AI INFUSED CHATBOT FOR TREATMENT OF MENTAL ILLNESS

A PROJECT REPORT

Submitted by

ANISH. M	**(412720104003)**
MONESH. R	**(412720104303)**
RAJASUDAR. M	**(412720104305)**
AKASH. R	**(412720104301)**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

TAGORE ENGINEERING COLLEGE, CHENGALPATTU

ANNA UNIVERSITY::CHENNAI 600 025

MAY 2024

ANNA UNIVERSITY::CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"AI INFUSED CHATBOT FOR TREATMENT OF MENTAL ILLNESS"** is the bonafide work of **"M.RAJASUDAR (412720104305), MONESH.R(412720104303), AKASH. R (412720104301), ANISH. M (412720104003)** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. S. Surendran M.E, Ph.D.,	D. Meenakshi M.E,
Head of the Department,	Supervisor
Professor	Associate Professor
Department of CSE	Department of CSE
Tagore Engineering College,	Tagore Engineering College,
Rathinamangalam,	Rathinamangalam,
Chennai-600127	Chennai- 600127

Submitted for Project Report viva voice held on____________________.

INTERNAL EXAMINAR **EXTERNAL EXAMINAR**

ACKNOWLEDGEMENT

Our heartfelt thanks go to **Prof. Dr. M. MALA M.A., M.Phil.,** Chairperson of Tagore Engineering College, Rathinamangalam, for having provided us with all necessary infrastructure and other facilities, for their support to complete this project successfully

We extend our sincere gratitude to. **Dr. R. RAMESH M.E., Ph.D., FIE, MIETE, MISTE.**, Principal Tagore Engineering College for this degree of encouragement and moral support during the course of this project.

We are extremely happy for expressing our heart full gratitude to our department HOD and Project Coordinator **Dr. S. SURENDRAN M.E., Ph.D.,** for his valuable suggestions which helped us to complete the project successfully.

Our sincere gratitude to our supervisor **D. MEENAKSHI M.E.,** for extending all possible help for this project work.

Our sincere thanks to all teaching and non-teaching staff who have rendered help during various stage of our work.

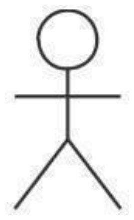
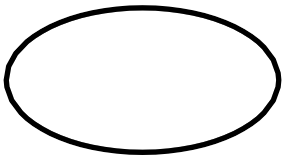
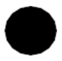
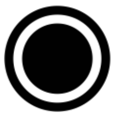
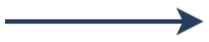
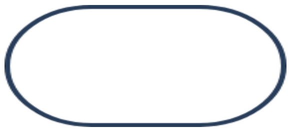
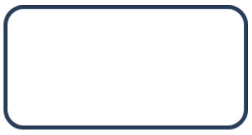
ABSTRACT

Mental health is a crucial aspect of human well-being, yet many people face barriers to access professional help and support. Chatbots, powered by artificial intelligence and machine learning, offer a promising solution to provide information, guidance, and empathy to individuals seeking mental health assistance. AI and ML chatbot for mental health suggestion is a type of application that uses artificial intelligence and machine learning to provide users with a safe and confidential space to talk about their mental health concerns and receive support and guidance from a trained professional. We use ChatGPT which is a powerful language model that has been trained on a large dataset of human conversation and is able to generate human-like responses to user input. This makes ChatGPT an ideal choice for building a chatbot that can provide support and guidance to users seeking help with their mental health.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

S.NO	SYMBOL NAME	SYMBOLS
1	Actor	
2	Use Case	
3	Initial State	
4	Final State	
5	Flow Line	
6	Activity	
7	State	

CHAPTER 1
INTRODUCTION

The introduction of the project report provides a concise overview of the project's context, objectives, and significance. It outlines the problem or need that the project addresses, articulates its main goals, and defines its scope. Additionally, it highlights the importance of the project and its potential impact. Finally, the introduction offers a brief preview of the structure of the project report, guiding the reader on what to expect in the subsequent sections.

1.1Mental Health Chatbots as an Emerging Technology

A chatbot is a system that can converse and interact with human users using spoken, written, and visual languages. In recent years, chatbots have been used more frequently in various industries, including retail, customer service, education, and so on because of the advances in artificial intelligence (AI) and machine learning (ML) domains. Facebook Messenger currently offers more than 300,000 text-based chatbots. Chatbots have primarily been used for commercial purposes and profitable businesses. However, more recent research has demonstrated that chatbots have considerable promise in the health care industry in treating patients and offering them support in a cost-effective and convenient manner.

In the context of mental health (MH), chatbots may encourage interaction with those who have traditionally been reluctant to seek health-related advice because of stigmatization. Chatbots are an emerging technology that shows potential for mobile MH apps to boost user engagement and adherence. The effectiveness of chatbots has been explored for self-disclosure and expressive writing. Young people with MH issues have experienced various types of social support such as appraisal, informational, emotional, and instrumental support from chatbots. In addition, chatbots have been designed to educate underprivileged communities on MH and stigmatized topics. Emerging evidence has shown user acceptance of chatbots for supporting various MH issues and early promises in boosting health outcomes in the physical and MH domains.

The adoption of new technology, especially those heavily related to AI and ML, relies first on ascertaining the levels of safety, effectiveness, and user comfort. Despite the increasing adoption and benefits of emerging technologies such as chatbots to support MH and well-being, little research has been conducted to gain an understanding of consumers' real-life user experiences of interacting with MH chatbot apps. Recent research on MH apps in general points out that patient safety is rarely examined, health outcomes are evaluated on a small scale, and no standard evaluation methods are present, and these findings also apply to MH chatbot apps. Similar to many other emerging technologies, recent developments in chatbots are because of a massive technology push, with little attention paid to human needs and experiences. This can lead to unintended negative consequences, such as biases, inadequate and failed responses, and privacy issues, all of which can negatively affect the quality of the experience of chatbots as a source of support. Thus, it is critical to gain an understanding of the nuances in users' perceptions and experiences of using MH chatbots.

Commercially available MH chatbot apps for popular platforms are used by a large user base with varying demographic backgrounds. These users can provide feedback through ratings and text reviews. These platforms can be leveraged to gain a holistic understanding of the features that recently developed MH chatbots offer and how users assess them. Knowledge of user perceptions from real-life experiences can inform future research and the design of more effective chatbots. Previous studies have identified user reviews as a great source for understanding the benefits and drawbacks of technology. This allows researchers to incorporate community values and needs into product design and improves user-friendliness. Consumers often make decisions about using new tools based on user rating scores and reviews in web-based marketplaces. According to previous studies, users trust reviews and feel at ease based on their decisions them. Moreover, previous literature emphasizes analyzing user reviews of mobile MH apps that have chatbot features to obtain in-depth knowledge about this new technology intervention in mobile MH apps. For this study, we decided to analyze commercially available well-known chatbot-based mobile MH apps and their corresponding user reviews from the Apple App Store and Google Play Store. To obtain a comprehensive overview of these apps and understand the nuances of user opinions, we aimed to answer the following 2 research questions (RQs):

RQ1: What are the state-of-the-art features and properties of chatbot-based mobile MH apps?

RQ2: What concerns and opinions are expressed in user reviews published on web-based app store platforms regarding the usability and efficiency of chatbot-based mobile MH apps?

We conducted an exploratory observation of 10 apps that offer support and treatment for a variety of MH concerns with a built-in chatbot feature and qualitatively analyzed their user reviews available on the Google Play Store and Apple App Store. Publicly available data (user reviews) provide in-depth analyses of consumers' personal app user experiences. We found that although chatbots' personalized, humanlike interactions were positively received by users, improper responses and assumptions about the personalities of users led to a loss of interest. As chatbots are always accessible and convenient, users can become overly attached to them and prefer them over interacting with their friends and family members. Furthermore, a chatbot may offer support for a crisis whenever the user needs it because of its 24/7 availability, but even the recently developed chatbots lack the understanding of properly identifying a crisis. Chatbots in this study fostered a judgment-free environment and helped users feel more comfortable sharing sensitive information.

