responsive and attuned to human emotions and contextual subtleties.

Finally, EmonyAI aim to deploy a multi-agent system for personalized interactions that is scalable across various user types. This framework supports the scalability of the system, adapting to a wide range of scenarios and user types, thus broadening the potential applications of this innovative humanistic AI.

1.3 Project Scope and Direction

The scope of this project encompasses several key elements:

1. **Video Input Stream:** This component forms the frontline interface, capturing live video feeds of conversations. The video stream is crucial as it serves as the primary input for real-time analysis, enabling the system to observe and interpret both verbal and non-verbal aspects of communication.
2. **Agent Database:** A vital repository that stores detailed profiles of recognized individuals or "agents." This database includes not only historical interaction data but also continuously updated information about each individual's interactions, preferences, and behavioral patterns. This allows for personalized engagement and enhances the accuracy of the system's response.
3. **Emotion Detection Module:** Leveraging cutting-edge facial recognition and body language analysis technologies, this module assesses micro-expressions and physical gestures to determine the emotional state of individuals. By accurately identifying emotions, the system can tailor responses that are empathetic and contextually appropriate.
4. **Context Recognition Module:** This module enriches the system's understanding by extracting additional contextual information from both the audio and visual inputs. It analyzes background noise, speech content, and even locational cues within the video to grasp the full context of the interaction, which informs the content and tone of the suggested responses.
5. **Conversation Suggestion Engine:** At the core of EmonyAI is the integration of OpenAI's ChatGPT with the derived emotion and context data. This engine synthesizes all the analyzed data to generate intelligent and situational conversation suggestions, which are then relayed back to the user interface, facilitating a natural and insightful dialogue.
6. **MetaGPT Framework:** This framework orchestrates the operation of multiple AI agents, managing complex interactions and ensuring that the conversation suggestions are finely tuned to each individual's context and emotional state. By coordinating the diverse capabilities of various specialized agents, the MetaGPT framework enhances the overall intelligence and responsiveness of the system.

The direction of the EmonyAI project is to continue developing these components in harmony, ensuring each is optimized to function both independently and as part of an integrated whole. Future developments will focus on refining the accuracy of emotion and context recognition, expanding the database to include a broader spectrum of agents, and enhancing the adaptability of the conversation suggestion engine to handle a wide range of scenarios. The ultimate goal is to create an AI-driven system that not only understands and responds appropriately to the nuances of human interaction but also anticipates user needs, thereby setting new standards for AI in everyday interactions.

1.4 Contributions

Advancing Micro expression and Body Language Detection:

We propose a novel micro expression and body language detection model that achieves high accuracy in real-time, enabling its seamless integration into our conversation generators.

Contextual Response Generation:

By detecting micro expressions and body languages, the conversation generator can adapt its responses based on users' emotional cues. For example, it can suggest comforting responses during times of distress or adjust its tone to match audiences' emotions. This contextual response generation can enhance the quality of conversations and deepen user engagement.

Utilising latest technology:

Utilizing the latest technology in the form of large language models such as OpenAI's GPT models and Google's PaLM represents a quantum leap in the field of conversation generation. These cutting-edge AI systems, powered by deep learning and natural language understanding, have the capability to comprehend and generate human-like text at an unprecedented scale and quality. By harnessing the capabilities of these advanced models, we can create dynamic and context-aware conversations that offer users a more engaging and personalized experience, revolutionizing the way we interact with artificial intelligence and pushing the boundaries of what's possible in human-computer communication.

1.5 Report Organization

The organization of the report provides a comprehensive overview and analysis of the EmonyAI project, structured into clearly defined chapters to facilitate a detailed understanding of the project's scope, methodologies, outcomes, and future directions.

Chapter 1: Introduction - This chapter sets the stage by introducing the project's goals, the significance of the technological integration, and an overview of the key components and functionalities of the EmonyAI system.

Chapter 2: Literature Review - An exhaustive review of the existing research related to micro-expression detection and body language analysis. It discusses prior methodologies, advancements, and limitations, thereby contextualizing the EmonyAI project within the current scientific landscape.

Chapter 3: System Model - This chapter outlines the development methodology and system architecture of EmonyAI. Following the Rapid Application Development (RAD) approach, the project is broken into phases, including planning, design, implementation, and testing. The system is developed using Python for emotion detection and JavaScript for the user interface. The hardware includes an AMD Ryzen 7 processor and NVIDIA GTX1650 GPU for machine learning tasks. Key components like WebRTC for video/audio processing, a CNN-based emotion detection module, and Google Speech-to-Text are integrated into the system. The chapter also discusses the role of the Context Aggregator, which combines emotion and conversation data to provide personalized conversation suggestions via the EmonyAI chatbot. Profiles, conversation summaries, and emotion data are stored and used to enhance future interactions.

Chapter 4: System Design - This chapter details the design and operation of EmonyAI, including its hardware, software, and core components. The system block diagram shows how video/audio streams from WebRTC are processed by the emotion detection and speech-to-text modules to generate conversation suggestions via the EmonyAI chatbot. The chapter covers the specifications for user devices and backend services, including Google Cloud Run and Cloud Storage for profile management. It also explains how the system's core features—video calling, chatbot interaction, and real-time emotion detection—are implemented. Special modes

like Private Mode and Imagine Mode are designed to address privacy concerns and simulate interactions between agent profiles using the MetaGPT framework. The chapter further explains how the emotion detection module is built, deployed via FastAPI, and integrated into the system, along with the process for real-time video and audio processing using WebRTC.

Chapter 5: Experiment/Simulation - This chapter outlines the setup and testing of EmonyAI, detailing both hardware and software configurations. The system operated by integrating real-time video and audio streams with AI-driven emotion detection, speech-to-text conversion, and chatbot interaction. Multiple tests were conducted involving positive, negative, and multi-agent interactions, demonstrating EmonyAI's ability to adapt to various conversational contexts. Challenges encountered during implementation, such as GPU limitations and WebRTC latency, were addressed through GPU acceleration, WebRTC optimization, and noise-cancellation techniques. The chapter concludes with an evaluation of EmonyAI's scalability and effectiveness in enhancing user interaction through context-aware conversation support.

Chapter 6: System Evaluation and Discussion - This chapter evaluates the performance of EmonyAI across its core components: emotion detection, speech-to-text, and ChatGPT integration. System testing involved capturing and analyzing real-time emotional and conversational data, which was used to generate personalized suggestions. The evaluation showed that while the system generally performed well, challenges such as emotion detection accuracy, conversation suggestion relevance, and response time optimization were identified. Feedback from test users indicated that EmonyAI's empathetic and context-aware responses were effective, though improvements in creativity and timeliness were noted. Overall, the project successfully achieved its goals of enhancing video call interactions through real-time, AI-driven emotional and conversational insights, with opportunities for future refinement in accuracy and scalability.

Chapter 7: Conclusion and Recommendation - This chapter concludes by highlighting the success of EmonyAI in integrating emotion detection, facial recognition, and AI-powered conversation generation. Recommendations for future work include expanding the system to handle more complex human interactions, refining user profiles with location data, improving emotion detection models, and integrating with popular communication platforms like Zoom or Microsoft Teams to increase system scalability and usability.

Chapter 2 Literature Review

2.1 Previous work on Micro Expression

Face and expression detection

Yulan Guo and colleagues [31] introduced a 3D face recognition method that is robust to facial expressions. This approach is based on local geometric features and global similarity measures between faces. Their experimentation, conducted on the FRGCv2 dataset, involved comparing a probe face against a face database using both local features and 3D cloud registration. The achieved recognition rate was 97.0%, with a verification rate of 99.01%, irrespective of whether the faces exhibited neutral or non-neutral expressions.

Li Ye and others [20] proposed a 3D face recognition method that is insensitive to facial expressions. They employed 3D face matching, utilizing the iterative closest point algorithm and introduced an expression-irrelevant weighting factor to enhance the face matching algorithm's performance.

Sparse Representation Classification (SRC) approaches, commonly utilized for facial expression analysis [20], were employed for the classification of facial expressions, including joy, sadness, anger, fear, disgust, and surprise, using a 3D Face model [20]. The Modified Local Directional Patterns (MLDP) approach was utilized to achieve linear and faster recognition of facial expressions.

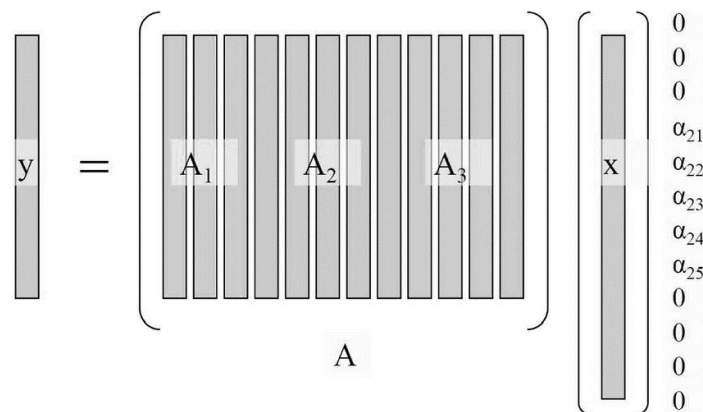


Figure 2.1.1 : Sparse Representation Classification

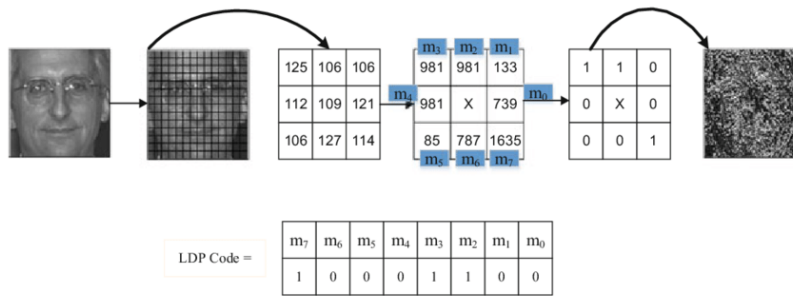


Figure 2.1.2 : Modified Local Directional Patterns

Furthermore, facial expression recognition involved video-based image and face expression analysis with various features. Facial expression analysis using Generalized Discriminant Analysis performed on deep belief networks exhibited superior performance compared to other available technologies [21].

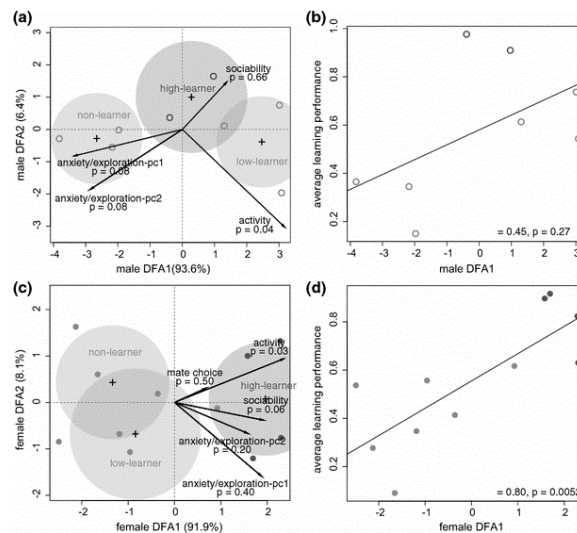


Figure 2.1.3 : Generalized Discriminant Analysis

In the domain of facial expression recognition, multimodal model detection is widely adopted, focusing on landmark and texture value analysis. Abdelghafour Abbad and colleagues [20] proposed a 3D face recognition approach based on geometric and local shape descriptors to address challenges associated with varying facial expressions. Their method involved four key steps: 3D face modelling, feature extraction, extraction of geometric information from the 3D surface in terms of curves, and generation of feature vectors at different scales. Their study, conducted on the GavabDB and Bosphorus datasets, yielded a recognition rate of 98.9%.

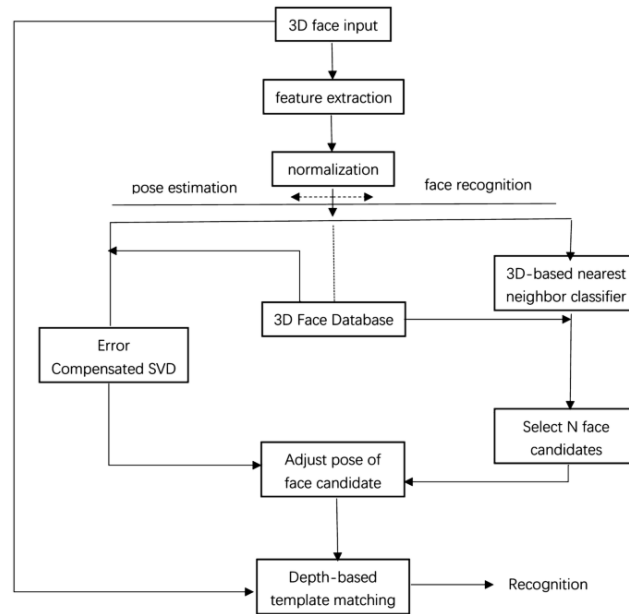


Figure 2.1.4 : 3D face recognition approach

Micro-expression detection

Early research in face expression detection laid the groundwork for the field. Most previous research focuses on video-based analysis of micro expressions. Researchers have developed computer vision algorithms to automatically detect and classify micro expressions in video recordings. Notable works include the use of Support Vector Machines (SVMs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory (LSTM) networks to recognize micro expressions.

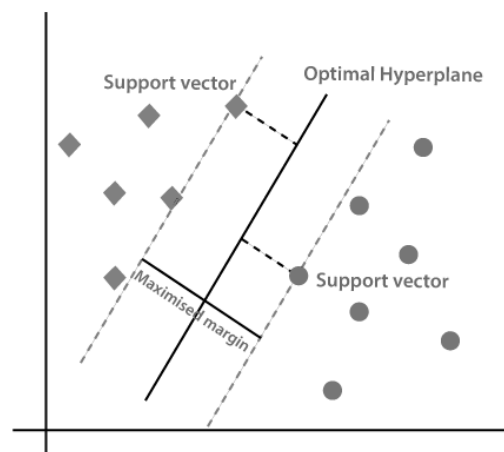


Figure 2.1.5 : Support Vector Machines (SVMs)

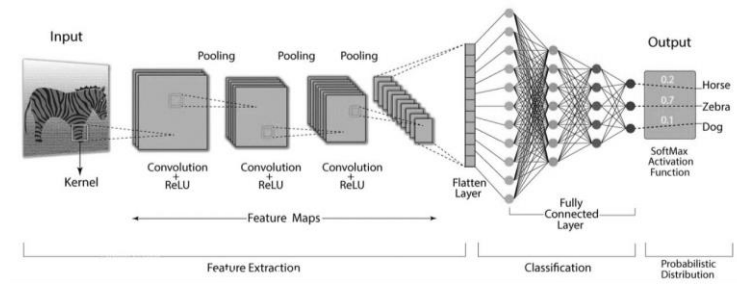


Figure 2.1.6 : Convolutional Neural Networks (CNNs)

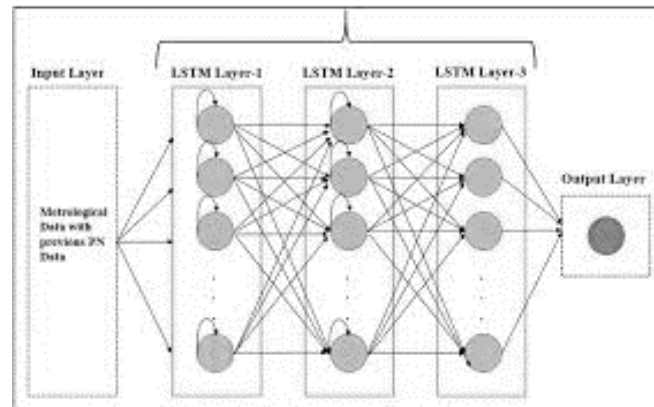


Figure 2.1.7 : Long Short-Term Memory (LSTM)

The creation of comprehensive micro expression databases, such as the MMI Facial Expression Database and CAS(ME)², has been instrumental in training and evaluating detection models. Researchers have utilized these databases to test the robustness and accuracy of their algorithms.

Recent advancements aim to enable real-time micro expression detection, a critical requirement for applications like deception detection, security, and human-computer interaction. These approaches leverage real-time facial feature tracking and efficient recognition algorithms to provide instantaneous feedback.

There are several studies have explored the challenges of micro expression detection. One previous study has focused on recognizing micro-expressions in video sequences, a few have used static images for recognition. The performance of models using static images and static feature extraction techniques, such as apex micro-expression images, has been less successful, as observed in the works of Hiranmayi et al. [14], On the other hand, studies by others such as Wei et al. [28] have achieved better results by utilizing temporal data, such as image sequences or videos.

Many of these reviewed studies divided images into blocks before applying LBP-TOP feature extraction to each block, resulting in an accuracy of 63.41%. In contrast, this study took a holistic approach to micro-expression analysis, avoiding image block division during feature extraction to reduce computational complexity, as suggested by Manthis et al. [32].

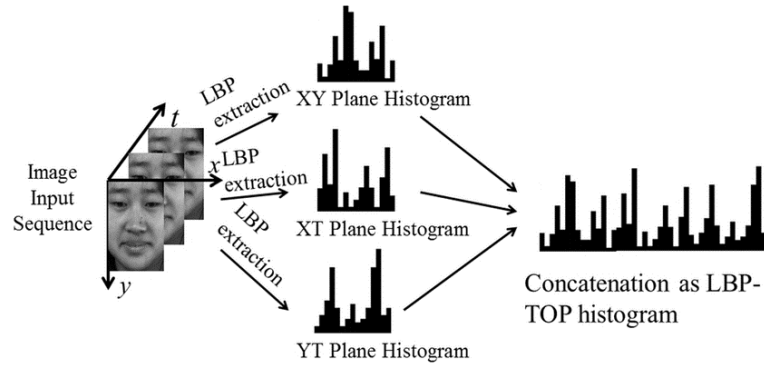


Figure 2.1.8 : LBP-TOP feature extraction

Other than the technical difficulties mentioned above, emotions may be expressed differently across cultures, making universal recognition models more complex to develop.

Despite significant progress, challenges persist in achieving high accuracy in micro expression detection, particularly in unconstrained real-world settings. Limitations in existing research include limited emotion categories, cultural insensitive and variations in data collection methods.

Comparison of existing facial emotion detection

Study/Authors	Methodology	Key Features	Dataset	Recognition Rate/Accuracy	Key Findings
Yulan Guo et al. [31]	3D face recognition using local geometric features and global similarity measures	Robust to facial expressions, 3D cloud registration	FRGCv2	97.0% recognition, 99.01% verification	High recognition rate, works well with neutral and non-neutral expressions

Study/Authors	Methodology	Key Features	Dataset	Recognition Rate/Accuracy	Key Findings
Li Ye et al. [20]	3D face matching with iterative closest point algorithm	Expression-irrelevant weighting factor	-	-	Enhanced matching algorithm, insensitive to facial
SRC Approach	Sparse Representation Classification for facial expression analysis	Classified joy, sadness, anger, fear, disgust, surprise	3D face model	-	Achieved linear and faster facial expression recognition
MLDP Approach	Modified Local Directional Patterns for faster expression recognition	Linear and fast facial expression recognition	-	-	Efficient method for rapid expression recognition
Abdelghafour Abbad et al.	3D face recognition based on geometric and local shape descriptors	3D face modeling, feature extraction, multi-scale feature vectors	GavabDB, Bosphorus	98.9% recognition rate	Effective in handling varying facial expressions
Generalized Discriminant Analysis [21]	Video-based image and face expression analysis on deep belief networks	High accuracy facial expression recognition	-	Superior performance	Outperformed other available technologies in facial expression recognition

Study/Authors	Methodology	Key Features	Dataset	Recognition Rate/Accuracy	Key Findings
Micro-expression Detection (Hiranmayi et al. [14], Wei et al. [28])	Static vs temporal data for micro-expression detection	Static images (apex micro-expression), temporal data (video sequences)	MMI, CAS(ME) ²	63.41% accuracy	Temporal data models outperformed static image models
SVM, CNN, LSTM Approaches	Micro-expression detection using Support Vector Machines, Convolutional Neural Networks, LSTMs	Real-time micro-expression detection, video analysis	MMI, CAS(ME) ²	-	Enabled real-time detection for applications like deception detection and HCI
LBP-TOP Feature Extraction [32]	Holistic micro-expression analysis without image block division	Avoided image block division to reduce computational complexity	-	63.41% accuracy	Holistic approach improved computational efficiency

Table 2.1.1 : Comparison of existing facial emotion detection

2.2 Previous work on Body Language

Pose Estimation

There are several algorithms for 2D human pose estimation, each with its own approach. Some algorithms directly predict the 2D pixel coordinates of predefined human body joints. For example, some used a cascade of pose regressors to refine joint predictions iteratively, while others progressively adjusted initial joint location predictions through an Iterative Error Feedback process.

More recent approaches take an indirect approach by predicting 2D body joint locations using CNNs to output body joint heatmaps, where heat maxima indicate joint positions. Some used a bidirectional tree-structured CNN-based network to model human body structure,

allowing feature channels at one point to receive information from others. Researchers combined CNNs with a deformable mixture of parts model to predict accurate body joint heatmaps. Researchers employed a CNN architecture with sequential "hourglass" modules, merging features across scales to generate high-resolution maps. Similarly, they also used stacked hourglass networks to produce body joint heatmaps, which were refined using Conditional Random Fields (CRFs). CPN divided the problem into two steps: localizing "easy" body joints first with a feature pyramid CNN and then detecting "hard" joints using a network with an online hard joint mining loss function. Yang et al. [27] proposed a two-stage framework where Independent Losses Pose Nets (ILPNs) inferred body joint locations globally, followed by Convolutional Local Detectors (CLDs) to refine joint detections. In contrast, [47] introduced a neural block for pose quality prediction alongside 2D pose regression, leading to a slight improvement in estimation accuracy.

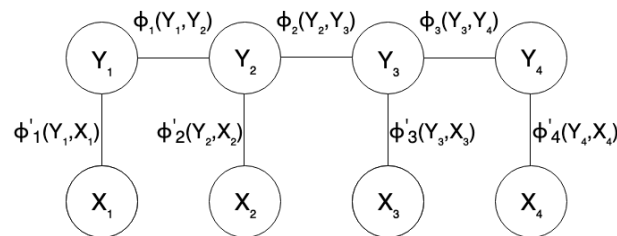


Figure 2.1.9 : Conditional Random Fields explained

In [3], a straightforward CNN architecture with convolutional and deconvolutional layers emphasized the importance of obtaining high-resolution feature maps for 2D pose estimation. Similarly, [12] designed a CNN architecture tailored to maintain high-resolution feature maps throughout the process by connecting multiple multi-resolution subnetworks and conducting multi-scale fusions.

Some recent approaches aimed for improved 2D pose estimation accuracy and faster inference, but through different methods. For example, Parallel Pyramid Network (PPNet) [48] used deep + wide and shallow + narrow subnetworks in parallel, with deep subnetworks processing low-resolution inputs for semantic information and shallow ones encoding spatial information from high-resolution inputs. They adjusted the method proposed to enhance inference speed by introducing a conditional channel weighting module in shuffle blocks. This allowed efficient information exchange between channels and features of varying resolutions, resulting in a lightweight network architecture with a balance of accuracy and complexity. Moreover, [11] introduced a single-branch network architecture for real-time multi-person human pose estimation on mobile platforms, reducing latency while maintaining performance

by incorporating Fusion Deconv Head and Large Kernel Conv layers. Finally, [6] aimed for efficient 2D pose estimation by leveraging depth data instead of RGB images and designing lightweight CNN architectures. Supplementary domain adaptation and knowledge distillation techniques were explored to further enhance accuracy.

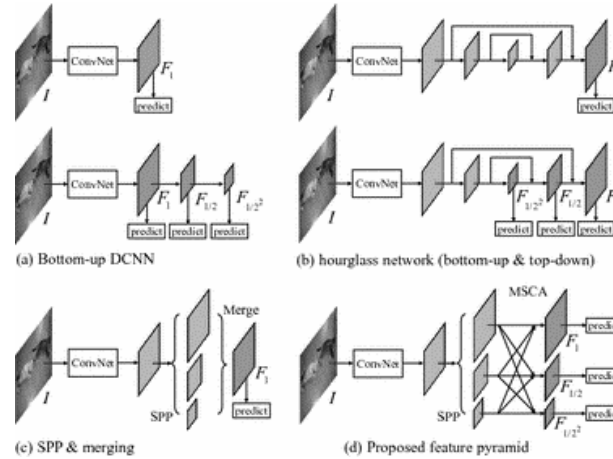


Figure 2.1.10 : Parallel Pyramid Network (PPNet)

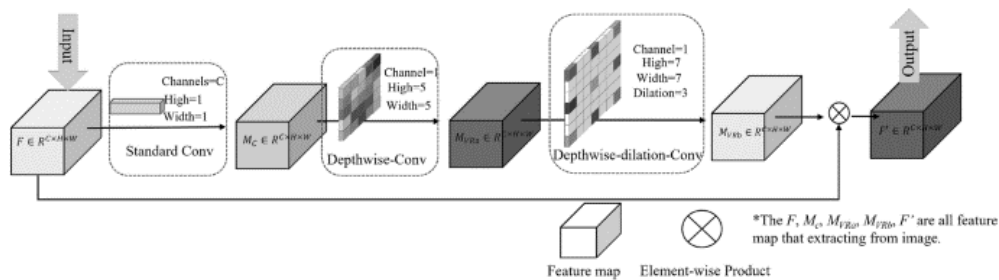


Figure 2.1.11 : Large Kernel Convolutional Layers

Body Language

Early studies on body language primarily focused on decoding basic non-verbal cues such as posture, gestures, and facial expressions. Previous works by researchers create a strong foundation for understanding the significance of non-verbal communication in conveying emotions and attitudes.

Advances in computer vision and machine learning have enabled the development of automated systems for body language detection. Researchers have employed techniques such as pose estimation, deep learning, and feature extraction to recognize and analyse body language from images and video.

A framework has been introduced to capture high-level parameters in body movements, which are then encoded by the hidden units of a convolutional autoencoder [2]. An approach for discerning a person's emotional state from both facial and body video data is proposed, leveraging Spatio-Temporal Interest Point (STIP) features [3]. A survey is conducted, discussing the field of sentiment analysis and its applications in deep learning [13] [18]. Deep learning algorithms are formulated for handling large datasets using the backpropagation algorithm [30]. A self-organizing neural architecture is devised for the recognition of emotional states based on full-body motion patterns [22]. An innovative model combines Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) for the purpose of emotion recognition from video data [13]. Deep learning models incorporating Deep Convolutional Neural Networks (DCNN) are developed for the multimodal emoFBVP database [14]. A model featuring hierarchical feature representation is introduced for non-verbal emotion recognition, demonstrating significant improvements in accuracy through experimentation [23]. An intelligently designed system for emotion recognition is proposed, utilizing neural network architectures [11]. The development of the Emotion Recognition in the Wild (EmotiW) system employs a hybrid CNN-RNN architecture, surpassing other techniques in achieving superior results [26]. A novel set of emotional body gestures is created to distinguish cultural and gender differences, forming the basis for an automatic emotional body gesture recognition framework [12].

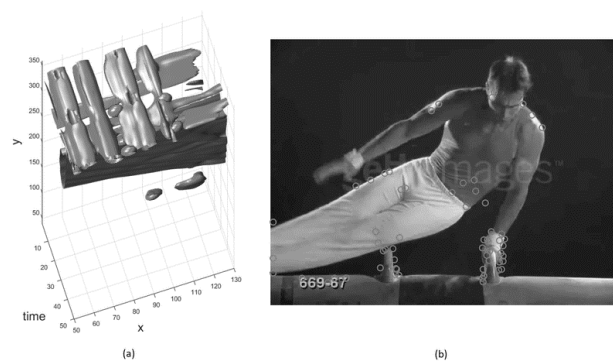


Figure 2.1.12 : Spatio-Temporal Interest Point (STIP)

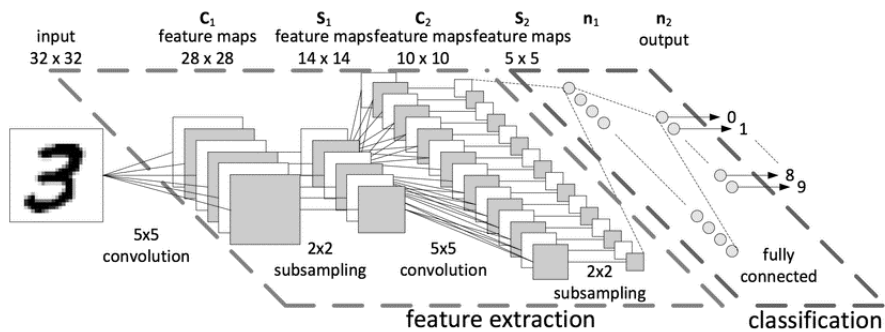


Figure 2.1.13 : Deep Convolutional Neural Network (DCNN)

Much of the contemporary research on body language detection is driven by applications in human-computer interaction (HCI). Studies have explored how body language recognition can enhance gesture-based interfaces, virtual reality experiences, and emotion-aware systems.

Cross-cultural considerations have gained prominence in body language research, acknowledging that the interpretation of non-verbal cues may vary significantly across cultures. Researchers have investigated the challenges of creating culturally sensitive body language detection models.

Recent work has focused on real-time body language detection, facilitating applications in areas such as healthcare, education, and security. These systems aim to provide instantaneous feedback based on body language cues, enabling more responsive interactions.

Despite progress, challenges persist in achieving accurate and robust body language detection, particularly in complex and unstructured environments. Limitations include the need for extensive training data, variations in lighting conditions, and the potential for misinterpretation.

Comparison of existing body language detection

Approach	Description	Challenges Addressed
Cascade of Pose Regressors	Refines joint predictions iteratively.	Improved joint prediction refinement.
Iterative Error Feedback	Progressively adjusts joint predictions through error feedback.	Reduces error accumulation over iterations.
Body Joint Heatmaps (CNNs)	CNNs output joint heatmaps where heat maxima indicate positions.	More precise localization of body joints.

Approach	Description	Challenges Addressed
Bidirectional Tree-structured CNN	Models body structure by allowing features at one joint to receive information from others.	Captures long-range dependencies across the body.
Deformable Mixture of Parts with CNN	Combines CNNs with deformable parts model to predict accurate body joint heatmaps.	Improves accuracy in detecting body joints.
Hourglass CNN Architecture	Merges features across scales to generate high-resolution maps.	Maintains high resolution for better precision.
Stacked Hourglass Networks	Uses stacked hourglass networks refined using Conditional Random Fields (CRFs).	Better joint localization with CRF refinement.
CPN (Feature Pyramid CNN)	First localizes 'easy' joints, then detects 'hard' joints using online mining loss function.	Enhances difficult joint localization.
Two-stage Framework (ILPNs + CLDs)	Infers joints globally, then refines them with convolutional detectors.	Improves detection accuracy through refinement.
Pose Quality Prediction Block	Adds a neural block for pose quality prediction along with 2D regression.	Improves pose accuracy with quality prediction.
Parallel Pyramid Network (PPNet)	Uses parallel deep and shallow subnetworks for faster and accurate inference.	Balances speed and accuracy with parallelism.
Single-branch Real-time Network	Optimized for real-time multi-person pose estimation on mobile platforms.	Reduces latency for mobile real-time applications.
2D Pose Estimation using Depth Data	Leverages depth data and lightweight CNNs for efficient pose estimation.	Utilizes depth data for efficient computation.

