Thesis

Vietnamese - German University

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Apply algorithms for chatbot psychology

Author: Le Duc Nghia

Program: Bachelor of Science (B.Sc.) Computer Science and Engineering Student ID: 14706

Intake: 2018 - 2023

Supervisor: Dr. Tran Hong Ngoc

Co-Supervisor: Dr. Dinh Quang Vinh

Intentionally left blank

October 31, 2023

Contents

List of Figures

Declaration

I affirm that this bachelor's thesis, submitted jointly to the Vietnamese-German University and Frankfurt University of Applied Sciences, represents my own original work except where duly cited. I further attest that I have not previously published this thesis, either wholly or in part, at any other academic institutions or universities.

Acknowledgements

The successful completion of this bachelor's thesis could not have been accomplished without the invaluable support and guidance of numerous individuals. I wish to extend my sincere gratitude to all those who assisted me throughout this research project.

In particular, I am deeply indebted to Professor Tran Hong Ngoc and Professor Dinh Quang Vinh, my esteemed thesis supervisors, for their expert direction and constant encouragement during this endeavor. Their insightful feedback and steadfast mentorship were instrumental in shaping and enriching this body of work.

Additionally, I owe profound thanks to my cherished friends and family for their unwavering love and reassurance, which inspired me to persevere during challenging times. I also wish to recognize the devoted professors, faculty, and staff at the Vietnamese-German University who laid the educational foundation that equipped me to take on this ambitious thesis.

The opportunity to pursue significant research and make a meaningful contribution in my chosen field of study was a privilege made possible through the support of many. I am sincerely grateful to have been surrounded by such inspiring and dedicated individuals during this rewarding chapter of personal and academic growth.

Abstract

This thesis explores the development of an artificial intelligence-powered chatbot to provide mental health support for individuals struggling with depression in Vietnam. As mental health gains recognition in Vietnam, innovative solutions like conversational agents can expand access to therapy. The chatbot proposed leverages capabilities including natural language processing, voice and image sentiment analysis, and psychology-focused language models to understand users and offer personalized guidance. The methodology centered on Python tools like PyCharm, Tkinter, and FlowiseAI to create an engaging interface and customized language learning model. Training resources in psychological domains equipped the chatbot to comprehend key concepts and advise users. Additional features like emotion recognition and adaptive dialogues allowed nuanced responses aligned to users' mental states. Overall, the thesis demonstrates chatbots' potential in enhancing mental healthcare availability through customized interactions. The solution aims to bring innovation to a vital but overlooked area in Vietnam. Further development could continue improving the chatbot's capabilities in providing empathetic therapy.

Chapter 1

Introduction

Mental health is a prominent aspect of human well-being that affects individuals, communities, and societies as a whole. In recent years, there has been a growing recognition of the importance of mental health in developed countries such as the United States, Europe, and Japan. These nations have prioritized the development of psychology and mental health services to enhance the quality of life for their citizens.

In contrast, countries like Vietnam have traditionally placed more emphasis on economic growth rather than mental health. Over the past 10-20 years, Vietnam has experienced significant economic development and progress, which has led to improvements in various aspects of people's lives. However, mental health has often been overlooked and considered a distant concern.

Nevertheless, as Vietnam's economy continues to grow and its citizens experience an improvement in their overall living standards, there is now a noticeable shift in the country's perspective towards mental health. There is a growing interest in understanding and addressing mental health challenges among the population. This change in attitude reflects an increasing awareness of the importance of mental well-being and its impact on individuals, families, and communities.

In light of these developments, the need for effective mental health therapy and support services has become evident. Healthcare professionals play a crucial role in addressing the mental health needs of individuals, and they require adequate tools and resources to provide effective care. This is where

technology, such as chat bots, can play a significant role.

This chat bot aims to assist healthcare professionals in providing mental health therapy and improving the quality of life for individuals. By leveraging the capabilities of artificial intelligence and natural language processing, this chat bot can engage in meaningful conversations, offer support, and provide valuable insights into mental well-being. Additionally, the chatbot can contribute to raising awareness about the importance of mental health and promoting a happier and healthier society.

In the following sections, we will explore the challenges and opportunities surrounding mental health in Vietnam, the role of psychology in understanding and addressing mental health issues, and the potential benefits of incorporating technology, such as chat bots, in mental health care. By examining these aspects, we aim to shed light on the importance of mental health and the potential of innovative solutions in improving the well-being of individuals and society as a whole.

Chapter 2

Methodology

2.1 Important Features

2.1.1 Pycharm

PyCharm is a popular integrated development environment (IDE) designed specifically for Python programming. Developed by JetBrains, it offers a range of features to enhance the Python development experience. With its user-friendly interface, PyCharm provides advanced code editing capabilities, including code completion, refactoring tools, and syntax highlighting. It also supports version control integration, allowing developers to manage their code repositories seamlessly. PyCharm's debugging and testing features help identify and resolve issues in the code efficiently. Additionally, it offers virtual environment support, enabling developers to manage project dependencies effectively. Overall, PyCharm is a powerful IDE that simplifies Python development tasks and boosts productivity (Figure 2.1).

Figure 2.1: Pycharm

2.1.2 Tkinter

Tkinter is a widely adopted Python library that facilitates the development of graphical user interfaces (GUIs). With Tkinter, developers can create intuitive and visually appealing desktop applications by leveraging its comprehensive set of tools and widgets. Whether it's designing windows, buttons, menus, or handling user interactions, Tkinter offers a simple and intuitive API for building interactive applications (Figure 2.2).

A notable advantage of Tkinter is its cross-platform compatibility, allowing applications to run seamlessly on various operating systems. This versatility makes Tkinter a reliable choice for developers seeking to reach a broader user base. Moreover, Tkinter's extensibility empowers developers to customize the appearance of widgets, create their own custom widgets, and integrate additional functionalities using third-party libraries .

Figure 2.2: Tkinter

2.1.3 Huggingface

Hugging Face is a renowned organization and platform specialized in natural language processing (NLP) and machine learning. They have made significant contributions to the advancement and promotion of NLP models, datasets, and tools (Figure 2.3).

The Hugging Face platform provides a vast array of resources catered to NLP practitioners and researchers. Notably, they have developed the Transformers library, an open-source library built on PyTorch and TensorFlow. Transformers encompass a collection of pre-trained models, including popular ones such as BERT, GPT, and RoBERTa, which have demonstrated impressive capabilities in various NLP tasks like text classification, language translation, and text generation .

Figure 2.3: Huggingface

2.1.4 OpenCV

OpenCV (Open Source Computer Vision Library) is a widely-used open-source computer vision and machine learning software library. It provides a comprehensive set of tools and functions for image and video processing, object detection and recognition, and various computer vision tasks (Figure 2.4).

One of the key strengths of OpenCV is its robust support for real-time computer vision applications. It provides modules for video capturing, video streaming, and video analysis, enabling developers to build applications that process and analyze video streams in real-time. OpenCV also includes pre-trained models and methods for object detection, facial recognition, and tracking, making it valuable for tasks like surveillance, augmented reality, and robotics.

Figure 2.4: OpenCV

2.1.5 Kaggle

Kaggle is an online community and platform that provides a collaborative environment for data scientists, machine learning practitioners, and researchers to work on and solve complex data-related challenges. It hosts a wide range of datasets, competitions, and notebooks, allowing users to explore, analyze, and build models on real-world data.

Kaggle also serves as a comprehensive repository of datasets. Users can access and download a vast collection of public datasets, spanning various domains and sizes. This provides a valuable resource for data exploration, model development, and research. (Figure 2.5) Furthermore, Kaggle offers a cloud-based computational environment known as Kaggle Kernels. Kernels allow users to write and execute code in popular programming languages like Python and R, with built-in support for popular data science libraries. Kernels facilitate collaborative coding, where users can share their code, analysis, and findings with the community.

The Kaggle community is highly active and supportive, fostering collaboration and knowledge sharing. Users can engage in discussion forums, ask questions, and share insights, enabling a vibrant exchange of ideas.