Before implementing a technological solution for MH, researchers in digital health communities are constantly interested in the support needs and preferences of groups or communities. Researchers have analyzed the effectiveness of technologies used for MH assistance, proposing ethical concerns, policy recommendations, and designing automated or human-in-the-loop interactive systems. These studies stressed the significance of designing and evaluating systems for susceptible populations, such as people with MH issues, from the perspective of users. To contribute to this body of work, we discussed our study’s findings with respect to the research and design implications for future MH chatbots. We outlined specific recommendations for customizing certain features, careful consideration of incorporating persuasive strategies, and trust building. Finally, we discussed the impact of excessive reliance on chatbots for MH support. We believe that considering these insights while developing a chatbot-based MH support system will make the design user centric and, thus, more effective.

CHAPTER 2
LITERATURE SURVEY

A literature review is a body of text that aims to review the critical points of current knowledge on and/or methodological approaches to a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes information in a particular subject area within a certain time period. Its ultimate goal is to bring the reader up to date with current literature on a topic and forms the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources. Usually, it has an organizational pattern and combines both summary and synthesis.

2.1 RELATED WORKS & STUDIES

TITLE: CHATBOT-BASED ASSESSMENT OF EMPLOYEES' MENTAL HEALTH

AUTHOR: Ines Hungerbuehler , Kate Daley , Kate Cavanagh , Heloísa Garcia Claro , Michael Kapps

YEAR: 2020

Stress, burnout, and mental health problems such as depression and anxiety are common, and can significantly impact workplaces through absenteeism and reduced productivity. To address this issue, organizations must first understand the extent of the difficulties by mapping the mental health of their workforce. Online surveys are a cost-effective and scalable approach to achieve this but typically have low response rates, in part due to a lack of interactivity. Chatbots offer one potential solution, enhancing engagement through simulated natural human conversation and use of interactive features.

The aim of this study was to explore if a text-based chatbot is a feasible approach to engage and motivate employees to complete a workplace mental health assessment. This paper describes the design process and results of a pilot implementation.

ISSUES: In this implementation, several key issues may arise.Integrating the chatbot seamlessly into existing workplace systems and workflows may pose a challenge, requiring coordination with IT departments and potentially complex technical implementations. Finally, assessing the effectiveness of the chatbot in improving survey response rates and overall mental health awareness requires clear metrics and evaluation methods.

TITLE: ASSESSING THE USABILITY OF A CHATBOT FOR MENTAL HEALTH CARE

AUTHOR: Gillian Cameron, David Cameron, Gavin Megaw, Raymond Bond, Maurice Mulvenna, Siobhan O'Neill, Cherie Armour & Michael McTear

YEAR: 2019

The aim of this paper is to assess the usability of a chatbot for mental health care within a social enterprise. Chatbots are becoming more prevalent in our daily lives, as we can now use them to book flights, manage savings, and check the weather. Chatbots are increasingly being used in mental health care, with the emergence of "virtual therapists". In this study, the usability of a chatbot named iHelpr has been assessed. iHelpr has been developed to provide guided self-assessment, and tips for the following areas: stress, anxiety, depression, sleep, and self-esteem. This study used a questionnaire developed by Chatbot test, and the System Usability Scale to assess the usability of iHelpr. The participants in this study enjoyed interacting with the chatbot, and found it easy to use.

However, the study highlighted areas that need major improvements, such as Error Management and Intelligence. A list of recommendations has been developed to improve the usability of the iHelpr chatbot.

ISSUES: In this project, several issues are identified that need to be addressed to improve the usability of the iHelpr chatbot for mental health care. The study highlights the need for major improvements in Error Management and Intelligence. Error Management refers to the chatbot's ability to handle user errors or misunderstandings gracefully, providing helpful feedback and suggestions. Intelligence refers to the chatbot's capacity to understand and respond appropriately to user queries, offering personalized and effective support. These issues suggest that while participants enjoyed interacting with iHelpr and found it easy to use, there is room for enhancement in its functionality and responsiveness to better serve users' mental health needs

TITLE: A MENTAL HEALTH CHATBOT FOR REGULATING EMOTIONS (SERMO) - CONCEPT AND USABILITY TEST

AUTHOR: Kerstin Denecke, Sayan Vaaheesan, Aaganya Arulnathan

YEAR:2020

Mental disorders are widespread in countries all over the world. Nevertheless, there is a global shortage in human resources delivering mental health services. Leaving people with mental disorders untreated may increase suicide attempts and mortality. To address this matter of limited resources, conversational agents have gained momentum in the last years. In this work, we introduce SERMO, a mobile application with integrated chatbot that implements methods from cognitive behaviour therapy (CBT) to support mentally ill people in regulating emotions and dealing with thoughts and feelings. SERMO asks the user on a daily basis on events that occurred and on emotions. It determines automatically the basic emotion of a user from the natural language input using natural language processing and a lexicon-based approach. Depending on the emotion, an appropriate measurement such as activities or mindfulness exercises are suggested by SERMO. Additional functionalities are an emotion diary, a list of pleasant activities, mindfulness exercises and information on emotions and CBT in general. User experience was studied with 21 participants using the User Experience Questionnaire (UEQ). Findings show that efficiency, perspicuity and attractiveness are considered as good. The scales describing hedonic quality (stimulation and novelty), i.e., fun of use, show neutral evaluations.

ISSUE: There are several potential issues that need to be considered. Firstly, the accuracy of determining the user's basic emotion from natural language input using a lexicon-based approach may be limited, as emotions can be complex and context-dependent. Secondly, while suggesting activities or exercises based on the user's emotion is helpful, the effectiveness of these suggestions may vary greatly depending on individual preferences and circumstances. Additionally, the neutral evaluations for hedonic quality indicate that users may not find the app particularly enjoyable or engaging, which could impact long-term engagement and effectiveness. Lastly, while the User Experience Questionnaire (UEQ) provides valuable feedback, a larger and more diverse sample size would be beneficial for a more comprehensive evaluation of the app's effectiveness and user satisfaction.

TITLE: CAREBOT: A MENTAL HEALTH CHATBOT

AUTHOR: Reuben Crasto, Lance Dias, Dominic Miranda, Deepali Kayande

YEAR: 2021

A large number of students face some sort of mental health illness such as depression, anxiety, stress, etc. Due to the lack of willingness or financial ability many students do not visit a college counsellor or seek professional help. The proposed system aims to ease this problem by providing a chatbot to students that would provide the required support similar to a counsellor or therapist. Recent use of technology in aiding with Mental Health recovery has proven to be highly effective in terms of machine learning. The method involves surveys, questionnaires, data analysis and natural language processing. The aim is to build an online platform through which the tool will function.

ISSUE: The project faces several challenges, including ensuring the chatbot's effectiveness in providing support comparable to a human counsellor or therapist, especially in understanding and responding to the nuances of mental health issues. Another challenge is maintaining user privacy and data security, given the sensitive nature of mental health information. Additionally, the chatbot must be designed to handle a wide range of mental health issues and situations, requiring robust natural language processing capabilities and a deep understanding of mental health concepts. Finally, ensuring the chatbot's accessibility and usability for a diverse student population, including those with varying levels of technological proficiency, will be critical for its success.

TITLE: TOWARDS THE DEVELOPMENT OF A TRUSTWORTHY CHATBOT FOR MENTAL HEALTH APPLICATIONS

AUTHORS: Wolfgang Minker, Philip Seldschopf, Matthias Kraus

YEAR: 2021

Research on conversational chatbots for mental health applications is an emerging topic. Current work focuses primarily on the usability and acceptance of such systems. However, the human-computer trust relationship is often overlooked, even though being highly important for the acceptance of chatbots in a clinical environment. This paper presents the creation and evaluation of a trustworthy agent using relational and proactive dialogue. A pilot study with non-clinical subjects showed that a relational strategy using empathetic reactions and small-talk failed to foster human-computer trust. However, changing the initiative to be more proactive seems to be welcomed as it is perceived more reliable and understandable by users.