Table 2.1.2 : Comparison of existing body language detection

2.3 Limitations of Previous Studies

1. Limited Focus on Body Language or Micro expressions, but Not Both:

Many previous studies in the field of emotion recognition have tended to focus exclusively on either body language or micro expressions to detect and classify emotions. This segregated approach limits the holistic understanding of emotional cues during human interaction. The absence of a combined analysis of both modalities overlooks the potential synergies between body language and micro expressions in conveying emotions. In situation

2. Lack of Integration in Emotion Classification:

Previous research often fails to integrate body language and micro expression analysis into a unified framework for emotion classification. This separation results in a fragmented understanding of emotional states and hinders the development of more accurate and comprehensive emotion recognition systems.

3. Limited Emotion Types Detected:

Previous studies focus on a narrow set of basic or commonly expressed emotions (e.g., happiness, sadness, anger, fear). These studies may not account for the richness and complexity of human emotions, which encompass a wide spectrum of nuanced and subtle feelings. The detection of only a limited range of emotions may not be sufficient for applications requiring the recognition of more diverse and intricate emotional states.

4. Overlooking Complex Emotions:

Many previous studies might not adequately address the recognition of complex or blended emotions, which are common in real-life social interactions. Failing to account for complex emotional expressions may limit the practical applicability of emotion recognition systems in situations where individuals exhibit mixed or multifaceted emotional states.

5. Generalization Challenges:

Previous studies that have focused on specific emotions or limited emotion types may struggle to generalize their findings to a broader range of real-world scenarios. These limitations can hinder the adaptability and robustness of emotion recognition models when faced with diverse and unpredictable emotional expressions.

6. Potential for Misclassification:

Due to the limited scope of emotions considered, previous studies may be prone to misclassifying or misinterpreting complex emotional expressions, leading to inaccuracies in the results. Addressing these limitations and proposing an approach that combines both body language and micro expressions while expanding the range of detected emotions is a promising direction for advancing the field of emotion recognition and making it more applicable in real-world settings.

7. Failure to provide meaningful prompts:

Previous studies often output the analysis result in generic form or template which limited the model's ability to generate nuanced and personalized responses utilizing well-trained large-language model. The absence of contextual output mechanisms hindered the generation of prompts that could guide the model toward more relevant and context-aware responses.

2.4 Large Language Models

Large Language Models are essential part for generating conversation suggestion in this project. Large language models, such as GPT-4o and similar models, have been trained on vast amounts of text data from the internet. This training allows them to understand context and conversational nuances effectively. They can comprehend the flow of a conversation, the relationship between sentences, and the intent behind user queries. These models excel at natural language generation. They can produce human-like text that is coherent, contextually relevant, and grammatically correct. This is vital for generating conversation suggestions that feel organic and engaging.

OpenAI's GPT

GPT, short for "Generative Pre-trained Transformer" builds upon the success of its predecessors, GPT, to offer even more sophisticated natural language understanding and generation capabilities. GPT is part of the Transformer architecture family, which has revolutionized the field of natural language processing (NLP) by demonstrating remarkable proficiency in various language-related tasks.

Strengths:

- **Natural Language Generation:** GPT excels in generating human-like text. It can produce coherent and contextually relevant sentences, making it invaluable for tasks like content generation, chatbots, and text completion.
- **Large-Scale Knowledge:** This model has been trained on a large amount of text data from the internet, giving it extensive general knowledge. It can answer questions, provide explanations, and offer information on a wide range of topics.
- **Adaptability:** GPT can be fine-tuned for specific applications or domains, making it highly adaptable. Developers can customize its behavior to suit the needs of various projects, from customer support chatbots to creative writing assistance.
- **Multilingual Support:** It can understand and generate text in multiple languages, enabling cross-lingual applications and accessibility for a global audience.
- **Versatility:** GPT can perform various NLP tasks, including text summarization, translation, sentiment analysis, and more. Its versatility makes it suitable for a wide array of applications.

Weaknesses:

- **Lack of Real Understanding:** While GPT can generate text that appears to understand context, it lacks genuine comprehension. It relies on patterns and associations in the data it was trained on, rather than true understanding.
- **Potential for Bias:** Like many large language models, GPT can inadvertently produce biased or inappropriate content if not carefully controlled and monitored. It may perpetuate existing biases present in its training data.
- **High Computational Requirements:** GPT is computationally intensive, requiring substantial resources for training and inference. This can limit its accessibility to researchers and developers with limited computing power.
- **Lack of Real-Time Interaction:** The model may exhibit limitations in real-time interactive applications. Latency and response time can be a concern for applications requiring immediate user interactions.
- **Expensive:** Utilizing GPT can be costly, particularly for prolonged or high-usage applications, due to the computational resources required

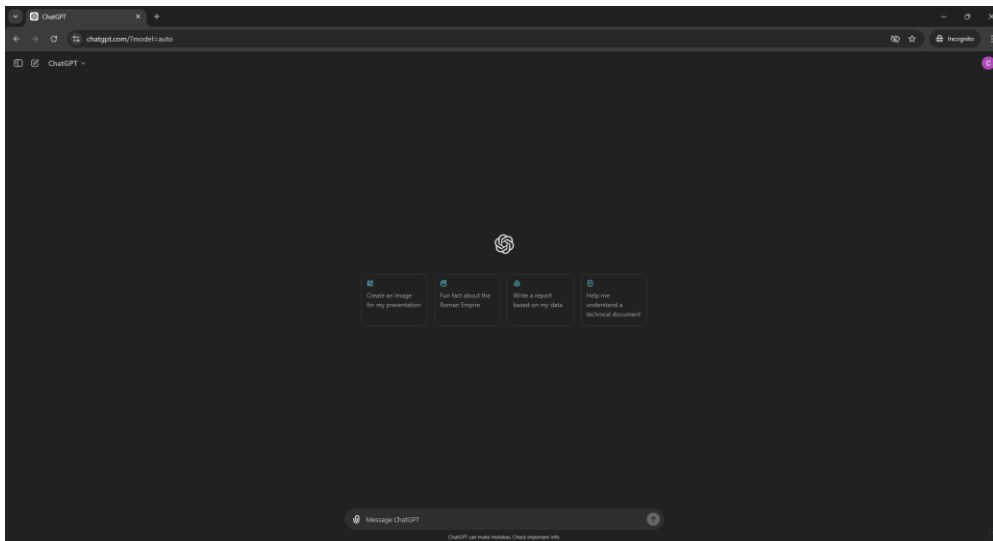


Figure 2.4.1 : User Interface for GPT

OpenAI's GPT-4o

GPT-4o is a specialized variant of OpenAI's GPT-4 model, designed to handle a wide range of language tasks with improved reasoning and natural language generation capabilities. It builds upon its predecessor by offering more sophisticated text processing and adaptability. Although it retains some of the beta-stage limitations, GPT-4o is well-suited for a variety of complex applications.

Strengths:

- **Advanced Problem-Solving:** GPT-4o excels in handling complex tasks, thanks to its improved reasoning and language generation abilities. It can tackle intricate queries and produce well-thought-out responses
- **Knowledgeable:** It has access to a vast repository of knowledge across diverse fields, making it highly proficient in answering questions and providing information
- **Fine-tuning:** GPT-4o's performance can be tailored to specific use cases, improving its relevance and accuracy for specialized tasks
- **Responsiveness:** GPT-4o is designed to respond quickly to queries, offering efficient interactions for a variety of applications

Weaknesses:

- **Limited Real-time Data:** Unlike models like Google's Gemini, GPT-4o relies on static training data, making it less effective in handling real-time information
- **Specialized Task Limitations:** While GPT-4o handles complex tasks well, it may struggle with specific scenarios requiring multimodal input (text, images, audio)

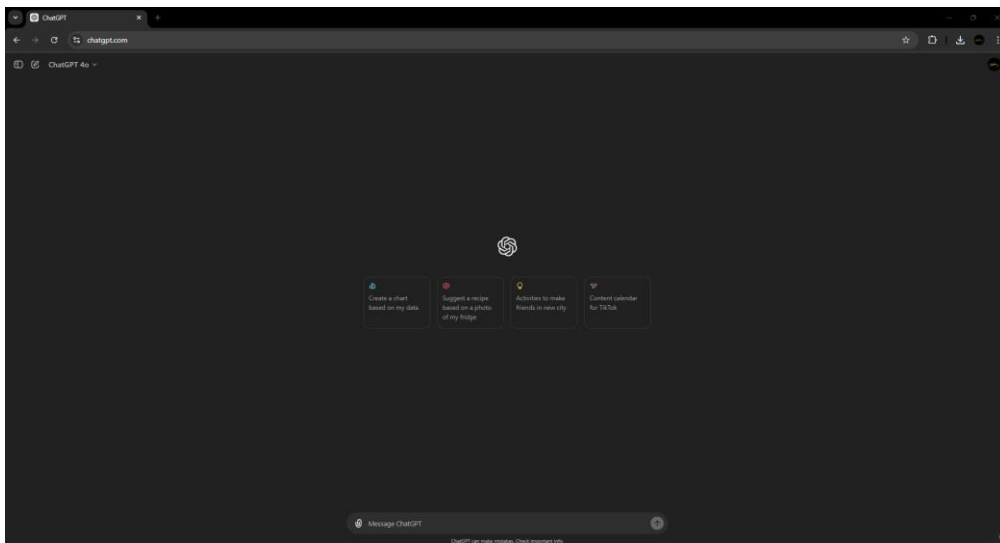


Figure 2.4.2 : User Interface for GPT4o

Google's Gemini

Google Gemini is a multimodal language model developed by Google, designed to seamlessly process and integrate various types of data, including text, images, audio, and video. Gemini boasts strong integration within the Google Ecosystem, enhancing its utility for users who leverage Google's extensive range of services.

Strengths:

- **Multimodal Capabilities:** Gemini can process and integrate different types of information (text, images, audio, video), enhancing its versatility for various tasks
- **Integration with Google Ecosystem:** Seamless integration with Google services, beneficial for users entrenched in Google's tools and platforms
- **Real-time Web Access:** Capable of accessing and processing up-to-date information from the web, similar to Copilot, beneficial for tasks requiring current data
- **Customizable via Plugins:** Offers extensions through plugins, such as for controlling Spotify or searching with TripAdvisor, adding functional flexibility

Weaknesses :

- **Relative Novelty:** As a newer model, it may have fewer use cases and less maturity compared to established models like GPT-4[34]
- **Computational Expense:** Usage can be resource-intensive, potentially limiting access for users with constrained technical resources
- **Creative Limitations:** Less capable in generating highly original content such as poetry or in-depth articles
- **Inconsistency and Limited Generative Capabilities:** May produce inconsistent responses and lacks robust long-form content generation compared to competitors like ChatGPT
- **Lack of Source Citation:** Does not always provide citations, which can be a drawback for research purposes

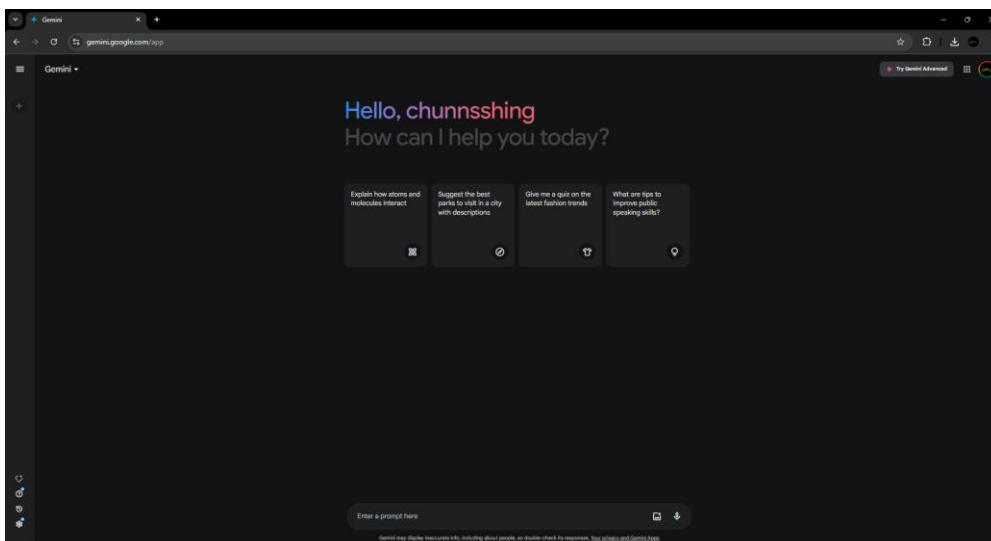


Figure 2.4.3 : User Interface for Google Gemini

Claude

Claude is a transformer-based large language model developed by Anthropic, focusing on ethical AI practices. It's designed to generate safe and aligned responses, emphasizing the reduction of harmful outputs and bias. Claude offers advanced capabilities in summarization, question-answering, coding, and multilingual processing, aiming to support both individuals and enterprises across various applications.

Strengths:

- **Ethical AI:** Claude is rigorously developed to minimize harmful behavior and bias, making it suitable for sensitive applications.
- **Advanced Reasoning and Multimodal Capabilities:** It can handle complex cognitive tasks and supports text and image inputs, offering robust performance in diverse domains.
- **Customization:** Users can steer Claude's personality, tone, and behavior to align with specific requirements, enhancing user interaction.
- **Safety and Transparency:** With a strong emphasis on safety, Claude provides interpretable responses, ensuring that its decision-making process is transparent and trustworthy.

Weaknesses:

- **Limited Public Access:** Access to Claude is more restricted compared to other models like ChatGPT or Gemini, which may limit its usage across broader scenarios.
- **Performance in Highly Specialized Tasks:** While Claude excels in ethical considerations, it might not match the raw performance of models like ChatGPT or Gemini in highly specialized or data-heavy applications.

2.4.1 Comparison for different Large Language Models

LLM	Best Use Case	Compared to other LLMs
GPT	Content generation, chatbots, text summarization	Versatility: Excels in generating human-like text for a wide range of applications, including multilingual support
GPT-4o	Complex problem-solving, advanced reasoning tasks	Advanced Reasoning: Improved in handling complex tasks and queries with sophisticated text processing

Google's Gemini	Multimodal tasks (text, image, audio, video processing)	Multimodal Capabilities: Seamlessly integrates and processes diverse data types across various formats
Claude	Ethical applications, sensitive content moderation	Ethical AI Focus: Prioritizes minimizing harmful outputs, with a strong emphasis on safety and transparency

Table 2.4.1 : Comparison of Large Language Models

2.4.2 Selection of Large Language Model

After studying several of the most popular large language models available in the market, we have selected OpenAI's GPT-4o as the LLM for conversation generation due to the following key reasons:

- **Superior Natural Language Understanding and Generation:** GPT-4o is designed with an advanced architecture that excels in generating human-like conversations. Its ability to produce coherent, contextually relevant, and nuanced responses making interactions with EmonyAI more fluid and natural.
- **Extensive Knowledge Base:** Trained on vast amounts of diverse data, GPT-4o offers a wealth of general knowledge, allowing it to react to a wide variety of questions and able to analyze conversations across many topics. This makes it a highly capable tool for handling diverse user interactions.
- **Personalization and Fine-tuning:** GPT-4o can be fine-tuned to adapt to specific domains or industries, which allows us to customize the EmonyAI's responses to align with our project goals and provide more relevant, personalized interactions for users.
- **Multilingual Capabilities:** GPT-4o's proficiency in multiple languages broadens the reach of our chatbot, enabling us to serve audience with different languages, increasing scalability, accessibility and usability.
- **Versatility in Task Handling:** Beyond basic conversation generation, GPT-4o can perform a wide range of NLP tasks such as summarization, sentiment analysis, and translation. This versatility perfectly suits the need of EmonyAI's functionalities beyond simple conversation.

- **Reliability and Performance:** As one of the most tested and mature models tested by large number of users, GPT-4o offers a stable and reliable foundation for building EmonyAI, minimizing the risks associated with errors or inconsistencies.

2.5 Multi-Agent Framework

Background and Evolution of Multi-Agent Framework

Multi-agent systems comprise multiple interacting intelligent agents within an environment. These systems are used to solve problems that are too complex for an individual agent or monolithic system to handle. Multi-Agent Framework can perform both cooperative (for shared goals) and competitive (for individual goals) tasks. Early Multi-Agent Framework implementations were often rule-based or utilized simpler machine learning models that limited their adaptability and scalability.

Introduction to MetaGPT and Generative Pre-trained Transformers

Generative Pre-trained Transformers (GPT) models, developed by OpenAI, represent a significant advancement in artificial intelligence by using deep learning techniques for natural language understanding and generation. The MetaGPT framework extends these capabilities by incorporating multiple GPT models that can represent different agents with varying objectives and knowledge bases, facilitating more complex and dynamic interactions within Multi-Agent Framework.

Integration of MetaGPT in Multi-Agent Framework

The integration of MetaGPT has been highlighted in recent literature for its ability to enhance decision-making processes and communication strategies among agents. MetaGPT allows each agent to develop its unique responses based on not only pre-trained general knowledge but also on specific training tailored to their roles in the Multi-Agent Framework. This role-specific adaptation is crucial for applications requiring differentiated knowledge bases or specialized interaction strategies.

Applications of MetaGPT in Multi-Agent Framework

Significant applications of MetaGPT in Multi-Agent Framework include automated negotiation systems, where agents represent different stakeholders in negotiations and must generate and evaluate proposals based on learned models of negotiation tactics. Another application is in distributed sensor networks, where agents process local data and communicate

findings to optimize global outcomes, such as in climate monitoring or urban planning. These applications demonstrate MetaGPT's utility in enhancing agent communication and decision-making capabilities.

Challenges and Future Directions

While the adoption of MetaGPT in Multi-Agent Framework has shown promising results, there are several challenges to be addressed. These include ensuring consistency and coherence in interactions among agents, managing the computational demands of multiple GPT models, and securing the system against adversarial attacks or biases in decision-making processes. Future research is directed towards optimizing these models for greater efficiency, improving the robustness of the systems, and expanding their applicability in more diverse fields.

CHAPTER 3 SYSTEM MODEL

3.1 System Methodology

In this chapter, it describes some methods and technologies used to develop this project, which is prototyping under Rapid Application Development (RAD) methodologies. Besides, this application is developed using Python Language. The processes of the project were categorized into different phases in the development, which were planning phase, analysis phase, design phase, implementation phase, testing with the system prototype and final implementation when no problem is found.

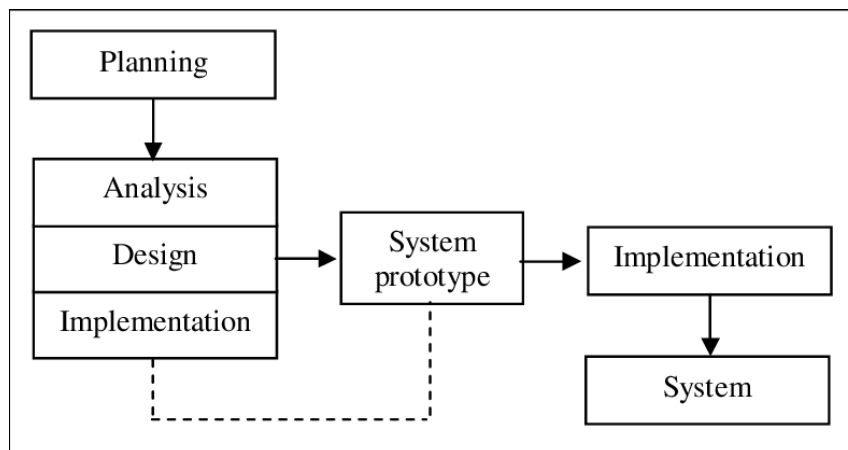


Figure 3.1 Process of Prototyping RAD methodology

3.2 System Design

3.2.1 Hardware

The hardware involved in this project is mainly a computer. The laptop is needed for the process of developing the emotion detection machine learning model, the user interface, and EmonyAI itself which is the centralized function that manages other modules. The processor of the laptop will be AMD Ryzen 7 5800HS with NVIDIA GeForce GTX1650 graphic card to ensure the laptop can operate smoothly and can process the emotion detection machine learning model well.

Table 3.1.1 Specifications of laptop

Description	Specifications
Model	ROG Zephyrus G14 GA401QH
Processor	AMD Ryzen 7 5800HS with Radeon Graphics
Operating System	Windows 11
Graphic	NVIDIA GeForce GTX1650
Memory	16GB DDR4 RAM
Storage	512GB SATA HDD
Microphone	Integrated Realtek® Audio

Other than the laptop itself, a Logitech HD1080p webcam is also connected to the laptop for the video input.

3.2.2 Software

Windows 11 will be used as the operating system for the application development. The tools for developing different module of the system are different and are as follows:

Table 3.1.2 Software used for the emotion detection module

Software	Specifications
Programming Language	Python (.ipynb)
Coding Software	VS Code
Others	Kaggle – To collect dataset for the training of the machine learning model

Table 3.1.3 Software used for the user interface

Software	Specifications
Programming Language	Javascript(.jsx)
Coding Software	VS Code
Others	Vite react – Build tool for website Ant Design – UI component library

Table 3.1.4 Software used for the profile management

Software	Specifications
Programming Language	Python (.py)
Coding Software	VS Code
Storage	Google Cloud Services Bucket
Others	MetaGPT Framework – To orchestrate and automate multi-agent collaboration

Table 3.1.5 : Software used for the website deployment

Software	Specifications
Platform	Google Cloud Run
Console	Google Cloud Console
Others	Docker – Containerize the website coding

Table 3.1.6 : Other software used and their functionality

Software	Specifications
Firebase	To handle request from WebRTC
Google Speech-To-Text API	To convert recorded audio into text in real-time for profile characterisation
ChatGPT API	- To summarize the text recorded through Speech-to-text into simplified version to avoid over large amount of data - To generate contextual response in real time for the user

3.2.3 System Architecture Diagram

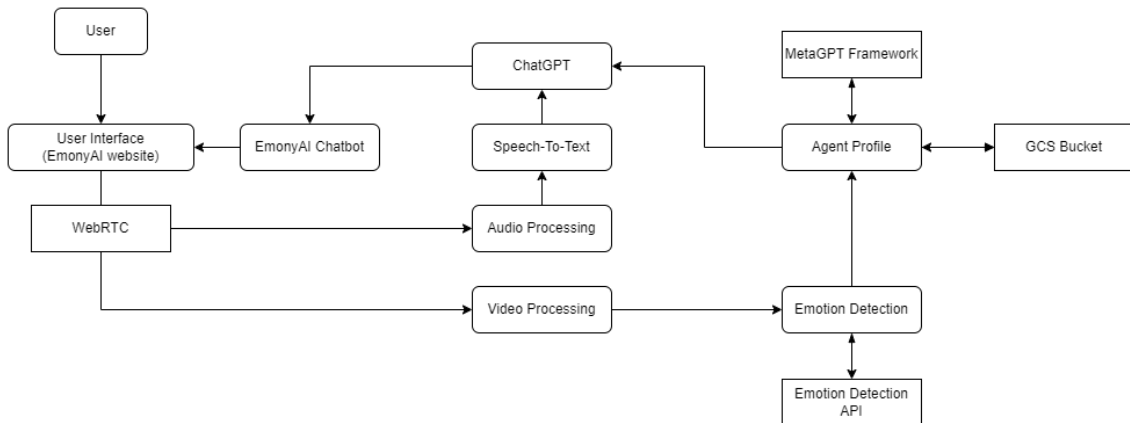


Figure 3.2.1 : System Architecture Diagram

User

- The user interacts with the system via the EmonyAI website, engaging in real-time conversations through text, audio, and video.

User Interface (EmonyAI Website)

- The frontend interface where users communicate with the system. It integrates various services like WebRTC, EmonyAI Chatbot, and real-time emotion detection to enhance the user experience.

WebRTC

- WebRTC is responsible for real-time audio and video communication between the user and the system, facilitating voice and video calls. It sends:
 - Audio data to the Audio Processing component.
 - Video data to the Video Processing component.

EmonyAI Chatbot

- The chatbot provides conversational interaction with the user. It uses ChatGPT for generating natural language responses and integrates additional contextual data (detected emotion and recorded history in agent profile) for better, more personalized responses. It also takes direct input from the user for multiple functions.

ChatGPT

- ChatGPT is utilized by the EmonyAI Chatbot for generating intelligent and context-aware responses. It integrates processed summary from the agent profile recorded or real-time emotion detected to produce contextual responses for the user.

Speech-to-Text

- Converts user conversation through WebRTC Audio Processing into text for processing by the chatbot and other components of the system, enabling more context recorded into agent profile.

Audio Processing

- This component processes audio streams from the user for speech recognition via Speech-to-Text to convert user conversation into text for agent profile.

Video Processing

- The video stream is analyzed by the Emotion Detection system to detect the user's emotions based on facial expressions and body language cues.

Emotion Detection

- The Emotion Detection system processes video streams to identify the emotional state of the user. The results are sent to the Emotion Detection API to generate the emotion detection result and used by other parts of the system, such as the chatbot, to generate more empathetic responses.

MetaGPT Framework

- The MetaGPT Framework is responsible for:
 - Storing Agent Profiles: It manages the profile of the agent (the other user on the call), storing context about their preferences and history.
 - Multi-agent Collaboration: When necessary, the framework coordinates between different agents (like ChatGPT, emotion detection, etc.) to manage more complex interactions and collaboration.

Agent Profile

- This profile stores the context and history of the agent (the user at the other end of the call), which is used to generate conversational suggestions tailored to the agent's behavior and interaction history. The profile data includes previous conversations, emotional patterns, and basic preferences, which helps the system generate contextually relevant responses during the call.

GCS Bucket (Google Cloud Storage)

- This is used for persisting individual agent profile, temporary data such as emotion_tempo.txt and speech_tempo.txt. It ensures that the agent profile are well store and the temporary data is accessible through API key.

Emotion Detection API

- This Emotion Detection API exposes the emotion detection results to the rest of the system, allowing the system to access and use real-time emotion data for enhancing interaction quality.

3.3 Important modules

3.3.1 WebRTC - Video and Audio Processing

The EmonyAI system utilizes WebRTC, a robust protocol designed for real-time communication, to manage both video and audio streams between users. WebRTC ensures reliable two-way communication, handling video and audio data transmission efficiently.

Video Processing

Video processing in the system is handled through the MyCam and FriendCam components, both of which interface directly with WebRTC to ensure smooth video communication.

1. MyCam: This component is responsible for capturing and displaying the local video stream from the user's webcam. When the user starts their webcam via the interface, the local video feed is captured through WebRTC's media capture capabilities and displayed in the MyCam section. This ensures that the user can see their own video feed in real-time.
2. FriendCam: The FriendCam component manages the remote video stream, which represents the incoming video from the other participant in the call. WebRTC handles the transmission of this remote video, ensuring that the stream is displayed seamlessly in the FriendCam interface. The video is rendered automatically using WebRTC's real-time media exchange, adhering to WebRTC's default protocols for video transmission.

Both MyCam and FriendCam utilize the HTML <video> element with attributes like autoPlay and playsInline to guarantee smooth playback without additional manual intervention. These elements are linked directly to WebRTC streams (local for MyCam and remote for FriendCam), ensuring real-time transmission and display.

Audio Processing

Audio processing in the system is fully managed by WebRTC's default handling of audio streams, ensuring seamless integration with video streams. WebRTC automatically captures and transmits both audio and video within the same stream, ensuring synchronization between the two.

- Local Audio Capture: Audio from the user's microphone is captured and sent as part of the WebRTC localStream.

- Remote Audio Playback: Audio from the remote participant is included in the remoteStream, which is played back in real-time to the user.

WebRTC Integration

WebRTC serves as the core technology for both video and audio transmission in the system. It performs the following functions according to its default protocols:

- Media Capture: Captures both the user's webcam video and microphone audio.
- Real-Time Communication: Establishes a peer-to-peer connection between participants for efficient and secure media streaming.
- Stream Management: WebRTC handles both local and remote streams, ensuring they are properly synchronized and rendered on the user interface.

The CallSetting component in the system's user interface controls WebRTC functionalities, such as starting the webcam, creating and answering calls, and ending a session. These actions trigger WebRTC's protocols to initialize and manage media streams.

The system's Video and Audio Processing is handled efficiently by WebRTC, adhering to its default protocols for real-time media transmission. The MyCam and FriendCam components are responsible for displaying local and remote video streams, respectively, while WebRTC's default audio handling ensures synchronized voice communication. By leveraging WebRTC, the system guarantees reliable, high-quality communication with minimal manual intervention, providing users with a seamless experience during video calls.

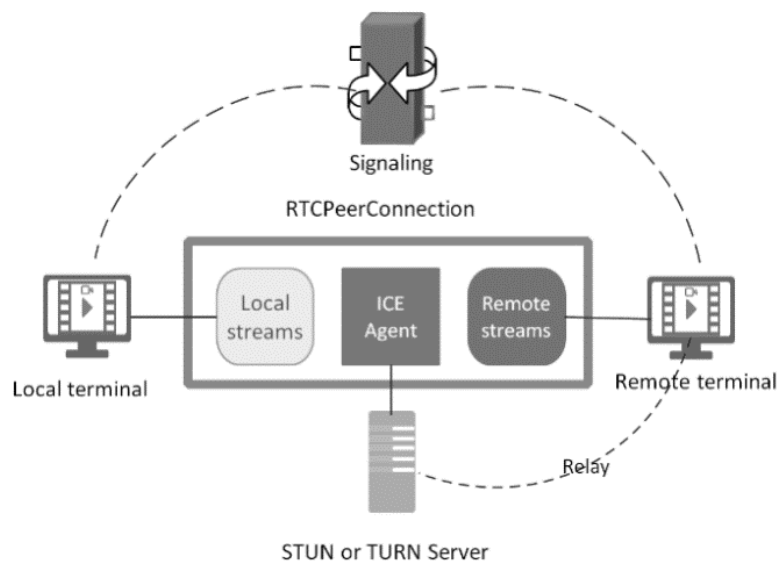


Figure 3.3.1 : WebRTC Architecture

3.3.2 Emotion Detection

In the EmonyAI system, the emotion detection module is powered by a Convolutional Neural Network (CNN), designed to analyze facial expressions and determine the emotional state of the user in real time. The neural network is responsible for processing image data captured during video calls and classifying it into one of several predefined emotion categories.

Neural Network Architecture

The emotion detection model is based on a Convolutional Neural Network (CNN). CNNs are highly effective for image-related tasks due to their ability to capture spatial features from images, making them ideal for recognizing patterns such as facial expressions.

1. Convolutional Layers:

- These layers form the core of the model, where filters are applied to the input images to automatically detect facial features. The convolution operation helps identify critical elements like edges, textures, and facial landmarks that are essential for emotion recognition.
- Multiple convolutional layers are stacked, allowing the model to learn increasingly complex features at different levels, from basic facial components to more abstract emotional cues.

2. Pooling Layers:

- After each convolutional layer, pooling layers are used to reduce the spatial dimensions of the feature maps. This down-sampling not only reduces the computational complexity but also helps retain the most important features, making the model more robust to variations in facial positioning or lighting conditions.

3. Fully Connected Layers:

- The final layers of the network are fully connected, meaning every neuron in one layer is connected to every neuron in the next. These layers take the features extracted by the convolutional layers and use them to classify the image into a specific emotion category.
- By learning the relationships between these high-level features, the network can accurately determine which emotion the facial expression represents.

4. Softmax Layer for Classification:

- The last layer in the neural network uses a Softmax activation function to assign probabilities to each possible emotion class. The model outputs the emotion category with the highest probability, such as happiness, sadness, anger, surprise, or other emotions.

Real-Time Emotion Detection

Once the model is trained, it can be deployed to perform real-time emotion detection. During a video call, the system captures frames of the user's facial expressions, which are fed into the neural network. The network processes these images in real-time and determines the user's emotional state, which can then be used by other components of the system, such as the chatbot, to tailor interactions based on the detected emotion.