Figure 2.5: Kaggle

2.1.6 Flowise AI

FlowiseAI is an open-source low-code/no-code drag and drop tool that allows users to build customized Language Learning Models (LLMs). It provides a user-friendly interface for visualizing and creating LLM apps without requiring extensive coding knowledge. Here is some key information about FlowiseAI: Features of FlowiseAI: (Figure 2.6)

• Drag & Drop Interface: FlowiseAI offers a visual interface that allows users to easily create workflows by dragging and dropping components.

•Customization: Users can customize their LLM apps according to their specific requirements by selecting and configuring different components within the tool .

• LangchainJS: FlowiseAI is built using LangchainJS, a Node Typescript/Javascript framework, which makes it accessible to developers and language enthusiasts .

• Free and Open Source: FlowiseAI is an open-source tool, meaning it is available for free and can be used for personal and commercial purposes .

• Docker Support: FlowiseAI supports Docker, allowing users to containerize their language models and simplify deployment across different environments.

• Community Collaboration: FlowiseAI encourages community-driven development and provides channels for users to engage with the FlowiseAI team and fellow developers through Discord, Twitter, and email

Figure 2.6: Flowise AI

2.2 Application description

To fulfill the noble aspirations of creating a transformative psychological chatbot that assists individuals in Vietnam with depression, a methodology was devised using PyCharm and Tkinter. The interface consists of three key components: an intelligent conversational agent powered by PyCharm's natural language processing, a visually appealing GUI designed with Tkinter, and additional features like soothing audio and adaptive responses. Together, these elements aim to provide personalized support and guide individuals towards emotional well-being and renewed joy in life. The application consist 3 parts:

2.2.1 Image Sentiment Analysis

The proposed approach includes an Image Sentiment Analysis component, which utilizes the huggingface library and specifically the openai/clip-vit-large-patch14 model for detecting emotions based on facial expressions. This model, categorized as an image zero-shot classifier, possesses the ability to classify images with custom labels. To facilitate emotion recognition, a set of carefully selected emotion labels, including Sad, Happy, Neutral, Fear, Anger, Pleasant-Surprise, and Disgust, is incorporated. The process commences with capturing the image using the openCV Python library, after which it undergoes analysis using the aforementioned model, ultimately yielding the dominant emotion along with corresponding percentage probabilities. This integration of image analysis enhances the chatbot's understanding of user emotions, allowing for a more nuanced and empathetic response tailored to the individual's emotional state (Figure 2.7).

Figure 2.7: Image sentiment classification

2.2.2 Voice Sentiment Analysis

Another crucial component of the system is Voice Sentiment Analysis, which aims to detect emotions based on the tone of the user's voice. This functionality relies on a model trained using a Kaggle dataset, which includes default emotion labels such as Sad, Happy, Neutral, Fear, Anger, and Pleasant-Surprise. The process begins by recording and saving the user's voice at specific intervals. The recorded voice is then analyzed using the trained model, providing the highest detected emotion along with corresponding percentage probabilities. By incorporating Voice Sentiment Analysis, the chatbot becomes capable of understanding and responding to the user's emotional state, enabling it to provide tailored support and guidance based on the detected emotions in the user's voice. This integration adds depth and sensitivity to the chatbot's interactions, fostering a more personalized and empathetic user experience (Figure 2.8).

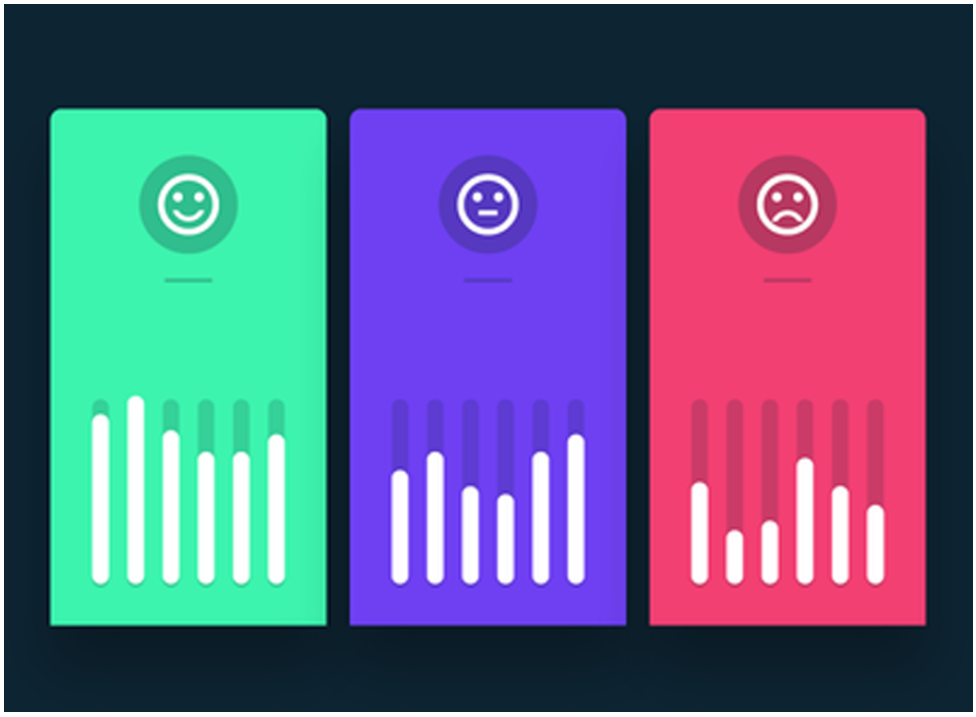

Figure 2.8: Voice sentiment classification

2.2.3 Chatbot Therapy

Large language models

In order to establish an effective Language Learning Model (LLM) for the chatbot, FlowiseAI, an open-source tool that simplifies the process by offering a user-friendly drag and drop interface. This tool empowers me to construct a customized LLM model by seamlessly integrating multiple features and

functionalities, all with the convenience of low-code or no-code development. Leveraging FlowiseAI, I can define and configure the specific requirements and parameters that will enable the chatbot to engage in meaningful language learning interactions, ensuring an optimized and personalized learning experience for users (Figure 2.9).

Figure 2.9: Chatbot

Psychology Preparation

FlowiseAI offers the advantage of OpenAI Embeddings features. These features enable the chatbot to comprehend the main ideas and concepts within various file formats such as CSV, Word, and PDF. Leveraging this capability, I can gather a curated collection of PDF resources covering several key topics in psychology, including Developmental Psychology, Social Psychology, Educational Psychology, Depression Psychology, Anxiety Psychology, and Behavioral Psychology. By understanding these concepts, the chatbot will be equipped to assist patients in comprehending their mental state and guide them on strategies to overcome challenges related to depression, anxiety, and other psychological issues. This integration of comprehensive psychology resources enhances the chatbot's ability to provide valuable insights and support to individuals seeking assistance in their mental well-being journey (Figure 2.10).

Figure 2.10: Therapy

Chapter 3

App background

To create the GUI for your psychology application, you can follow these steps. (Figure 3.1)

Step 1: Install Tkinter library using the command pip install tkinter. Tkinter is a Python library that allows you to create GUIs easily.

Step 2: Begin by creating the main window of your application. Set the width to 1200 and the height to 800. You can also set a background image for the window to enhance the visual appeal.