ISSUES: The project faces several key issues. Firstly, the initial relational strategy, which incorporated empathetic reactions and small talk, did not effectively foster human-computer trust, indicating a need for more nuanced approaches to empathy and conversation. Secondly, the shift towards a proactive dialogue approach was more positively received, suggesting that the initiative and direction of the conversation play a crucial role in building trust. Additionally, the study was conducted with non-clinical subjects, so the findings may not fully translate to a clinical environment where trust dynamics could differ significantly. Further research with clinical populations is needed to validate these findings and understand how trust can be effectively established and maintained in mental health chatbots.

TITLE: ELOMIA CHATBOT: THE EFFECTIVENESS OF ARTIFICIAL INTELLIGENCE IN THE FIGHT FOR MENTAL HEALTH

AUTHORS: Oleksandr Romanovskyi, Nina Pidbutska and Anastasiia Knysh

YEAR: 2021

The article is presenting results of controlled study of effectiveness of Elomia chatbot in reducing tendency to depression, anxiety and negative emotional effects. Elomia wasdeveloped using artificial intelligence technologies. In the process of developing a chatbotfollowing technologies were used: RoBERTa(NER) for identification names and locations; COSMIC for identification of human emotions; DIALOGPT for answers generation; DistilBERT (SQuAD) for pointing the most relevant information in the script; GECToR for spelling check. Elomia is able to identify the main psychological problems of the client and offer him the most suitable support option using first aid techniques and cognitive-behavioral psychotherapy. To check the effectiveness of the chatbot it was conducted a study that included three stages: 1) the formation of experimental and control samples; 2) baseline testing; 3) final testing. The study used psychological research methods: 1) Patient Health Questionnaire-9 (PHQ-9) to diagnose a tendency to depression; 2) General Anxiety Disorder-7 (GAD-7) to diagnose a tendency to generalized anxiety disorder; 3) Positive and Negative Affect Schedule (PANAS) to diagnose of prevailing (positive / negative) emotional affects. 412 volunteers (202 women, 210 men, ages 19 to 23 years old) who identified their tendency to depression, anxiety and low mood were selected in students social networks groups ofKharkiv region as participants of research. It was found that regular usage of Elomiacontributes to a significant reduction in the high tendency to depression (up to 28%), anxiety(up to 31%), and negative affects (up to 15%).

ISSUE: This project faces several potential issues. Firstly, the selection of participants from social network groups may lead to a biased sample, as these individuals may not be representative of the general population. Additionally, relying solely on self-reported measures like the PHQ-9, GAD-7, and PANAS may introduce response biases and inaccuracies, as participants may overstate or understate their symptoms. The use of artificial intelligence technologies like RoBERTa, COSMIC, DIALOGPT, DistilBERT, and GECToR is innovative, but their effectiveness and accuracy in identifying psychological issues and providing suitable support options need to be validated through rigorous testing and comparison with traditional diagnostic methods. The study's design, while including baseline and final testing, could be strengthened by including a more diverse range of outcome measures and longer-term follow-ups to assess the chatbot's sustained impact on mental health.

TITLE: AN OVERVIEW OF THE FEATURES OF CHATBOTS IN MENTAL HEALTH: A SCOPING REVIEW

AUTHORS: Alaa A. Abd-alrazaq, Mohannad Alajlani, Ali Abdallah Alalwan, Bridgette M. Bewick, Peter Gardner, Mowafa Househ.

YEAR: 2019

Chatbots are systems that are able to converse and interact with human users using spoken, written, and visual languages. Chatbots have the potential to be useful tools for individuals with mental disorders, especially those who are reluctant to seek mental health advice due to stigmatization. While numerous studies have been conducted about using chatbots for mental health, there is a need to systematically bring this evidence together in order to inform mental health providers and potential users about the main features of chatbots and their potential uses, and to inform future research about the main gaps of the previous literature.

ISSUE: In this project, the main issues include the need to systematically review existing evidence on using chatbots for mental health, which involves gathering and synthesizing a large amount of information from various studies. Additionally, the project aims to inform mental health providers and potential users about the main features and potential uses of chatbots, requiring clear and concise communication of complex information. Furthermore, identifying gaps in the previous literature for future research involves critical analysis and interpretation of the findings from existing studies.

TITLE: EFFECTIVENESS AND SAFETY OF USING CHATBOTS TO IMPROVE MENTAL HEALTH: SYSTEMATIC REVIEW AND META-ANALYSIS

AUTHORS: Alaa Ali Abd-Alrazaq, Asma Rababeh, Mohannad Alajlani, Bridgette M Bewick, Mowafa Househ.

YEAR: 2020

The global shortage of mental health workers has prompted the utilization of technological advancements, such as chatbots, to meet the needs of people with mental health conditions. Chatbots are systems that are able to converse and interact with human users using spoken, written, and visual language. While numerous studies have assessed the effectiveness and safety of using chatbots in mental health, no reviews have pooled the results of those studies. This study aimed to assess the effectiveness and safety of using chatbots to improve mental health through summarizing and pooling the results of previous studies.A systematic review was carried out to achieve this objective. The search sources were 7 bibliographic databases (eg, MEDLINE, EMBASE, PsycINFO), the search engine "Google Scholar," and backward and forward reference list checking of the included studies and relevant reviews. Two reviewers independently selected the studies, extracted data from the included studies, and assessed the risk of bias. Data extracted from studies were synthesized using narrative and statistical methods, as appropriate.

TITLE: VICKYBOT, A CHATBOT FOR ANXIETY-DEPRESSIVE SYMPTOMS AND WORK-RELATED BURNOUT IN PRIMARY CARE AND HEALTH CARE PROFESSIONALS: DEVELOPMENT, FEASIBILITY, AND POTENTIAL EFFECTIVENESS STUDIES

AUTHORS: Aleix Solanes, Joaquim Radua, María Antonieta Also, Elisenda Sant, Sandra Mur,

Alicia Ruiz, Carlota Sorroche, Giulia Virgil, Marta Espinosa, Claudia Rodriguez-Torrella,Francisco José Valdesoiro, Flavia Piazza, Arturo Rodríguez-Rey, Julia-

Parisad Esteva, Alejandro Caballo,Inés Martín-Villalba, Ariadna Mas, Victoria Ruiz.

YEAR: 2023

Many people attending primary care (PC) have anxiety-depressive symptoms and work-related burnout compounded by a lack of resources to meet their needs. The COVID-19 pandemic has exacerbated this problem, and digital tools have been proposed as a solution. Healthy controls (HCs) tested Vickybot for reliability. For the simulation study, HCs used Vickybot for 2 weeks to simulate different clinical situations. For feasibility and effectiveness study, people consulting PC or health care workers with mental health problems used Vickybot for 1 month. Self-assessments for anxiety (Generalized Anxiety Disorder 7-item) and depression (Patient Health Questionnaire-9) symptoms and work-related burnout (based on the Maslach Burnout Inventory) were administered at baseline and every 2 weeks. Feasibility was determined from both subjective and objective user-engagement indicators (UEIs). Potential effectiveness was measured using paired 2-tailed t tests or Wilcoxon signed-rank test for changes in self-assessment scores. We aimed to present the development, feasibility, and potential effectiveness of Vickybot, a chatbot aimed at screening, monitoring, and reducing anxiety-depressive symptoms and work-related burnout, and detecting suicide risk in patients from PC and health care workers.

CHAPTER 3
PROBLEM STATEMENT

A problem statement succinctly describes an issue, its impact, and desired resolution, guiding project direction and scope. It's a concise summary outlining the problem's current state, why it matters, and what needs to change. This statement acts as a road map, ensuring efforts are focused on addressing the core challenge effectively.