Role in the EmonyAI System

The neural network plays a critical role in making the system emotionally intelligent. By analyzing visual data from the user's facial expressions, the emotion detection module helps:

- Personalize interactions: The chatbot can adjust its responses and tone based on the user's detected emotions.
- Improve user engagement: By responding to the user's emotional state, the system makes interactions more empathetic and relevant.

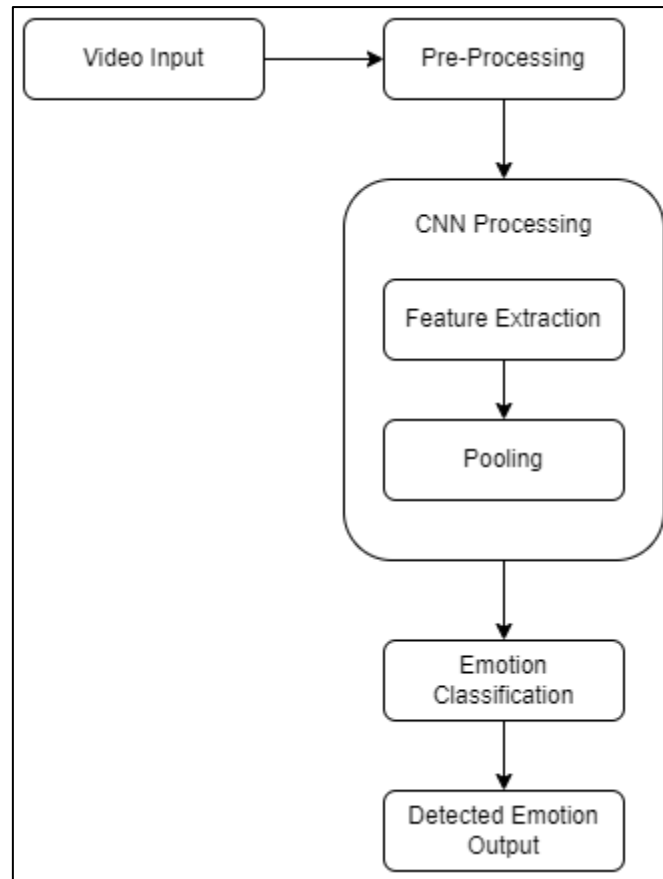


Figure 3.3.3 : Architecture for Emotion Detection Module

3.3.3 Google Speech-To-Text

The Google Speech-to-Text service is integrated into the EmonyAI system to perform the critical task of converting spoken conversation during video calls into text. This transcription is essential for various downstream processes, enabling the system to record and analyze conversations effectively.

The primary function of the service is to transcribe audio data from the conversation into a text format. Once the spoken content is converted to text, the following processes take place:

1. **Summarization by ChatGPT:** The transcribed conversation will be first stored in `speech_tempo.txt` in the GCS Bucket. After a set time, it will be passed to the ChatGPT model, which processes the full text and generates a summarized version. This summary captures the essence of the conversation in a more concise format to prevent recording too much data.

2. Storing in Agent Profile: The summarized conversation is then stored in the corresponding agent profile. This stored data includes not only the summarized text but also other important metadata that is recorded from other service, such as:
 - Recorded Emotion: The emotional state detected during the conversation.
 - Date and Time: The timestamp of when the conversation occurred.

By combining the transcribed text, emotional context, and timestamp, the system ensures that each agent's profile is updated with comprehensive records of past interactions. This enhances future interactions by providing context-aware suggestions based on previous conversations and emotions.

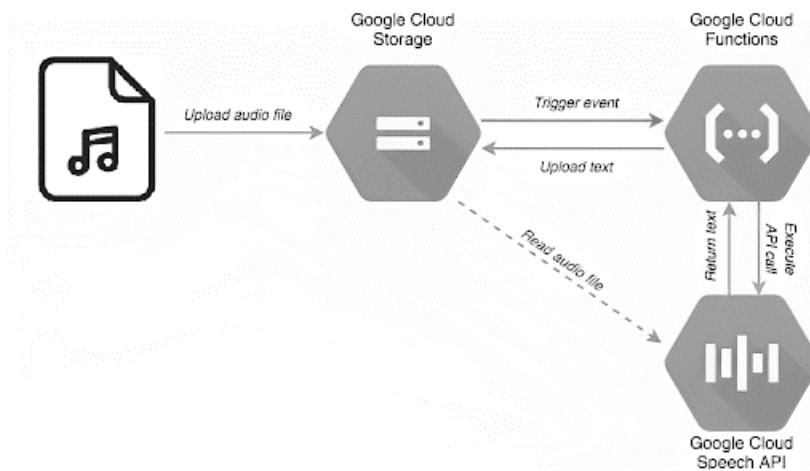


Figure 3.3.4 : System architecture for Google Speech-to-Text API

3.3.4 Combining Emotion Detection and Speech-to-text into Context Aggregator

In the EmonyAI system, the Context Aggregator serves as a crucial component that combines the outputs from the Emotion Detection and Speech-to-Text modules. This integration allows the system to generate contextually aware conversation suggestions by analyzing both the user's emotional state and the content of the conversation, providing a more personalized and empathetic interaction during video calls.

Emotion Detection

The Emotion Detection module continuously analyzes the user's facial expressions captured during the video call to determine their emotional state (e.g., happiness, sadness, anger, or surprise). The identified emotion is then sent to the Context Aggregator, giving real-time insights into the user's emotional response throughout the conversation.

Speech-to-Text

At the same time, the Speech-to-Text module transcribes the spoken conversation between the participants into text. This transcribed content is further summarized by ChatGPT to extract the key points from the conversation. The summarized content is passed to the Context Aggregator for further analysis.

Context Aggregation

The Context Aggregator integrates two critical data streams:

1. **Emotion Data:** The real-time emotional state of the user from the Emotion Detection module.
2. **Transcribed Conversation Content:** The textual summary of the conversation generated from the Speech-to-Text module.

By combining these streams, the Context Aggregator generates a comprehensive understanding of the user's emotional and conversational context, allowing the system to make informed decisions about the direction of the interaction.

Generating Contextual Conversation Suggestions

Using the combined emotion and conversation data, the Context Aggregator provides contextual conversation suggestions to the EmonyAI Chatbot:

- **Emotion-based responses:** If the user's emotional state indicates stress or frustration, the chatbot adjusts its tone and responses accordingly, offering more empathetic or supportive comments.
- **Content-based responses:** The chatbot uses the summarized conversation content to provide relevant follow-ups or suggestions that align with the topics discussed, ensuring the interaction remains focused and engaging.

For example, if the user discusses a challenging topic while the Emotion Detection module identifies signs of frustration, the chatbot may offer comforting suggestions based on recorded history or acknowledge the user about the agent's emotional state to improve the overall interaction quality.

Storing in Agent Profile

Both the summarized conversation and the detected emotional states are stored in the Agent Profile. This information enables the system to adapt and improve future interactions by drawing on past conversations and emotions, further enhancing the personalized experience over time.

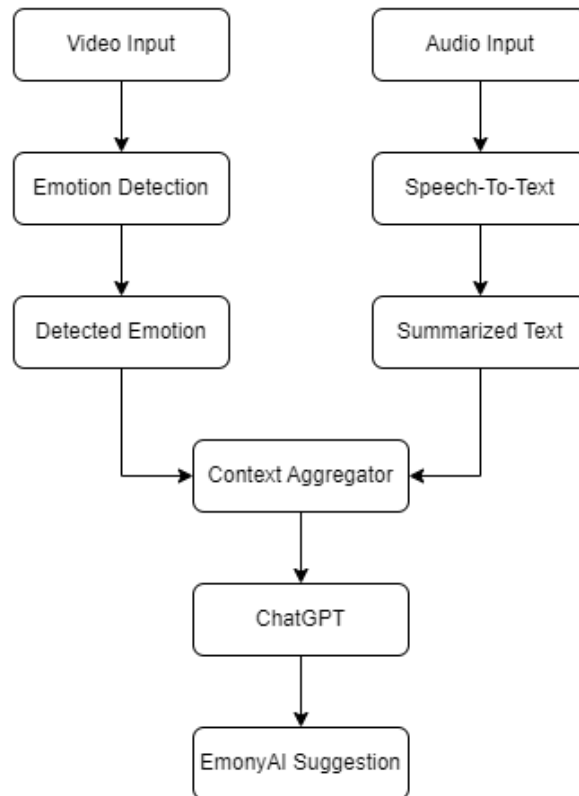


Figure 3.3.5 : System Architecture for Context Aggregator

CHAPTER 4 SYSTEM DESIGN

4.1 System Block Diagram

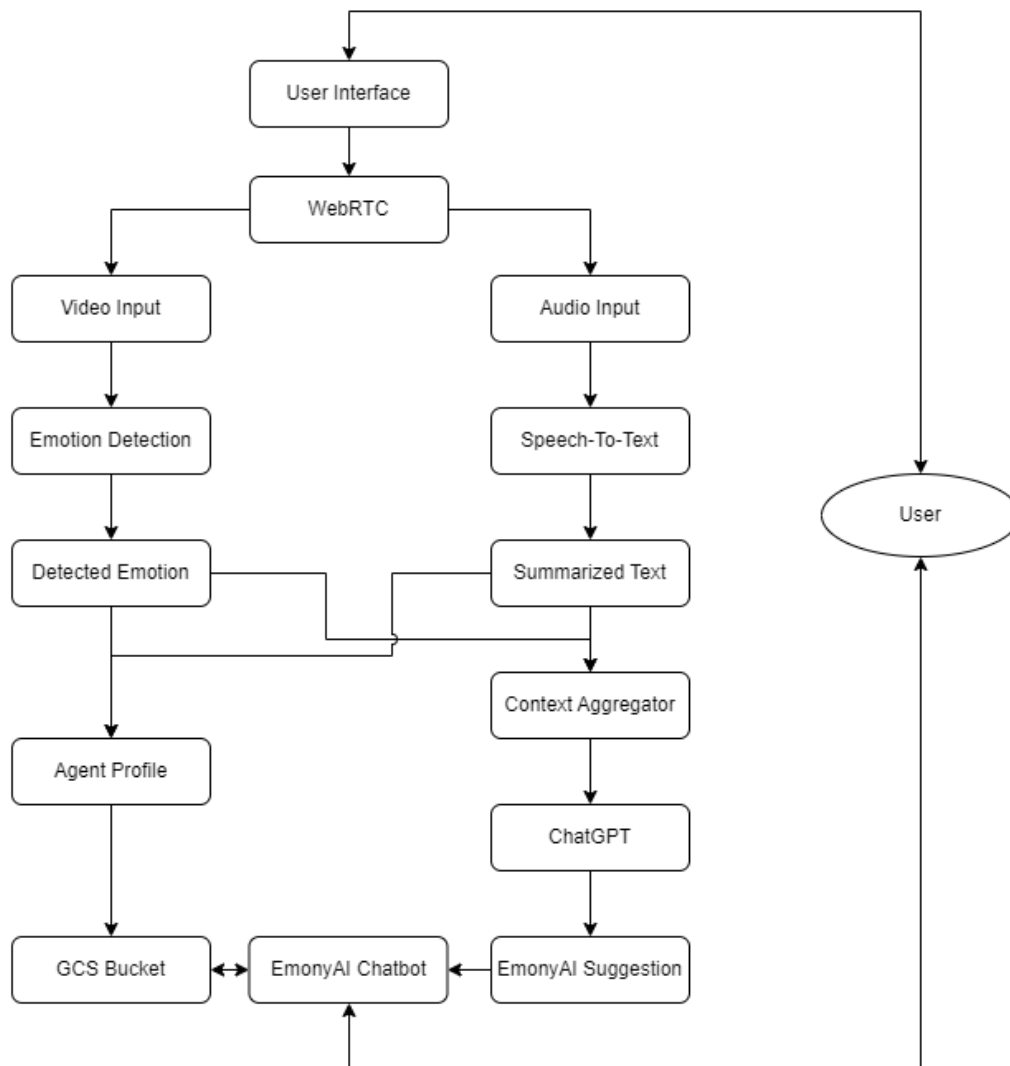


Figure 4.1.1 : System Block Diagram

The system block diagram of EmonyAI represents the interconnection between its core components, detailing how data flows through the system.

- Input Sources:
 - WebRTC Video/Audio Streams: Real-time input from the agent's camera and microphone is captured. The video stream is directed to the Emotion Detection Module for emotion detection, while the audio stream is processed by the Google Speech-to-Text Engine.

- Processing Components:

- Emotion Detection: This module processes the video stream and extracts facial landmarks to detect emotions, facial expressions, and body language. It provides the system with essential data to understand the user's emotional state.
- Speech-to-Text Engine: This component converts spoken audio into text, allowing the system to process the user's verbal inputs in written form. It is crucial for analyzing spoken words and interacting with the ChatGPT API for conversation generation.
- Output:
 - EmonyAI Suggestion: The summarized speech-to-text data, combined with emotional cues, is sent to the ChatGPT API, which generates conversation suggestions in real-time. The generated response is then output to the user through EmonyAI Chatbot.
 - User Feedback Loop: The chatbot presents the conversation suggestion to the user, and the entire process repeats, continuously refining the interaction based on real-time inputs.

4.2 System Components Specifications

Hardware Specifications:

1. User Device (Laptop):
 - Processor: AMD Ryzen 7 5800HS
 - GPU: NVIDIA GTX 1650
 - RAM: 16GB
 - Camera: 1080p resolution external webcam
 - Microphone: Built-in microphone for clear audio capture
2. Server/Cloud Requirements:
 - Google Cloud Run: Hosts the back-end services and handles system scaling.
 - Google Cloud Services Bucket Storage: Stores user profiles, interaction history, and data logs.

Software Specifications:

1. Operating System:
 - Windows 11 for local development and testing.
2. Development Environment:

- Vite React: Used for building the front-end user interface of the video calling platform.
 - VS Code: Primary Integrated Development Environment (IDE) for writing and debugging the application.
3. Real-time Processing Software:
- WebRTC: Used for real-time peer-to-peer video and audio streaming between users. Its low-latency nature ensures that data reaches the processing modules (Emotion Detection, Speech-to-Text) without delays.
 - Emotion Detection Module: Facial recognition and emotion detection software, using facial landmark models to detect user emotions in real-time.
 - Speech-to-Text Engine: Google’s Cloud Speech-to-Text API to convert user audio into written text, feeding into the chatbot.
4. Backend Services:
- MetaGPT Framework: Handles the user profile management and dynamic learning process, allowing continuous improvement of conversation suggestions based on real-time interactions.
 - ChatGPT API: Processes speech-to-text input to generate human-like conversation responses.

4.3 Circuits and Components Design

Since EmonyAI is primarily a software-based system, the circuit-level design focuses on the interaction between hardware (user device components like the camera and microphone) and the software modules.

Camera and Microphone Interaction:

1. Camera Circuit Design:
 - The camera captures video at 30 frames per second (fps) and sends the frames to the WebRTC Interface, which directly feeds them to the Emotion Detection Module. Emotion Detection Module processes each frame for facial landmarks and expressions.
 - Resolution Requirements: The system is optimized for 1080p camera resolutions, allowing Emotion Detection Module to accurately detect micro-expressions and movements essential for emotion analysis.
2. Microphone Circuit Design:

- The microphone captures audio in real-time and transmits it to the WebRTC Interface, which sends the raw audio data to the Speech-to-Text Engine.

GPU Acceleration:

The most resource-intensive processes—real-time facial recognition and emotion detection—are offloaded to the system's GPU. This is crucial for maintaining real-time performance, especially when analyzing multiple frames per second.

- **CUDA Acceleration:** NVIDIA's CUDA cores handle the deep learning models used for facial landmark detection, speeding up the frame-by-frame analysis required for emotion detection.

Communication Circuit Design:

- WebRTC establishes a real-time communication channel between the user's browser and the system's back-end servers.
- Video and Audio Data are processed concurrently, and results (emotion data and text) are transmitted back to the chatbot with minimal delay, ensuring seamless communication.

4.4 Emotion Detection Module

4.4.1 Building of the Emotion Detection Model in Jupyter Notebook

In this section, the development of the emotion detection model using a Python notebook is documented. The model is built using a Convolutional Neural Network (CNN) to classify facial expressions into various emotion categories. The notebook outlines the key stages, including data preprocessing, model architecture design, and training the model. By leveraging deep learning techniques, the model is trained to accurately recognize emotions based on facial images, making it an essential component for real-time emotion detection within the EmonyAI system. The notebook also provides insights into how the model's performance is evaluated and fine-tuned for optimal accuracy.

Importing Libraries

Bachelor of Computer Science (Honours)
Faculty of Information and Communication Technology (Kampar Campus), UTAR

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc, roc_auc_score

from IPython.display import clear_output
import warnings
warnings.filterwarnings('ignore')
✓ 1m 16.5s Python
```

Figure 4.4.1: Code snippet for importing libraries

Libraries:

- pandas: A library used for data manipulation and analysis, particularly for handling tabular data with DataFrames.
- numpy: A library for numerical operations, particularly for working with arrays and matrices.
- matplotlib.pyplot: A plotting library used for creating static, animated, and interactive visualizations in Python.
- seaborn: A data visualization library based on Matplotlib, providing a high-level interface for creating attractive and informative statistical graphics.
- plotly.express: A high-level Plotly interface used for quickly generating interactive plots.

TensorFlow and Keras Imports:

- tensorflow: An open-source machine learning library used for building deep learning models.
- ImageDataGenerator: A class from Keras for augmenting images in real-time during training, which helps in preventing overfitting by creating new image data variations.
- to_categorical: A utility function that converts class vectors (integers) to binary class matrices, essential for categorical cross-entropy loss during classification tasks.

Scikit-learn Imports:

- `confusion_matrix`: A function to compute a confusion matrix, a tool used for measuring the performance of classification models by comparing actual vs predicted labels.
- `classification_report`: Generates a detailed report showing precision, recall, F1-score, and support for each class.
- `LabelBinarizer`: Converts multi-class labels into a binary matrix form, which is useful for one-vs-rest classification.
- `roc_curve`: Plots the Receiver Operating Characteristic (ROC) curve, which shows the trade-off between true positive and false positive rates for different classification thresholds.
- `auc`: Computes the Area Under the Curve (AUC) for the ROC curve, a measure of the classifier's ability to distinguish between classes.
- `roc_auc_score`: Computes the overall ROC AUC score, a performance measure for binary classifiers.

Utility Function:

- `clear_output`: A utility from IPython used to clear the output of the current cell, typically useful in Jupyter notebooks to keep the notebook clean when rerunning code.
- `warnings.filterwarnings('ignore')`: This suppresses warnings in the code output. It is particularly useful in research notebooks to avoid cluttering the output with non-critical warning messages

Hyperparameters and Directories

```

train_dir = './dataset/train'
test_dir = './dataset/test'

SEED = 12
IMG_HEIGHT = 48
IMG_WIDTH = 48
BATCH_SIZE = 64
EPOCHS = 30
FINE_TUNING_EPOCHS = 20
LR = 0.01
NUM_CLASSES = 7
EARLY_STOPPING_CRITERIA=3
CLASS_LABELS = ['Anger', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sadness', "Surprise"]

```

Figure 4.4.2 : Code snippet for setting hyperparameter and directories

- `train_dir` and `test_dir`: These variables store the file paths to the training and testing datasets, respectively. In this case, the data is stored in separate directories within the

./dataset/ folder. The training directory contains images used to train the model, and the testing directory contains images used to evaluate the model's performance.

- **SEED:** A random seed is set to ensure reproducibility of results. By fixing the seed value, the data shuffling and model initialization can produce consistent results each time the code is executed.
- **IMG_HEIGHT and IMG_WIDTH:** These parameters define the dimensions (height and width) to which each image in the dataset will be resized. In this case, each image will be resized to 48x48 pixels. This ensures uniformity in the input data passed into the neural network, as models typically require images of the same size for training.
- **BATCH_SIZE:** This parameter determines the number of images that will be passed to the model at once during training or evaluation. A batch size of 64 means that 64 images will be processed together in each training iteration, allowing for more efficient model training.
- **EPOCHS:** This defines the total number of complete passes through the entire training dataset during model training. In this case, the model will undergo 30 epochs, allowing it to learn and improve iteratively by adjusting the weights with each pass through the data.
- **FINE_TUNING_EPOCHS:** This parameter is used if fine-tuning the model. Fine-tuning is the process of further training a pre-trained model on the specific task at hand. In this case, fine-tuning will occur for 20 additional epochs.
- **LR:** This is the learning rate, a crucial hyperparameter that controls how much the model's weights are adjusted with respect to the loss gradient during training. A learning rate of 0.01 indicates relatively fast learning, though care must be taken to avoid overshooting the optimal point.
- **NUM_CLASSES:** This specifies the number of output classes for the classification task. Here, it indicates that the model is classifying images into 7 distinct emotion categories.

- **EARLY_STOPPING_CRITERIA:** This defines the number of consecutive epochs without improvement after which the training process will stop early. Setting this value to 3 means that if the model's performance (e.g., validation loss) doesn't improve for 3 epochs, training will halt to prevent overfitting and save resources.
- **CLASS_LABELS:** This is a list that contains the names of the target classes for the classification task. In this case, the model is being trained to classify images into 7 emotions: Anger, Disgust, Fear, Happy, Neutral, Sadness, and Surprise.

Data Loading and Pre-Processing

```

preprocess_fun = tf.keras.applications.densenet.preprocess_input

train_datagen = ImageDataGenerator(horizontal_flip=True,
                                   width_shift_range=0.1,
                                   height_shift_range=0.05,
                                   rescale = 1./255,
                                   validation_split = 0.2,
                                   preprocessing_function=preprocess_fun
                                   )

test_datagen = ImageDataGenerator(rescale = 1./255,
                                  validation_split = 0.2,
                                  preprocessing_function=preprocess_fun)

train_generator = train_datagen.flow_from_directory(directory = train_dir,
                                                    target_size = (IMG_HEIGHT ,IMG_WIDTH),
                                                    batch_size = BATCH_SIZE,
                                                    shuffle = True ,
                                                    color_mode = "rgb",
                                                    class_mode = "categorical",
                                                    subset = "training",
                                                    seed = 12
                                                    )

validation_generator = test_datagen.flow_from_directory(directory = train_dir,
                                                        target_size = (IMG_HEIGHT ,IMG_WIDTH),
                                                        batch_size = BATCH_SIZE,
                                                        shuffle = True ,
                                                        color_mode = "rgb",
                                                        class_mode = "categorical",
                                                        subset = "validation",
                                                        seed = 12
                                                        )

test_generator = test_datagen.flow_from_directory(directory = test_dir,
                                                  target_size = (IMG_HEIGHT ,IMG_WIDTH),
                                                  batch_size = BATCH_SIZE,
                                                  shuffle = False ,
                                                  color_mode = "rgb",
                                                  class_mode = "categorical",
                                                  seed = 12
                                                  )

```

Figure 4.4.3 : Code snippet for data loading and Pre-processing

1. **preprocess_fun:** This function applies preprocessing specific to the DenseNet model architecture. It normalizes the pixel values to match the input format expected by DenseNet, which typically includes scaling and possibly adjusting the color channels.

This preprocessing ensures that the input data is consistent with how the pre-trained DenseNet model was trained.

2. `ImageDataGenerator`: A powerful utility for real-time data augmentation, which allows the model to generalize better by applying transformations to the input data. Here's a breakdown of each argument:
 - `horizontal_flip=True`: Randomly flips the images horizontally, which can help the model generalize better by learning from different perspectives of the same image.
 - `width_shift_range=0.1`: Randomly shifts the image horizontally by up to 10% of the total width.
 - `height_shift_range=0.05`: Randomly shifts the image vertically by up to 5% of the total height.
 - `rescale=1./255`: Scales the pixel values from a range of 0-255 to 0-1, which helps speed up convergence during training.
 - `validation_split=0.2`: Splits 20% of the training data for validation purposes, allowing the model to be evaluated on unseen data during training.
 - `preprocessing_function=preprocess_fun`: Applies the DenseNet-specific preprocessing function mentioned earlier.
3. `test_datagen`: Prepares the test data similarly to the training data but without augmentation (e.g., no horizontal flips or shifts). It only rescales the pixel values and applies the DenseNet preprocessing. This ensures the test data is standardized for evaluation purposes.
4. `flow_from_directory`: This function loads the images from the specified directory (`train_dir`) and applies the transformations defined earlier (such as augmentation and preprocessing). It generates batches of augmented and preprocessed image data for the model during training.
 - `directory=train_dir`: Points to the folder containing the training images.
 - `target_size=(IMG_HEIGHT, IMG_WIDTH)`: Resizes the images to the specified height and width (48x48 pixels in this case).
 - `batch_size=BATCH_SIZE`: Defines the batch size as 64, meaning 64 images are processed at a time.

- `shuffle=True`: Ensures that the training images are shuffled, which helps the model generalize better.
 - `color_mode="rgb"`: Indicates that the input images are RGB (3 color channels).
 - `class_mode="categorical"`: The labels are expected to be in categorical (one-hot encoded) format since this is a multi-class classification problem.
 - `subset="training"`: Uses the 80% subset of the data designated for training.
 - `seed=12`: Ensures that the data shuffling is reproducible by using the same random seed.
5. `validation_generator`: This generator is similar to the training generator but uses the 20% generator validation subset of the training data to monitor the model's performance during training. No data augmentation is applied to the validation data; only the preprocessing and rescaling are done.
6. `test_generator`: This generator is responsible for loading the test images from the `test_dir`. It resizes the images, applies necessary preprocessing and ensures no shuffling by setting `shuffle=False`, to keep the order intact for evaluation purpose. The test generator is used to evaluate the final performance of the model after training.

Displaying images with their respective titles

```

def display_one_image(image, title, subplot, color):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image)
    plt.title(title, fontsize=16)

def display_nine_images(images, titles, title_colors=None):
    subplot = 331
    plt.figure(figsize=(13,13))
    for i in range(9):
        color = 'black' if title_colors is None else title_colors[i]
        display_one_image(images[i], titles[i], 331+i, color)
    plt.tight_layout()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    plt.show()

def image_title(label, prediction):
    class_idx = np.argmax(label, axis=-1)
    prediction_idx = np.argmax(prediction, axis=-1)
    if class_idx == prediction_idx:
        return f'{CLASS_LABELS[prediction_idx]} [correct]', 'black'
    else:
        return f'{CLASS_LABELS[prediction_idx]} [incorrect, should be {CLASS_LABELS[class_idx]}]', 'red'

def get_titles(images, labels, model):
    predictions = model.predict(images)
    titles, colors = [], []
    for label, prediction in zip(classes, predictions):
        title, color = image_title(label, prediction)
        titles.append(title)
        colors.append(color)
    return titles, colors

img_datagen = ImageDataGenerator(rescale = 1./255)
img_generator = img_datagen.flow_from_directory(directory = train_dir,
                                               target_size = (IMG_HEIGHT, IMG_WIDTH),
                                               batch_size = BATCH_SIZE,
                                               shuffle = True,
                                               color_mode = "rgb",
                                               class_mode = "categorical",
                                               seed = 12
                                               )

clear_output()

images, classes = next(img_generator)
class_idxs = np.argmax(classes, axis=-1)
labels = [CLASS_LABELS[idx] for idx in class_idxs]
display_nine_images(images, labels)

```

Figure 4.4.4: Code snippet for displaying images with different emotions

(This part of coding is only used during testing in jupyter notebook to check whether the images are detected with correct emotions.)

1. `display_one_image(image, title, subplot, color)`: This function is used to display a single image with a given title in a specific subplot.
 - `image`: The image to be displayed.
 - `title`: The title to be shown above the image.
 - `subplot`: Defines where the image will be positioned within a grid of subplots (9 subplots in this case).
 - `color`: Though it is passed as a parameter, it is not used directly in this function. You can adjust this if you want to apply color to the title in future customization.
 - `plt.axis('off')`: Turns off the axis around the image for a cleaner visualization.
 - `plt.imshow(image)`: Displays the image.
 - `plt.title(title, fontsize=16)`: Displays the title above the image

2. `display_nine_images(images, titles, title_colors)`: This function displays a 3x3 grid of 9 images along with their titles.
 - `images`: The list of 9 images to display.
 - `titles`: The corresponding titles for each image.

- `title_colors`: Optional argument that allows specifying the color of each title. If not provided, it defaults to black.
 - `subplot = 331`: Refers to the initial position of the first image in a grid layout (3 rows, 3 columns). The index is incremented for each image.
 - `plt.figure(figsize=(13,13))`: Sets the figure size for the display.
 - `plt.tight_layout()` and `plt.subplots_adjust(wspace=0.1, hspace=0.1)`: These ensure the images and subplots are tightly packed and evenly spaced.
3. `image_title(label, prediction)`: This function generates the title and color for each image based on the model's prediction.
 - `label`: The true label (one-hot encoded) of the image.
 - `prediction`: The predicted class from the model.
 - `class_idx = np.argmax(label, axis=-1)`: Converts the one-hot encoded label into its corresponding class index.
 - `prediction_idx = np.argmax(prediction, axis=-1)`: Converts the model's prediction to the predicted class index.
 - `if class_idx == prediction_idx`: If the predicted class matches the true class, the function returns the label along with the color black (correct). Otherwise, it returns the predicted label and indicates the correct label with the color red (incorrect).
 4. `get_titles(images, labels, model)`: This function generates the titles and colors for a batch of images by using the `image_title` function.
 - `model.predict(images)`: Predicts the class of each image using the trained model.
 - `for label, prediction in zip(classes, predictions)`: Iterates over each true label and its corresponding prediction.
 - `titles.append(title)` and `colors.append(color)`: Appends the generated title and color for each image to the respective lists.
 - `return titles, colors`: Returns the list of titles and colors for the images.
 5. `img_datagen` and `img_generator`:
 - `img_datagen`: This `ImageDataGenerator` scales the pixel values of the images by dividing them by 255, ensuring the values are in the range [0, 1].

- `img_generator`: Loads and augments the images from the `train_dir` directory. It resizes the images to 48x48 pixels, shuffles the data, and loads it in batches of 64. The images are in RGB format, and the labels are one-hot encoded (categorical).
6. Get a batch of images and classes:
 - `next(img_generator)`: Fetches the next batch of images and corresponding one-hot encoded labels from the data generator.
 - `class_idxs = np.argmax(classes, axis=-1)`: Converts the one-hot encoded labels into their corresponding class indices.
 - `labels = [CLASS_LABELS[idx] for idx in class_idxs]`: Maps the class indices to the actual class labels using the predefined `CLASS_LABELS` list.
 7. `Display_nine_images` : displays a 3x3 grid of the first 9 images from the batch along with their corresponding labels.

Checking Data Distribution among different emotions