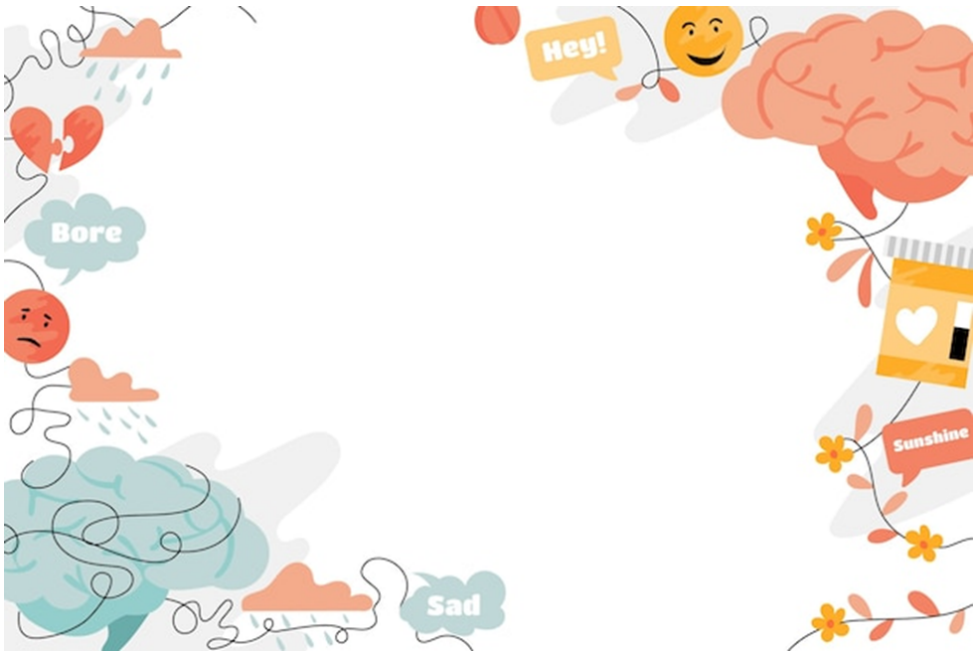

Figure 3.1: Chatbot background

Step 3: Application Diagram Map: Your psychology application consists of three main parts. The first part is Image Sentiment Classification, where you will classify facial emotions using openCV. This involves analyzing images and determining the emotions expressed by the faces in those images. The second part is Voice Sentiment Classification, where you will record voice samples, play them back, and classify the sentiment expressed in the voice. Lastly, you have the Psychology Chatbot, which serves as a user interface for interacting with the application. The chatbot will guide users through the different functionalities and provide assistance as needed (Figure 3.2).

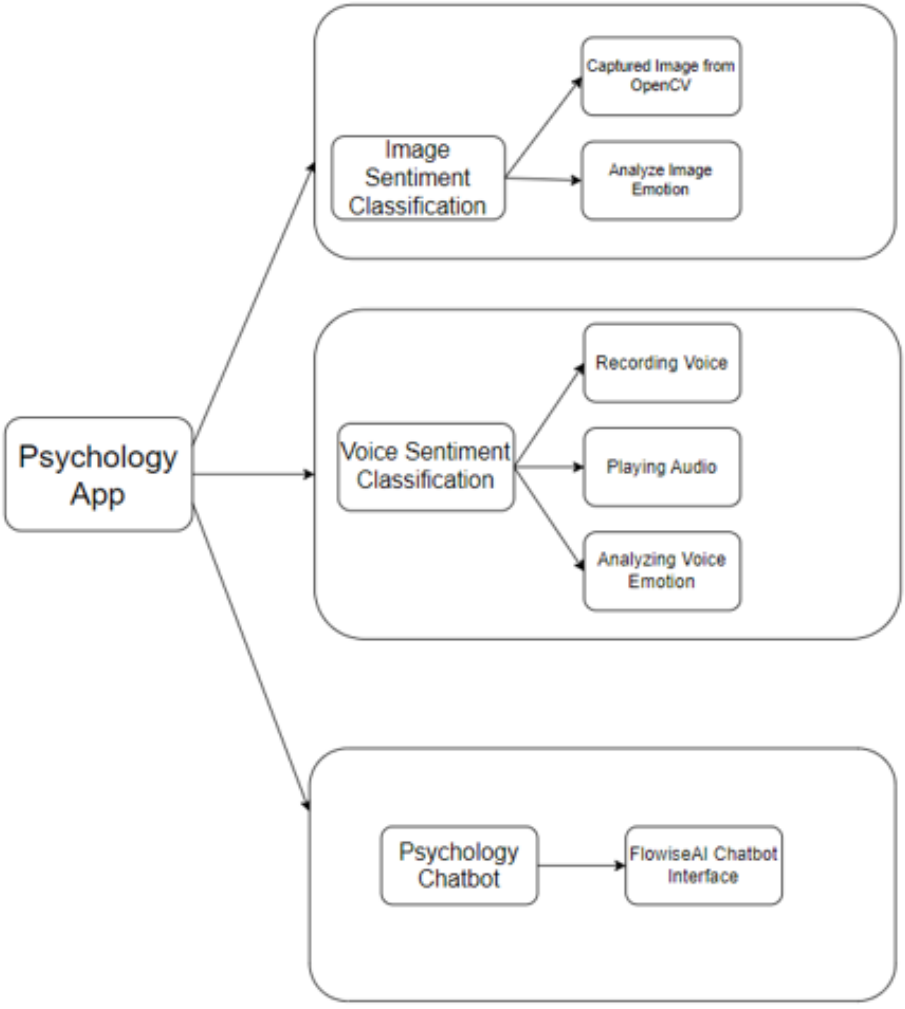

Figure 3.2: Psychology diagram

By structuring your application in this way, users will be able to explore image sentiment classification, voice sentiment classification, and interact with the psychology chatbot seamlessly. This GUI-based approach will enhance user experience and make your psychology application more accessible and user-friendly.

Chapter 4

Image Emotion Classification

4.1 Zeroshot image classification

Zero-shot image classification is a fascinating topic that has seen significant advancements in recent years, and it's closely connected to the capabilities offered by the Hugging Face library. In zero-shot image classification, the challenge is to classify images into different categories without the need for specific training on labeled data for those particular categories. Traditionally, image classification models rely on extensive training on labeled images to learn associations between visual features and labels. However, in the context of Hugging Face's model ecosystem, zero-shot image classification takes a different approach. Models like openai/clip-vit-large-patch14 are trained on datasets that include images and their corresponding descriptions, enabling them to acquire aligned vision-language representations (Figure 4.1).

```
>>> predictions = detector(image, candidate_labels=["fox", "bear", "seagull", "owl"])
>>> predictions
[{'score': 0.9996670484542847, 'label': 'owl'},
 {'score': 0.000199399160919711, 'label': 'seagull'},
 {'score': 7.392891711788252e-05, 'label': 'fox'},
 {'score': 5.96074532950297e-05, 'label': 'bear'}]
```

Figure 4.1: Zeroshot image classification

Hugging Face's model zoo provides a versatile framework for leveraging these pre-trained models for a wide range of downstream tasks, including zero-shot image classification. In this thesis, we aim to use the openai/clip-vit-large-patch14 model from Hugging Face to perform zero-shot image classification based on specific emotion labels such as Sad, Happy, Neutral, Fear, Anger, Pleasant-Surprise, and Disgust. This approach underscores the practicality of Hugging Face's model ecosystem in enabling the transfer of knowledge acquired during training to classify new classes, even those not present in the original training data.

4.2 Feature explain

To use the openai/clip-vit-large-patch14 model, you will need to install the transformers library (which allows you to employ pre-trained models from Hugging Face) and use the provided code snippet:

Explanation some important feature from code above (Figure 4.2):

```python
from transformers import CLIPProcessor, CLIPModel

model = CLIPModel.from_pretrained("openai/clip-vit-large-patch14")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-large-patch14")
```

Figure 4.2: Zeroshot image classification

from transformers import CLIPProcessor, CLIPModel: This line imports necessary classes from

the Transformers library, specifically the CLIPProcessor and CLIPModel classes.

model = CLIPModel.from pretrained(openai/clip-vit-large-patch14): This line creates an instance of the CLIPModel class by loading the pre-trained CLIP model called openai/clip-vit-large-patch14. This model, developed by OpenAI, is based on the Vision Transformer (ViT) architecture.

processor = CLIPProcessor.from pretrained(openai/clip-vit-large-patch14) This line creates an instance of the CLIPProcessor class by loading the associated pre-trained processor for the CLIP model. The processor handles tokenization and other preprocessing tasks specific to the CLIP model.

4.3 Image Sentiment Model

Combining additional technologies such as image capturing using OpenCV and image analysis, the identified emotion with the highest score (expressed as a percentage) is displayed in the top left corner of the frame.