3.1 AIM

AI-Infused Chatbot for the Treatment of Mental Illness" is to develop a sophisticated and empathetic chatbot that leverages artificial intelligence (AI) to provide effective mental health support and treatment.

3.2 OBJECTIVES

- Provide Accessible Mental Health Support to the users.
- Provide Emotional Support to the users.
- Deploy the chatbot using containerization technology to ensure that it can scale easily to accommodate a large number of users.

3.3 SCOPE OF THE PROJECT

- The project will involve the development and training of a sophisticated AI model using technologies like OpenAI's GPT-3.5 Turbo to enable the chatbot to understand and respond to user inputs in a conversational manner.

- The project will include the design of a user-friendly interface for the chatbot, ensuring that it is easy for users to interact with and navigate.

- The chatbot will be designed in collaboration with mental health professionals to ensure that it provides accurate and effective support and treatment recommendations.

- The project will include mechanisms for continuous monitoring and improvement of the chatbot based on user feedback and evolving mental health research.
- The chatbot will be designed to be accessible to a wide range of users, including those with disabilities, ensuring that everyone can benefit from its support.

3.4 EXISTING SYSTEM

The existing systems for chatbots designed for the treatment of mental illness vary widely in terms of complexity, functionality, and effectiveness.

1. **Rule-Based Chatbots:** These chatbots use a set of predefined rules to respond to user inputs. Disadvantages include limited ability to understand natural language and lack of adaptability to new or complex situations.
2. **Scripted Chatbots:** Scripted chatbots use predefined scripts to simulate conversation. While they can provide useful information, they lack the ability to engage in meaningful dialogue or provide personalized responses.
3. **Machine Learning-Based Chatbots:** These chatbots use machine learning algorithms to improve their responses over time. However, they may require large amounts of training data and ongoing maintenance to remain effective.
4. **Hybrid Chatbots:** Hybrid chatbots combine rule-based and machine learning approaches to improve their performance. However, they can be complex to develop and may still struggle with complex or nuanced conversations.

3.4.1 DISADVANTAGES OF EXISTING SYSTEM:

There are several drawbacks associated with the current system:

1. **Ethical and Privacy Concerns:** All chatbots for mental health must address concerns about privacy, confidentiality, and the potential for harm. Ensuring

that user data is protected and that the chatbot provides accurate and safe advice is crucial.

2. **Limited Understanding of Context:** Many existing chatbots struggle to understand the context of a user's conversation, leading to responses that may be irrelevant or confusing.
3. **Lack of Emotional Intelligence:** While some chatbots attempt to simulate empathy, they often fall short in providing the kind of emotional support that a human therapist can offer.
4. **Dependency on Internet Connection:** Most chatbots require an internet connection to function, which may limit their accessibility in areas with poor connectivity.

3.5 PROPOSED SYSTEM

The proposed system for the chatbot for the treatment of mental illness combines several key components to deliver effective and user-friendly support. The frontend interface will be developed using Python Gradio, providing an intuitive platform for users to interact with the chatbot. The backend, also implemented in Python, will manage the communication between the frontend and OpenAI's powerful GPT-3.5 Turbo model. This model, trained on a vast dataset, will generate responses to user inputs, enabling the chatbot to engage in meaningful and empathetic conversations. The entire system will be deployed on the AWS cloud using Docker containers, ensuring scalability, reliability, and security. This architecture leverages the strengths of each component to create a chatbot that offers personalized and effective mental health support.

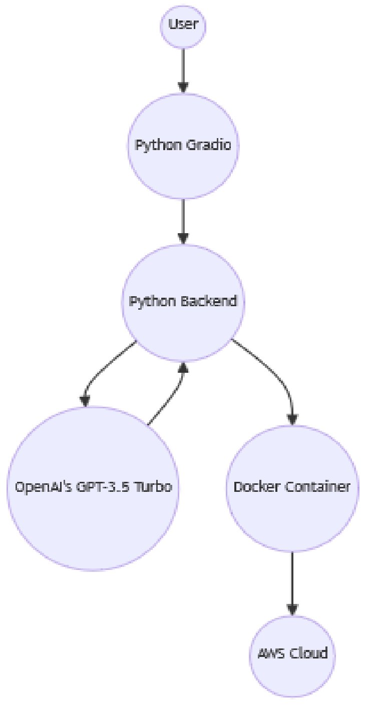


Figure 3.1 Proposed Model

3.5.1 ADVANTAGES OF PROPOSED SYSTEM:

- **Effective Support:** The use of OpenAI's GPT-3.5 Turbo model allows the chatbot to provide personalized and effective mental health support, enhancing the user experience.
- **User-Friendly Interface:** The frontend interface developed using Python Gradio provides an intuitive platform for users to interact with the chatbot, making it more accessible and easier to use.
- **Scalability:** Deploying the entire system on the AWS cloud using Docker containers ensures scalability, allowing the chatbot to handle a large number of users and adapt to changing demands.
- **Reliability:** The use of Docker containers and AWS cloud ensures reliable performance, with high availability and minimal downtime.
- **Security:** Deploying on AWS cloud provides robust security features, protecting user data and ensuring compliance with data protection regulations.

CHAPTER 4

SYSTEM REQUIREMENT SPECIFICATION

GENERAL

Requirements are the basic constrains that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Environmental requirements
   a. Software requirements
   b. Hardware requirements

4.1 SOFTWARE REQUIREMENTS

Operating System: Windows 10 or later Tool:

1. Amazon Web Services.
2. Python version 3.10 or later.
3. Microsoft Visual Studio Code.
4. Docker Desktop.

4.2 HARDWARE REQUIREMENTS

Processor: Intel i7 orRyzen7

Hard disk: Minimum 256GB

RAM: Minimum 16GB

4.3 SOFTWARE FEATURES

AMAZON WEB SERVICES:

Amazon Web Services (AWS) is a comprehensive and widely-used cloud computing platform offered by Amazon. It provides a vast array of services that enable businesses and developers to build and deploy applications and services quickly and securely. AWS offers a pay-as-you-go pricing model, which allows users to pay only for the computing resources and services they use, without the need to invest in costly infrastructure upfront.

One of the key advantages of AWS is its scalability, allowing users to easily scale their applications up or down based on demand. This scalability is supported by a global network of data centres, which ensures high availability and low latency for users around the world.

AWS provides a wide range of services, including computing power with Amazon EC2, storage options with Amazon S3, and database services with Amazon RDS. Additionally, AWS offers services for machine learning, analytics, networking, security, and more, making it a versatile platform for a variety of use cases.

Overall, AWS is known for its reliability, scalability, and flexibility, making it a popular choice for businesses of all sizes looking to leverage the power of the cloud to innovate and grow.

PYTHON:

Python is designed to be beginner-friendly and emphasizes code readability, using indentation and a clean syntax that reduces the need for excessive punctuation. This makes it easier to understand and write Python code, even for individuals who are new to programming.

Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming styles. It provides a comprehensive standard library with a wide range of modules and functions that enable developers to perform various tasks efficiently.

One of Python's strengths lies in its extensive ecosystem of third-party libraries and frameworks. These libraries, such as NumPy, pandas, TensorFlow, and Django, offer specialized functionality for scientific computing, data analysis, machine learning, web development, and more. The availability of these libraries contributes to Python's popularity among developers working in diverse domains.

Python's versatility allows it to be used for a wide array of applications. It is commonly used for web development, scripting, automation, scientific computing, data analysis, machine learning, artificial intelligence, and system administration, among other fields.

FEATURES IN PYTHON

There are many features in Python, some of which are discussed below as follows:

1. Free and Open Source

Python language is freely available at the official website and since it is open-source, this means that source code is also available to the public. So, you can download it, use it as well as share it.

2. Easy to code

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, JavaScript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

MICROSOFT VISUAL STUDIO CODE:

Visual Studio Code (VS Code) is a free, open-source code editor developed by Microsoft. It's designed to be lightweight yet powerful, making it a popular choice among developers. VS Code supports a wide range of programming languages, including popular ones like JavaScript, Python, and Java, as well as lesser-known languages through extensions. These extensions are available through the Visual Studio Code Marketplace and can add functionality such as language support, debugging capabilities, and integration with other tools and services.