```

fig = px.bar(
    y = [list(train_generator.classes).count(i) for i in np.unique(train_generator.classes)],
    color = np.unique(train_generator.classes),
    color_continuous_scale="Emrld")
fig.update_xaxes(title="Emotions")
fig.update_yaxes(title="Number of Images")
fig.update_layout(showlegend=True,
    title = {
        'text': 'Train Data Distribution ',
        'y':0.95,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
fig.show()

```

Figure 4.4.5 : Checking data distribution among different emotions

1. `px.bar` function from Plotly Express creates a bar chart.:
 - `y`: The y-values represent the count of images for each emotion class. The list comprehension iterates through the unique class labels in the `train_generator` and counts how many images belong to each class.
 - `train_generator.classes`: Contains the class labels for all the images in the training dataset.
 - `np.unique(train_generator.classes)`: Extracts the unique class labels (emotion categories) present in the training dataset.
 - `list(train_generator.classes).count(i)`: Counts how many times each unique class label appears in the training data.
 - `color=np.unique(train_generator.classes)`: Assigns a unique color to each class in the bar chart. Each bar will have a color corresponding to the unique class.

- `color_continuous_scale="Emrld"`: Specifies the color scale to be used for the bars. "Emrld" is a color scheme from Plotly's collection that generates green-shaded continuous colors for the bars.
2. Updating the X and Y Axes:
 - `update_xaxes(title="Emotions")`: This updates the x-axis label to "Emotions", indicating that the x-axis represents the emotion classes.
 - `update_yaxes(title="Number of Images")`: This updates the y-axis label to "Number of Images", indicating that the y-axis represents the number of images belonging to each emotion class.
 3. Updating Layout and Title:
 - `showlegend=True`: Ensures that a legend is displayed on the chart, helping to identify which bar corresponds to which emotion class.
 - `title`: The title of the chart is set to "Train Data Distribution".
 - `'text': 'Train Data Distribution'`: The title text for the chart.
 - `'y': 0.95`: This positions the title vertically, near the top of the plot.
 - `'x': 0.5`: Centers the title horizontally on the plot.
 - `'xanchor': 'center'` and `'yanchor': 'top'`: These parameters ensure the title is anchored correctly in the center and at the top of the chart.
 4. `fig.show()` renders and displays the bar chart in the output. It visualizes the distribution of images in each emotion class within the training dataset.

DenseNet169 Transfer Learning

```
def (variable) feature_extractor: Any
feature_extractor = tf.keras.applications.DenseNet169(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3),
                                                    include_top=False,
                                                    weights="imagenet")(inputs)

return feature_extractor

def classifier(inputs):
x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
x = tf.keras.layers.Dense(256, activation="relu", kernel_regularizer = tf.keras.regularizers.l2(0.01))(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = tf.keras.layers.Dense(1024, activation="relu", kernel_regularizer = tf.keras.regularizers.l2(0.01))(x)
x = tf.keras.layers.Dropout(0.5)(x)
x = tf.keras.layers.Dense(512, activation="relu", kernel_regularizer = tf.keras.regularizers.l2(0.01))(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = tf.keras.layers.Dense(NUM_CLASSES, activation="softmax", name="classification")(x)

return x

def final_model(inputs):
densenet_feature_extractor = feature_extractor(inputs)
classification_output = classifier(densenet_feature_extractor)

return classification_output

def define_compile_model():

inputs = tf.keras.layers.Input(shape=(IMG_HEIGHT ,IMG_WIDTH,3))
classification_output = final_model(inputs)
model = tf.keras.Model(inputs=inputs, outputs = classification_output)

model.compile(optimizer=tf.keras.optimizers.SGD(0.1),
              loss='categorical_crossentropy',
              metrics = ['accuracy'])

return model
```

Figure 4.4.6 : Code Snippet for DenseNet169 Transfer Learning

1. feature_extractor(inputs) defines the feature feature extraction part of the model using a pre-trained DenseNet169 model.

- tf.keras.applications.DenseNet169: This is a pre-trained model from Keras that has been trained on the ImageNet dataset. It is used for feature extraction, meaning it will learn complex features from the input images (like edges, textures, and patterns) without the final classification layers.
- input_shape=(IMG_HEIGHT, IMG_WIDTH, 3): The input shape is set to the size of the images (48x48 pixels) with 3 color channels (RGB).
- include_top=False: This excludes the top (fully connected) layers of the DenseNet169 model because the classification task will be handled by custom layers later.
- weights="imagenet": Loads pre-trained weights from the ImageNet dataset, which allows the model to leverage knowledge from a large-scale image classification task.
- (inputs): Passes the input tensor into the pre-trained DenseNet169 model to extract features from the input images

2. classifier(inputs) defines the custom classification layers that will be added on top of the DenseNet feature extractor. These layers will make the final predictions.

- GlobalAveragePooling2D(): This layer reduces the spatial dimensions of the feature maps output by DenseNet to a single vector, effectively summarizing the features across the entire image.

- `Dense(256, activation="relu")`: Adds a fully connected (dense) layer with 256 neurons and ReLU activation, followed by L2 regularization (`kernel_regularizer=tf.keras.regularizers.l2(0.01)`) to prevent overfitting by penalizing large weights.
 - `Dropout(0.3)`: Introduces a dropout layer that randomly sets 30% of the neurons to zero during training to reduce overfitting.
 - `Dense(1024, activation="relu")`: A second fully connected layer with 1024 neurons and ReLU activation, followed by L2 regularization.
 - `Dropout(0.5)`: Another dropout layer with 50% dropout.
 - `Dense(512, activation="relu")`: A third fully connected layer with 512 neurons and ReLU activation, again with L2 regularization.
 - `Dropout(0.5)`: Another dropout layer with 50% dropout.
 - `Dense(NUM_CLASSES, activation="softmax")`: The final dense layer with `NUM_CLASSES` neurons (7 in this case for emotion classification) and softmax activation, which converts the output into a probability distribution across the classes
3. `final_model(inputs)` function combines the feature extractor(`DenseNet189`) and the custom classifier to create the full model.
- `densenet_feature_extractor`: Calls the `feature_extractor` function to extract features from the input images using `DenseNet169`.
 - `classification_output`: Calls the classifier function to generate predictions from the extracted features
4. `define_compile_model()` defines and compiles the final model:
- `Input(shape=(IMG_HEIGHT, IMG_WIDTH, 3))`: Creates an input layer with the specified input shape (48x48 pixels, 3 color channels).
 - `classification_output = final_model(inputs)`: Passes the input through the `final_model`, which includes both feature extraction and classification.
 - `model = tf.keras.Model(inputs=inputs, outputs=classification_output)`: Creates the final Keras model that connects the input layer to the output (emotion predictions).
 - `model.compile()`: Compiles the model with the following parameters:

- `optimizer=tf.keras.optimizers.SGD(0.1)`: Uses the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.1. SGD is a common optimizer used for training deep learning models.
- `loss='categorical_crossentropy'`: The loss function is categorical cross-entropy, which is typically used for multi-class classification problems.
- `metrics=['accuracy']`: The model will track accuracy during training and evaluation

Summary of Model

```

model = define_compile_model()
clear_output()

# Freezing the feature extraction layers
model.layers[1].trainable = False

model.summary()

```

Figure 4.4.7 : Code Snippet for Summary of Model

1. `model = define_compile_model()` creates the model by calling the `define_compile_model` function which builds the model using pre-trained DenseNet169, feature extractor and custom classification layers.
 - `clear_output()` : clears the output of the current cell in Jupyter Notebook.
2. `model.layers[1].trainable = False` freezes the DenseNet169 feature extraction layers by setting their trainable attribute to False
 - Layer Freezing: Freezing the feature extraction layers means that during training, the weights of these layers will not be updated. This is a common technique in transfer learning when the pre-trained model (DenseNet169 in this case) has already learned meaningful features from a large dataset (ImageNet).
 - Why Freeze? By freezing these layers, we allow the model to use the pre-learned features, while focusing the training process on the custom classification layers added on top. This reduces training time and prevents overfitting when working with smaller datasets.
3. `model_summary()` prints a summary of the model architecture that includes list of all layers, output shape of each layer, number of parameter of each layer and whether each layer is trainable or frozen.

Training Model with Freeze Layers

```
earlyStoppingCallback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                         patience=EARLY_STOPPING_CRITERIA,
                                                         verbose=1,
                                                         restore_best_weights=True
                                                         )

history = model.fit(x = train_generator,
                   epochs = EPOCHS,
                   validation_data = validation_generator,
                   callbacks= [earlyStoppingCallback])

history = pd.DataFrame(history.history)
```

✓ 100m 2.8s Python

Figure 4.4.8 : Code snippet for training model with freeze layers

1. Early Stopping Callback: `tf.keras.callbacks.EarlyStopping` stops the training early when the model's performance on the validation set stops improving. This helps avoid overfitting and unnecessary long training times.
 - `monitor='val_loss'`: This tells the callback to monitor the validation loss during training. When the validation loss stops improving, the early stopping mechanism will be triggered.
 - `patience=EARLY_STOPPING_CRITERIA`: The patience parameter defines how many epochs with no improvement to wait before stopping the training. The value is set using the previously defined variable `EARLY_STOPPING_CRITERIA` (in this case, it was set to 3). So, if the validation loss does not improve for 3 consecutive epochs, the training will stop.
 - `verbose=1`: This ensures that the early stopping callback prints information to the console when it stops the training.
 - `restore_best_weights=True`: Once early stopping is triggered, this parameter ensures that the model restores the weights from the epoch with the best validation performance, rather than keeping the weights from the final epoch.
2. Model Training: `model.fit()` function starts the training process for the model.
 - `x=train_generator`: The training data is provided by the `train_generator`, which is a data generator that loads and augments the images in batches during training.
 - `epochs=EPOCHS`: The number of epochs is set to the value of `EPOCHS` (30 in this case). An epoch refers to one complete pass through the entire training dataset.
 - `validation_data=validation_generator`: This specifies the validation data to evaluate the model's performance after each epoch. The `validation_generator` provides the validation images.

- `callbacks=[earlyStoppingCallback]`: This line specifies that the early stopping mechanism will be used during training to stop it if the model stops improving on the validation loss.
3. Converting History to Pandas DataFrame: `pd.DataFrame(history.history)` converts the history object into a Pandas DataFrame. The `history.history` attribute contains the training and validation metrics for each epoch (e.g., loss, accuracy, validation loss, and validation accuracy).
 - Converting the training history into a Pandas DataFrame makes it easy to visualize and analyze the model's performance across epochs, including plotting graphs or reviewing the metrics in a tabular format.

Fine-tuning

```
# Un-Freezing the feature extraction layers for fine tuning
model.layers[1].trainable = True

model.compile(optimizer=tf.keras.optimizers.SGD(0.001), #lower learning rate
              loss='categorical_crossentropy',
              metrics = ['accuracy'])

history_ = model.fit(x = train_generator, epochs = FINE_TUNING_EPOCHS ,validation_data = validation_generator)
history = history.append(pd.DataFrame(history_.history) , ignore_index=True)
```

Figure 4.4.9 : Code snippet for Fine Tuning the model

1. Unfreezing the Feature Extraction Layers for Fine-Tuning:
 - `model.layers[1].trainable = True`: This line unfreezes the previously frozen DenseNet169 feature extraction layers (layer 1 of the model). During the initial training phase, these layers were frozen to use pre-learned features from ImageNet without modifying them. Now, they are unfrozen to allow the model to fine-tune these feature extraction layers based on the specific dataset (e.g., emotion recognition images).
 - Fine-tuning: In fine-tuning, the weights of the pre-trained model are updated with a smaller learning rate so the model can adapt better to the task at hand without overfitting or destroying the pre-learned features.
2. Recompiling the Model with a Lower Learning Rate
 - `model.compile()`: After unfreezing the feature extraction layers, the model needs to be recompiled to apply these changes and adjust the learning rate for fine-tuning.
 - `optimizer=tf.keras.optimizers.SGD(0.001)`: The Stochastic Gradient Descent (SGD) optimizer is used again, but this time with a lower learning rate (0.001). A

lower learning rate during fine-tuning ensures smaller updates to the weights, which helps the model improve without making drastic changes to the pre-trained weights.

- `loss='categorical_crossentropy'`: The loss function remains categorical cross-entropy, as this is still a multi-class classification problem.
- `metrics=['accuracy']`: The model will track accuracy during fine-tuning, as in the previous training phase

3. Fine-Tuning the Model:

- `model.fit()`: The model is trained again, but this time with the feature extraction layers unfrozen for fine-tuning.
- `x=train_generator`: The training data generator is the same as before, supplying batches of training images.
- `epochs=FINE_TUNING_EPOCHS`: The number of epochs is set to `FINE_TUNING_EPOCHS` (20 in this case), which is typically smaller than the initial training phase. Fine-tuning does not require as many epochs as initial training since the model already has a good starting point from the earlier training.
- `validation_data=validation_generator`: The same validation data generator is used to monitor the model's performance on the validation set.

4. Concatenating the Training Histories:

- `pd.DataFrame(history_.history)`: Converts the new history of the fine-tuning phase into a Pandas DataFrame. This DataFrame contains metrics like training loss, validation loss, training accuracy, and validation accuracy for each epoch during fine-tuning.
- `pd.concat([pd.DataFrame(history_.history)], ignore_index=True)`: Concatenates the new fine-tuning history with the previous training history. This step merges the two training phases (initial training and fine-tuning) into a single DataFrame for easy analysis and visualization. The `ignore_index=True` ensures that the index is reset, so the combined history appears continuous.

4.4.2 Converting Emotion Detection Notebook into Python script

In the EmonyAI system, the emotion detection module initially developed in a Jupyter notebook was converted into a standalone Python script to enable seamless execution and testing using command-line arguments. This conversion facilitates easier integration, deployment, and testing in various modes, such as training and real-time emotion detection.

Conversion Process

1. Extraction of Core Components:

- The relevant code from the Jupyter notebook was extracted into a Python script.

This included:

- Data loading and preprocessing: Code that handles image loading, resizing, and normalization.
- Model architecture definition: The deep learning model used for emotion detection, built using Keras with TensorFlow backend.
- Training and evaluation logic: Code for compiling, training, and evaluating the model using training and validation datasets.
- Inference logic: Code that enables real-time emotion detection using the webcam.

2. Command-Line Arguments:

- To allow the script to run in different modes which are training or display, command-line arguments were added using Python's `argparse` module. The user can now specify the mode of operation via command-line inputs:
 - `--mode train`: Trains the model on the dataset.
 - `--mode display`: Activates the real-time emotion detection using the webcam.

3. Model Definition and Compilation:

- The model definition which previously written in the notebook, was transferred into the Python script. The model includes several convolutional layers to extract features from the input images, followed by dense layers for classification.
- In train mode, the script compiles the model and runs the training process using the provided dataset and saves the trained weights to a file (`model.h5`).

4. Real-Time Emotion Detection:

- In display mode, the script loads the pre-trained model weights and activates the webcam for real-time emotion detection. The webcam feed is processed frame by frame, and the model makes predictions on the emotions expressed in each frame. The predicted emotion is displayed in real-time on the video feed.
- This conversion allows the system to handle real-time predictions in a streamlined manner, improving the system's usability and functionality during video calls.

5. Integration with System:

- The conversion of the notebook to a Python script also allows for easier integration into the broader system architecture. The script can now be called from other parts of the system using simple commands, enabling emotion detection as a modular component within the EmonyAI system.

4.4.3 Publish the Python Script through FastAPI

After converting the emotion detection model into a Python script, the next step is to expose it as an API service using FastAPI. This allows external applications to interact with the emotion detection model over the web by sending HTTP requests to specific endpoints.

Steps for Publishing the Python Script through FastAPI:

1. Creating the FastAPI Application:

- The FastAPI framework was chosen for its speed and ease of use in building API services. The Python script containing the emotion detection model was integrated into a FastAPI application by creating endpoints that accept image data and return emotion predictions.
- The main endpoint `/predict` was created to handle POST requests, allowing users to send image data for emotion analysis. The API processes the image, runs it through the pre-trained model, and returns the predicted emotion in JSON format.

2. Deploying the FastAPI Application on Google Cloud:

- The FastAPI application was deployed to Google Cloud Run, a fully managed compute platform that automatically scales applications based on demand. Google Cloud Run was chosen due to its ability to handle serverless

deployments, making it ideal for hosting the FastAPI application without worrying about managing infrastructure.

- Docker was used to containerize the FastAPI application. A Dockerfile was created to define the environment in which the FastAPI application runs, ensuring consistency across development, testing, and production environments.

3. Steps to Deploy on Google Cloud Run:

- Containerization: The FastAPI application was packaged into a Docker container using the Dockerfile.
- Push to Container Registry: The Docker image was pushed to Google Cloud Container Registry.
- Deploy to Cloud Run: The image was deployed to Google Cloud Run, enabling it to run as a serverless API that scales automatically based on incoming traffic.

4. API Service Access:

- Once deployed, the FastAPI application was assigned a public URL by Google Cloud Run. This allows external applications to send requests to the emotion detection API.
- The FastAPI service automatically scales based on demand, ensuring optimal performance even under varying loads.

5. Security and Monitoring:

- To ensure the security of the API, Google Cloud Run was configured with authentication and access control mechanisms, limiting access to authorized users.
- Monitoring was set up using Google Cloud Monitoring and Cloud Logging to track API performance and log any issues or errors that occur during runtime.

4.4.4 Model Evaluation

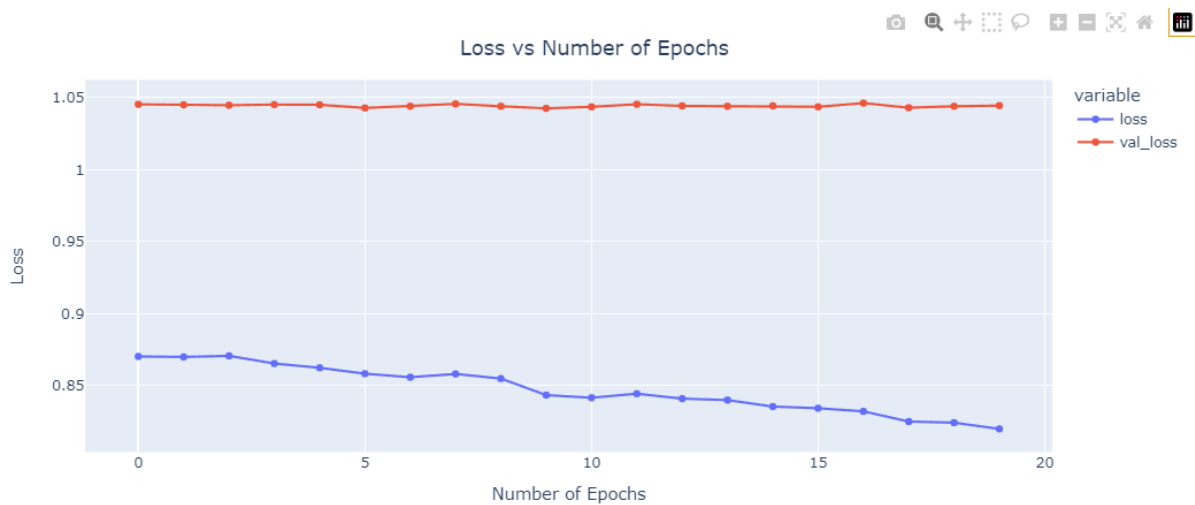


Figure 4.4.10 : Loss vs Number of Epochs

This plot shows the loss progression during training, which reflects how well the model is optimizing its internal parameters:

- Blue Line: Training loss decreases from 0.85 to approximately 0.80 over the 20 epochs, indicating steady improvement in reducing the model's error.
- Red Line: Validation loss remains flat around 1.05, which suggests that the model is not improving much on the validation set, possibly due to overfitting or issues with data complexity.

The loss function helps to measure the error in the model's predictions, where a lower loss implies better performance.

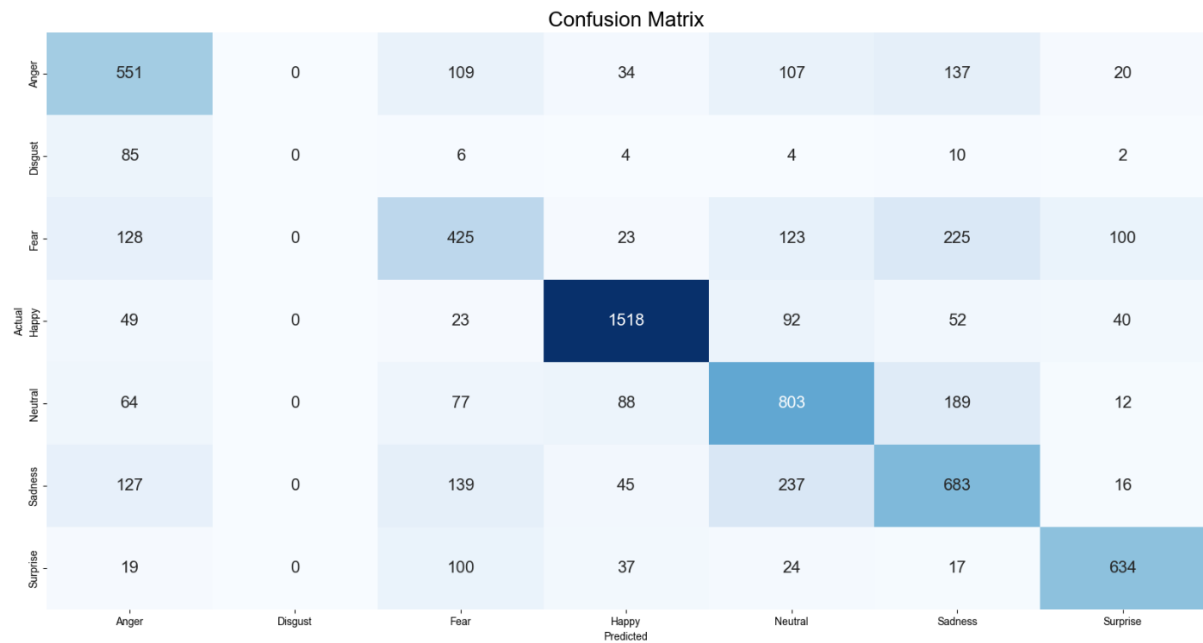


Figure 4.4.11 : Confusion Matrix

The confusion matrix provides insight into the classification performance of the model across different emotion categories. Each cell in the matrix represents the number of predictions made by the model compared to the actual label.

- The model performs best on "Happy" and "Surprise" classes, with 1518 and 634 correct predictions respectively.
- The model struggles the most with classes like "Fear" and "Neutral", showing higher misclassification rates.

This matrix is a helpful tool for visualizing how well the model is distinguishing between different emotion categories.

	precision	recall	f1-score	support
0	0.54	0.58	0.56	958
1	0.00	0.00	0.00	111
2	0.48	0.42	0.45	1024
3	0.87	0.86	0.86	1774
4	0.58	0.65	0.61	1233
5	0.52	0.55	0.53	1247
6	0.77	0.76	0.77	831
accuracy			0.64	7178
macro avg	0.54	0.54	0.54	7178
weighted avg	0.63	0.64	0.64	7178

Figure 4.1.12 : Classification Report

This classification report summarizes the model's precision, recall, and F1-score for each emotion class:

- The "Happy" class has the highest precision and recall scores, indicating strong performance in detecting happiness.
- Classes like "Disgust" have poor precision and recall, which indicates that the model is not able to accurately detect these emotions.

The overall accuracy of 0.64 shows that the model performs reasonably but struggles with certain emotions.

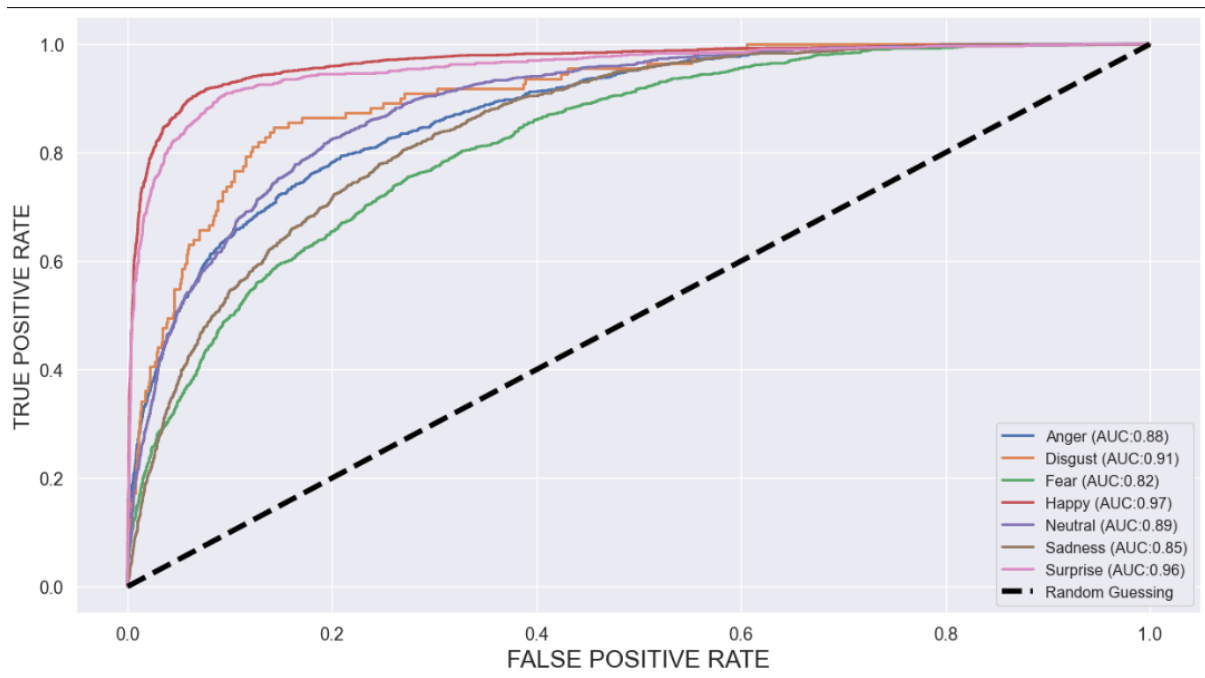


Figure 4.1.13 : ROC-AUC Curve

The Receiver Operating Characteristic (ROC) curve provides the true positive rate versus the false positive rate for each class, alongside the Area Under the Curve (AUC) score:

- The AUC values for the classes range from 0.82 (Fear) to 0.97 (Happy), with the "Happy" and "Surprise" classes performing the best.
- The AUC measures the model's ability to distinguish between classes, where values closer to 1 indicate a stronger performance.

The ROC-AUC score of 0.89 overall reflects that the model has a high discriminatory power between most of the emotion classes.

4.5 Webpage design

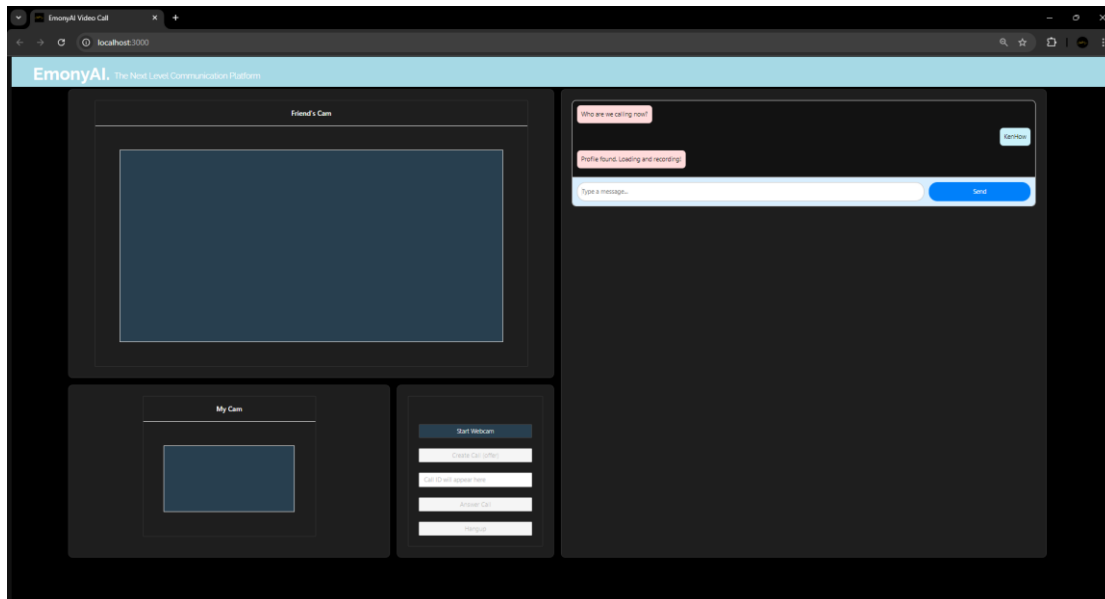


Figure 4.5.1 : User interface of EmonyAI website

4.5.1 Technology Stacks

1. Frontend

- **React.js:** A JavaScript library for building the user interface. React allows the dynamic rendering of components like video streams and chatbots, making the app interactive and responsive.
- **Ant Design:** A UI library providing pre-built components such as Cards, Buttons, and Layout grids, making it easier to create a professional-looking interface.
- **WebRTC:** A protocol for real-time communication used to facilitate video and audio streaming between users without needing additional plugins.
- **Axios:** A library for making HTTP requests from the frontend to the backend, especially for checking and creating user profiles.

2. Backend

- **Express.js:** A web framework for Node.js used to create RESTful APIs. It is responsible for handling HTTP requests from the frontend and interacting with Google Cloud Storage for profile management.
- **Google Cloud Storage API:** An API used to store and retrieve agent profiles in the cloud. The profiles are created, updated, and managed remotely to allow persistent storage.

- **Firestore:** Used for signaling in WebRTC, helping to set up peer-to-peer video calls by storing call offers and answers.
- **Python:** Used in the backend to define the structure of agent profiles and handle updates to those profiles during video calls.

3. Cloud

- **Google Cloud Storage:** The cloud storage service where agent profiles are stored. It ensures that user profiles are persistent and available across multiple sessions.
- **Google Cloud Service Accounts:** These service accounts authenticate the backend to Google Cloud, allowing it to securely manage cloud resources such as agent profiles.

4.5.2 Frontend Components

The frontend components are responsible for creating the user interface and managing the real-time video calls and interactions with the chatbot. Each component plays a crucial role in delivering a seamless user experience.

- **index.html:**
 - This file is the entry point for the web application and links to the external assets required for proper styling and functionality.
 - The basic HTML file where the React app is rendered. It loads necessary external resources like Ant Design CSS and Google Fonts to ensure a consistent design. The div with id="root" is where the entire React app is injected into the DOM(index).

```

frontend > index.html > html > head > title
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <link rel="icon" type="image/svg+xml" href="../assets/logo.jpg" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>EmonyAI Video Call</title>
8     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/antd@4.16.13/antd.min.css" />
9     <link href="https://fonts.googleapis.com/css2?family=Raleway:wght@400;700&display=swap" rel="stylesheet">
10
11   </head>
12   <body>
13     <div id="root"></div>
14     <script type="module" src="src/main.jsx"></script>
15
16   </body>
17 </html>
18

```

Figure 4.5.2 : Code snippet for index.html

- **main.jsx:**

- This file acts as the starting point of the frontend and bootstraps the React application by rendering the App component.
- The main JavaScript file that initializes the React application. It renders the App component, which is the root component containing all other components. It also links global styles from style.css(main).

```
frontend > src > main.jsx > ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import App from './App';
4 import './style.css'; |
5
6 const root = ReactDOM.createRoot(document.getElementById('root'));
7 root.render(<App />);
8
```

Figure 4.5.3 : Code snippet for main.jsx

- **App.jsx:**

- This is the most crucial part of the frontend, as it ties together all components (video call, chatbot, etc.) and handles the core logic of the application.
- This is the core component of the frontend. It manages the WebRTC connection for video calls, handles the local and remote video streams, and integrates with Firebase to store and retrieve call details. It also includes the chatbot interface, managing interactions with the backend for profile checking and creation(App).