With the combination of Capture Image using Open CV2, analyzing the image and it will the emotion with highest score (in percentage) at the top left of the frame like this figure (Figure 4.3)

Figure 4.3: Emotion CV

Chapter 5

Voice Sentiment Classification

An alternative method for detecting human emotions involves analyzing the sentiment expressed in voice recordings. In this approach, emotions are classified into seven categories: Sad, Happy, Neutral, Fear, Anger, Pleasant-Surprise, and Disgust, similar to how sentiment analysis is performed on images.

5.1 Download and training dataset

To evaluate this model, data was obtained from Kaggle, specifically from a dataset available at the following link: https://www.kaggle.com/datasets/ejlok1/toronto-emotional-speech-set-tess. After installing the necessary components and training the model, a system was developed to analyze emotions in audio recordings (Figure 5.1 and Figure 5.2).

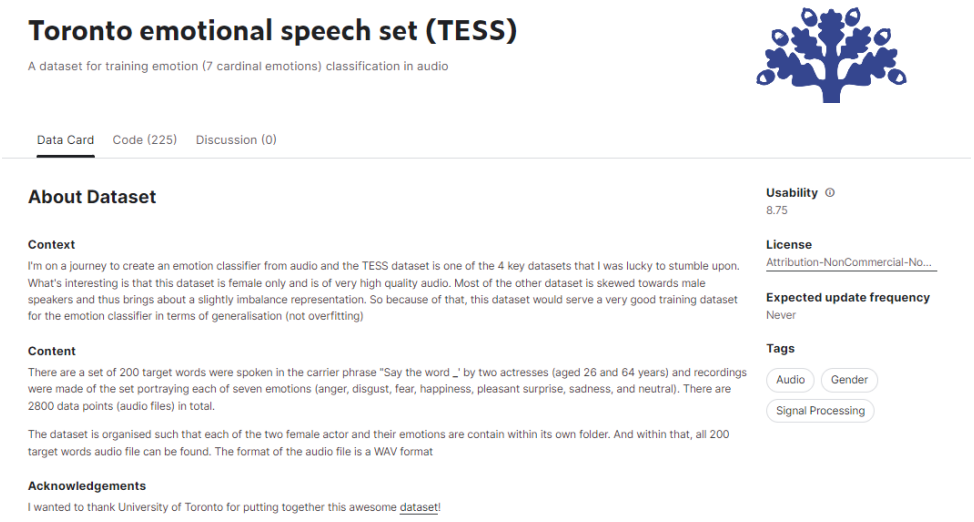

Figure 5.1: Toronto 1

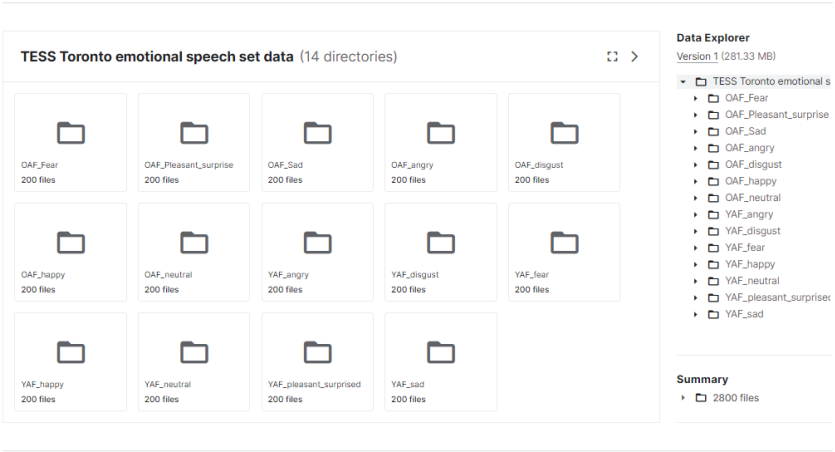

Figure 5.2: Toronto 2

5.2 Display model

The Voice sentiment classification system comprises three primary functions (Figure 5.3):

Figure 5.3: Voice emotion classification model

Recording function

This function records audio for a specific duration and signals completion after a predefined time (Figure 5.4).

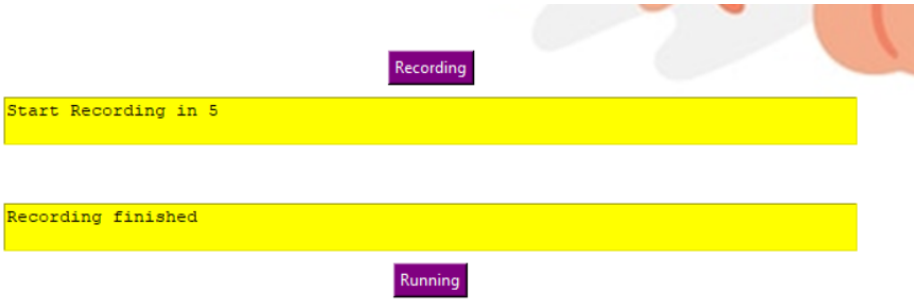

Figure 5.4: Recording

Playing function

The playing function allows the user to listen to the most recent audio recording to verify the recording process.

Analyzing function

The analyzing function utilizes the previously trained model to assess the emotions expressed in the recorded audio and returns the emotion with the highest probability as a percentage (Figure 5.5).

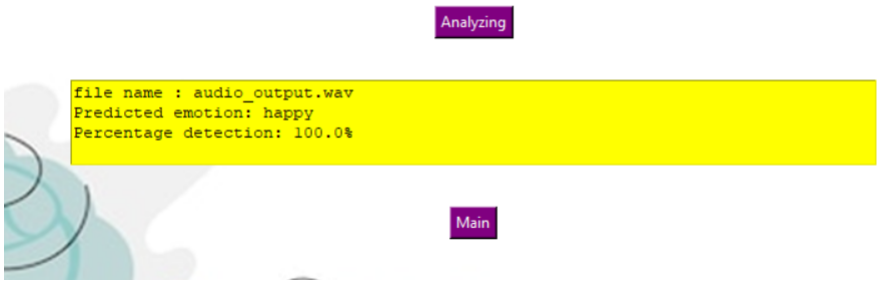

Figure 5.5: Analyzing

Chapter 6

Large Language Model

6.1 FlowiseAI quickview

FlowiseAI is an open-source platform that allows users to build customized LLM apps without extensive coding knowledge. It offers a drag-and-drop interface for creating personalized LLM flows.

6.2 Setup Flowise AI

Step 1: Get started by visiting the Flowise AI GitHub repository at https://github.com/FlowiseAI/Flowise (Figure 6.1).

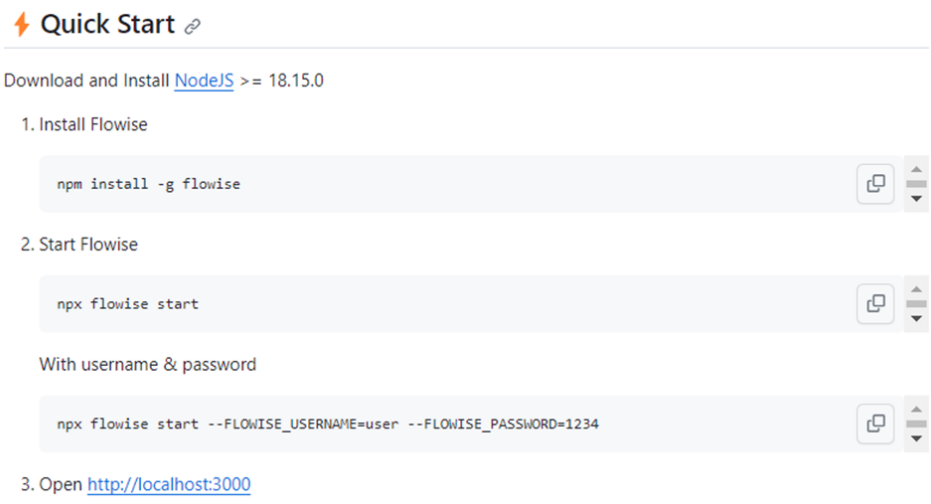

Figure 6.1: Flowise github

Step 2: Open PyCharm and go to the terminal. Install Flowise by running the following command: npm install -g flowise. Once installed, start Flowise by entering your username and password using the following command: npx flowise start –FLOWISE-USERNAME=user –FLOWISE-PASSWORD=1234. Wait for the process to finish (Figure 6.2).