One of the key features of VS Code is IntelliSense, which provides intelligent code completion, code suggestions, and auto-completion based on variable types, function definitions, and imported modules. This helps developers write code faster and with fewer errors. VS Code also includes built-in Git integration, allowing developers to manage version control directly within the editor.

Another notable feature of VS Code is its debugging capabilities. It includes a built-in debugger that supports breakpoints, step-through debugging, and variable inspection, making it easier for developers to identify and fix issues in their code.

VS Code's user interface is highly customizable, allowing developers to choose from a variety of themes, icon sets, and keyboard shortcuts to suit their preferences. It also supports extensions for additional customization, such as customizing the layout, adding new features, or integrating with external services.

Overall, Visual Studio Code is a versatile and powerful code editor that has gained popularity among developers for its ease of use, extensive feature set, and strong community support. Its cross-platform compatibility and wide range of language support make it a valuable tool for developers working on a variety of projects.

DOCKER DESKTOP:

Docker Desktop is a powerful tool for building, shipping, and running applications in containers. It provides an easy-to-use interface for managing containerized applications on both Windows and macOS operating systems. Docker Desktop includes Docker Engine, Docker CLI client, Docker Compose, Docker Content Trust, Kubernetes, and other Docker tools, making it a comprehensive solution for container development and deployment.

One of the key features of Docker Desktop is its ability to create lightweight, portable containers that can run on any machine that has Docker installed, regardless of the underlying operating system. This makes it easy to develop and test applications in a consistent environment, reducing the risk of compatibility issues when moving applications between development, testing, and production environments.

Docker Desktop also includes Docker Compose, which is a tool for defining and running multi-container Docker applications. Compose uses a YAML file to define the services, networks, and volumes required for an application, making it easy to manage complex multi-container applications with a single command.

Another feature of Docker Desktop is its integration with Kubernetes, which is an open-source platform for automating the deployment, scaling, and management of containerized applications. Docker Desktop includes a lightweight Kubernetes distribution that can be used for local development and testing of Kubernetes applications.

ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

Artificial intelligence (AI) is intelligence demonstrated by machines, as opposed to the natural intelligence displayed by humans or animals. Leading AI textbooks define the field as the study of "intelligent agents" any system that perceives its environment and takes actions that maximize its chance of achieving its goals. Some popular accounts use the term "artificial intelligence" to describe machines that mimic "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving", however this definition is rejected by major AI researchers.

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, and speech recognition and machine vision.

CHATBOT

A chatbot is a computer program that can simulate a conversation or chat with a user in natural language through messaging applications, website, or mobile applications and interact with users according to their input and should be available 24/7. Chatbots are developed and became so popular due to the increased use of smart devices and IoT technology.

Types of chatbots

a. Base-line chatbot:

It is a chatbot that is based on a database and uses if / then logic to create a conversation flow and that takes a lot of time to ensure the understanding of the question and the answer needed.

b. AI chatbot:

This type of chatbot is more complex than base-line but it is more interactive and personalized and needs big data training to be impressive if the problem is matched to their capabilities.

c. Hybrid Model:

A hybrid approach mixes the Base-line & AI chatbot to make it smart and his behavior more expected by depending on database and Ai algorithm to work together.

How do chatbots work?

Briefly and as mentioned in the definition, humans interact with chatbots. There are two ways to interact with a chatbot:

a. Text

Chatbot analyzes the inputted text and matches the text with predefined data called intents which are categorized to manage the conversation. The user utterance is tagged with one of these intents, even if what the user says stretches over two or more intents. Most chatbots will take the intent with the highest score and take the conversation down that avenue.

b. Voice

Some chatbots can interact and understand the voice of the user using a set of application programming interfaces (api's) that converts the recorded voice to the language and then convert the voice to words of that language and then deal with the transformed text as mentioned above.

Options to build a chatbot.

a. From Scratch

At first we have to identify the opportunities for our chatbot and decide its field and scope to achieve efficiency and accuracy. and a precise understanding of the customer needs is required to solve the operational challenges. Then the design of the bot comes to be a significant stage to decide the user engagement with your app or website and we can categorize chatbot interactions as structured and unstructured interactions.

- Structured interaction. You already know about this kind of interaction. You know what your customers will ask and can design it easily — it's just like an FAQ section of your app or website. This information will link to your contact information, services, products, etc.
- Unstructured interaction. The unstructured conversation flow includes freestyle plain text. It's hard to predict what queries will emerge and it looks like an extempore speech competition for your chatbot. the role of AI comes to lights here, it decodes the context of the text based on NLP analysis. while the same NLP will provide a voice to the chatbot. The later choice will need specialized chatbot developers with an understanding of programming languages, machine learning, and AI. We can use some of the code-based frameworks to build and handle the chatbot like wit.ai and api.ai.

b. Using platforms

It is similar to scratch chatbots but the only difference is that you do not have to hire a specialized developer and use the chatbot builder platforms like Chatfuel, Botsify and Rasa, it’s not hard or impossible to achieve it. but it’s not possible to create a NLP-enabled chatbot that can deal with unstructured data.

Selected Platform

We will use OpenAI's gpt-3.5 turbo and gardio as Ui to build the required chatbot.

Well, it has a lot of advantages such as:

- The advanced natural language processing capabilities of GPT-3.5 Turbo, which allows the chatbot to understand and respond to a wide range of user inputs in a conversational manner. This makes the chatbot more engaging and user-friendly, as it can simulate a natural conversation with users.
- The large amount of pre-trained data that GPT-3.5 Turbo has been trained on, which enables the chatbot to generate more accurate and contextually relevant responses. This can help improve the overall user experience and make the chatbot more useful in various applications.

But as we know, nothing is perfect and has its disadvantages like :

- One potential issue is the cost, as using the API for large-scale applications can be expensive.
- GPT-3.5 Turbo may not always provide accurate or relevant responses, especially in complex or specialized domains where the model may not have been trained extensively.

CHAPTER 5
SYSTEM ARCHITECTURE

A system architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

5.1 ARCHITECTURAL DIAGRAM

An architectural diagram is a visual representation that maps out physical implementation for components of a software system. It shows the general structure of the software system and the association, limitations and boundaries between each element.

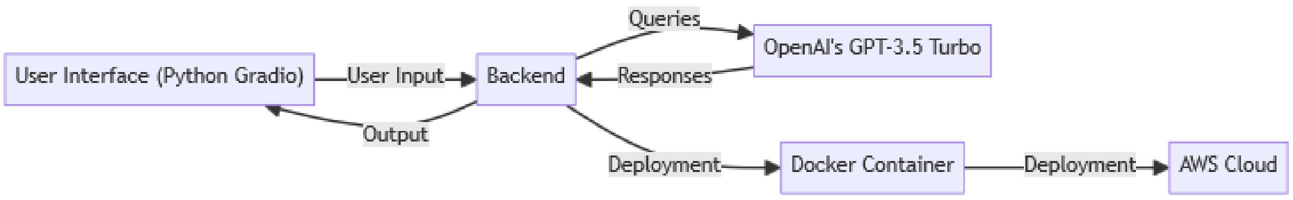


Figure 5.1 Architecture Diagram

5.2 WORKFLOW DIAGRAM

A workflow diagram is a visual layout of a process, project or job in the form of a flow chart. It’s a highly effective way to impart the steps more easily in a business process, how each one will be completed, by whom and in what sequence.