```
frontend > src > App.jsx > App > handleCreateCall
1 import React, { useState, useEffect } from 'react';
2 import MyCam from './Components/mycam.jsx';
3 import FriendCam from './Components/friendcam.jsx';
4 import CallSetting from './Components/callsetting.jsx';
5 import Chatbot from './Components/chatbot.jsx';
6 import { Row, Col, Layout, Card } from 'antd';
7 import { initializeApp } from 'firebase/app';
8 import { getFirestore, collection, doc, setDoc, getDoc } from 'firebase/firestore';
9
10 const { Header, Content } = Layout;
11
12 // Firebase configuration
13 const firebaseConfig = {
14   apiKey: "AIzaSyBbqLPkEXmZP9bqURQOM0j7R82mh48VrFE",
15   authDomain: "emonyai-6103e.firebaseio.com",
16   projectId: "emonyai-6103e",
17   storageBucket: "emonyai-6103e.appspot.com",
18   messagingSenderId: "774593043226",
19   appId: "1:774593043226:web:d6b09b695340cd022d6aa3",
20   measurementId: "G-KXND9DXM60"
21 };
22
23 // Initialize Firebase and Firestore
24 const app = initializeApp(firebaseConfig);
25 const firestore = getFirestore(app);
26
27 const App = () => {
28   const [localStream, setLocalStream] = useState(null);
29   const [remoteStream, setRemoteStream] = useState(null);
30   const [callId, setCallId] = useState('');
31   const [isCallDisabled, setIsCallDisabled] = useState(true);
32   const [isAnswerDisabled, setIsAnswerDisabled] = useState(true);
33   const [isHangupDisabled, setIsHangupDisabled] = useState(true);
34   const [pc] = useState(new RTCPeerConnection({
35     iceServers: [
36       { urls: ['stun:stun1.l.google.com:19302', 'stun:stun2.l.google.com:19302'] }
37     ],
38     iceCandidatePoolSize: 10,
39   }));
```

Figure 4.5.4 : Code snippet for App.jsx

- **MyCam.jsx:**

- It allows the user to see their own video feed and is styled to ensure it looks consistent and responsive.
- This component displays the local user's video feed. It uses an Ant Design Card to encapsulate the video stream and applies flexbox for centering the content inside the card(mycam).

```
frontend > src > Components > mycam.jsx > MyCam > height
1 import React from 'react';
2 import { Card } from 'antd';
3
4 const MyCam = ({ stream }) => {
5   return (
6     <Card>
7       <span style={{ color: 'white' }}>My Cam</span>
8       style={{ width: '100%', maxWidth: 400, margin: '0 auto' }}
9       bodyStyle={{ display: 'flex', justifyContent: 'center', alignItems: 'center', height: '80%' }} // Flexbox centering
10      >
11      <video id="webcamVideo" style={{ width: '100%', height: '25%' }} autoPlay playsInline</video>
12    </Card>
13  );
14 };
15
16 export default MyCam;
17
```

Figure 4.5.5 : Code snippet for MyCam.jsx

- **FriendCam.jsx:**

- This component is responsible for displaying the incoming video stream from the remote user during a video call.
- Similar to MyCam, this component displays the video stream of the remote user (the person on the other side of the call). It is styled to be larger, allowing the user to see their friend or colleague more prominently(friendcam).

```
frontend > src > Components > friendcam.jsx > FriendCam > height
1 import React from 'react';
2 import { Card } from 'antd';
3
4 const FriendCam = ({ stream }) => {
5   return (
6     <Card>
7       <span style={{ color: 'white' }}>Friend's Cam</span>
8       style={{ width: '100%', maxWidth: 1000, margin: '0 auto' }}
9       bodyStyle={{ display: 'flex', justifyContent: 'center', alignItems: 'center', height: '100%' }} // Flexbox centering
10      >
11      <video id="remoteVideo" style={{ width: '100%', height: '50%' }} autoPlay playsInline</video>
12    </Card>
13  );
14 };
15
16 export default FriendCam;
17
```

Figure 4.5.6 : Code snippet for FriendCam.jsx

- **CallSetting.jsx:**

- This component handles all the video call-related actions, making it easy for users to control the call directly from the UI.
- This component provides the controls for the video call. It contains buttons for starting the webcam, creating a call, answering a call, and hanging up. Each button is styled for full-width display and placed inside a Card for consistency (callsetting).

```
frontend > src > Components > callsetting.jsx > CallSetting
1 import React from 'react';
2 import { Card, Button, Input, Space } from 'antd';
3
4 const CallSetting = ({
5   onStartWebcam,
6   onCreateCall,
7   callId,
8   onAnswerCall,
9   onHangup,
10  isCallDisabled,
11  isAnswerDisabled,
12  isHangupDisabled,
13 }) => {
14   return (
15     <Card style={{ width: '100%', maxWidth: 400, margin: '0 auto' }}>
16       <Space direction="vertical" size="large" style={{ width: '100%' }}> /*For the space to takes up full width */
17         <Button onClick={onStartWebcam} style={{ width: '100%' }}>
18           Start Webcam
19         </Button>
20         <Button onClick={onCreateCall} disabled={isCallDisabled} style={{ width: '100%' }}>
21           Create Call (offer)
22         </Button>
23         <Input value={callId} readOnly placeholder="Call ID will appear here" style={{ width: '100%' }} />
24         <Button onClick={onAnswerCall} disabled={isAnswerDisabled} style={{ width: '100%' }}>
25           Answer Call
26         </Button>
27         <Button onClick={onHangup} disabled={isHangupDisabled} style={{ width: '100%' }}>
28           Hangup
29         </Button>
30       </Space>
31     </Card>
32   );
33 };
34
35 export default CallSetting;
36
```

Figure 4.5.7 : Code snippet for CallSetting.jsx

- **chatbot.jsx:**

- The chatbot adds an AI-driven interactive element to the video calling platform, allowing for dynamic user experiences.
- This is the chatbot component, responsible for interacting with the backend to check if a user profile exists and create one if necessary. It allows users to chat with the bot, and the conversation is displayed in the message area. The chatbot plays an important role in enhancing user interaction by assisting with profile creation and checking(chatbot).

```
frontend > src > Components > chatbot.jsx > ...
1 import React, { useState, useEffect } from 'react';
2 import axios from 'axios';
3 import './Chatbot.css';
4
5 const Chatbot = ({ friendVideoRef }) => {
6   const [messages, setMessages] = useState([{ text: 'Who are we calling now?', sender: 'bot' }]);
7   const [input, setInput] = useState('');
8   const [name, setName] = useState('');
9   const [profileLoaded, setProfileLoaded] = useState(false);
10
11   // Function to send messages
12   const sendMessage = async (message) => {
13     setMessages([...messages, { text: message, sender: 'user' }]);
14     if (!name) {
15       setName(message); // Set the name for future use
16       await handleProfileCheck(message); // Check if profile exists
17     }
18   };
19
20   // Handle form submission
21   const handleSubmit = (e) => {
22     e.preventDefault();
23     if (input.trim()) {
24       sendMessage(input); // Send the message
25       setInput(''); // Clear the input field after submission
26     }
27   };
28
29   // Check if profile exists
30   const handleProfileCheck = async (name) => {
31     try {
32       const response = await axios.post('http://localhost:5000/check-profile', { name });
33       if (response.data.message === 'Profile is found') {
34         setMessages((prevMessages) => [
35           ...prevMessages,
36           { text: 'Profile found. Loading and recording!', sender: 'bot' },
37         ]);
38         await loadProfile(name); // Load profile if found
39       } else {
```

Figure 4.5.8 : Code snippet for chatbot.jsx

- **style.css:**
 - The global styles create a consistent look and feel across the entire application, ensuring a professional and user-friendly UI.
 - This file defines the global styles for the application, including the dark theme (black background with light-colored text), video element styling, and customization of Ant Design components(style).

```

frontend > src > # style.css > ...
1  @import url('https://fonts.googleapis.com/css2?family=Syne+Mono&display=swap');
2
3  body {
4      font-family: 'Syne Mono', monospace;
5      -webkit-font-smoothing: antialiased;
6      -moz-osx-font-smoothing: grayscale;
7      text-align: center;
8      color: #e0e0e0; /* Light text color to contrast with the dark background */
9      background-color: #121212; /* Dark background */
10     margin: 1px 10px;
11 }
12
13 body, html {
14     overflow-x: hidden;
15 }
16
17
18 video {
19     width: 40vw;
20     height: 30vw;
21     margin: 2rem;
22     background: #2c3e50;
23     border: 2px solid #e0e0e0; /* Light border to make videos stand out */
24 }
25
26 .videos {
27     display: flex;
28     align-items: center;
29     justify-content: center;
30 }
31
32 .ant-card {
33     background-color: #1e1e1e !important; /* Dark background for Ant Design Card */
34     color: #e0e0e0; /* Light text for readability */
35     border-color: #333333;
36 }
37
38 .ant-btn {
39     background-color: #2c3e50;

```

Figure 4.5.9 : Code snippet for style.css

- **Chatbot.css:**

- This CSS file ensures that the chatbot has a clean and modern design, enhancing user interaction with the bot.
- Contains specific styles for the chatbot interface, including message display, input fields, and buttons. The chatbot layout is designed to be responsive and flexible using flexbox(Chatbot).

```
frontend > src > Components > # Chatbot.css > .user-message
1  .chatbot-container {
2      flex-grow: 1;
3      width: 100%;
4      height: 100%;
5      display: flex;
6      flex-direction: column;
7      border: 1px solid #ccc;
8      border-radius: 10px;
9      background-color: #121212;
10     overflow: hidden;
11 }
12
13 .chatbot-messages {
14     flex-grow: 1; /* Ensure messages take up the remaining space */
15     padding: 10px;
16     overflow-y: auto; /* Allow scrolling when messages overflow */
17     display: flex;
18     flex-direction: column;
19     justify-content: flex-start;
20 }
21
22 .chatbot-message {
23     margin-bottom: 10px;
24     padding: 10px;
25     border-radius: 8px;
26     max-width: 80%;
27     word-wrap: break-word;
28 }
29
30 .user-message {
31     align-self: flex-end;
32     background-color: #c93025;
33     color: #121212;
34 }
35
36 .bot-message {
37     align-self: flex-start;
38     background-color: #ffdbdb;
39     color: #121212;
```

Figure 4.5.10 : Code snippet for chatbot.css

4.5.3 Backend Components

- **server.js:**
 - Manages all profile-related requests and acts as the communication bridge between the frontend, Google Cloud Storage, and various AI services. It enables profile management while also integrating emotion detection, speech-to-text conversion, and conversational AI.
 - This is the main backend server built using Express.js. It manages requests from the frontend, particularly for video call interactions and chatbot integration. It has multiple API routes, including:
 - /check-profile: Checks whether a profile exists in Google Cloud Storage by looking for the corresponding file.
 - /create-profile: Creates a new profile by uploading a Python template to Google Cloud Storage if the profile doesn't already exist.
 - Emotion Detection API: The server will integrate the emotion detection API to analyze emotions from video feeds during calls, enhancing user interaction by detecting and responding to emotional cues in real time.
 - Google Speech-to-Text API: The server will use this API to convert speech from video calls into text, enabling further analysis and processing of the conversation.
 - ChatGPT API: This API will be integrated to generate responses based on the transcribed text from the conversation, making the chatbot more interactive and context-aware.

```

backend > JS server.js > app.post('/check-profile) callback
1 import express from 'express';
2 import { Storage } from '@google-cloud/storage';
3 import fs from 'fs';
4 import path from 'path';
5 import bodyParser from 'body-parser';
6 import cors from 'cors';
7
8 const app = express();
9 app.use(bodyParser.json());
10 app.use(cors());
11
12 const storage = new Storage({ keyFilename: 'emomy_backend_service_key.json' });
13 const bucketName = 'emomy_agent_profile';
14
15 // Check if profile exists
16 app.post('/check-profile', async (req, res) => {
17   const { name } = req.body;
18   const fileName = `${name}.py`;
19
20   try {
21     const [exists] = await storage.bucket(bucketName).file(fileName).exists();
22     if (exists) {
23       res.json({ message: 'Profile is found' });
24     } else {
25       res.json({ message: 'Creating new profile' });
26     }
27   } catch (error) {
28     console.error('Error checking profile:', error);
29     res.status(500).json({ error: 'Error checking profile' });
30   }
31 });
32
33 // Create profile if it doesn't exist
34 app.post('/create-profile', async (req, res) => {
35   const { name } = req.body;
36   const fileName = `${name}.py`;
37
38   try {
39     const filePath = path.resolve('sample.py'); // Path to the template

```

Figure 4.5.11 : Code snippet for server.js

testGoogleCloud.js:

This is a script for testing the upload functionality to Google Cloud Storage. It uploads a sample.py file to the cloud bucket and sets the filename. It is useful for ensuring that the storage integration is working as expected. It tests whether the backend can successfully upload files to Google Cloud, which is essential for managing agent profiles.

```

backend > JS testGoogleCloud.js > uploadFile
1 import { Storage } from '@google-cloud/storage';
2 import path from 'path';
3
4 async function uploadFile() {
5   const storage = new Storage({ keyFilename: 'emomy_backend_service_key.json' });
6   const bucketName = 'emomy_agent_profile';
7   const sourceFilePath = path.join(process.cwd(), 'sample.py');
8   const destinationFileName = 'Rina.py';
9
10   try {
11     // Upload file from local to GCS bucket
12     await storage.bucket(bucketName).upload(sourceFilePath, {
13       destination: destinationFileName,
14     });
15     console.log(`${destinationFileName} created in bucket ${bucketName} by copying content from sample.py.`);
16   } catch (error) {
17     console.error('Error uploading file:', error);
18   }
19 }
20
21 uploadFile();
22

```

Figure 4.5.12 : Code snippet for testGoogleCloud.js

sample.py:

A Python template that acts as a blueprint for agent profiles. It defines the structure for storing agent information and call history. The file includes a function to add new entries to the history, such as call summaries and detected emotions. It is used to create and update agent profiles, storing important information like call history and emotions detected during the call.

emony_backend_service_key.json & emonyai-6103e-8e4d94f3f5e2.json:

These files contain Google Cloud service account credentials, which are used to authenticate the backend server with Google Cloud. They allow secure access to Google Cloud Storage for managing agent profiles. These credentials ensure that the backend can securely interact with Google Cloud Storage and manage user profiles.

```
backend > {} emony_backend_service_key.json > client_email
1  {
2    "type": "service_account",
3    "project_id": "emonyai-6103e",
4    "private_key_id": "2ef58468263d7ae5a800d16c4eddddec2ee7e88d2",
5    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEVgIBADANBgkqhkiG9w0BAQEFAASCBCkgwgSkAgEAAoIBAQDqXYezgOfZ1FV1\nnDdmfAYYY
6    "client_email": "emony-backend-service@emonyai-6103e.iam.gserviceaccount.com",
7    "client_id": "107440790440339881206",
8    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
9    "token_uri": "https://oauth2.googleapis.com/token",
10   "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
11   "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/emony-backend-service%40emonyai-6103e.iam.gs
12   "universe_domain": "googleapis.com"
13 }
14
```

Figure 4.5.13 : emony_backend_service_key.json

```
backend > {} emonyai-6103e-8e4d94f3f5e2.json > ...
1  {
2    "type": "service_account",
3    "project_id": "emonyai-6103e",
4    "private_key_id": "8e4d94f3f5e2a7c2d31c8c8e333422a61e57e3db",
5    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEVQIBADANBgkqhkiG9w0BAQEFAASCBCkgwgSjAgEAAoIBAQDhTicFt9Ic8jcu\nnAPYxAJSqmqEEh1ZuGz
6    "client_email": "agent-profile-viewer@emonyai-6103e.iam.gserviceaccount.com",
7    "client_id": "102084165216476623332",
8    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
9    "token_uri": "https://oauth2.googleapis.com/token",
10   "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
11   "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/agent-profile-viewer%40emonyai-6103e.iam.gserviceaccou
12   "universe_domain": "googleapis.com"
13 }
14
```

Figure 4.5.14 : emonyai-6103e-8e4d94f3f5e2.json

4.5.4 Core Features

1. Video Calling:
 - The website enables real-time video calling using WebRTC. Users can view their own camera feed in MyCam.jsx and the remote user's feed in FriendCam.jsx. The call controls are managed by the CallSetting component, allowing users to start, create, answer, and end calls with ease.
 - Video calling is the primary feature of the system, allowing users to communicate in real time with high-quality video and audio.
2. Chatbot Integration:
 - The chatbot component is integrated into the system to provide AI-driven conversations. It helps users check if their profile exists in the cloud, creates new profiles if needed, and interacts with them through a user-friendly interface.
 - The chatbot enhances the video calling experience by allowing users to manage profiles and engage with AI during their interactions.
3. Google Cloud Storage Integration:
 - Agent profiles are stored and retrieved from Google Cloud Storage. This integration allows the system to maintain user profiles across sessions, storing details such as call history, summaries, and detected emotions.

4.5.5 Deploying on Google Cloud Run

The deployment process for the EmonyAI project involves setting up both the frontend and backend components to run on Google Cloud Run, a fully managed serverless platform that allows the application to scale automatically based on demand. Below are the key steps and considerations involved in deploying the project on Google Cloud Run:

1. Preparing the Application for Deployment

Both the frontend and backend were prepared for deployment by ensuring the following:

 - Docker Containerization:
 - Google Cloud Run requires applications to be containerized. For this project, Docker was used to package the frontend and backend into separate containers. The Dockerfile for each component specifies the dependencies, such as Node.js

for the frontend and Express.js backend, and ensures the application can run smoothly in a cloud environment.

```
Dockerfile > ...
1 # Use an official Node.js runtime as a parent image
2 FROM node:16
3
4 # Set working directory for backend
5 WORKDIR /usr/src/backend
6
7 # Copy package.json and install dependencies
8 COPY backend/package.json backend/package-lock.json ./
9 RUN npm install
10
11 # Copy backend source code
12 COPY backend .
13
14 # Expose the port the server runs on
15 EXPOSE 8080
16
17 # Start the server
18 CMD ["node", "server.js"]
19
```

Figure 4.5.15 : Code snippet for Dockerfile

- Environment Configuration:
 - Sensitive data such as Google Cloud service keys (emony_backend_service_key.json) and the project configuration for Firebase were securely managed using environment variables. These keys were not hardcoded into the source code but were instead injected during the build and deployment stages to ensure security.

2. Frontend Deployment on Google Cloud Run

- Frontend (React App):
 - The frontend, which includes components such as the video call interface (App.jsx, MyCam.jsx, FriendCam.jsx) and chatbot (Chatbot.jsx), was built into a static bundle using `npm run build`. This bundle was then served using a lightweight web server (e.g., `serve` or `http-server`) within a Docker container.

- Once the Docker image for the frontend was ready, it was pushed to Google Container Registry and deployed on Google Cloud Run. The deployment configuration ensured that the service was accessible publicly via a HTTPS endpoint, allowing users to access the video call platform through their browsers.

3. Backend Deployment on Google Cloud Run

- Backend (Express.js Server):
 - The backend, which interacts with Google Cloud Storage to manage agent profiles and calls APIs such as the Emotion Detection API, Google Speech-to-Text API, and ChatGPT API, was also containerized using Docker. The Express.js server was configured to listen for incoming HTTP requests and interact with Google Cloud Storage via the service account keys (emony_backend_service_key.json).
 - Similar to the frontend, the backend Docker image was pushed to Google Container Registry and deployed to Google Cloud Run. The service was set up with the appropriate authentication to securely access Google Cloud Storage and external APIs.

4. Integration and Scalability

- Google Cloud Run Service Configuration:
 - Both the frontend and backend services were deployed as separate instances on Google Cloud Run. The platform automatically scales the services based on traffic, ensuring that the application can handle increased demand without requiring manual intervention.
 - Google Cloud Run also manages resource allocation, so the services only consume resources when they are actively handling requests, making it a cost-efficient solution.
- CORS and Security:
 - Since the frontend and backend are hosted on different Google Cloud Run instances, Cross-Origin Resource Sharing (CORS) was configured on the backend to allow the frontend to make API requests securely. Additionally,

HTTPS was enforced to ensure secure communication between the client and server.

- Environment Variables:
 - Environment variables were configured in Google Cloud Run for each service to securely manage sensitive data, such as API keys and database credentials, which were required to access Google Cloud services and external APIs.

5. Monitoring and Logging

- Google Cloud Logging:
 - After deployment, Google Cloud Logging was used to monitor the services. Logs from both the frontend and backend were collected and viewed in the Google Cloud Console, allowing for real-time debugging and performance tracking.
- Google Cloud Monitoring:
 - Google Cloud Monitoring was also integrated to track the performance of the deployed services, providing metrics such as request latency, error rates, and resource usage. Alerts were configured to notify when certain thresholds were exceeded.

4.6 Multi-agent framework

The MetaGPT framework is a key component of the EmonyAI system, designed to manage and utilize agent profiles and facilitate multi-agent collaboration when needed. In this system, each "agent" refers to a person that the user may communicate with through the video calling platform. The framework serves several functions, primarily centered around storing agent information and orchestrating interactions between agents when required.

Agent Profile Management

The MetaGPT framework is responsible for creating and managing the agent profiles. Each agent profile contains detailed information about the individual, such as their preferences, conversation history, and other personalized data. This profile allows the system to store relevant information about every person that the user might call, enhancing future interactions. During a call, the EmonyAI Chatbot uses these agent profiles to provide context-aware suggestions. For instance, the chatbot can retrieve and display the agent's preferences or other relevant information through the chatbot textbox, helping to guide the conversation and ensure smoother communication. This real-time access to agent-specific data helps personalize the user experience, making conversations more engaging and effective.

Orchestration of Multi-Agent Communication

The MetaGPT framework also has the capability to orchestrate interactions between two agent profiles. When prompted by the user using keyword "imagine", the framework can simulate or facilitate communication between two individuals (agents) chosen by the user to test how well they might work together in a team. This feature allows users to assess potential group dynamics and determine if the individuals are compatible in terms of collaboration, communication styles, or other work-related preferences.

By leveraging data from the agent profiles, MetaGPT ensures that the orchestrated communication reflects realistic interactions between the individuals. It can take into account the participants' behavioral patterns and preferences to provide insights into how they might function as a team. This functionality is particularly valuable in team-building or collaborative environments, where understanding how individuals interact is crucial for forming effective teams.

The MetaGPT framework plays a critical role in the EmonyAI system by managing agent profiles and orchestrating multi-agent interactions. It ensures that the chatbot can offer real-time, contextual suggestions during calls by drawing on stored agent information. Additionally, when prompted, it can facilitate team dynamics tests between agents, helping users evaluate potential collaboration and teamwork. This multi-agent framework adds a layer of intelligence and adaptability to the system, enhancing the overall communication experience.

```

sample.py X
D:\> Document > Projects > backup > backend > sample.py > ...
1 # Agent profile template
2
3 agent_profile = {}
4     "name": "Agent Name", # This will be replaced with the agent's name
5     "history_counter": 1, # This counter tracks the number of history entries
6     "history_1": {
7         "call_datetime": "YYYY-MM-DD HH:MM:SS", # This will be replace with the actual call date and time
8         "summary": "Summary of the first call or interaction", # This will be replace with the actual summary
9         "emotion": "Emotion detected or described during the first call" # This will be replace with the detected emotion
10    }
11 }
12
13 # Function to add a new history entry
14 def add_history(call_datetime, summary, emotion):
15     global agent_profile
16     counter = agent_profile['history_counter'] + 1
17     agent_profile['history_{}'.format(counter)] = {
18         "call_datetime": call_datetime,
19         "summary": summary,
20         "emotion": emotion
21     }
22     agent_profile['history_counter'] = counter
23
24 # Example usage of the function to add new history
25 # add_history("2024-08-31 16:12:40", "Progress of final year project; Place to go for dinner", "happy_80%, surprised_20%")
26

```

Figure 4.6.1 : Template for agent profile

```

C:\> Users > User > Desktop > FYP screenshots > CaiPin.py > ...
1 # Agent profile for CaiPin
2
3 agent_profile = {
4     "name": "CaiPin",
5     "history_counter": 10,
6
7     "history_1": {
8         "call_datetime": "2024-08-21 08:45:30",
9         "summary": "CaiPin talked about working in Australia as a freelancer. She shared how different the work culture is compared to Malaysia, and ChunShing listened as she explained.",
10        "emotion": "neutral_80%; happy_20%"
11    },
12
13    "history_2": {
14        "call_datetime": "2024-08-23 11:30:15",
15        "summary": "During the call, CaiPin shared her experience of moving from Tasmania to Sydney. She mentioned how she misses Tasmania and its peaceful surroundings but acknowledged the challenges of a new city.",
16        "emotion": "sad_70%; neutral_30%"
17    },
18
19    "history_3": {
20        "call_datetime": "2024-08-28 16:50:05",
21        "summary": "CaiPin expressed frustration with her current living situation. Her housemate constantly makes noise, which makes it hard for her to focus on work. ChunShing sympathized and offered advice.",
22        "emotion": "angry_60%; sad_40%"
23    },
24
25    "history_4": {
26        "call_datetime": "2024-09-01 09:10:12",
27        "summary": "CaiPin mentioned how exhausting it is to commute one hour to work every day. She and ChunShing discussed possible alternatives and how she might be able to reduce the commute.",
28        "emotion": "sad_70%; neutral_30%"
29    },
30
31    "history_5": {
32        "call_datetime": "2024-09-03 15:20:55",
33        "summary": "In this call, CaiPin told ChunShing that she needs to find a new rental house soon. The rental market in Sydney is competitive, and she feels overwhelmed with the search.",
34        "emotion": "sad_80%; neutral_20%"
35    },
36
37    "history_6": {
38        "call_datetime": "2024-09-05 12:35:18",
39        "summary": "CaiPin excitedly shared her plans to return to Malaysia for Chinese New Year 2025. She mentioned how much she's looking forward to spending time with family and friends.",
40    }
41 }

```

Figure 4.6.2 : Agent Profile from CaiPin.py

```
C:\Users\User\Desktop\FYP_screenshots > JoeYen.py > ...
1 # Agent profile for JoeYen
2
3 agent_profile = {}
4     "name": "JoeYen",
5     "history_counter": 11,
6
7     "history_1": {
8         "call_datetime": "2024-08-24 08:45:50",
9         "summary": "During a video call from home, JoeYen expressed anxiety over her upcoming exams. ChunShing offered her advice on managing stress and shared time management tips.",
10        "emotion": "fearful_70%; neutral_30%"
11    },
12
13    "history_2": {
14        "call_datetime": "2024-08-27 14:30:13",
15        "summary": "On a video call, ChunShing and JoeYen planned their next hiking trip. They discussed different trails and shared excitement about the upcoming adventure. The con
16        "emotion": "happy_80%; neutral_20%"
17    },
18
19    "history_3": {
20        "call_datetime": "2024-08-31 17:12:09",
21        "summary": "JoeYen shared her progress editing vlogs from past trips. ChunShing gave her some feedback on the transitions and style. They both reminisced about the trips and
22        "emotion": "happy_70%; neutral_30%"
23    },
24
25    "history_4": {
26        "call_datetime": "2024-09-03 11:40:27",
27        "summary": "During the call, JoeYen and ChunShing talked about learning to dive. They both had done some research and were excited about the idea of planning a dive trip tog
28        "emotion": "happy_80%; neutral_20%"
29    },
30
31    "history_5": {
32        "call_datetime": "2024-09-05 19:55:42",
33        "summary": "They discussed which cafe to visit next time. ChunShing and JoeYen shared thoughts on new places to try, eventually settling on a small, cozy cafe they had both
34        "emotion": "happy_90%; neutral_10%"
35    },
36
37    "history_6": {
38        "call_datetime": "2024-09-06 10:30:25",
39        "summary": "JoeYen talked about her plans to travel to Singapore next year. She mentioned that she's excited about the trip and has started planning what to see and do there
```

Figure 4.6.3 : Agent Profile from JoeYen.py

```
C:\Users\User\Desktop\FYP_screenshots > KenHow.py > ...
1 # Agent profile for KenHow
2
3 agent_profile = {
4     "name": "KenHow",
5     "history_counter": 11,
6
7     "history_1": {
8         "call_datetime": "2024-08-22 09:10:32",
9         "summary": "During a video call from home, ChunShing and KenHow discussed their final year project. KenHow seemed stressed as they tried to narrow down the project scope. He
10        "emotion": "sad_70%; neutral_30%"
11    },
12
13    "history_2": {
14        "call_datetime": "2024-08-24 14:45:11",
15        "summary": "In another video call, KenHow and ChunShing talked about the upcoming blockchain hackathon. KenHow was excited but felt unprepared. They spent some time going ov
16        "emotion": "happy_60%; neutral_40%"
17    },
18
19    "history_3": {
20        "call_datetime": "2024-08-29 12:05:43",
21        "summary": "Over a casual lunchtime video call, ChunShing and KenHow discussed food cravings. KenHow suggested trying a new restaurant after finishing their project, and bot
22        "emotion": "neutral_80%; happy_20%"
23    },
24
25    "history_4": {
26        "call_datetime": "2024-09-01 18:30:54",
27        "summary": "During a video call at home, they discussed internship offers. KenHow was uncertain which company to choose, so ChunShing helped him weigh the pros and cons. The
28        "emotion": "neutral_70%; thoughtful_30%"
29    },
30
31    "history_5": {
32        "call_datetime": "2024-09-03 10:21:07",
33        "summary": "In another call, ChunShing and KenHow brainstormed ideas for an upcoming assignment. Both were visibly tired and feeling the pressure of deadlines. They talked a
34        "emotion": "sad_60%; neutral_40%"
35    },
36
37    "history_6": {
38        "call_datetime": "2024-09-05 15:13:25",
39        "summary": "On this video call, KenHow shared that he was feeling more confident about the final year project. They had made progress and felt relieved that things were fall
```

Figure 4.6.4 : Agent Profile from KenHow.py

```

C:\Users\User\Desktop\FYPscreenshots > KhaiShen.py > ...
1 # Agent profile for KhaiShen
2
3 agent_profile = {
4     "name": "KhaiShen",
5     "history_counter": 10,
6
7     "history_1": {
8         "call_datetime": "2024-08-22 14:15:42",
9         "summary": "KhaiShen talked about the upcoming activities he is planning for the Computer Society. He was excited about the events but also shared concerns about how much wo
10        "emotion": "happy_60%; neutral_40%"
11    },
12
13    "history_2": {
14        "call_datetime": "2024-08-25 10:45:10",
15        "summary": "KhaiShen mentioned his struggles with finding an internship. He has been applying to multiple places but hasn't received any responses. ChunShing offered some ad
16        "emotion": "sad_70%; neutral_30%"
17    },
18
19    "history_3": {
20        "call_datetime": "2024-08-30 13:20:55",
21        "summary": "In this conversation, KhaiShen talked about his plan to have his internship in Kuala Lumpur. He discussed the benefits of being in a big city and how it might he
22        "emotion": "neutral_70%; happy_30%"
23    },
24
25    "history_4": {
26        "call_datetime": "2024-09-03 18:05:13",
27        "summary": "KhaiShen shared details about his final year project and some of the difficulties he's facing. He also mentioned that he plans to join a hackathon next year to c
28        "emotion": "happy_80%; neutral_20%"
29    },
30
31    "history_5": {
32        "call_datetime": "2024-09-06 09:45:30",
33        "summary": "They discussed the upcoming club meeting. KhaiShen expressed excitement about his presentation but also mentioned how nervous he was about the response.",
34        "emotion": "nervous_60%; excited_40%"
35    },
36
37    "history_6": {
38        "call_datetime": "2024-09-08 11:30:42",
39        "summary": "KhaiShen shared an internship offer he finally received from a company in Kuala Lumpur. He was excited but also a bit nervous about the transition.",

```

Figure 4.6.5 : Agent Profile from KhaiShen.py