Figure 6.2: Flowise Pycharm Terminal

Step 3: Once the setup is complete, open your browser and navigate to http://localhost:3000 (Figure 6.3).

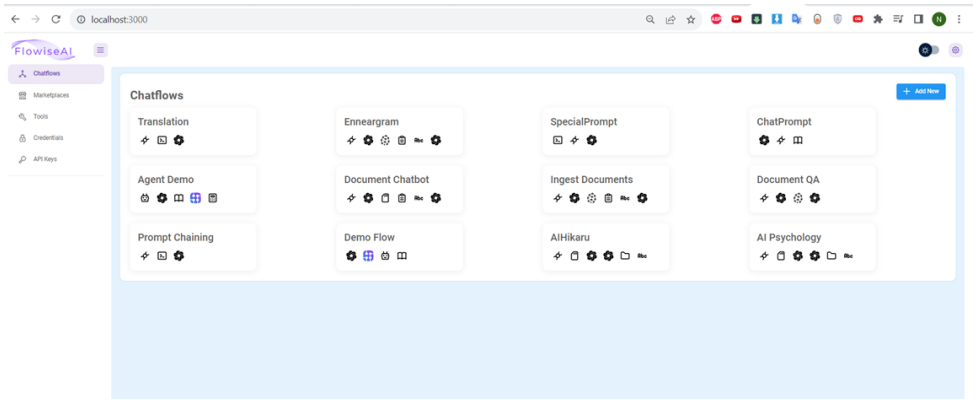

Figure 6.3: Flowise Forum

Step 4: Now you can create your own chatbot interface with a customized model (Figure 6.4).

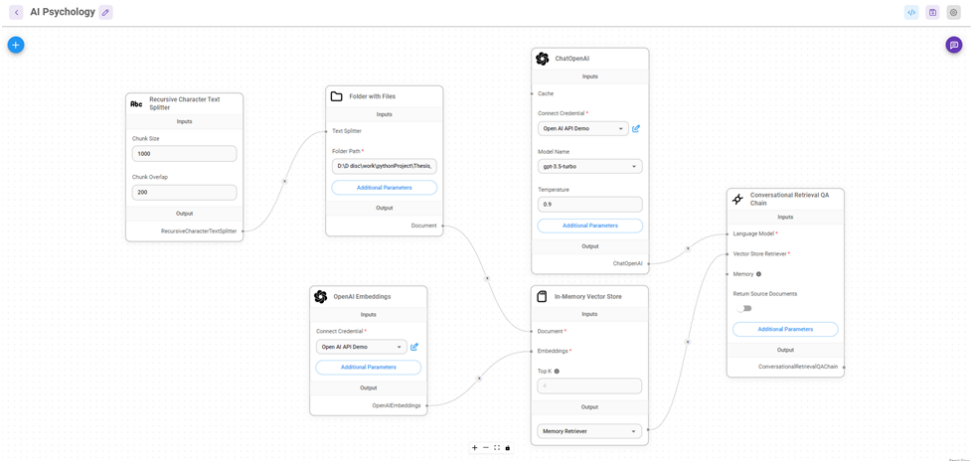

Figure 6.4: Flowise AI Psychology

6.3 Explain Features

Let's discuss the important features of Flowise: Nodes:

• Nodes are individual components or building blocks in the chatbot's logic. Each node represents a specific function or operation that the chatbot can perform.

• Different types of nodes are used to carry out various tasks and processes. In our case, we'll be using the following types of nodes:

• AI Models: These nodes connect to AI models, such as OpenAI's GPT-3, and generate responses to user queries based on the given input. You can configure the AI model to control its behavior and responses.

• Vector Stores: A vector store is a database that stores information in vector form. The chatbot can query this store for information, and the vectors help retrieve relevant data quickly.

• Document Loaders: These nodes load text data, such as psychology PDFs, into the chatbot's knowledge base.

• Text Splitters: Text splitter nodes are used to break down large text documents into smaller, more manageable chunks, which is especially useful when dealing with extensive documents. Chains:

• Chains define the flow of conversation and how the chatbot processes user queries. They connect

nodes together to create a structured conversation flow.

• Each chain represents a specific conversation path or a task the chatbot can perform.

• For example, we can create a "Conversational Retrieval QA Chain" to retrieve answers to user questions based on the information stored in the vector store and the responses generated by the AI model.

Here's how nodes and chains work together when building a chatbot:

1. Configuration: Nodes are configured to perform their respective tasks. For example, the AI model node is connected to an AI model (e.g., GPT-3) and set up with specific parameters for generating responses. Vector store nodes are linked to relevant data sources and configured to manage document data and embeddings. Document loader nodes are configured to load text data from specific locations, and text splitter nodes break down this data into manageable pieces.

2. Connection: Nodes are then connected within chains. For instance, the chain responsible for answering user questions might link the AI model node, the vector store node, and potentially other nodes such as the text splitter. The configuration and connection of these nodes within a chain determine how the chatbot processes user queries.

3. Conversation Flow: When a user interacts with the chatbot by sending a message or query, the chatbot's response is determined by the nodes and chains. The user's input is processed through the chain, which involves AI models generating responses based on the available knowledge in the vector store. The chatbot may also provide system messages and follow conversational rules defined in the chain.

4. Personalization and Knowledge Retrieval: To personalize the chatbot, you can define a system message that sets its personality and instructs it to respond in the first person. The chatbot's knowledge is drawn from the data stored in the vector store. When users ask questions, the chatbot searches its knowledge base in the vector store to retrieve answers (Figure 6.5).