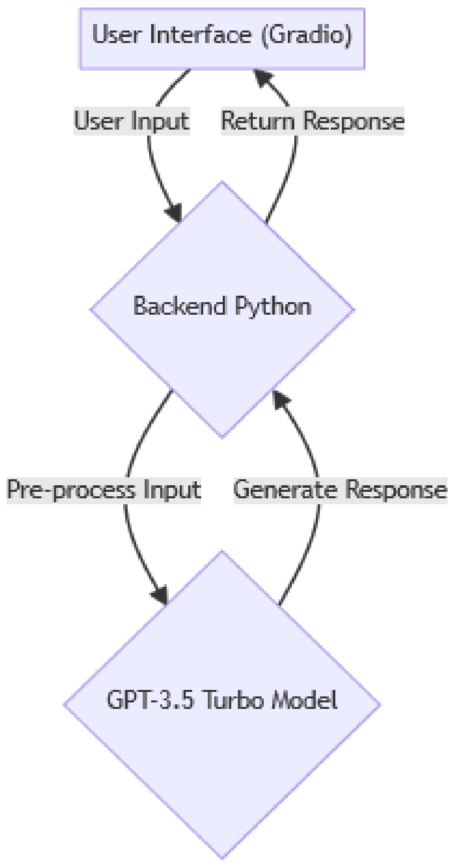


Figure 5.2 Workflow Diagram

CHAPTER 6
SYSTEM DESIGN

System Design is defined as a process of creating architecture for different components, interfaces and modules of the system providing corresponding data helpful in implementing such elements in systems.

System design refers to the process of defining the architecture, modules, interfaces, data for a system to satisfy specified requirements. It is a multi-disciplinary field that involves trade-off analysis, balancing conflicting requirements, and making decisions about design choices that will impact the overall system.

6.1 USE CASE DIAGRAM

Use case diagrams are considered for high level requirement analysis of a system. So, when the requirements of a system are analyzed, the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

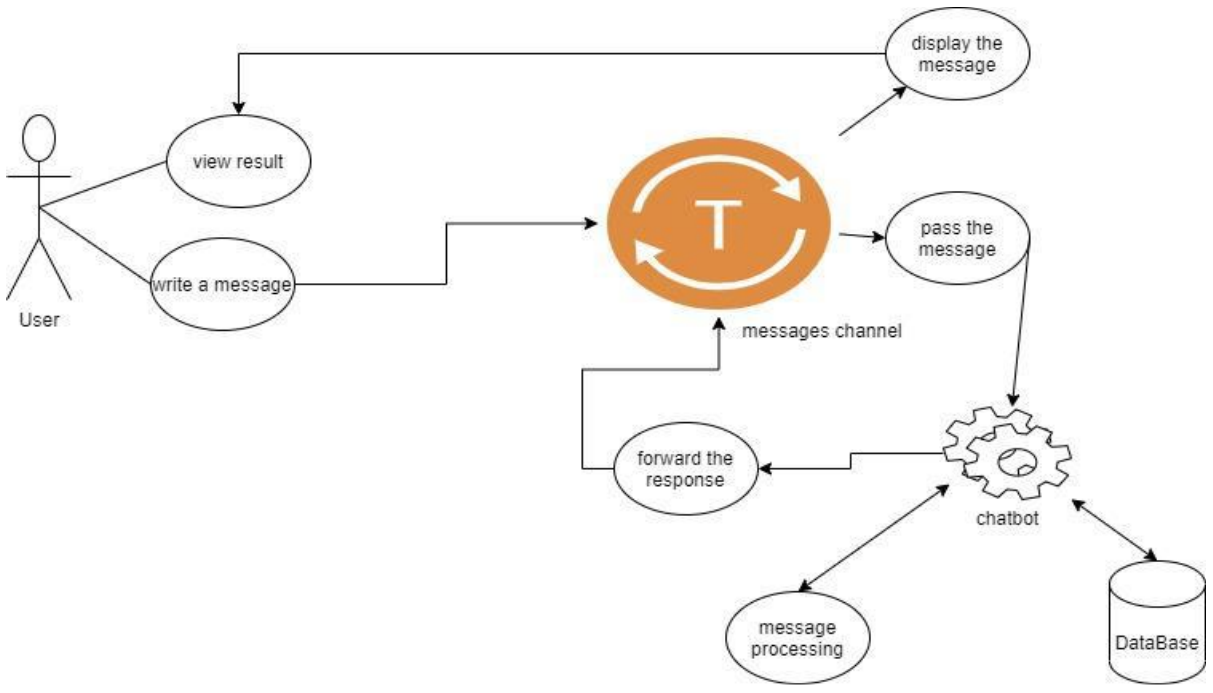


Figure 6.1 Use Case Diagram

6.2ACTIVITY DIAGRAM

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams look like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

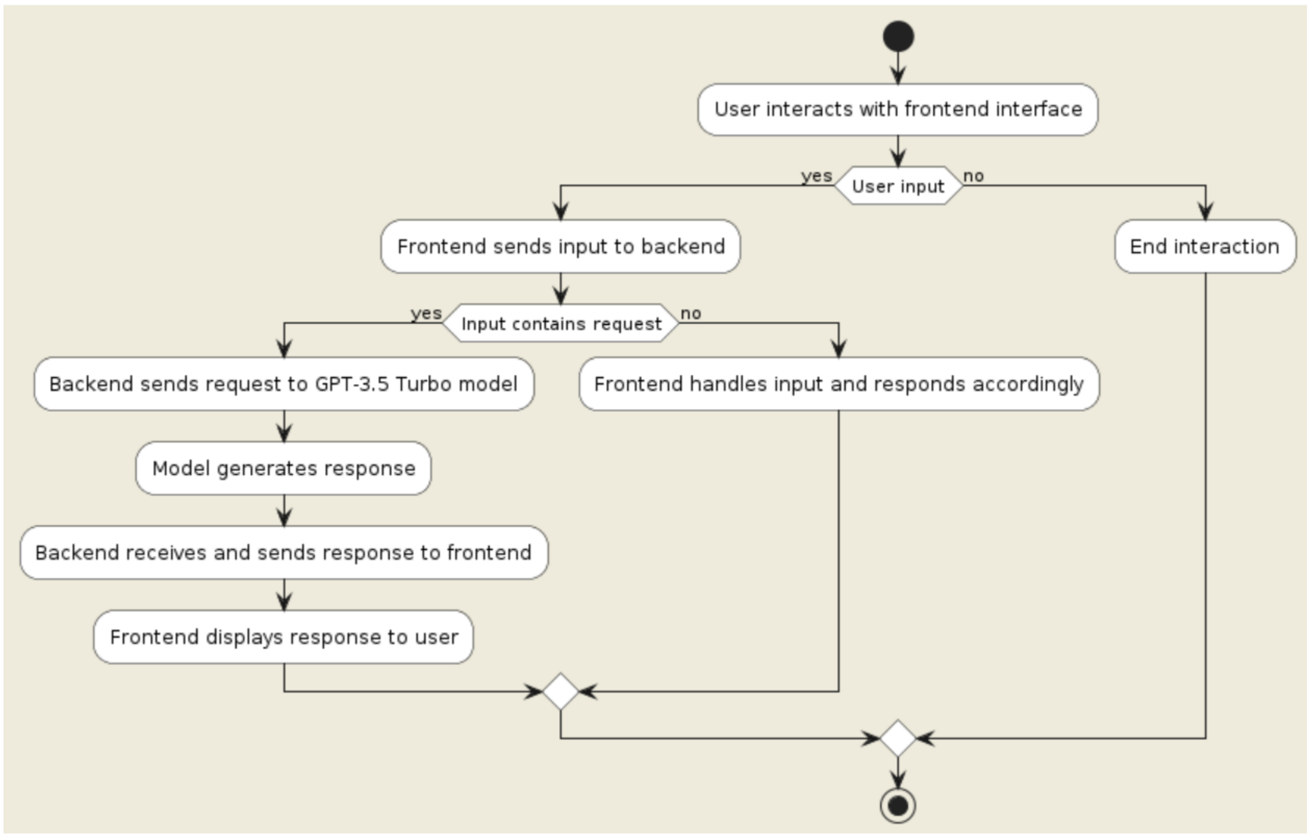

User interacts with frontend interface
yes
User input
no
Frontend sends input to backend
End interaction
yes
Input contains request
no
Backend sends request to GPT-3.5 Turbo model
Frontend handles input and responds accordingly
Model generates response
Backend receives and sends response to frontend
Frontend displays response to user

Figure 6.2. Activity Diagram

6.3 SEQUENCE DIAGRAM

Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behavior within your system. Other dynamic modelling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are in my opinion the most important design-level models for modern business application development.