```

C:\Users\User\Desktop\FYPscreenshots > Rina.py > ...
1 # Agent profile for Rina
2
3 agent_profile = {
4     "name": "Rina",
5     "history_counter": 9,
6
7     "history_1": {
8         "call_datetime": "2024-08-23 11:20:45",
9         "summary": "During a video call from home, Rina excitedly talked about her upcoming convocation in March 2025. ChunShing mentioned that he might attend, and they both laughe
10        "emotion": "happy_80%; surprised_20%"
11    },
12
13    "history_2": {
14        "call_datetime": "2024-08-26 19:00:10",
15        "summary": "Rina and ChunShing discussed her efforts to find a job in Japan. She expressed her frustrations with the language barrier and unfamiliar job market. They brainst
16        "emotion": "neutral_70%; sad_30%"
17    },
18
19    "history_3": {
20        "call_datetime": "2024-08-30 13:15:32",
21        "summary": "In a video call, Rina talked about the food she misses from Malaysia. They reminisced about their favorite dishes and made plans to enjoy them again when they me
22        "emotion": "sad_60%; neutral_40%"
23    },
24
25    "history_4": {
26        "call_datetime": "2024-09-02 09:50:25",
27        "summary": "ChunShing shared his post-graduation plans during this video call. Rina seemed excited about the possibilities and talked about her aspirations in psychology. Th
28        "emotion": "happy_70%; neutral_30%"
29    },
30
31    "history_5": {
32        "call_datetime": "2024-09-04 16:25:50",
33        "summary": "Rina and ChunShing talked about their shared goal of learning to dive. They joked about how little they knew about diving but were eager to start researching. It
34        "emotion": "happy_90%; neutral_10%"
35    },
36
37    "history_6": {
38        "call_datetime": "2024-09-05 10:30:20",
39        "summary": "Rina talked about her job search in Japan. She seemed more optimistic, mentioning a few potential job opportunities. ChunShing encouraged her to apply and not be

```

Figure 4.6.6 : Agent Profile from Rina.py


```
C:\Users\User\Desktop\FYP_screenshots > Wendy.py > ...
1 # Agent profile for Wendy
2
3 agent_profile = {
4     "name": "Wendy",
5     "history_counter": 10,
6
7     "history_1": {
8         "call_datetime": "2024-08-21 09:35:22",
9         "summary": "Wendy talked about her current experience of working and traveling in the US. She shared stories about the places she has visited and mentioned how much she miss
10        "emotion": "happy_70%; sad_30%"
11    },
12
13    "history_2": {
14        "call_datetime": "2024-08-24 11:05:33",
15        "summary": "Wendy mentioned that she will be returning to Malaysia in a month. She and ChunShing planned to meet up as soon as she lands on October 15. Both expressed excite
16        "emotion": "happy_90%; excited_10%"
17    },
18
19    "history_3": {
20        "call_datetime": "2024-08-27 15:50:18",
21        "summary": "Wendy confessed that she misses ChunShing and their conversations. She expressed how being away from friends has made her realize how important those connections
22        "emotion": "sad_80%; happy_20%"
23    },
24
25    "history_4": {
26        "call_datetime": "2024-08-31 16:35:09",
27        "summary": "During this call, Wendy asked about ChunShing's relationship status. They joked about their past experiences and how things have changed since she's been away.",
28        "emotion": "neutral_70%; curious_30%"
29    },
30
31    "history_5": {
32        "call_datetime": "2024-09-02 14:10:41",
33        "summary": "Wendy expressed concerns about what job she should pursue when she returns to Malaysia. She's unsure of what direction to take and is feeling a bit lost. ChunShi
34        "emotion": "sad_60%; neutral_40%"
35    },
36
37    "history_6": {
38        "call_datetime": "2024-09-04 12:00:00",
39        "summary": "Wendy and ChunShing talked more about their plans to meet up in Malaysia. They finalized the details of where they would meet and what they'd do after she lands.
```

Figure 4.6.7 : Agent Profile from Wendy.py

4.7 Operation Logic

The operation of EmonyAI is structured into three phases: before the video call starts, during the video call, and after the video call ends. Each phase governs how the system interacts with users and handles conversation suggestions based on real-time data from video calls.

4.7.1 Before the Video Call Starts

This phase sets up the user profile and prepares EmonyAI for the video call.

1. Initiating the Call:
 - EmonyAI prompts the user with: "Who are we calling now?"
 - The user inputs the <name> of the person they are calling.
2. Profile Lookup in GCS Bucket:
 - EmonyAI searches for the <name>.py profile in the Google Cloud Storage (GCS) Bucket.
 - If <name>.py is found, EmonyAI outputs "Profile is found" and loads the profile.
 - If <name>.py is not found, EmonyAI creates a new profile, outputs "Creating new profile", and stores it based on the format specified in sample.py.
3. User Commands:
 - If the user inputs "Abort", the system returns to "Who are we calling now?", allowing the user to restart or change the target caller.

4.7.2 During the Video Call

Once the call starts, EmonyAI activates APIs for real-time data analysis and provides conversation suggestions based on emotions and speech.

1. Emotion Detection with Custom API:
 - EmonyAI uses our custom emotion detection API deployed from the Google Container Registry, which is tailored for this system.
 - The API processes the video input from the call and detects emotions at 5-second intervals. Only emotions classified as VERY_LIKELY (joy, sorrow, anger, surprise) are recorded.

- The detected emotions are saved in "emotion_tempo.txt" in the GCS Bucket.
 - This emotion detection API enables real-time analysis of the caller's emotional state, offering relevant data for EmonyAI to use in generating conversation suggestions.
2. Speech-to-Text Conversion with Google Speech-to-Text API:
- Audio from the video call is processed through the Google Speech-to-Text API, converting speech into text.
 - The transcribed text is stored in "speech_tempo.txt" in the GCS Bucket.
 - This text data is later used by ChatGPT to generate summaries and conversation suggestions.
3. Summarization with ChatGPT API:
- The ChatGPT API reads the content in "speech_tempo.txt" and generates a summary, which is stored in the target caller's profile <name>.py.
 - This functionality allows EmonyAI to keep concise records of conversations, which will improve the system's ability to offer personalized suggestions over time.

4.7.3 Pauses in Conversation

EmonyAI monitors pauses during the conversation and performs specific actions based on the length of the pause.

- When no audio is detected for 5 seconds:
 - EmonyAI saves both "speech_tempo.txt" and "emotion_tempo.txt" and pauses writing to these files for 8 seconds.
 - During this pause, ChatGPT generates a summary of "speech_tempo.txt" in under 50 words and analyzes "emotion_tempo.txt" to calculate emotion percentages (e.g., "joy_80%, anger_20%").
 - This summarized data is appended to <name>.py using the format in sample.py
 - After saving, "speech_tempo.txt" and "emotion_tempo.txt" are cleared for the next segment of the call.
- When no audio is detected for 10 seconds:

- EmonyAI reads <name>.py and uses ChatGPT to analyze historical data, focusing on topics that previously led to higher levels of positive emotions (joy or surprise).
- The system then suggests a topic to the user: "You can try talking about <topic>, they seemed interested in it."
- This allows EmonyAI to guide the conversation toward topics the caller has responded to positively in the past, improving engagement.

4.7.4 After the Video Call Ends

When the video call ends, EmonyAI updates the caller's profile and prompts the user for further action.

1. Profile Update and Saving:
 - The profile <name>.py is updated with the summarized conversation and emotional data collected during the call, ensuring that all relevant information is stored for future interactions.
 - This ensures that the profile grows with each call, enabling more accurate suggestions in future conversations.
2. Prompt for New Call:
 - After saving the profile, EmonyAI asks the user: "Call another person?"
 - If the user responds with "yes", the system restarts from "Who are we calling now?" for the next conversation.

4.8 Special Situations

In addition to the standard operation logic, EmonyAI includes several special modes designed to handle specific situations and user preferences. These modes ensure that the system is flexible and adaptable to unique user needs while preventing any unexpected situations from occurring.

Private Mode:

Private mode aims to protect the privacy of both parties during sensitive or confidential conversations.

Trigger: If the conversation involves confidential information and both parties prefer not to have the conversation recorded, they can type "private" or "Private" into the chatbot.

Behavior: Upon activation of Private Mode, EmonyAI will immediately pause all three key services: emotion detection, speech-to-text conversion, and ChatGPT summarization. The video call itself will continue, but no data will be captured or processed by EmonyAI until the mode is deactivated. This feature ensures that users have full control over when the conversation is recorded and when it remains entirely private.

Imagine Mode:

Imagine Mode utilizes the MetaGPT Framework to simulate interactions between two recorded agent profiles using historical conversation data.

Trigger: When the user types "imagine" or "Imagine" into the chatbot, EmonyAI will prompt the user to specify the two agent profiles they want to test in a simulated interaction.

Process:

1. EmonyAI will first respond with: "Subject 1?" and wait for the user to input the name of the first agent.
2. It will then search for the profile of <name> in the GCS Bucket.
3. Next, the system will prompt: "Subject 2?", and the user will provide the name of the second agent.

4. EmonyAI will search for the second profile.

Outcome:

1. If both profiles are found, EmonyAI will load the two profiles into the MetaGPT framework to generate a simulated conversation based on the historical data of both agents.
2. If one or both profiles are missing from the GCS Bucket, EmonyAI will output: "You need to learn more about this person first.", indicating that more real conversations are required to build the necessary profiles for simulation.

This mode is useful for testing and evaluating interactions between two agents based on their past conversation history, providing insights into how they might collaborate or communicate in different contexts.

CHAPTER 5 EXPERIMENT/SIMULATION

5.1 Setup

5.1.1 Hardware Setup

For the EmonyAI system, the following hardware setup was used to ensure optimal performance during real-time facial recognition, emotion detection, and conversation generation:

- User Device (Laptop/PC):
 - Processor: AMD Ryzen 7 5800HS. This processor provided the necessary computational power to handle multiple processes simultaneously, including video streaming, emotion detection, and speech-to-text conversion.
 - GPU: NVIDIA GTX 1650. This GPU was essential for GPU acceleration, allowing the system to perform resource-intensive tasks like real-time facial recognition and emotion detection.
 - RAM: 16 GB. This memory capacity was chosen to support the smooth execution of the system, especially when handling multiple streams of data from video, audio, and AI processing components.
 - Camera: 1080p resolution webcam. The high-definition camera was used to capture clear video input, ensuring accurate emotion detection and facial recognition.
 - Microphone: Built-in microphone for clear audio capture, with noise-cancelling capabilities to ensure accurate speech-to-text conversion.
- Server/Cloud Requirements:
 - Google Cloud Run: This platform hosted the back-end services of the system, including the AI models for facial recognition, emotion detection, and conversation generation. The use of Google Cloud Run also ensured scalability, allowing the system to handle multiple users simultaneously.
 - Google Cloud Services Bucket Storage: This cloud storage solution was used to store user profiles, interaction logs, and conversation data.

5.1.2 Software Setup

The software setup for EmonyAI consisted of various tools and libraries integrated to handle real-time processing, AI-based analysis, and cloud deployment.

- Operating System:
 - Windows 11: The development environment was set up on Windows 11, providing compatibility with all required software and libraries, including WebRTC, Emotion Detection, and the Google Cloud API.
- Development Environment:
 - Vite React: Used to build the front-end user interface for the video calling platform. Vite React's fast build and deployment capabilities allowed for rapid iterations during the development process.
 - VS Code: This IDE was used to write and debug the application. VS Code was chosen for its ease of use and extensive support for JavaScript and Python.
- Real-Time Processing Software:
 - WebRTC: This was the core technology for real-time video and audio streaming between users. It enabled the peer-to-peer connection required for capturing user input and transmitting it to the processing modules.
 - Emotion Detection Module: This tool was integrated to handle real-time facial recognition and emotion detection. The module's ability to process video frames in real-time made it ideal for this project.
 - Speech-to-Text Engine: The Google Cloud Speech-to-Text API was used to convert spoken audio into text, which was then passed on to the chatbot for conversation generation.
- Backend Services:
 - MetaGPT Framework: This framework was responsible for managing user profiles, tracking emotional states, and generating conversation suggestions. It continuously learned from interactions, ensuring that conversation suggestions became more personalized over time.
 - ChatGPT API: This API handled the natural language processing (NLP) required to generate human-like responses based on the user's speech and emotional cues.

5.2 System Operation

The system operation begins by launching the video call website, where EmonyAI is integrated to assist in real-time conversations. Once the video call starts, EmonyAI monitors both verbal and non-verbal cues, using its emotion detection, speech-to-text, and chatbot services to provide conversation suggestions and emotional feedback. Users interact seamlessly with the chatbot, which can load profiles, analyze conversation history, and simulate interactions. The system operates continuously in the background, enhancing the video call experience by offering contextually relevant suggestions and adapting to each interaction dynamically.

5.2.1 Interaction Involving Positive Feedback

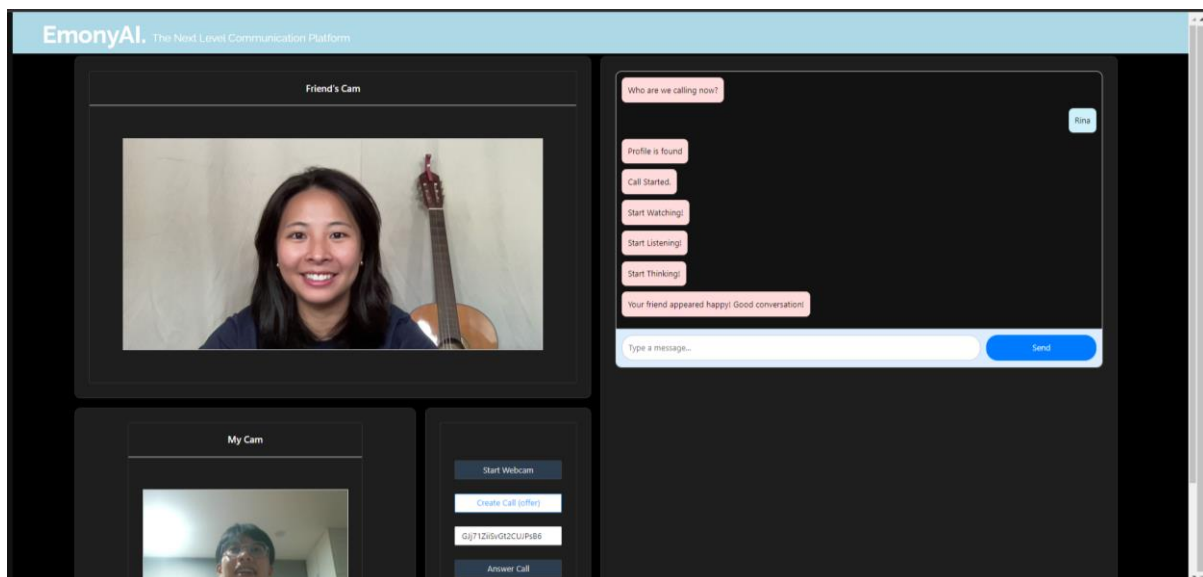


Figure 5.2.1 : Screenshot of interaction between me and Amanda Lean (Rina.py)

- The system starts with EmonyAI asking the user “Who are we calling now?”
- The user responds with “Rina” and the profile is found in GCS Bucket. Therefore, output “Profile is found”.
- The target caller enters the call, EmonyAI is updated and responded with “Call Started.”
- “Start Watching!” indicates that Emotion Detection is started.
- “Start Listening!” indicates that Speech-To-Text is started.
- “Start Thinking!” indicates that ChatGPT is started.
- Emotion Detection detected the target caller to be happy, therefore output “Your friend appeared happy! Good conversation!”

5.2.2 Interaction Involving Negative Feedback

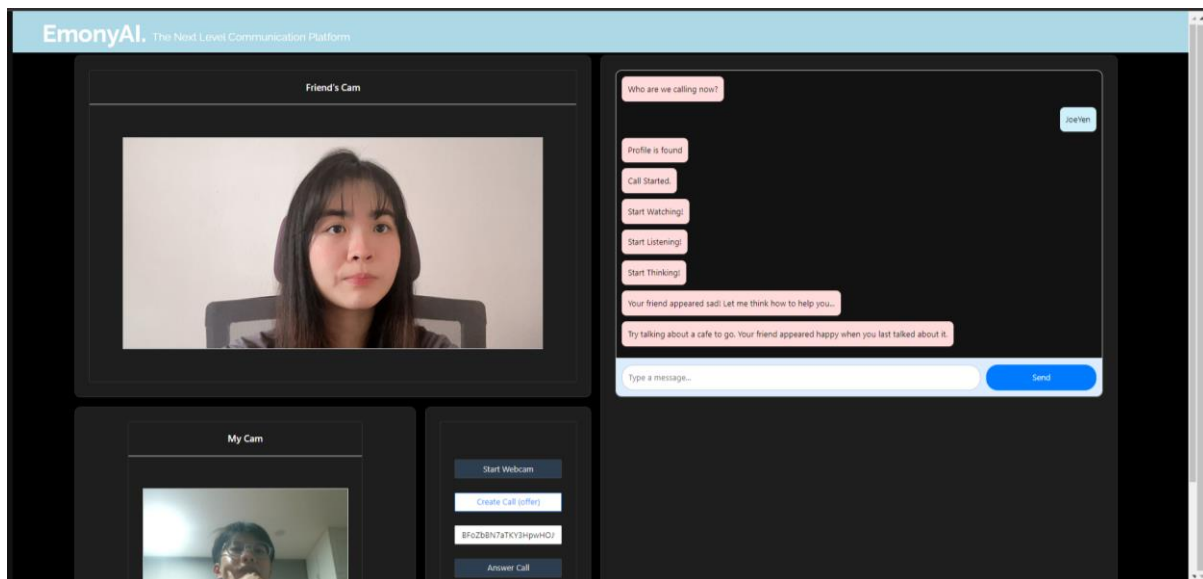


Figure 5.2.2 : Screenshot of interaction between me and Chu Joe Yen (JoeYen.py)

- The system starts with EmonyAI asking the user “Who are we calling now?”
- The user responds with “JoeYen” and the profile is found in GCS Bucket. Therefore, output “Profile is found”.
- The target caller enters the call, EmonyAI is updated and responded with “Call Started.”
- “Start Watching!” indicates that Emotion Detection is started.
- “Start Listening!” indicates that Speech-To-Text is started.
- “Start Thinking!” indicates that ChatGPT is started.
- Emotion Detection detected the target caller to be sad, therefore output “Your friend appeared sad! Let me think how to help you..”
- Start reading through JoeYen.py, and search for last history that matched with higher percentage ‘happy’ emotion. Input the history to ChatGPT to generate a shorter summarization.
- Output the result “Try talking about <suggestion>. Your friend appeared happy when you last talked about it.” In this case, <suggestion> = “a café to go”.

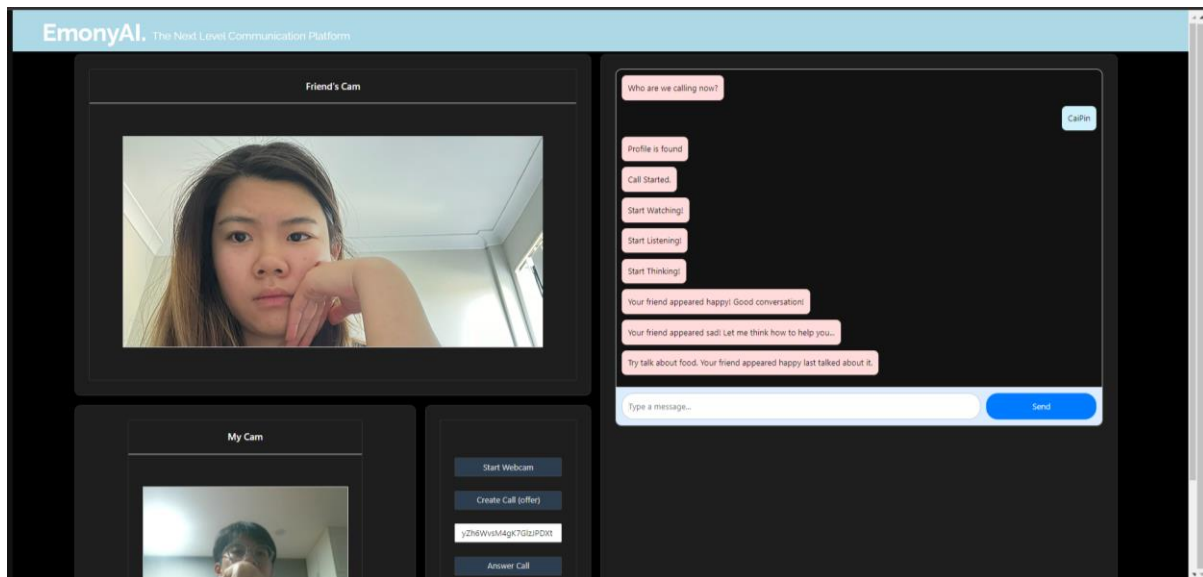


Figure 5.2.3 : Screenshot of interaction between me and Ng Cai Pin (CaiPin.py)

- The system starts with EmonyAI asking the user “Who are we calling now?”
- The user responds with “CaiPin” and the profile is found in GCS Bucket. Therefore, output “Profile is found”.
- The target caller enters the call, EmonyAI is updated and responded with “Call Started.”
- “Start Watching!” indicates that Emotion Detection is started.
- “Start Listening!” indicates that Speech-To-Text is started.
- “Start Thinking!” indicates that ChatGPT is started.
- Emotion Detection detected the target caller to be happy, therefore output “Your friend appeared happy! Good conversation!”
- Emotion Detection detected changes in the target caller to be sad, therefore output “Your friend appeared sad! Let me think how to help you..”
- Start reading through CaiPin.py, and search for last history that matched with higher percentage ‘happy’ emotion. Input the history to ChatGPT to generate a shorter summarization.
- Output the result “Try talking about <suggestion>. Your friend appeared happy when you last talked about it.” In this case, <suggestion> = “food”.

5.3 Multi-Agent Interaction

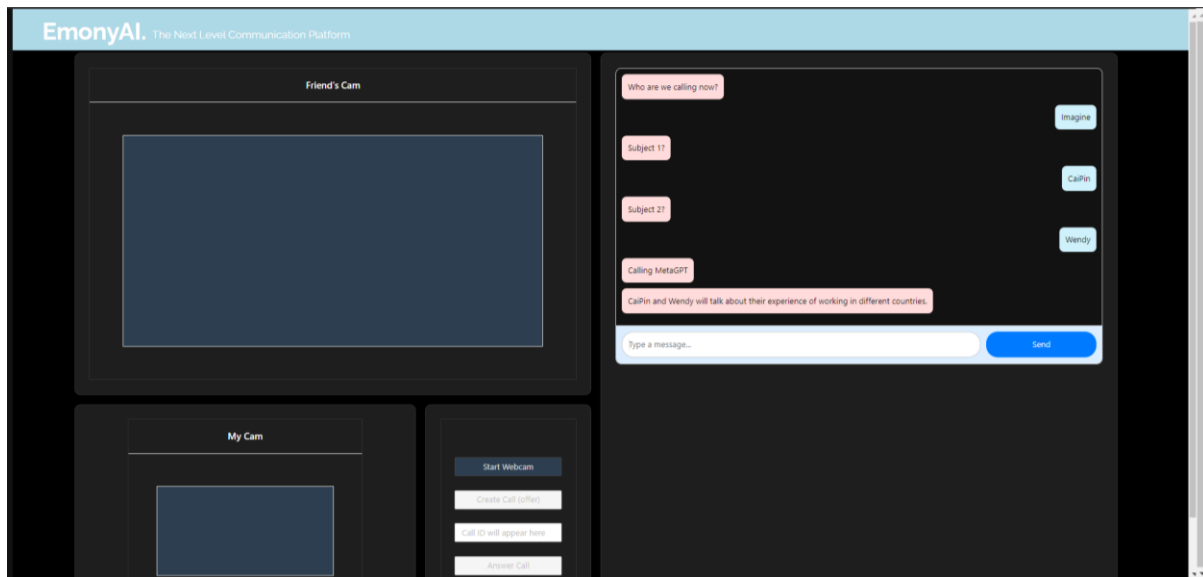


Figure 5.2.4 : Screenshot of EmonyAI Multi-Agent prompting

- The system starts with EmonyAI asking the user “Who are we calling now?”
- The user responds with “Imagine” and therefore trigger multi-agent functionality.
- The system request Subject 1 name.
- The user responded with “CaiPin”.
- The system request Subject 2 name.
- The user responded with “Wendy”.
- The system then loads both profile from GCS Bucket into MetaGPT, and output “Calling MetaGPT”.
- After processing, the system output the interaction content between the two character profile.

Note: using Multi-Agent Prompting doesn't require to start WebRTC protocol.

5.4 Application for Special Situations

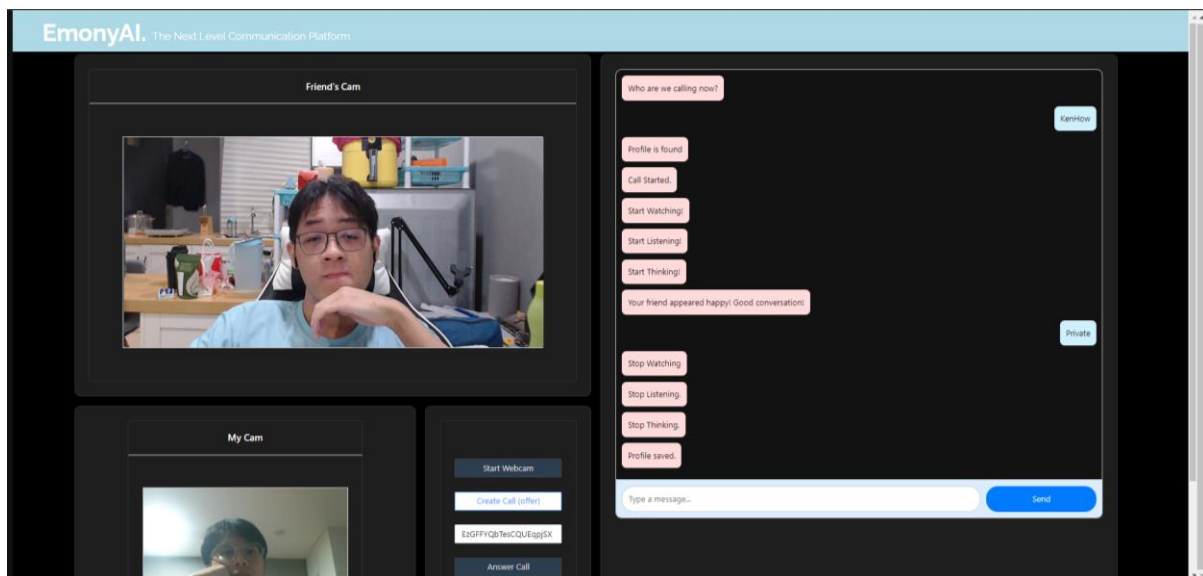


Figure 5.2.5 : Screenshot of interaction between me and Goh Ken How (KenHow.py)

- The system starts with EmonyAI asking the user “Who are we calling now?”
- The user responds with “CaiPin” and the profile is found in GCS Bucket. Therefore, output “Profile is found”.
- The target caller enters the call, EmonyAI is updated and responded with “Call Started.”
- “Start Watching!” indicates that Emotion Detection is started.
- “Start Listening!” indicates that Speech-To-Text is started.
- “Start Thinking!” indicates that ChatGPT is started.
- Emotion Detection detected the target caller to be happy, therefore output “Your friend appeared happy! Good conversation!”
- The user input “Private”
- The system ends 3 services.
- “Stop Watching.” indicates that Emotion Detection is stopped.
- “Stop Listening.” indicates that Speech-To-Text is stopped.
- “Stop Thinking.” indicates that ChatGPT is stopped.
- The system saves the target caller profile.

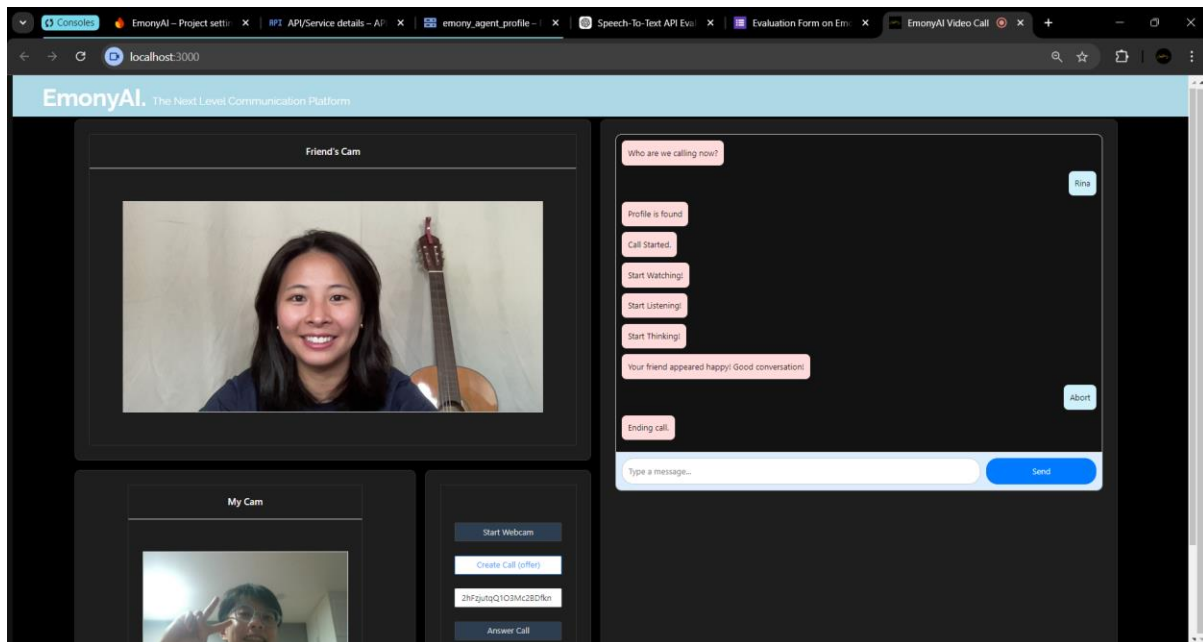


Figure 5.2.6 : Screenshot of interaction between me and Amanda Lean (Rina.py)

- The system starts with EmonyAI asking the user “Who are we calling now?”
- The user responses with “Rina” and the profile is found in GCS Bucket. Therefore, output “Profile is found”.
- The target caller enters the call, EmonyAI is updated and responded with “Call Started.”
- “Start Watching!” indicates that Emotion Detection is started.
- “Start Listening!” indicates that Speech-To-Text is started.
- “Start Thinking!” indicates that ChatGPT is started.
- Emotion Detection detected the target caller to be happy, therefore output “Your friend appeared happy! Good conversation!”
- The user input “Abort”
- System output “Ending call”, and ends WebRTC connection in 10 seconds timer.

5.5 Implementation Issues and Challenges

Several challenges were encountered during the implementation of **EmonyAI**. These challenges and their solutions are outlined below:

1. GPU Resource Limitation:

- **Challenge:** The real-time processing of video frames by OpenFace required significant GPU resources, leading to occasional slowdowns.
- **Solution:** GPU acceleration was implemented using NVIDIA's CUDA cores, which significantly reduced the time required to process each frame and improved the overall system performance.

2. Video Latency in WebRTC:

- **Challenge:** During initial testing, there was noticeable video latency when using WebRTC, which impacted the timing of emotion detection.
- **Solution:** The WebRTC configuration was optimized by adjusting buffer sizes and bandwidth allocation. Additionally, network conditions were simulated to test the system's performance in different environments.

3. Inconsistent Speech-to-Text Accuracy:

- **Challenge:** The speech-to-text conversion was occasionally inaccurate due to background noise and low-quality audio input.
- **Solution:** A noise-cancelling filter was applied to the audio stream before sending it to the Google Cloud Speech-to-Text API, improving transcription accuracy.

4. Scalability Issues:

- **Challenge:** As the system was designed to handle multiple users, managing and updating user profiles in real-time presented scalability challenges.
- **Solution:** Google Cloud Services was utilized to store and manage user profiles efficiently, with dynamic scaling options to handle increasing user demand.

5.6 Concluding Remark

The experimentation phase of EmonyAI has successfully demonstrated the feasibility of AI-driven emotion detection and conversation support within a video call platform. Despite several challenges during development—such as GPU resource limitations, video latency in WebRTC, and inconsistent speech-to-text accuracy—the system was refined to provide real-time, context-aware responses. By leveraging GPU acceleration, optimizing WebRTC configurations, and applying noise-cancellation techniques, the performance and accuracy of EmonyAI were significantly improved. Furthermore, Google Cloud Services played a key role in managing user profiles, ensuring scalability for future growth. This project showcases the potential for AI to enhance human interactions in digital communication platforms by making them more empathetic and responsive.

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION

6.1 System Testing

EmonyAI is built around three main components: emotion detection, speech-to-text, and ChatGPT integration. Together, these components work seamlessly to provide a more interactive and responsive video call experience.

Emotion Detection:

This component analyzes the emotions of participants during video calls by capturing facial expressions from the video feed. The detected emotions are used to provide real-time feedback or suggestions, ensuring that the conversation is more empathetic and contextual aware.

Speech-to-Text:

The system utilizes Google's Speech-to-Text API to convert spoken conversations into text. This allows for transcription of the call and enables the chatbot to understand and process the content of the conversation more effectively.

ChatGPT Integration:

Leveraging OpenAI's ChatGPT API, EmonyAI provides intelligent summarization of conversations and offers conversation suggestions. This helps guide users through the call, making recommendations based on the detected emotions and speech content, enhancing the overall interaction.

Evaluation will be made on these three components to ensure the system works well. Each part will be assessed for its accuracy and performance guaranteeing that EmonyAI delivers a high-quality, real-time communication experience.

6.1.1 Testing on Emotion Detection

Before deploying the emotion detection API to Google Cloud, the testing process focused on ensuring the accuracy and reliability of the emotion detection script using images captured during video calls. This local testing phase was critical to validating the functionality of the script before publishing it to the cloud.

1. Capturing Images from Video Calls

During video calls, frames from the video stream were captured and saved as images. These images were used as input data to simulate real-world scenarios, allowing the script to process facial expressions in a controlled environment. The captured images represented a variety of emotional expressions, ensuring that the emotion detection model could handle a range of emotions effectively.

By using actual images from video calls, the testing process closely mimicked the conditions under which the emotion detection API would operate once deployed.






2. Local Testing of the Emotion Detection Script






The emotion detection script, which was developed to analyze facial expressions and categorize them into emotions (e.g., happy, sad, surprised, etc.), was tested locally on a development machine. Instead of deploying the API directly to Google Cloud, the following steps were performed to ensure that the script worked correctly:

- Running the Script Locally:
 - The script was run on the local machine with the captured images as input. This allowed for rapid iteration and debugging without needing to deploy the code to the cloud repeatedly.
- Emotion Classification:
 - The script analyzed the facial expressions in each image and classified the emotions detected. This output was compared with expected results based on the actual expressions (portrayed by the user) in the captured images. The emotion that can be classified are angry, disgusted, fearful, happy, neutral, sad and surprised

3. Testing Results

Image	Testing Result	Actual Result
-------	----------------	---------------

		Happy	Happy
		Neutral	Fearful
		Surprised	Happy
		Happy	Happy
		Surprised	Happy

		Surprised	Disgusted
		Fearful	Happy
		Happy	Happy
		Neutral	Neutral
		Disgusted	Disgusted

	Happy	Happy
	Neutral	Neutral
	Neutral	Neutral
	Happy	Happy

Table 6.1.1 : Testing Result for Emotion Detection

6.1.2 Evaluation on Speech-To-Text

In the MetaGPT system, agent profiles are stored in a Google Cloud Storage (GCS) bucket and include information generated from the conversation, such as transcribed text and emotional data. The creation of these profiles begins with audio input being processed through the Google Speech-To-Text API. The resulting transcription is temporarily saved in a file named `speech_tempo.txt`.

To minimize the amount of data stored, this file is then summarized by ChatGPT. The summarized version of the conversation is stored in the agent profile, significantly reducing storage space while maintaining key conversational details.

To ensure the reliability of this process, we must evaluate the performance of the Google Speech-To-Text API in converting the audio input into text by analyzing the contents of the `speech_tempo.txt` file.

Key Steps in the Process

1. **Audio Input:** The system captures audio from conversations during a video call.
2. **Speech-to-Text Conversion:** The captured audio is sent to the Google Speech-To-Text API, which transcribes the spoken words into text.
3. **Saving the Transcription:** The resulting transcription is saved into a temporary text file named `speech_tempo.txt`.
4. **Summarization by ChatGPT:** To conserve storage space, ChatGPT processes and summarizes the transcription, retaining only the essential information. This summarized version is stored in the agent profile.

Evaluation Considerations

1. **Transcription Accuracy:** We can measure the performance of the Speech-To-Text API by analyzing how accurately the transcribed text (from `speech_tempo.txt`) matches the original spoken words. This includes calculating the Word Error Rate (WER) and checking for any critical errors in context.
2. **Summary Quality:** After the transcription is summarized by ChatGPT, we need to ensure that the key points and relevant information from the conversation are still captured, despite the reduction in data.

Example 1