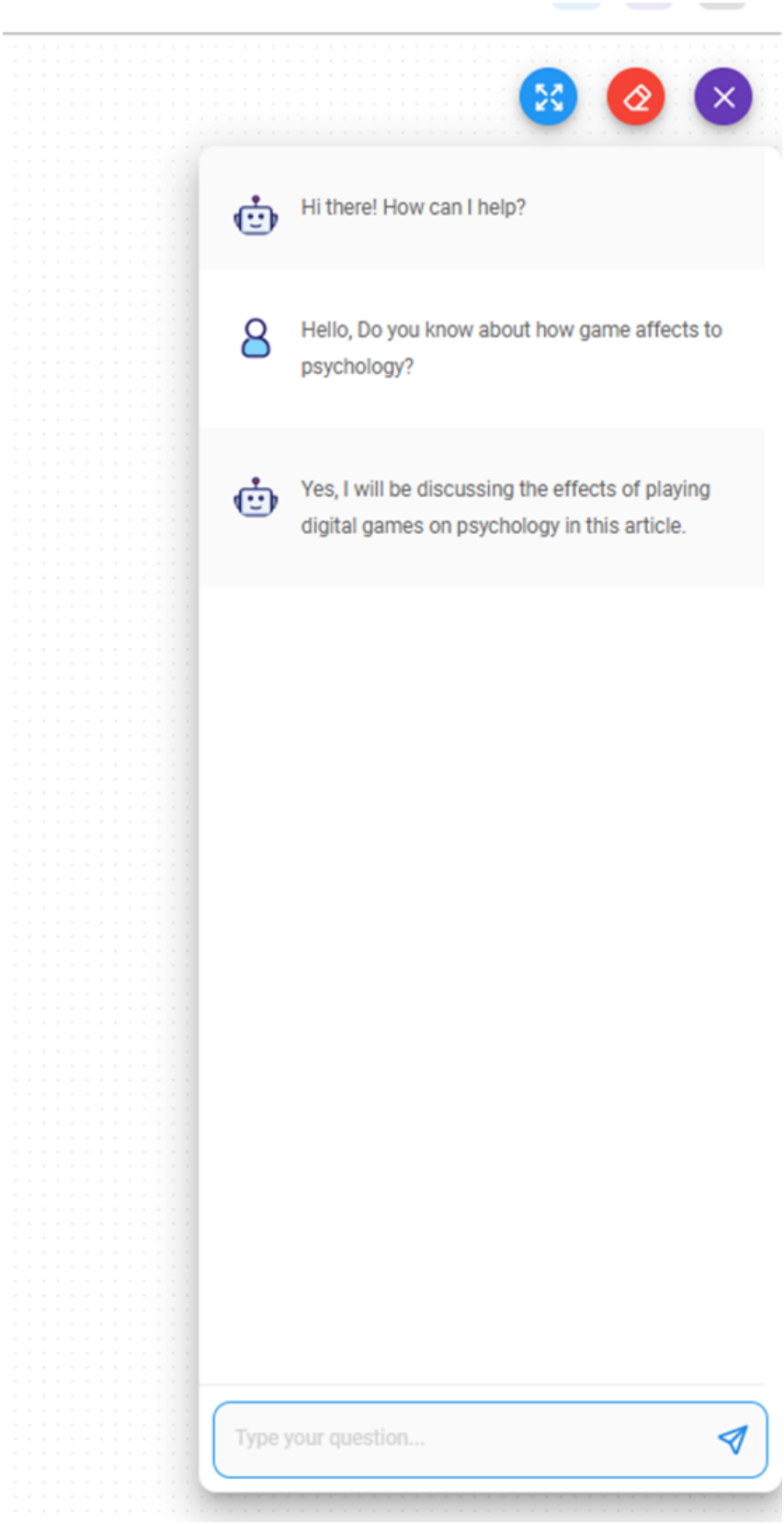

Figure 6.5: Flowise Converstation

Chapter 7

Psychology Knowledge Preparation

The establishment of an AI system capable of engaging in conversations about psychology topics necessitates the inclusion of comprehensive knowledge encompassing developmental psychology, social psychology, educational psychology, depression psychology, anxiety psychology, and behavioral psychology. This entails the integration of relevant theories, concepts, and empirical findings associated with each of these psychology domains. By incorporating a wide range of information, the AI system can effectively address various aspects of human development, social influences, learning processes, mental health conditions, and behavioral patterns. The availability of such a diverse knowledge base will enable the AI system to engage in meaningful and informed discussions on psychology topics, providing valuable insights and support to users seeking information or assistance in these areas.

7.1 Developmental Psychology

7.1.1 Quick view

Developmental psychology focuses on the study of human development across the lifespan, from infancy to old age.

7.1.2 Characteristics

It examines physical, cognitive, emotional, and social changes that occur as individuals age and progress through different life stages.

7.1.3 Therapy Support

Developmental psychology can inform therapeutic approaches by understanding the unique challenges and needs at different developmental stages. Therapists may use developmental theories and interventions tailored to specific age groups to address developmental issues and promote healthy development.

7.2 Social Psychology

7.2.1 Quick view

Social psychology explores how individuals' thoughts, feelings, and behaviors are influenced by the presence of others and the social environment.

7.2.2 Characteristics

It examines topics such as social perception, attitudes, conformity, obedience, group dynamics, prejudice, and interpersonal relationships.

7.2.3 Therapy Support

Social psychology can provide insights into how social factors impact mental health. Therapists may use social psychological principles to understand and address issues related to social anxiety, social skills deficits, prejudice, and interpersonal conflicts.

7.3 Educational Psychology

7.3.1 Quick view

Educational psychology focuses on understanding how people learn and develop in educational settings.

7.3.2 Characteristics

It investigates factors that influence learning, instructional strategies, motivation, cognitive development, assessment, and classroom management.

7.3.3 Therapy Support

Educational psychology informs the development of effective teaching methods, learning interventions, and educational programs. Therapists may collaborate with educational psychologists to address learning difficulties, design individualized education plans, and provide support for students with special needs.

7.4 Depression Psychology

7.4.1 Quick view

Depression psychology involves the study and treatment of depression, a mood disorder characterized by persistent feelings of sadness, hopelessness, and a lack of interest or pleasure.

7.4.2 Characteristics

It explores the causes, symptoms, risk factors, and impact of depression on individuals' thoughts, emotions, and behaviors.

7.4.3 Therapy Support

Therapy approaches for depression psychology may include cognitive-behavioral therapy (CBT), interpersonal therapy (IPT), psychodynamic therapy, or medication. Treatment aims to alleviate depressive symptoms, identify and modify negative thought patterns, improve coping skills, and enhance overall well-being.

7.5 Anxiety Psychology

7.5.1 Quick view

Anxiety psychology focuses on understanding and treating anxiety disorders, which involve excessive and persistent worry, fear, and anxiety that significantly interfere with daily functioning.

7.5.2 Characteristics

It examines different anxiety disorders, such as generalized anxiety disorder, panic disorder, social anxiety disorder, and specific phobias.

7.5.3 Therapy Support

Therapy approaches for anxiety psychology may include cognitive-behavioral therapy (CBT), exposure therapy, relaxation techniques, and medication. Treatment aims to reduce anxiety symptoms, challenge irrational beliefs, develop coping strategies, and improve overall functioning.

7.6 Behavioral Psychology

7.6.1 Quick view

Behavioral psychology, also known as behaviorism, emphasizes the study of observable behaviors and the environmental factors that influence them.

7.6.2 Characteristics

It focuses on how behaviors are learned, shaped, and modified through conditioning and reinforcement processes.

7.6.3 Therapy Support

Behavioral therapy techniques, such as operant conditioning, classical conditioning, and behavior modification, are used to address various psychological issues. Therapists may help individuals identify

maladaptive behaviors, develop new skills, and implement behavior change strategies to improve their well-being.

Chapter 8

Conclusion

In summary, this thesis aimed to develop an innovative chatbot capable of providing mental health support to individuals in Vietnam struggling with depression and other psychological challenges. Driven by the growing need for accessible and effective mental health services in the country, the proposed solution leverages artificial intelligence and natural language processing to deliver personalized therapy and guidance.

The methodology centered on building an intuitive conversational agent using tools like PyCharm, Tkinter, and FlowiseAI. Key capabilities included image and voice sentiment analysis for detecting user emotions, psychology-focused language learning to comprehend mental health concepts, and an engaging interface to foster meaningful dialogues. Together, these features empower the chatbot to understand users' mental states and respond with tailored advice to guide them towards improved wellbeing.

The integration of psychology resources using FlowiseAI's OpenAI Embeddings enables the chatbot to grasp critical concepts like developmental, social, educational, and behavioral psychology. This domain knowledge, alongside emotion recognition and adaptive conversations, allows the chatbot to provide nuanced support aligned with users' specific needs and challenges.

In conclusion, this thesis demonstrates the potential of AI-powered chatbots in expanding access to mental healthcare, especially in regions like Vietnam where such services are traditionally limited. The proposed solution aims to make therapy and emotional healing more available through technology. While further research and development may be needed, the foundational framework presented offers a promising starting point for providing customized mental health support through intelligent chatbot

agents. With its vision of nurturing wellbeing and bringing hope to struggling individuals, this project illustrates how emerging innovations can be harnessed for social good.

Reference

1.https://huggingface.co/docs/transformers/tasks/zero_shot_image_classification

2.https://www.kaggle.com/datasets/ejlok1/toronto-emotional-speech-set-tess

3.https://flowiseai.com/

4.https://github.com/FlowiseAI/Flowise

5.https://docs.flowiseai.com/

Appendix A

Code

Here is some inline code:

Main.py

```
import tkinter as tk
import ImageEmotionClassification
import LLM
import VoiceEmotionClassification
from tkinter import PhotoImage
from PIL import Image, ImageTk


def navigate_to_ImageEmotionNavigation():
    print("Navigate to Image Emotion Classification")
    root.destroy()
    ImageEmotionClassification.create_ImageEmotionClassification_window()


def navigate_to_VoiceEmotionNavigation():
    print("Navigate to Voice Emotion Classification")
    root.destroy()
    VoiceEmotionClassification.create_VoiceEmotionClassification_window()