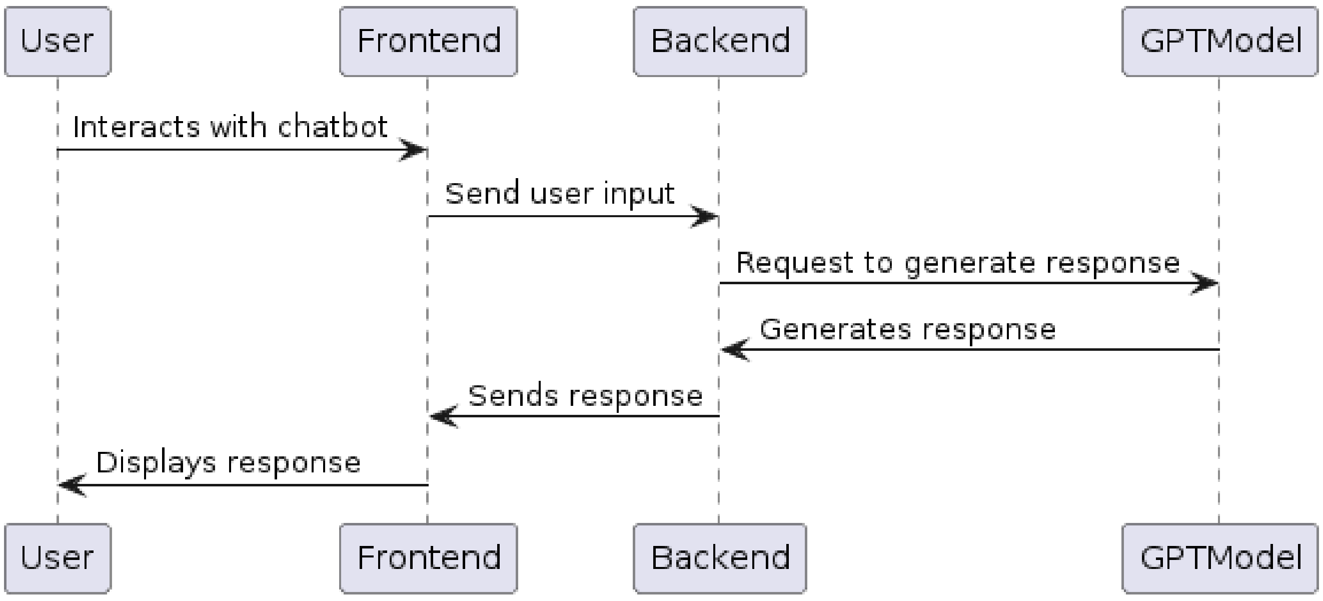


Figure 6.3 Sequence Diagram

CHAPTER 7

MODULES

In a project report, a module refers to a distinct functional unit or component that contributes to the project's overall functionality. Each module serves a specific purpose within the project and may encompass various tasks or features. The module section of the report provides a comprehensive overview of each module, including its objectives, functionalities, implementation details, dependencies, and any challenges encountered. This section helps stakeholders understand the structure and organization of the project, facilitating better comprehension of its components and their interrelations.

7.1 LIST OF MODULES:

8 Python Frontend (Gradio)
9 Python Backend (API Call)
10 Fine tuning Gpt 3.5 turbo
11 Deploying in Cloud

7.2 PYTHON FRONTEND (GRADIO):

The Frontend Interface implemented using Gradio for the mental health chatbot offers a highly interactive and user-friendly experience, leveraging the platform's capabilities for rapid development of machine learning interfaces. Here's a breakdown of its key components:

1. **Input Elements:** Gradio allows the inclusion of various input elements such as text boxes, dropdowns, and sliders, enabling users to input their queries or select options in a structured manner. These elements are designed to be intuitive and easy to use, enhancing the overall user experience.

2. **Output Display:** The interface includes a designated area to display the chatbot's responses. These responses are generated by the backend AI model and are presented in a format that is clear and easy to understand, ensuring effective communication between the user and the chatbot.
3. **Dynamic Interaction:** Gradio's dynamic interface features enable real-time updates and interactions. For example, as users type their queries, the chatbot may suggest completions or provide immediate feedback, enhancing the conversational flow.
4. **Customization Options:** Gradio offers a range of customization options for the interface's appearance and behavior. This includes the ability to customize colors, fonts, and layouts to match the chatbot's branding or to create a visually appealing interface that aligns with the chatbot's purpose.
5. **Accessibility Features:** Gradio interfaces can be designed to be accessible to users with disabilities. This includes support for screen readers, keyboard navigation, and other accessibility features to ensure that all users can engage with the chatbot effectively.
6. **Error Handling:** The interface includes robust error handling mechanisms to manage situations where the chatbot may not understand a query or encounter other issues. Clear error messages and suggestions for rephrasing queries help users navigate these situations smoothly.

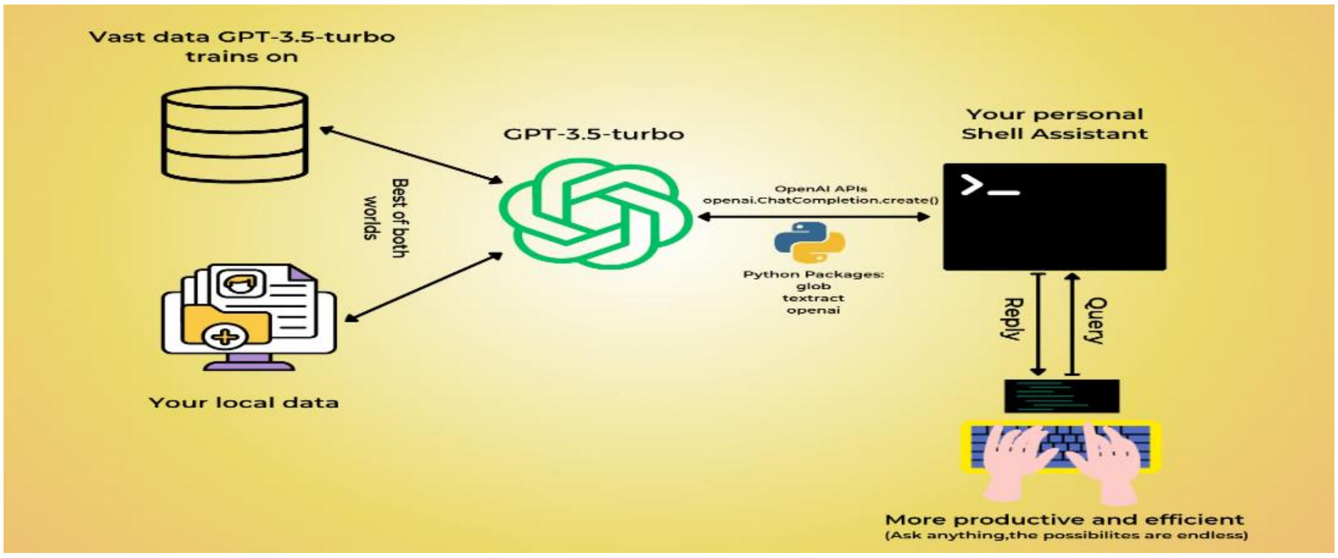
7.3 PYTHON BACKEND (API CALL)

The Backend module of the proposed chatbot system for mental health support manages the communication between the Frontend Interface and OpenAI's GPT-3.5 Turbo model, facilitating the exchange of messages and responses. It consists of several key components:

1. **Input Processing:** Upon receiving a user input from the Frontend Interface, the Backend module processes the input to remove any unnecessary characters or

formatting. This ensures that the input is in a suitable format for the ChatGPT API call.