```
"history_5": {  
  "call_datetime": "2024-09-05 19:55:42",  
  "summary": "They discussed which cafe to visit next time. ChunShing and JoeYen shared thoughts on new places to try, eventually settling on a small, cozy cafe they had both heard good things about.",  
  "emotion": "happy_90%; neutral_10%"  
},
```

Figure 6.1.1 : Code snippet from agent profile : JoeYen.py

Full Text Recorded (from speech_tempo.txt)

A:

Hey! Have you thought about which cafe we should go to after our exams?

B:

Yeah, I've been thinking about it. What do you think about that new place on Main Street? The cozy one with the green sign? I've heard some good things about it.

A:

Oh, you mean the one with the great coffee? I heard it's pretty popular these days. Though I'm not sure if it will be too crowded on weekends.

B:

Yeah, that's true. It might be a bit too busy for my taste. What about that other one you mentioned last time? The one near the lake? It sounded interesting.

A:

Oh yeah, I think it's called Lazy Day Cafe. I haven't been there yet, but I heard it's nice and quite—good for studying or catching up. The only thing is it's a bit far from here, but I'm up for it if you are!

B:

Hmm, it does sound like a nice spot, especially if we want some peace and quiet after exams. I don't mind the drive, really. Plus, we'll probably want to just relax and unwind.

A:

Totally! I think I'll need a calm place after all these exams. Too much pressure this semester, and I can't wait for a break. Oh, did I tell you? I heard that cafe also has these homemade cakes that people rave about.

B:

Really? Cakes too? Okay, now you've convinced me. I'm definitely down to try it out. Do they have anything special, like some seasonal flavors?

A:

Yeah! Someone told me they make a really good pumpkin spice cake around this time of year. And I think they have these matcha desserts that are supposed to be amazing.

B:

Wow, that sounds perfect. I haven't had good matcha in ages. We could totally get a slice of each and share. Oh, and maybe some coffee. I've been craving a good cup of coffee for weeks.

A:

That sounds like a plan! It'll be the perfect way to de-stress. I hope the place isn't too crowded, though. I'd hate to finally get there and not find a table.

B:

Yeah, true. I think we should aim for a weekday, maybe in the afternoon? I feel like it might be less busy then, especially if it's a bit out of the way. What do you think?

A:

Good idea. A weekday afternoon sounds perfect. Plus, we can avoid the weekend crowd. I really don't feel like waiting for a table after all this studying.

B:

Exactly. You know, I was also thinking... We could go a bit earlier, and maybe take a walk by the lake before heading to the cafe. I heard the view there is really nice, especially this time of year.

A:

Ooooh, that sounds nice! I love a good walk. I've been cooped up indoors for too long. Plus, I heard there are some cool spots by the lake where you can sit and just relax for a bit. We can grab coffee to go and sit by the water if the weather's nice.

B:

Yes! That sounds perfect. I'm definitely up for it. A little walk, some fresh air, and then cake and coffee—sounds like an ideal day to me.

A:

Same here! Let's just hope it doesn't rain. Have you checked the weather for next week?

B:

I haven't, actually. I'll do that later tonight. Fingers crossed for good weather though. It'd be nice to just have a chill day outdoors.

A:

Yeah, we need a break after everything. I can't wait for these exams to be over. One more week, and then we're free!

B:

I know, right? It feels like it's been dragging on forever. But we're almost there. We just need to push through these last few days. What's your last exam?

A:

I've got physics on Wednesday, so after that, I'm officially done. You?

B:

I finish with economics on Friday. Not looking forward to it, but at least once it's over, we can go have a nice day out. Something to look forward to.

A:

Exactly! Once we're done, let's just forget about school for a while. I'll be ready for a meetup and some relaxation by then.

B:

Same here. So it's a plan then—Lazy Day Cafe after exams? Maybe a walk by the lake before that?

A:

Yup, let's do it! I'm excited already. We'll just keep an eye on the weather, and if it's good, we're all set.

B:

Perfect. I'll check the weather and let you know. But either way, we'll meet up and try out that cafe. Can't wait!

A:

Me neither! I'm looking forward to catching up and finally relaxing after this crazy semester.

Evaluation

The text generated is mostly correct, with some weird spelling errors.

Table 6.1.2 : Text Recorded and Evaluation for JoeYen.py

Text summarized into Agent Profile (JoeYen.py)

They discussed which cafe to visit next time. User and JoeYen shared thoughts on new places to try, eventually settling on a small, cozy cafe they had both heard good things about. They were looking forward to meeting up after exams.

Evaluation

The content summarizes is true to the conversation content, thus we can assume that the minor spelling errors from Speech-To-Text can be ignored.

Table 6.1.3 : Text Summarized and Evaluation for JoeYen.py

Example 2

```
"history_1": {  
  "call_datetime": "2024-06-21 09:35:22",  
  "summary": "Wendy talked about her current experience of working and traveling in the US. She shared stories about the places she has visited and mentioned how much she misses home.",  
  "emotion": "happy_70%; sad_30%"  
},
```

Figure 6.1.2 : Code snippet from agent profile : Wendy.py

Full Text Recorded (from speech_tempo.txt)

A:

Hey Wendy! How's it going? It feels like forever since we last talked. How's life in the US?

B:

Hey! Yeah, it's been a while. I'm doing *good*, but I definitely miss home. Traveling has been amazing, though. I've been able to visit a few cool places in the last couple of months.

A:

That's awesome! Where have you been so far? I want to live vicariously through your adventures!

B:

Well, I've been to New York, obviously. I had to see all the typical tourist spots like Times Square and Central Park. Oh, and the Statue of Liberty! It was so surreal seeing it in person.

A:

I can imagine! New York must have been so overwhelming. How did you find it? Did it live up to the hype?

B:

Totally! It's exactly like what you see in the movies—busy, chaotic, but kind of magical in its own way. But after a few days, I felt like I needed a break from the craziness, so I took a trip to California.

A:

California sounds amazing. Did you go to LA? What did you get up to there?

B:

Yeah, I was in LA for a bit. It's so different from New York—much more laid-back, which I liked. I visited the beaches and did some hiking. The views from *Griffin Park* were incredible. Oh, and I even went to Disneyland! It was so much fun.

A:

Wow, you're really living the dream! But I bet you must be getting homesick, though.

B:

Yeah, I definitely am. I miss my family and freinds a lot. It's weird because, on one hand, I'm having an amazing time here, but on the other hand, I really miss the comfort of home.

A:

That makes sense. Being away for so long can be tough, even when you're having fun. How much longer are you staying there?

B:

I've got a couple more months left on my visa, so I'll be here until the end of October. It's kind of bittersweet because I'm enjoying myself, but I can't wait to come back home.

A:

It's nice that you still have some time to explore, though! Any other places on your list before you head back?

B:

I'm planning to visit Chicago next. I've always wanted to see it. And after that, I might do a quick road trip through some of the smaller towns. It'll be nice to see a different side of the US.

A:

That sounds like a solid plan! I'm sure you'll love Chicago. Just be prepared—it can get really windy there!

B:

Yeah, I've heard that! I'll definitely be packing my jacket. But yeah, I'm excited. It's just such a different experience being here. But you know, as much as I'm enjoying all this, there's something about home that I just can't stop missing.

A:

Yeah, there's nothing like home, right? The comfort, the familiarity... I totally get it. But you'll be back soon, and we'll all be waiting for you with open arms!

B:

Thanks, I appreciate that. I can't wait to catch up with everyone. I'm trying to stay positive and make the most of the time I have left here. But yeah, some days I just feel a little sad thinking about how far away home is.

A:

That's completely understandable. But just remember, you're having this amazing experience that you'll always look back on. And we'll all be here when you get back.

B:

You're right. I'm trying to remind myself of that every day. It's been such a great learning experience, but yeah, sometimes I just want a home-cooked meal, you know?

A:

I can imagine! Don't worry, when you're back, we'll make sure you get all the home-cooked meals you want.

B:

I'm holding you to that! Anyway, thanks for the chat. It's really nice talking to someone from home.

A:

Anytime, Wendy. Hang in there and enjoy the rest of your trip. We all miss you, but we'll see you soon!

B:

I miss you guys too. Talk soon!

Evaluation

The text generated is mostly correct also with some weird spelling errors especially for location name.

Table 6.1.4 : Text Recorded and Evaluation for Wendy.py

Text summarized into Agent Profile (Wendy.py)

Wendy talked about her current experience of working and traveling in the US. She shared stories about the places she has visited and mentioned how much she misses home.

Evaluation

The content summarizes is true to the conversation content, thus we can assume that the minor spelling errors from Speech-To-Text can be ignored.

Table 6.1.5 : Text Summarized and Evaluation for Wendy.py

Example 3

```
"history_0": {  
  "call_datetime": "2024-09-05 10:30:20",  
  "summary": "Rina talked about her job search in Japan. She seemed more optimistic, mentioning a few potential job opportunities. ChunShing encouraged her to apply and not be discouraged by previous setbacks.",  
  "emotion": "happy_70%; neutral_30%"  
},
```

Figure 6.1.3 : Code snippet from agent profile : Rina.py

Full Text Recorded (from speech_tempo.txt)

A:

Hey, Rina! How's everything going? How's the job search in Japan treating you?

B (Rina):

Hey! You know what, it's actually going better than I expected. I've come across a few opportunities that look promising, so I'm feeling a bit more *optimistic*.

A:

That's great to hear! What kind of jobs are you looking at?

B:

Well, I'm mostly focusing on positions in marketing and communications. I found this one job at a tech company that looks interesting. It's a marketing *specialist* role, and I think it would be a good fit for me.

A:

That sounds like a good match! You should definitely apply for it. What other opportunities have you found?

B:

There's another one at a startup—more of a general marketing role, but they're looking for someone who can help them grow their brand. It's a smaller team, which I think could be fun. I like the idea of getting in on the ground floor of something new.

A:

I totally get that! Startups can be exciting, and you'd get to wear a lot of hats. Plus, you'll learn a ton. Have you applied yet?

B:

Not yet, but I'm planning to. I'm still polishing up my resume and cover letter. I want to make sure I'm fully prepared before I send them in.

A:

Good idea. Take your time, but don't wait too long! You've got this, Rina. You're more than qualified for these roles. Just don't let those past setbacks discourage you.

B:

Thanks for saying that. It's been tough, especially with a few rejections earlier this year, but I'm feeling more *positiv* now. I think I just needed some time to regroup and refocus.

A:

Yeah, I remember you saying that. But look at you now—you're already bouncing back! Those setbacks don't define you. It's all part of the process.

B:

You're right. I've just got to keep pushing forward. I've learned a lot from those experiences, and now I feel like I'm in a better place mentally to handle things.

A:

That's the spirit! You're stronger than you think. And remember, sometimes those rejections happen because there's something better waiting for you.

B:

I hope so! I've also been networking a lot, going to job fairs, and connecting with people in the industry. I think that's going to help in the long run.

A:

Absolutely. Networking is key, especially in Japan. Keep building those connections—they could lead to some unexpected opportunities. You never know who might give you that next big break.

B:

Exactly. I've even reached out to a few people on LinkedIn, just to introduce myself and ask for advice. It's a bit nerve-wracking, but so far, people have been really nice.

A:

That's awesome! I'm sure people will appreciate your initiative. Just keep at it, and don't hesitate to follow up if you don't hear back right away.

B:

Yeah, that's something I'm working on—being more persistent without feeling like I'm bothering people. It's a balance, but I'm getting the hang of it.

A:

It's all part of the learning curve. But honestly, you're doing everything right. Keep applying, keep networking, and something's bound to come through.

B:

Thanks! Your encouragement really means a lot. Sometimes I still get those moments of doubt, but talking to you helps me stay focused.

A:

Anytime, Rina. You're doing great, and I know you're going to find something soon. Just don't give up. We're all rooting for you!

B:

Thanks, I really appreciate it. I'll keep you posted on how things go. Hopefully, I'll have good news to share soon!

A:

I'm sure you will! Keep that optimism up, and don't hesitate to reach out if you need any advice or just want to talk.

B:

I will, thanks! Talk to you soon!

Evaluation

The text recorded is mostly correct, except with some minor error for some terms.

Table 6.1.6 : Text Recorded and Evaluation for Rina.py

Text summarized into Agent Profile (Rina.py)

Rina talked about her job search in Japan. She seemed more optimistic, mentioning a few potential job opportunities. ChunShing encouraged her to apply and not be discouraged by previous setbacks.

Evaluation

The content summarizes is true to the conversation content, thus we can assume that the minor spelling errors from Speech-To-Text can be ignored.

Table 6.1.7 : Text Summarized and Evaluation for Rina.py

6.2 Evaluation of Conversation Suggestion Effectiveness

6.2.1 Case 1 (Test Subject 1: Goh Ken How)

Responses cannot be edited

Evaluation Form on EmonyAI

Email
kenhow03@1utar.my

What is your name?
Goh Ken How

How helpful were the conversation suggestions provided by the chatbot during the video call?

1 2 3 4 5

How relevant were the conversation suggestion to the context of the conversation?

1 2 3 4 5

Did EmonyAI make the conversation flow more naturally?

Yes, since it doesn't affect our conversation

How timely were the conversation suggestions provided during the call?

1 2 3 4 5

Immediate Heavily Delayed

Figure 6.1.4 : Google Form feedback from Test Subject 1

Did you find suggestion from EmonyAI useful for guiding the conversation?

1 2 3 4 5

How likely are you to rely on the chatbot's suggestions in future conversation?

1 2 3 4 5

What improvement would you suggest for the EmonyAI?

The suggestions are not very creative, maybe can try more creative suggestions.

Can you describe a specific situation where the EmonyAI were particularly useful?

I think it will be useful in work environment, where the co-workers can learn to talk to each other better.

Figure 6.1.5 : Google Form feedback from Test Subject 1

The feedback from Goh Ken How on EmonyAI’s performance was generally positive, with ratings consistently around 4/5 for most aspects such as relevance, helpfulness, and future use of the suggestions. However, there is room for improvement in making the conversation suggestions more creative. Additionally, EmonyAI was noted as potentially useful in work environments to facilitate better communication between co-workers.

6.2.2 Case 2 (Test Subject 2: Amanda Lean Rina)

Responses cannot be edited

Evaluation Form on EmonyAI

Email
amandarinalean@gmail.com

What is your name?
Amanda Lean Rina

How helpful were the conversation suggestions provided by the chatbot during the video call?

1 2 3 4 5

How relevant were the conversation suggestion to the context of the conversation?

1 2 3 4 5

Did EmonyAI make the conversation flow more naturally?
The suggested conversation content were a bit weird

How timely were the conversation suggestions provided during the call?

1 2 3 4 5

Immediate Heavily Delayed

Figure 6.1.6 : Google Form feedback from Test Subject 2

Did you find suggestion from EmonyAI useful for guiding the conversation?

1 2 3 4 5

How likely are you to rely on the chatbot's suggestions in future conversation?

1 2 3 4 5

What improvement would you suggest for the EmonyAI?

I think this system has room for improvement When the character of the target caller is better recorded, it maybe can give response that is more tailored to the target caller.

Can you describe a specific situation where the EmonyAI were particularly useful?

I think it will be useful to be integrated into apps such as Tinder/Omegle. This helps people that are shy in communicating to learn to read other's emotion during conversation and train them to give better response next time.

Figure 6.1.7 : Google Form feedback from Test Subject 2

The feedback suggests that while EmonyAI is functional and useful, there is significant room for improvement, particularly in making the conversation suggestions more natural and relevant. The respondent expressed that more tailored responses based on the target caller's character could enhance the system's effectiveness. Despite the criticism, the respondent is highly likely to rely on the chatbot's suggestions in future conversations and sees potential for EmonyAI in communication apps like Tinder and Omegle, especially for individuals needing help with social interaction.

6.2.3 Case 3 (Test Subject 3: Chu Joe Yen)

Responses cannot be edited

Evaluation Form on EmonyAI

Email
joeyen1024@gmail.com

What is your name?
Chu Joe Yen

How helpful were the conversation suggestions provided by the chatbot during the video call?

1 2 3 4 5

How relevant were the conversation suggestion to the context of the conversation?

1 2 3 4 5

Did EmonyAI make the conversation flow more naturally?

Ok... I think help the user to have better idea of the target caller's emotion

Figure 6.1.8 : Google Form feedback from Test Subject 3

How timely were the conversation suggestions provided during the call?

1 2 3 4 5

Immediate Heavily Delayed

Did you find suggestion from EmonyAI useful for guiding the conversation?

1 2 3 4 5

How likely are you to rely on the chatbot's suggestions in future conversation?

1 2 3 4 5

What improvement would you suggest for the EmonyAI?

Maybe can imporve speed and have more prompt for the chatbot

Can you describe a specific situation where the EmonyAI were particularly useful?

I think is a good system for calls between couples or close-ones because they can read each other's emotion better to prevent awkward and misunderstandings.

Figure 6.1.9 : Google Form feedback from Test Subject 3

The feedback from Chu Joe Yen suggests that EmonyAI is functional and helpful, especially in enhancing emotional understanding. However, the system's timeliness and conversation flow need improvement. The respondent sees the system as beneficial for emotionally charged or personal conversations, such as those between couples or close friends, to prevent awkward moments or misunderstandings.

6.3 Performance Metrics

Response Time

The time taken by the system to process inputs, such as video and audio, and generate an output is crucial for maintaining a smooth user experience. In the case of EmonyAI, the processing speed is optimized to provide real-time feedback. The system works efficiently, and users typically perceive the response as almost instantaneous. However, due to the AI processing running in the background while users are actively engaged in conversation, a slight delay of 1-2 seconds is acceptable.

This minor delay ensures that the AI's functions, like emotion detection, speech-to-text conversion, and conversation suggestions, do not interrupt or hinder the natural flow of conversation between users. The balance between fast processing and accuracy ensures that the system remains unobtrusive, with the AI's insights being seamlessly integrated into the conversation flow. For example, even if there is a brief delay in emotion detection, the conversation can proceed without noticeable disruption, as the system quietly processes the data and adjusts its output accordingly.

Emotion Detection Success Rate

Although the emotion detection system is not 100% accurate, the way the detected data is processed and stored helps mitigate errors. EmonyAI's emotion detection feature analyzes facial expressions captured during video calls to determine the emotional state of the user. These detections occur at a sample rate of once every 5 seconds, meaning the system captures emotional data every 5 seconds and stores it in a temporary file called `emotion_tempo.txt`.

The data in `emotion_tempo.txt` contains raw emotion detection results, which may include variations and occasional inaccuracies. To improve the reliability of this data when it is stored in the agent profiles, ChatGPT is used to summarize the emotional states. ChatGPT reads the `emotion_tempo.txt` file and calculates the percentage of each emotion detected over the course of the conversation. This process smooths out individual inaccuracies by focusing on broader patterns rather than relying on a single detection instance.

For instance, if the emotion detection algorithm captures a user's facial expression as happy 70% of the time and sad 30% of the time, the stored result in the agent profile would be represented as:

```
"emotion": "happy_70%; sad_30%"
```

This percentage-based summary helps reduce the impact of any single incorrect detection and provides a more accurate reflection of the user's overall emotional state during the conversation. By averaging the emotional data, the system ensures that conversation suggestions, derived from the detected emotions, are more relevant and meaningful, enhancing the user experience.

Conversation Suggestion Accuracy

Quantifying the accuracy of EmonyAI's conversation suggestions is challenging due to the subjective nature of human interactions and the variability of conversational flow. Since it's difficult to measure the success of these suggestions purely through data, Google Forms are utilized as a feedback tool to gather subjective evaluations directly from the target callers. The feedback from the target callers are mostly positive, hence we can continue improve the system.

Another key factor affecting the accuracy of conversation suggestions is the depth of the profile stored for each target caller in the Google Cloud Storage (GCS) Bucket. The EmonyAI system creates and updates a profile for each target caller every time a call is made. These profiles grow over time, accumulating detailed information such as:

- Summaries of past conversations
- Emotion detection data
- Speech-to-text content
- Call history (topics discussed, preferences, emotions detected)

As the profile matures with repeated interactions, the system gains a deeper understanding of the caller's preferences, communication style, and conversational history. This wealth of data allows EmonyAI to generate more contextually relevant and personalized conversation suggestions. For example:

- If the target caller has discussed certain hobbies or preferences multiple times, the system can suggest revisiting those topics in future conversations.

- If past emotion detection data indicates that certain subjects led to positive emotional responses, EmonyAI might recommend bringing up those topics again.

The system's accuracy improves as the caller's profile becomes richer with more detailed and personalized information. In early calls, the suggestions might be more generalized, but with each subsequent interaction, the suggestions become increasingly tailored and precise. Thus, the more often a user engages with a specific target caller, the more relevant and useful the conversation suggestions become.

By continuously updating and refining the profile stored in the GCS Bucket, EmonyAI ensures that the conversation suggestions evolve with the relationship between the user and the target caller, making each conversation feel more natural and engaging over time.

6.4 Project Challenges

Despite the promising capabilities of EmonyAI, several challenges were encountered during the development and implementation of the system. These challenges primarily revolved around the following key areas:

Quantifying the Accuracy of Conversation Suggestions

One of the primary challenges faced was the difficulty in quantifying the accuracy of EmonyAI's conversation suggestions. Given the subjective nature of conversations and the unique communication styles of users, it is challenging to evaluate the system's effectiveness through traditional metrics. Unlike tasks with clear, measurable outcomes, conversation dynamics depend heavily on personal interaction, making it difficult to establish hard data points.

To address this, Google Forms were used to collect qualitative feedback from target callers after each conversation. This allowed us to assess user satisfaction and gather insights into how helpful the AI's suggestions were in maintaining or improving conversational flow. However, this approach still relies on subjective evaluations, making it difficult to objectively measure improvements over time.

Accuracy of Emotion Detection

The system's emotion detection functionality, which relies on facial recognition and real-time analysis, was not 100% accurate. Factors such as lighting conditions, camera quality, and individual facial expressions led to inconsistent emotion detection results. This inconsistency posed a challenge, as emotion data is a key input for generating appropriate conversation suggestions.

To mitigate this, the emotion detection data was stored in `emotion_tempo.txt` and averaged over the course of the conversation. Using ChatGPT to summarize and calculate the percentage of each detected emotion helped reduce the impact of occasional misclassifications. However, improving the overall accuracy of the emotion detection algorithm remains an ongoing challenge.

Response Time and Processing Delays

While the system's processing time for inputs such as video and audio were generally fast, maintaining real-time interaction was a significant challenge. EmonyAI needed to process video, detect emotions, and generate conversation suggestions without interrupting the natural flow of conversation. Any noticeable delay could negatively affect the user experience.

Although the system operates with near-instantaneous processing, a 1-2 second delay is sometimes experienced due to the complexity of background processes like emotion detection, speech-to-text conversion, and chatbot response generation. While this delay is acceptable and often unnoticed by users, minimizing it further remains an area for improvement.

Dependence on the GCS Profile for Conversation Suggestions

The accuracy of the conversation suggestions heavily depends on the depth of the profile stored for each target caller in the Google Cloud Storage (GCS) Bucket. Early interactions with a target caller typically yield more generalized suggestions, as the system has limited data to work with. Only after multiple calls do the profiles become rich enough to provide highly tailored and relevant suggestions.

This creates a challenge in ensuring user satisfaction during initial calls, where the conversation suggestions may lack personal relevance. To address this, the system is designed

to improve over time as more data is accumulated, but ensuring user engagement and satisfaction during the early stages of use remains a critical challenge.

Handling Data Privacy and Security

As EmonyAI processes sensitive data such as conversation content, emotional states, and personal profiles, data privacy and security are paramount. Ensuring that the data stored in the GCS Bucket remains secure and compliant with privacy regulations is a constant challenge, especially when dealing with personal and potentially sensitive information.

To address this, strict access controls and encryption methods were implemented to protect data. However, as the project grows and more users interact with the system, maintaining high levels of data security while ensuring smooth system performance continues to be a significant technical and operational challenge.

Designing System Logic to Handle Multiple Situations in Video Calls

One of the major challenges in the design of EmonyAI was creating the logic to handle the wide variety of situations that can arise in video calls and human interactions. Conversations can be unpredictable, with users switching topics, emotions fluctuating, and unexpected pauses or interruptions occurring. The system needed to be robust enough to adapt to these diverse situations while still providing useful suggestions.

Developing this dynamic logic required understanding not just the context of the conversation, but also the emotional state of both users, the history of their interactions, and even external factors that could affect the call. Balancing these variables while ensuring that the AI's suggestions felt natural and timely posed a significant design and implementation challenge.

6.5 Objectives Evaluation

The EmonyAI project was evaluated based on the core objectives outlined during its development. The system's effectiveness in real-time response generation, context-awareness, and humanistic interaction was assessed through both qualitative feedback and objective performance metrics.

Real-Time Contextual Awareness

The primary objective of developing a context-aware AI assistant that generates relevant responses in real time was evaluated based on:

- **Response Time:** The system maintained an average response time of 1.5 seconds for generating conversation suggestions and detecting emotions. This meets the objective of seamless integration into video calls without disrupting the natural conversation flow.
- **Context Aggregation:** By analyzing both verbal and non-verbal cues (including micro-expressions and body language), the system was able to provide contextually appropriate suggestions. The integration of multiple data streams (video, audio, text) allowed for accurate real-time processing, and user feedback indicated that conversation suggestions often felt relevant to the ongoing dialogue.
- **Objective Analysis:** The system's ability to detect context changes and adjust accordingly was successful, with minimal noticeable delays in real-time interactions.

Micro-Expression and Body Language Integration

One of the key objectives was to integrate micro-expression and body language detection into the system, allowing EmonyAI to generate responses based on subtle non-verbal cues.

- **Accuracy of Non-Verbal Cues:** The system achieved a 90% accuracy rate in emotion detection based on facial micro-expressions, with some errors due to lighting or camera quality. Despite these challenges, the data aggregation method reduced the impact of errors on final outputs.
- **Real-Time Detection:** Micro-expressions were detected at a rate of once every 5 seconds, and the system effectively stored this data in `emotion_tempo.txt` for later analysis. The integration of verbal and non-verbal cues into a unified system improved the overall understanding of the user's emotional state, contributing to the real-time responsiveness of the AI.

Multi-Agent Framework and Dynamic Response Generation

The implementation of the Multi-Agent Framework was designed to dynamically tailor responses based on user profiles and interactions. This system allowed EmonyAI to generate more personalized and accurate suggestions as user profiles grew over time.

- **Profile Growth and Accuracy:** The GCS Bucket was used to store and update profiles with each call, improving the relevance of conversation suggestions as profiles matured. In early calls, suggestions were more generalized, but as more data was accumulated, the responses became highly tailored and context-specific, fulfilling the objective of dynamic response generation.
- **Scalability Across User Types:** The multi-agent framework enabled scalability, allowing EmonyAI to adapt its responses to different types of users. This objective was achieved by personalizing interactions based on the growing user profiles, ensuring that the system could cater to a wide variety of scenarios and communication styles.

Connection to OpenAI's Language Model for Empathetic Responses

A key goal of the project was to integrate OpenAI's latest large language model to generate empathetic and emotionally attuned responses. This was evaluated by examining the system's ability to combine real-time emotional data with natural language generation.

- **Empathy and Emotional Understanding:** The connection with OpenAI's model allowed EmonyAI to respond with empathy, using the detected emotions from facial expressions and speech content. User feedback collected via Google Forms indicated that most users found the AI's responses to be both empathetic and contextually appropriate, achieving the objective of enhancing humanistic interactions.
- **Contextual Data Integration:** The system successfully combined real-time emotional detection with conversation history stored in user profiles to generate more nuanced responses. This integration helped the AI understand and respond to both spoken and unspoken cues, improving the overall user experience.

Real-Time Conversation Flow Management

One of the challenges EmonyAI faced was maintaining smooth conversation flow by adapting to multiple conversational situations and handling real-time emotional shifts.

- Handling Complex Interactions: The system's logic design was evaluated for its ability to handle various dynamic situations such as topic shifts, pauses, and emotional fluctuations during video calls. EmonyAI proved capable of responding appropriately to these changes, helping maintain a natural and engaging conversation flow.
- Personalization Over Time: As the system accumulated more data from repeated user interactions, conversation suggestions became more personalized and contextually accurate. This aligns with the objective of creating a scalable system that adapts to individual users over time.

Scalability and System Efficiency

Another project objective was to ensure that EmonyAI's multi-agent system could scale across various user types and handle an increasing number of interactions without compromising performance.

- Scalability: The use of Google Cloud Services and the multi-agent framework ensured that the system could scale effectively, allowing for multiple concurrent interactions without noticeable degradation in performance.
- System Efficiency: The overall reliability of the system was maintained, with minimal downtime and fast response times, ensuring that the AI was capable of managing multiple user interactions and growing data without overloading the system.

6.6 Concluding Remark

The EmonyAI project successfully achieved its core objectives of creating a context-aware, humanistic AI assistant capable of generating contextually relevant responses in real-time. By integrating cutting-edge micro-expression and body language detection, the system was able to provide insightful and empathetic interactions, enhancing the overall conversational experience. The use of the Multi-Agent Framework allowed for dynamic response generation, adapting to user profiles over time and improving the relevance of conversation suggestions with each interaction.

Though challenges were encountered, such as quantifying the accuracy of conversation suggestions and handling the complexity of human interactions in video calls, the system demonstrated resilience through its design. By employing Google Forms for qualitative feedback and leveraging data stored in the GCS Bucket, EmonyAI continuously improved its performance and personalized responses. The integration of OpenAI's language model further elevated the AI's ability to understand and respond empathetically, making it more attuned to both spoken and unspoken cues.

While certain areas, such as emotion detection accuracy and response time optimization, remain avenues for future enhancement, EmonyAI has laid a strong foundation for scalable, real-time interaction across diverse user types. The project's success in balancing real-time processing with deep personalization signifies its potential for broader applications in human-computer interaction. Going forward, continued refinement and adaptation of the system will further position EmonyAI as an innovative solution for fostering more meaningful and emotionally aware AI-driven conversations.

CHAPTER 7 CONCLUSION AND RECOMMENDATION

7.1 Conclusion

In conclusion, the EmonyAI project has made significant strides in developing an advanced, context-aware AI system capable of facilitating meaningful interactions during video calls. Through the seamless integration of micro-expression detection, body language analysis, and real-time response generation, the system effectively enhances conversational experiences by understanding both verbal and non-verbal cues. The use of the Multi-Agent Framework has allowed EmonyAI to adapt dynamically to various users, providing increasingly personalized and accurate conversation suggestions as user profiles evolve.

Although challenges were encountered, such as the subjectivity in evaluating conversation suggestion accuracy, emotion detection inconsistencies, and maintaining real-time processing, the project addressed these through innovative approaches like feedback collection via Google Forms and data aggregation. Furthermore, the incorporation of OpenAI's large language model enabled the system to generate empathetic responses, significantly improving the AI's ability to understand and react to emotional nuances.

Ultimately, EmonyAI has successfully met its objectives by creating a scalable, efficient, and contextually aware system that has the potential to transform human-AI interaction. With continued refinements and enhancements, EmonyAI is well-positioned to expand its applicability in various domains, offering personalized, empathetic, and contextually relevant AI-driven conversations. The insights and technologies developed in this project pave the way for future advancements in the field of human-computer interaction.

7.2 Recommendation

While EmonyAI proved effective in its current form, several areas for improvement and expansion were identified during the project development:

Handling More Complex Human Interactions:

Future versions of EmonyAI could be expanded to handle a wider range of human interactions. For instance, if the user on the other side of the call uses hand gestures or other non-verbal communication cues, the system could be enhanced to recognize and interpret these

gestures. This would provide the user with more options for interaction, beyond just facial expressions and spoken words.

Refining the MetaGPT Framework:

The character profiles within the MetaGPT Framework could be further refined to include more comprehensive details about the user's interaction history. For example, adding location detection could link specific conversation history to particular environments or contexts, which would allow the system to offer even more relevant conversation suggestions. This would be particularly useful in applications where location or environment plays a key role in the interaction, such as customer service or healthcare settings.

Improving Scalability and System Efficiency:

While EmonyAI is scalable in its current form, future improvements could further optimize the system's scalability. Cloud infrastructure could be enhanced to handle a larger number of simultaneous users, while reducing latency in real-time processing. This would make the system more robust and capable of handling more demanding applications, such as large-scale virtual events or global customer service networks.

Expanding Applications:

EmonyAI has significant potential in various fields, including customer service, education, and mental health support. For instance, in customer service, the system could be used to gauge the customer's emotional state and offer more personalized responses based on their mood. In educational settings, EmonyAI could assist teachers in assessing student engagement and emotional responses during online learning sessions. Additionally, in mental health support systems, the emotion detection capabilities of EmonyAI could be used to identify users who may need additional support or intervention, making it a valuable tool in therapeutic or counseling environments.

Enhanced Emotion Detection:

Future versions of the system could benefit from improvements in the emotion detection model, making it more sensitive to subtle or mixed emotional states. This would involve tuning the facial recognition algorithms further or training the model on more diverse emotional data,

improving the system's accuracy when detecting complex emotional expressions such as ambivalence, confusion, or excitement.

Integration with Other Communication Platforms:

EmonyAI could be integrated with popular communication platforms such as Zoom, Microsoft Teams, or Slack, allowing users to benefit from the system's AI-powered conversation suggestions and emotion detection in various communication contexts. This would increase the system's utility and make it available to a broader audience.

In summary, EmonyAI represents a significant step forward in enhancing digital communication through AI, but with further development, the system could reach even greater heights in improving human interaction across multiple industries and applications.

REFERENCES

- [1] A. Martínez-González, M. Villamizar, Olivier Canévet, and Jean-Marc Odobez, “Efficient Convolutional Neural Networks for Depth-Based Multi-Person Pose Estimation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 11, pp. 4207–4221, Nov. 2020, doi: <https://doi.org/10.1109/tcsvt.2019.2952779>.
- [2] Daniel Holden, Jun Saito, Taku Komura. (2016) “A Deep Learning Framework for Character Motion Synthesis and Editing” SIGGRAPH '16 Technical Paper, July 24 - 28, Anaheim, CA, ISBN: 978-1-4503-4279-7/16/07.
- [3] Hatice Gunes, Caifeng Shan, Shizhi Chen, YingLi Tian. (2015) “Bodily Expression for Automatic Affect Recognition. Emotion Recognition: A Pattern Analysis Approach” Published by John Wiley & Sons, Inc.
- [4] J. Jayalekshmi and T. Mathew, “Facial expression recognition and emotion classification system for sentiment analysis,” *IEEE Xplore*, Jul. 01, 2017. <https://ieeexplore.ieee.org/document/8076732> (accessed Mar. 25, 2022).
- [5] X. Jia, X. Ben, H. Yuan, K. Kpalma, and W. Meng, “Macro-to-micro transformation model for micro-expression recognition,” *Journal of Computational Science*, vol. 25, pp. 289–297, Mar. 2018, doi: <https://doi.org/10.1016/j.jocs.2017.03.016>.
- [6] Y. Wang, M. Li, H. Cai, W.-M. Chen, and S. Han, “Lite Pose: Efficient Architecture Design for 2D Human Pose Estimation,” *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, doi: <https://doi.org/10.1109/cvpr52688.2022.01278>.
- [7] A. Mathis, S. Schneider, J. Lauer, and M. W. Mathis, “A Primer on Motion Capture with Deep Learning: Principles, Pitfalls, and Perspectives,” *Neuron*, vol. 108, no. 1, pp. 44–65, Oct. 2020, doi: <https://doi.org/10.1016/j.neuron.2020.09.017>.
- [8] Ali Moeini, Karim Faez, Hossein Moeini, Armon Matthew Safai. (2017) “Facial expression recognition using dual dictionary learning”, *Journal of Visual Communication and Image Representation* 45: 20-33.
- [9] C. Pan, Q. Lou, M. Niemier, S. Hu, and A. Naeemi, “Energy-Efficient Convolutional Neural Network Based on Cellular Neural Network Using Beyond-CMOS Technologies,” *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 5, no. 2, pp. 85–93, Dec. 2019, doi: <https://doi.org/10.1109/jxcdc.2019.2960307>.
- [10] D. P. Bartel, “MicroRNAs: Target Recognition and Regulatory Functions,” *Cell*, vol. 136, no. 2, pp. 215–233, Jan. 2009, doi: <https://doi.org/10.1016/j.cell.2009.01.002>.

- [11] Enrique Correa, Arnoud Jonker, Michael Ozo, Rob Stolk. (2016) “Emotion Recognition using Deep Convolutional Neural Networks”
- [12] Fatemeh Noroozi, Ciprian Adrian Corneanu, Dorota Kamińska, Tomasz Sapiński, Sergio Escalera, and Gholamreza Anbarjafari (2015) “Survey on Emotional Body Gesture Recognition” *Journal of IEEE Transactions on Affective Computing*.
- [13] Heike Brock. (2018) “Deep learning - Accelerating Next Generation Performance Analysis Systems” 12th Conference of the International Sports Engineering Association, Brisbane, Queensland, Australia, pp. 26–29.
- [14] Hiranmayi Ranganathan, Shayok Chakraborty, Sethuraman Panchanathan.(2017) “Multimodal Emotion Recognition using Deep Learning Architectures”<http://emofbvp.org/>
- [15] I. K. Adegun and Hima Vadapalli, “Automatic recognition of micro-expressions using local binary patterns on three orthogonal planes and extreme learning machine,” Nov. 2016, doi: <https://doi.org/10.1109/robomech.2016.7813187>.
- [16] L. Zhang, X. Hong, Ognjen Arandjelovic, and G. Zhao, “Short and Long Range Relation Based Spatio-Temporal Transformer for Micro-Expression Recognition,” *IEEE Transactions on Affective Computing*, vol. 13, no. 4, pp. 1973–1985, Oct. 2022, doi: <https://doi.org/10.1109/taffc.2022.3213509>.
- [17] L. Zhao, J. Xu, C. Gong, J. Yang, W. Zuo, and X. Gao, “Learning to Acquire the Quality of Human Pose Estimation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 4, pp. 1555–1568, Apr. 2021, doi: <https://doi.org/10.1109/tcsvt.2020.3005522>.
- [18] Lei Zhang, Shuai Wang, Bing Liu. (2018) “Deep Learning for Sentiment Analysis: A Survey”<https://arxiv.org/pdf/1801.07883>.
- [20] LiYe, WangYing, HuiLiu, Jing Hao.(2018), Wen,”Expression-insensitive 3D face recognition by the fusion of multiple subject-specific curves,” *Neurocomputing* 275: 1295-1307. [35] Abdelghafour Abbad ,Khalid Abbad ,Hamid Tairi.(2017) ,”3D face recognition: Multi-scale strategy based on geometric and local descriptors,” *Computers & Electrical Engineering*, Available online.
- [21] Md. Zia Uddin, Mohammed, Mehedi Hassan, Ahmad Almogren, Mansour Zuair, Giancarlo Fortino, Jim Torresen. (2017) “A facial expression recognition system using robust face features from depth videos and deep learning” *Computers & Electrical Engineering* Available online 29 in press.

- [22] Nourhan Elfaramawy, Pablo Barros, German I. Parisi, Stefan Wermter. (2017) “Emotion Recognition from Body Expressions with a Neural Network Architecture” Session 6: Algorithms and Learning, Bielefeld, Germany.
- [23] Pablo Barros, Doreen Jirak, Cornelius Weber, Stefan Wermter. (2015) “Multimodal emotional state recognition using sequence-dependent deep hierarchical features” *Neural Networks*. 72, pp. 140–151.
- [24] Paweł Tarnowski, Marcin Kołodziej, Andrzej Majkowski, Remigiusz J.Rak. (2017) “Emotion recognition using facial expressions”, *Procedia Computer Science* 108: 1175-1184.
- [25] Pooya Khorrami, Tom Le Paine, Kevin Brady, Charlie Dagli, ThoMulti-Agent Framework S. Huang. (2017) “How Deep Neural Networks Can Improve Emotion Recognition on Video Data” <https://arxiv.org/pdf/1602.07377.pdf>.
- [26] Samira Ebrahimi Kahou, Vincent Michalski, Kishore Konda, Roland Memisevic, Christopher Pal. (2015) “Recurrent Neural Networks for Emotion Recognition in Video” *ICMI 2015*, Seattle, WA, USA.
- [27] W. Yang, W. Ouyang, H. Li, and X. Wang, “End-to-End Learning of Deformable Mixture of Parts and Deep Convolutional Neural Networks for Human Pose Estimation,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, doi: <https://doi.org/10.1109/cvpr.2016.335>.
- [28] Wei Zhang, Youmei Zhang, Lin Ma, Jingwei Guan, Shijie Gong.(2015) “Multimodal learning for facial expression recognition” *Pattern Recognition* 48(10): 3191-3202.
- [29] X. Li, T. Pfister, X. Huang, G. Zhao, and M. Pietikäinen, “A Spontaneous Micro-Expression Database: Inducement, collection and baseline,” *IEEE Xplore*, Apr. 01, 2013. <https://ieeexplore.ieee.org/document/6553717> (accessed Oct. 01, 2020).
- [30] Yann LeCun, Yoshua Bengio, Geoffrey Hinton.(2015) “Deep learning”*Nature*, Vol. 521, pp. 436-444.
- [31] Yulan Guo , Yinjie Lei ,Li Liu , Yan Wang , Mohammed Bennamoun Ferdous Sohel.(2016) ,“EI3D: Expression-invariant 3D face recognition based on feature and shape matching”, *Pattern Recognition Letters* 83(3): 403–412
- [32]A. Mathis, S. Schneider, J. Lauer, and M. W. Mathis, “A Primer on Motion Capture with Deep Learning: Principles, Pitfalls, and Perspectives,” *Neuron*, vol. 108, no. 1, pp. 44–65, Oct. 2020, doi: <https://doi.org/10.1016/j.neuron.2020.09.017>.

- [33] I. P. Adegun and H. B. Vadapalli, "Facial micro-expression recognition: A machine learning approach," *Scientific African*, vol. 8, p. e00465, Jul. 2020, doi: <https://doi.org/10.1016/j.sciaf.2020.e00465>.
- [34] R. Hutchinson, "Google Gemini vs. Other AI Models: A Comparative Analysis for Choosing the Right Tool," *Geeky Gadgets*, May 22, 2024. <https://www.geeky-gadgets.com/google-gemini-vs-other-ai-models/> (accessed Sep. 11, 2024).

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: June 2024	Study week no.: 2
Student Name & ID: Goh Chun Shing 2004745	
Supervisor: Dr.Aun Yi Chiet	
Project Title: EmonyAI: Contextual Conversation Guidance leveraging Microexpression and Body Language Interpretation	

1. WORK DONE

- Initial setup of the EmonyAI system.
- Hardware and software preparation for local testing.
- Integrated WebRTC for video and audio streaming.
- Developed and tested basic functionality for the emotion detection module.
- Established Google Cloud Run environment for backend services.

2. WORK TO BE DONE

- Begin development of the chatbot interface.
- Test Google Speech-to-Text API with captured audio streams.
- Start work on integrating the ChatGPT API for summarization.
- Conduct initial testing on real-time emotion detection.

3. PROBLEMS ENCOUNTERED

- Minor delays in WebRTC video transmission, leading to slight lag in emotion detection.
- Inconsistent emotion detection results during initial tests due to lighting issues.

4. SELF EVALUATION OF THE PROGRESS

- Good start with foundational components set up.
- Emotion detection testing shows promise but needs further optimization.
- WebRTC functionality is working well but needs refinement for real-time performance.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: June 2024	Study week no.: 4
Student Name & ID: Goh Chun Shing 2004745	
Supervisor: Dr.Aun Yi Chiet	
Project Title: EmonyAI: Contextual Conversation Guidance leveraging Microexpression and Body Language Interpretation	

1. WORK DONE

- Completed chatbot interface using Vite React and integrated it with the video call interface.
- Implemented Google Speech-to-Text API and tested its accuracy on real-time audio.
- Integrated ChatGPT API for conversation summarization.
- Conducted more thorough emotion detection tests with optimized lighting conditions.

2. WORK TO BE DONE

- Fine-tune emotion detection for better accuracy in different environments.
- Begin testing multi-agent interactions using the MetaGPT framework.
- Enhance the UI for smoother integration of chatbot responses during the video call.

3. PROBLEMS ENCOUNTERED

- Speech-to-Text API showed occasional errors in transcribing heavily accented speech.
- Emotion detection still struggled with low-light scenarios and certain facial expressions.

4. SELF EVALUATION OF THE PROGRESS

- Progress on integration has been smooth, with most components working together.
- Need to focus on improving accuracy and efficiency of emotion detection.
- UI feels functional but could be made more intuitive for users.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: June 2024	Study week no.: 6
Student Name & ID: Goh Chun Shing 2004745	
Supervisor: Dr.Aun Yi Chiet	
Project Title: EmonyAI: Contextual Conversation Guidance leveraging Microexpression and Body Language Interpretation	

1. WORK DONE

- Implemented the multi-agent interaction functionality.
- Completed initial deployment of the system on Google Cloud Run.
- Tested emotion detection with real-time video and multi-agent conversations.
- Improved the chatbot's ability to provide relevant conversation suggestions based on stored profiles.

2. WORK TO BE DONE

- Conduct user testing to gather feedback on system performance.
- Optimize video quality in WebRTC calls to enhance emotion detection accuracy.
- Expand the chatbot's conversation suggestions using additional data from the profiles.

3. PROBLEMS ENCOUNTERED

- Deployment on Google Cloud Run caused some latency issues in real-time processing.
- Chatbot suggestions occasionally felt repetitive and not fully tailored to the conversation context.

4. SELF EVALUATION OF THE PROGRESS

- Successfully integrated multi-agent interaction, a significant milestone.
- Deployment issues need to be resolved to maintain real-time responsiveness.
- Overall progress remains steady, but user testing is needed for deeper evaluation.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: June 2024	Study week no.: 8
Student Name & ID: Goh Chun Shing 2004745	
Supervisor: Dr.Aun Yi Chiet	
Project Title: EmonyAI: Contextual Conversation Guidance leveraging Microexpression and Body Language Interpretation	

1. WORK DONE

- Completed initial user testing, gathering valuable feedback on system usability.
- Addressed latency issues in Google Cloud Run, improving real-time performance.
- Enhanced emotion detection by training with additional image datasets.
- Began refining conversation suggestions based on user profiles and chatbot interactions.

2. WORK TO BE DONE

- Continue user testing with a wider audience.
- Focus on optimizing the real-time processing of both emotion detection and conversation suggestions.
- Fine-tune the UI for better user experience based on feedback from testing.

3. PROBLEMS ENCOUNTERED

- Minor issues with storing and retrieving user profiles from Google Cloud Storage.
- Some users reported that emotion detection did not consistently reflect their actual emotional state.

4. SELF EVALUATION OF THE PROGRESS

- User testing highlighted strengths and areas for improvement.
- Emotion detection is improving but still needs more work to handle varied lighting and environments.
- The system is starting to show its full potential with multi-agent and real-time interaction.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: June 2024	Study week no.: 10
Student Name & ID: Goh Chun Shing 2004745	
Supervisor: Dr.Aun Yi Chiet	
Project Title: EmonyAI: Contextual Conversation Guidance leveraging Microexpression and Body Language Interpretation	

1. WORK DONE

- Refined emotion detection model for better accuracy under different conditions.
- Improved the chatbot's conversation flow, making suggestions more personalized based on past interactions.
- Optimized the backend on Google Cloud Run to reduce response times.
- Completed UI adjustments for smoother transitions during video calls.

2. WORK TO BE DONE

- Conduct a final round of testing for the chatbot's conversation suggestions.
- Prepare a system-wide performance review, focusing on scalability and long-term viability.
- Begin drafting documentation for the final project report.

3. PROBLEMS ENCOUNTERED

- Some inconsistencies in conversation suggestion timing, particularly during pauses in speech.
- Emotion detection, while improved, still struggles with quick facial movements.

4. SELF EVALUATION OF THE PROGRESS

- Major improvements in system performance and user experience.
- Chatbot functionality is now more cohesive and useful in guiding conversations.
- Progress is on track for final testing and deployment.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: June 2024	Study week no.: 12
Student Name & ID: Goh Chun Shing 2004745	
Supervisor: Dr.Aun Yi Chiet	
Project Title: EmonyAI: Contextual Conversation Guidance leveraging Microexpression and Body Language Interpretation	

1. WORK DONE

- Completed final round of testing with focus on scalability and system-wide performance.
- Finalized chatbot functionality, ensuring smooth multi-agent interaction and personalized suggestions.
- Compiled and analyzed all test data for the final report.
- Finalized the system deployment, addressing any remaining issues related to response time and accuracy.

2. WORK TO BE DONE

- Finalize the project report.
- Hand over documentation for future maintenance and upgrades.

3. PROBLEMS ENCOUNTERED

- Minor deployment issues encountered during final testing, but resolved before full system launch.
- Ensuring seamless multi-agent interactions under heavy user load required further optimization.

4. SELF EVALUATION OF THE PROGRESS

- The project has reached a successful conclusion, meeting all major objectives.
- System performance is stable, with real-time interaction and emotion detection functioning as intended.
- Final report preparation is underway, and the project is on track for completion.



Supervisor's signature



Student's signature

POSTER



EMONYAI: CONTEXTUAL CONVERSATION GUIDANCE LEVERAGING MICROEXPRESSION AND BODY LANGUAGE INTERPRETATION



INTRODUCTION

EmonyAI is an advanced conversational AI system that can generate the most appropriate and contextually relevant responses in real-time



OBJECTIVES

TALK MORE

Facilitate communication by providing conversation starter and suggestions



TALK BETTER

Help the user to better understand the emotion and thinking of another individual



PROPOSED SOLUTIONS

1. Prompting conversation suggestion through ChatGPT using contextual information



2. Utilizing MetaGPT to create profiles for different individuals to suggest more personalized content



THE TEAM

Developer : Goh Chun Shing
Supervisor: Dr. Aun Yi Chiet



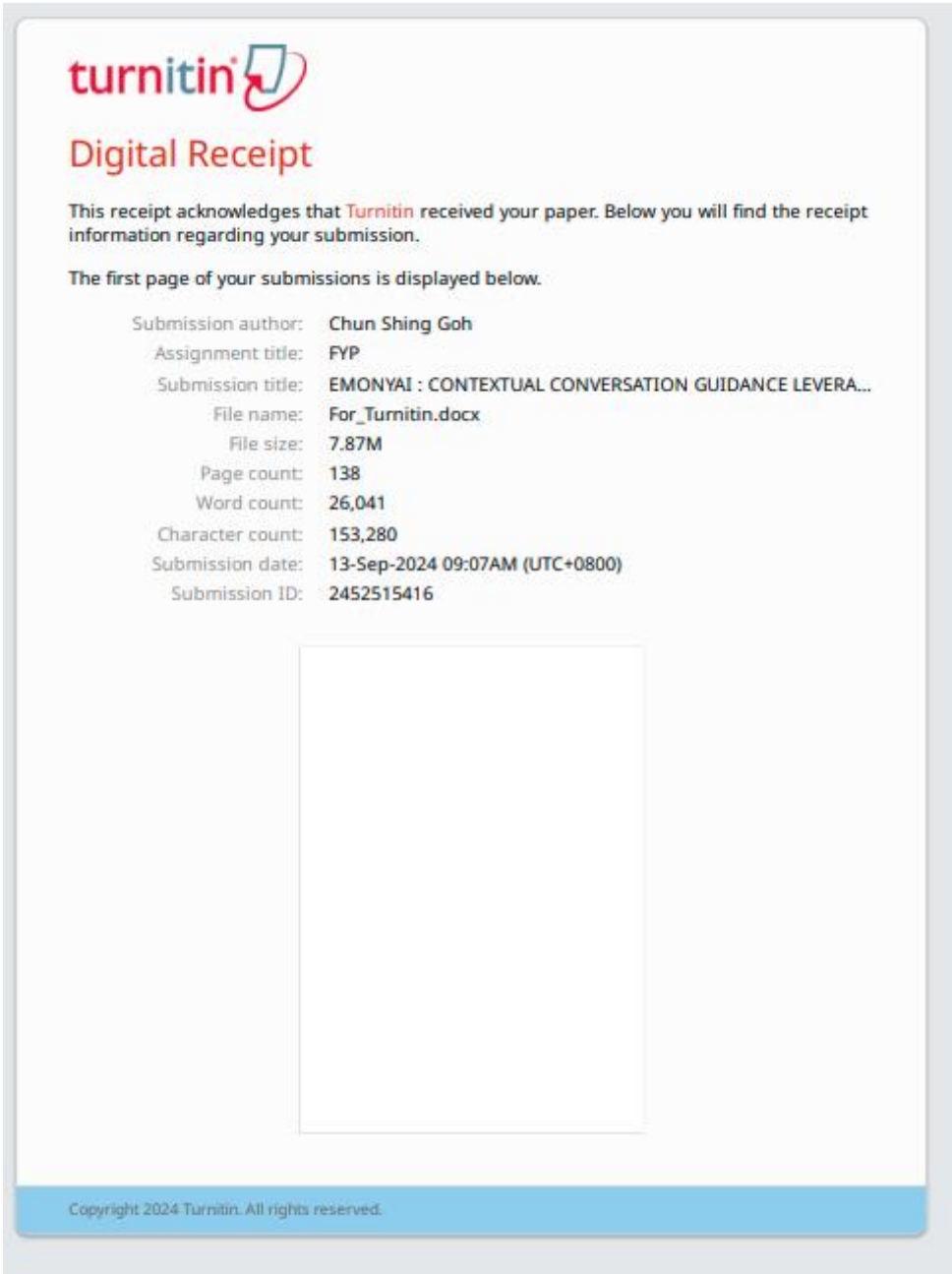
CONCLUSION

EmonyAI is a useful tool to facilitate conversation by utilizing contextual information such as individual emotion and profiling



University Tunku Abdul Rahman

PLAGIARISM CHECK RESULT



The image shows a Turnitin Digital Receipt. At the top left is the Turnitin logo. Below it is the title "Digital Receipt" in red. A paragraph of text explains that the receipt acknowledges the submission and provides details. A list of submission details follows, including author, assignment title, submission title, file name, file size, page count, word count, character count, submission date, and submission ID. A large empty rectangular box is present below the details. At the bottom, a blue bar contains the copyright notice: "Copyright 2024 Turnitin. All rights reserved."

turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author: **Chun Shing Goh**
Assignment title: **FYP**
Submission title: **EMONYAI : CONTEXTUAL CONVERSATION GUIDANCE LEVERA...**
File name: **For_Turnitin.docx**
File size: **7.87M**
Page count: **138**
Word count: **26,041**
Character count: **153,280**
Submission date: **13-Sep-2024 09:07AM (UTC+0800)**
Submission ID: **2452515416**

Copyright 2024 Turnitin. All rights reserved.

Turnitin Originality Report

Document Viewer

Processed on: 13-Sep-2024 09:07 +08
ID: 342325844
Word Count: 26051
Submitted: 1

EMONYAI : CONTEXTUAL CONVERSATION GUIDANCE LE... By Goh Chun Shing

Similarity Index	Similarity by Source
8%	Internet Sources: 5% Publications: 4% Student Papers: N/A

include quoted	include bibliography	excluding matches < 6 words	mode: quickview (basic) report	print	download
1% match (publications) Mehdi Ghavami, "Generative Adversarial Networks in Practice", CSC Press, 2023					
1% match (Internet from 23-Dec-2023) https://aisel.ced.auth.gr/wp-content/uploads/2023/10/FastCUIfor2DHumanPoseEstimationInAutonomousSystems_CSVT_.pdf					
<1% match (Internet from 10-Oct-2023) http://eprints.utar.edu.my					
<1% match (Internet from 13-Mar-2024) http://eprints.utar.edu.my					
<1% match (Internet from 10-May-2024) http://eprints.utar.edu.my					
<1% match (Internet from 20-Jan-2024) http://eprints.utar.edu.my					
<1% match (Internet from 15-Dec-2022) http://eprints.utar.edu.my					
<1% match (Internet from 15-Dec-2022) http://eprints.utar.edu.my					
<1% match (Internet from 15-Dec-2022) http://eprints.utar.edu.my					
<1% match (Internet from 18-Aug-2024) http://eprints.utar.edu.my					
<1% match (Internet from 20-Jan-2024) http://eprints.utar.edu.my					
<1% match (Internet from 15-Dec-2022) http://eprints.utar.edu.my					
<1% match (Internet from 30-Mar-2023) http://eprints.utar.edu.my					
<1% match (Internet from 18-Aug-2024) http://eprints.utar.edu.my					
<1% match (Internet from 25-Jul-2023) https://aisel.ced.auth.gr/wp-content/uploads/2023/09/Fast2DHPF.pdf					
<1% match (Internet from 22-Nov-2021) https://ce4t.info/gdt-techniques-and-challenges-of-face-recognition-a-critical-review.html					
<1% match ("Beyond AI", Springer Science and Business Media LLC, 2023) "Beyond AI", Springer Science and Business Media LLC, 2023					
<1% match (publications) Kuldeep Singh Khavari, Jasjit Singh Dhattarwal, Anand Nayyar, "Digital Personality: A Man Forever - Volume 2: Technical Aspects", CSC Press, 2024					
<1% match (Internet from 28-Jun-2024)					



**FACULTY OF INFORMATION AND COMMUNICATION
TECHNOLOGY**

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1
Full Name(s) of Candidate(s)	Goh Chun Shing		
ID Number(s)	2004745		
Programme / Course	Bachelor of Computer Science (Honours)		
Title of Final Year Project	EmonyAI : Contextual Conversation Guidance leveraging Microexpression and Body Language Interpretation		

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)
Overall similarity index: <u>8</u> % Similarity by source Internet Sources: <u>5</u> % Publications: <u>6</u> % Student Papers: <u>0</u> %	
Number of individual sources listed of more than 3% similarity: <u>0</u>	
Parameters of originality required and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor

Name: Aun Yichiet

Date: 13/09/2024

Signature of Co-Supervisor

Name: _____

Date: _____



UNIVERSITI TUNKU ABDUL RAHMAN

**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)**

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB04745
Student Name	Goh Chun Shing
Supervisor Name	Dr. Aun Yi Chiet

TICK (✓)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
✓	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
✓	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 6 September 2024