```

```
def navigate_to_LLMPsychology():
    print("Navigate to LLM Psychology")
    root.destroy()
    LLM.create_LLMPsychology_window()


def quit_application():
    print("Turn off")
    root.destroy()

def create_a_window():


    global root
    root = tk.Tk()
    HEIGHT = 800
    WIDTH = 1200
    root.geometry(f"{WIDTH}x{HEIGHT}")
    root.title("Main")
    # Load the modified image as a PhotoImage
    bg_image = tk.PhotoImage(file="background.png")

    # Create a Label widget to display the image
    bg_label = tk.Label(root, image=bg_image)
    bg_label.place(relwidth=1, relheight=1)



    button = tk.Button(root, text="ImageEmotionClassification",
```

```
        command=navigate_to_ImageEmotionNavigation, bg="red", fg="white")
    button.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

    button = tk.Button(root, text="VoiceEmotionClassification",
        command=navigate_to_VoiceEmotionNavigation, bg="red",
                       fg="white")
    button.place(relx=0.5, rely=0.4, anchor=tk.CENTER)

    button = tk.Button(root, text="LLMPsychology",
        command=navigate_to_LLMPsychology, bg="red",
                       fg="white")
    button.place(relx=0.5, rely=0.6, anchor=tk.CENTER)

    button = tk.Button(root, text="Quit", command=quit_application,
        bg="red",
                       fg="white")
    button.place(relx=0.5, rely=0.8, anchor=tk.CENTER)

    # Center the window on the screen.
    root.update_idletasks()
    x = (root.winfo_screenwidth() - root.winfo_width()) // 2
    y = (root.winfo_screenheight() - root.winfo_height()) // 2
    root.geometry("+%d+%d" % (x, y))

    root.mainloop()

if __name__ == "__main__":
    create_a_window()
```

ImageEmotionClassification.py

```python
import tkinter as tk
import Main
from ImgEmoClassification import ImgEmoClassification

def process():
    print(f"The result:")
    ImgEmoClassification.main()
def navigate_to_Main():
    root.destroy()
    Main.create_a_window()

def create_ImageEmotionClassification_window():
    global root
    root = tk.Tk()
    HEIGHT = 800
    WIDTH = 1200
    root.geometry(f"{WIDTH}x{HEIGHT}")
    root.title("Image Emotion Classification")
    # Load the modified image as a PhotoImage
    bg_image = tk.PhotoImage(file="background.png")

    # Create a Label widget to display the image
    bg_label = tk.Label(root, image=bg_image)
    bg_label.place(relwidth=1, relheight=1)

    button = tk.Button(root, text="Run", command=process, bg="purple",
        fg="white")
    button.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

```

```
    button = tk.Button(root, text="Main", command=navigate_to_Main,
        bg="purple", fg="white")
    button.place(relx=0.5, rely=0.8, anchor=tk.CENTER)



    # Center the window on the screen.
    root.update_idletasks()
    x = (root.winfo_screenwidth() - root.winfo_width()) // 2
    y = (root.winfo_screenheight() - root.winfo_height()) // 2
    root.geometry("+%d+%d" % (x, y))

    root.mainloop()
```

VoiceEmotionClassification.py

```
import tkinter as tk
import Main
from VoiceEmoClassification import Running, Analyzing, Recording
import os

def navigate_to_Main():
    root.destroy()
    Main.create_a_window()

def running():
    print("Running the audio")
    Running.run()

def recording():
    print("Recording the audio")
```

```
16
17      # Clear the existing text in the recording_text_widget
18      recording_text_widget.config(state=tk.NORMAL)  # Enable the widget
            for editing
19      recording_text_widget.delete('1.0', tk.END)  # Clear previous text
20      recording_text_widget.config(state=tk.DISABLED)  # Disable editing
21
22      # Display "Start Recording" immediately
23      start_recording = f"Start Recording in {Recording.TIMEOUT} seconds"
24      start_recording_widget.config(state=tk.NORMAL)  # Enable the widget
            for editing
25      start_recording_widget.delete('1.0', tk.END)  # Clear previous text
26      start_recording_widget.insert(tk.END, start_recording)  # Insert
            the new result
27      start_recording_widget.config(state=tk.DISABLED)  # Disable editing
28
29      # Use after to update the recording_text_widget after a delay
30      root.after(100, update_recording_text)
31
32  def update_recording_text():
33      recording_result = Recording.speech_to_text_and_save()
34
35      # Update the text widget with the result
36      recording_text_widget.config(state=tk.NORMAL)  # Enable the widget
            for editing
37      recording_text_widget.insert(tk.END, "Recording finished")  #
            Insert the new result
38      recording_text_widget.config(state=tk.DISABLED)  # Disable editing
39
```

```
40 def analyzing():
41     print("Analyzing the audio")
42 
43     # Clear the existing text in the analyze_text_widget
44     analyze_text_widget.config(state=tk.NORMAL)  # Enable the widget
       for editing
45     analyze_text_widget.delete('1.0', tk.END)  # Clear previous text
46     analyze_text_widget.config(state=tk.DISABLED)  # Disable editing
47 
48     # Perform the analysis and get the results
49     result_text = Analyzing.analyzing()
50 
51     # Use after to update the analyze_text_widget after a delay
52     root.after(100, update_analyze_text, result_text)
53 
54 def update_analyze_text(result_text):
55     # Update the text widget with the result
56     analyze_text_widget.config(state=tk.NORMAL)  # Enable the widget
       for editing
57     analyze_text_widget.insert(tk.END, result_text)  # Insert the new
       result
58     analyze_text_widget.config(state=tk.DISABLED)  # Disable editing
59 
60 
61 def create_VoiceEmotionClassification_window():
62     global root, analyze_text_widget, recording_text_widget,
       start_recording_widget
63     root = tk.Tk()
64     WIDTH = 1200
```

```
65      HEIGHT = 800
66      root.geometry(f"{WIDTH}x{HEIGHT}")
67      root.title("Voice Emotion Classification")
68
69      # Load the modified image as a PhotoImage
70      bg_image = tk.PhotoImage(file="background.png")
71
72      # Create a Label widget to display the image
73      bg_label = tk.Label(root, image=bg_image)
74      bg_label.place(relwidth=1, relheight=1)
75
76      button = tk.Button(root, text="Recording", command=recording,
            bg="purple", fg="white")
77      button.place(relx=0.5, rely=0.2, anchor=tk.CENTER)
78
79      start_recording_widget = tk.Text(root, wrap=tk.WORD,
            state=tk.DISABLED, height=2, bg="yellow")  # Adjust the height
            as needed
80      start_recording_widget.place(relx=0.5, rely=0.25, anchor=tk.CENTER)
81
82      recording_text_widget = tk.Text(root, wrap=tk.WORD,
            state=tk.DISABLED, height=2, bg="yellow")  # Adjust the height
            as needed
83      recording_text_widget.place(relx=0.5, rely=0.35, anchor=tk.CENTER)
84
85      button = tk.Button(root, text="Running", command=running,
            bg="purple", fg="white")
86      button.place(relx=0.5, rely=0.4, anchor=tk.CENTER)
87
```

```
    button = tk.Button(root, text="Analyzing", command=analyzing,
        bg="purple", fg="white")
    button.place(relx=0.5, rely=0.6, anchor=tk.CENTER)

    analyze_text_widget = tk.Text(root, wrap=tk.WORD,
        state=tk.DISABLED, height=4, bg="yellow")  # Adjust the height
        as needed
    analyze_text_widget.place(relx=0.5, rely=0.7, anchor=tk.CENTER)

    button = tk.Button(root, text="Main", command=navigate_to_Main,
        bg="purple", fg="white")
    button.place(relx=0.5, rely=0.8, anchor=tk.CENTER)

    # Center the window on the screen.
    root.update_idletasks()
    x = (root.winfo_screenwidth() - root.winfo_width()) // 2
    y = (root.winfo_screenheight() - root.winfo_height()) // 2
    root.geometry("+%d+%d" % (x, y))

    root.mainloop()
```

LLM.py

```
import tkinter as tk
import Main
from LLMPsychology import LLMPsychology

def process():
    print(f"The result:")
    LLMPsychology.run()
```

```
def navigate_to_Main():
    root.destroy()
    Main.create_a_window()

def create_LLMPsychology_window():
    global root
    root = tk.Tk()
    HEIGHT = 800
    WIDTH = 1200
    root.geometry(f"{WIDTH}x{HEIGHT}")
    root.title("LLM Psychology")
    # Load the modified image as a PhotoImage
    bg_image = tk.PhotoImage(file="background.png")

    # Create a Label widget to display the image
    bg_label = tk.Label(root, image=bg_image)
    bg_label.place(relwidth=1, relheight=1)

    button = tk.Button(root, text="Run", command=process, bg="purple",
        fg="white")
    button.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

    button = tk.Button(root, text="Main", command=navigate_to_Main,
        bg="purple", fg="white")
    button.place(relx=0.5, rely=0.8, anchor=tk.CENTER)

    # Center the window on the screen.
    root.update_idletasks()
    x = (root.winfo_screenwidth() - root.winfo_width()) // 2
```

```
    y = (root.winfo_screenheight() - root.winfo_height()) // 2
    root.geometry("+%d+%d" % (x, y))

    root.mainloop()
```

In the directory ImgEmoClassification

ImgEmoClassification.py

```
import cv2
from PIL import Image
import numpy as np
import time
from transformers import CLIPProcessor, CLIPModel

model = CLIPModel.from_pretrained("openai/clip-vit-large-patch14")
processor =
    CLIPProcessor.from_pretrained("openai/clip-vit-large-patch14")

labelData = ["Sad", "Happy", "Neutral", "Fear", "Anger",
    "Pleasant-Surprise", "Digust"]

WIDTH = 800
HEIGHT = 800




def main():
    print("Running Image Emotion Classification")
    run()

```

```
21 def run():
22
23     # Open camera
24     cap = cv2.VideoCapture(0)
25
26     # Set frame width and height (adjust as needed)
27     cap.set(cv2.CAP_PROP_FRAME_WIDTH, WIDTH)
28     cap.set(cv2.CAP_PROP_FRAME_HEIGHT, HEIGHT)
29
30     # Loop to capture and process frames
31     while True:
32         # Capture frame from video feed
33         ret, frame = cap.read()
34
35         # Convert BGR frame to RGB PIL image
36         image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
37
38         # Perform emotion analysis using CLIP
39         inputs = processor(text=labelData, images=image,
               return_tensors="pt", padding=True)
40         outputs = model(**inputs)
41         logits_per_image = outputs.logits_per_image
42         probs = logits_per_image.softmax(dim=1)
43         probs_list = probs.tolist()[0]
44         values = max(probs_list)
45         max_index = probs_list.index(values)
46         emotion_label = labelData[max_index]
47         confidence = "{:.2%}".format(values)
48         print(f"The most emotion is: {emotion_label} with confidence:
```

```
            {confidence}")

49

50          # Display frame with emotion label
51          cv2.putText(frame, f"Emotion: {emotion_label} ({confidence})",
                (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
52          cv2.imshow("Frame", frame)
53
54          # Check if the window is closed (close button clicked)
55          if cv2.getWindowProperty("Frame", cv2.WND_PROP_VISIBLE) < 1:
56              break
57
58          # Check for 'q' key press
59          key = cv2.waitKey(1) & 0xFF
60          if key == ord('q') or key == 27:  # 27 corresponds to the ASCII
                code of the 'Esc' key
61              break
62
63      # Release the capture and close the windows
64      cap.release()
65      cv2.destroyAllWindows()
```

In the VoiceEmoClassification directory

Recording.py

```
1  import pyaudio
2  import wave
3  import time
4  import os
5
6  TIMEOUT = 5  # Timeout duration in seconds
```

```
def record_audio(filename):
    CHUNK = 1024
    FORMAT = pyaudio.paInt16
    CHANNELS = 1
    RATE = 44100
    SILENCE_THRESHOLD = 5000  # Minimum silence duration in milliseconds



    p = pyaudio.PyAudio()

    stream = p.open(format=FORMAT,
                    channels=CHANNELS,
                    rate=RATE,
                    input=True,
                    frames_per_buffer=CHUNK)

    frames = []
    silent_frames = 0  # Counter to track consecutive silent frames
    start_time = time.time()
    if (TIMEOUT <=1):
        print(f"Recording in {TIMEOUT} second")
    else:
        print(f"Recording in {TIMEOUT} seconds")

    while True:
        data = stream.read(CHUNK)
        frames.append(data)

        # Check if the recorded audio is silent
```

```
        is_silent = max(abs(sample) for sample in data) <
            SILENCE_THRESHOLD

        if is_silent:
            silent_frames += 1
            if silent_frames >= int(RATE / CHUNK * TIMEOUT):
                elapsed_time = time.time() - start_time
                if elapsed_time > TIMEOUT:
                    break
        else:
            silent_frames = 0
            start_time = time.time()

    print("Finished recording.")

    stream.stop_stream()
    stream.close()
    p.terminate()

    wf = wave.open(filename, "wb")
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(p.get_sample_size(FORMAT))
    wf.setframerate(RATE)
    wf.writeframes(b"".join(frames))
    wf.close()


def speech_to_text_and_save():

```

```
64
65      filename = os.path.join(os.path.dirname(__file__),
            "audio_output.wav")
66
67      TIME = record_audio(filename)
68      print("Audio saved as", filename)
69
70      # Perform speech recognition on the recorded audio here
71      # and print out the result
```

Running.py

```
1  from pydub import AudioSegment
2  from pydub.playback import play
3  import os
4  def play_audio(audio_filename):
5      audio = AudioSegment.from_file(audio_filename)
6      play(audio)
7
8  def run():
9  # Example usage
10     path = "audio_output.wav"
11     play_audio_path = os.path.join(os.path.dirname(__file__), path)
12     play_audio(play_audio_path)
13     return path
```

Analyzing.py

```
1  import numpy as np
2  import librosa
3  from keras.models import load_model
4  import joblib
```

```
import os



model_path = os.path.join(os.path.dirname(__file__), "emotion_model.h5")
model = load_model(model_path)
label_encoder_path = os.path.join(os.path.dirname(__file__),
    "label_encoder.pkl")
label_encoder = joblib.load(label_encoder_path)

# Function to extract MFCC features
def extract_mfcc(filename, num_mfcc=40):
    try:
        # Load audio file
        audio, sr = librosa.load(filename)

        # Extract MFCC features
        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=num_mfcc)

        # Resize MFCC features if necessary
        if mfcc.shape[1] > num_mfcc:
            mfcc = mfcc[:, :num_mfcc]
        elif mfcc.shape[1] < num_mfcc:
            mfcc = np.pad(mfcc, ((0, 0), (0, num_mfcc -
                mfcc.shape[1])), mode='constant')

        return mfcc
    except Exception as e:
        print("Error occurred during audio processing:", str(e))
        return None
```

```
32
33 # Perform inference on new audio samples
34 new_audio = os.path.join(os.path.dirname(__file__), "audio_output.wav")
35
36
37 def analyzing():
38     new_audio_mfcc = extract_mfcc(new_audio, num_mfcc=40)
39     if new_audio_mfcc is not None:
40         new_audio_mfcc = np.mean(new_audio_mfcc, axis=0)  # Take the
              mean across the channel axis
41         new_audio_mfcc = np.expand_dims(new_audio_mfcc, axis=0)
42         prediction = model.predict(new_audio_mfcc)
43         emotion_label =
              label_encoder.inverse_transform(np.argmax(prediction,
              axis=1))[0]
44         percentage = np.max(prediction) * 100
45         file_name = os.path.basename(new_audio)
46
47         result = f'file name : {file_name}\n'
48         if emotion_label == "ps":
49             result += f"Predicted emotion: pleasant-surprised\n"
50         else:
51             result += f"Predicted emotion: {emotion_label}\n"
52         result += f"Percentage detection: {percentage}%\n"
53         print(f"result: {result}")
54         return result
```

In the LLM directory

LLMPsychology

```
import webbrowser

def run():
    url =
        "http://localhost:3000/canvas/b66e70f9-2fe2-40b4-bc92-cfb59c0f46d6"
    chrome_path = r"C:\Program
        Files\Google\Chrome\Application\chrome.exe"
    webbrowser.register('chrome', None,
        webbrowser.BackgroundBrowser(chrome_path))
    webbrowser.get('chrome').open(url)

# Call the run() function to open the browser immediately
```