2. **ChatGPT API Integration:** The Backend module interacts with the ChatGPT API to send the processed user input and receive a response from the AI model. This integration is crucial for leveraging the AI model's capabilities in generating meaningful and empathetic responses.
3. **Response Handling:** Once the Backend receives a response from the ChatGPT API, it processes the response to format it appropriately for display in the Frontend Interface. This may involve adding formatting, such as line breaks or emojis, to make the response more visually appealing and easier to read.
4. **Error Handling:** The Backend module includes error handling mechanisms to manage any issues that may arise during the API call or response processing. This ensures that the chatbot remains functional and responsive even in the event of API errors or other issues.
5. **Feedback Loop:** The Backend module may include a feedback loop mechanism to collect feedback from users on the chatbot's responses. This feedback can be used to improve the chatbot's performance over time, making it more effective in providing mental health support.


Vast data GPT-3.5-turbo trains on
GPT-3.5-turbo
Your personal Shell Assistant
Best of both worlds
OpenAI APIs
openai.ChatCompletion.create()
>_
Python Packages:
glob
textract
openai
Reply
Query
Your local data
More productive and efficient
(Ask anything,the possibilites are endless)

7.1 Python – ChatGpt Api Call

7.4 FINE-TUNE AN OPENAI CHATGPT MODE:

The ChatGPT 3.5 Turbo module is a key component of the proposed chatbot system for mental health support, serving as the AI model responsible for generating responses to user inputs. This module is based on OpenAI's GPT-3.5 architecture, which is a state-of-the-art language model known for its ability to understand and generate human-like text.

the following steps to build a fine-tuned model

- Preparing the training data
- Uploading the training data to OpenAI servers
- Creating a fine-tuned model with the training data

7.4.1 PREPARING THE TRAINING DATA

The fine-tuning process typically begins with a dataset that is carefully curated and labeled, and it involves raining the model on this dataset using techniques such as transfer learning. The model's parameters are adjusted during fine-tuning to make it more accurate and contextually appropriate to generate responses in the target domain. The model can acquire domain-specific knowledge, language patterns, and nuances by fine-tuning, enabling it to generate more relevant and coherent responses for specific applications or use cases.

During the fine-tuning process, it is necessary to provide the training data as a JSON file. You should create a diverse set of target conversations that expect the model to perform after the fine-tuning process is completed.

This training data can come from a variety of sources, such as books, articles, specialized datasets, or we can prepare it manually.

Training Data Format

The training data should be in conversational chat format and it is required to fine-tune *gpt-3.5-turbo*. For example, for our usecase, we have created the *input_data.jsonl* which has the required pattern. Note that each message appears in a new line.

```
    {"messages": [{"role": "user", "content": "Reset my password for
account ID 12345"}, {"role": "assistant", "content": "Your password
has been reset. Please check your email for the new login details."}]}
    {"messages": [{"role": "user", "content": "How do I update my
billing information?"}, {"role": "assistant", "content": "To update
your billing information, log in to your account, go to the 'Billing'
section, and follow the prompts."}]}
```

To start the fine-tuning a model, we must provide at least 10 examples. For optimal results surpassing base models, we should aim to provide a few hundred or more high-quality examples, preferably vetted by human experts. The right number of examples varies on the exact use case and the level of required accuracy.

7.4.2 UPLOADING THE TRAINING DATA TO OPENAI SERVERS:

The next step towards fine-tuning a model is to upload the training data file to OpenAI server using the **"client.files.create()"** API.

After the file is uploaded, a file ID is generated that we can refer to every time we need without uploading the file again and again.

```
"File has been uploaded to OpenAI with id file
EVvrt7QWovkOH4gz2NdrC20U"
```

After we upload the file, it may take some time to process. While the file is processing, we can still create a fine-tuning job but it will not start until the file processing has completed.

The maximum file upload size is 1 GB, though we do not suggest fine-tuning with that amount of data since you are unlikely to need that large of an amount to see improvements.

7.4.3 FINE-TUNING A MODEL WITH THE UPLOADED TRAINING DATA

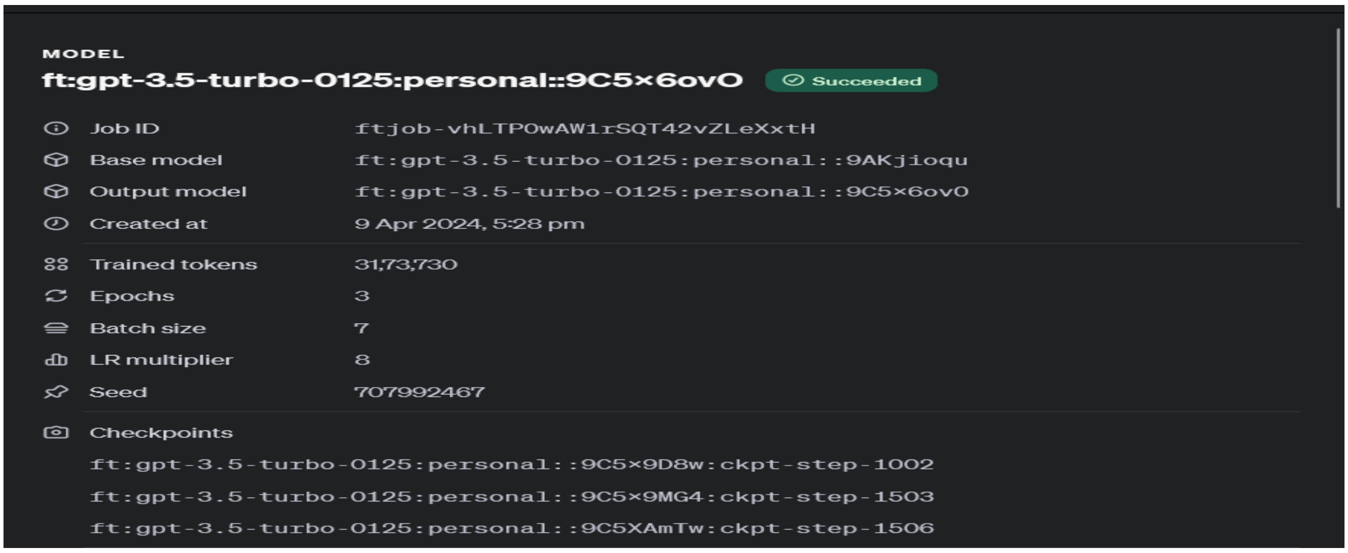
After the file has been uploaded to the OpenAI server, we can use the **"client.fine_tuning.jobs.create()"** API to fine-tune the selected chatGPT model with the supplied training data.The API returns a Job ID which refers to an asynchronous Job that has been created on the backend.

```
"Fine Tune Job has been created with id  ftjob-
vhLTPOwAW1rSQT42vZLeXxtH"
```

It's important to note that the process of creating the model may vary in duration, taking anywhere from a few minutes to hours. The OpenAI servers will continue processing your fine-tuned model until it reaches completion.

We can enquire about the progress of the fine-tuning job using the "**client.fine_tuning.jobs.list_events()"** API that returns the latest N statuses related to the job.Once the job has been completed, we can get the ID of the fine-tuned model in the final status message:

```
"{FineTuningJobEvent(id='ftevent-OqhBL7WqEkkYTQqFCjGS1BSu',
created_at=1702289919, level='info', message='New fine-tuned
model      created:ft:gpt-3.5-turbo-0613:personal::8UXexX8R',
object='fine_tuning.job.event', data={}, type='message')"}
```



MODEL
ft:gpt-3.5-turbo-0125:personal::9C5x6ovO Succeeded

Job ID	ftjob-vhLTPOwAW1rSQT42vZLeXxtH
Base model	ft:gpt-3.5-turbo-0125:personal::9AKjioqu
Output model	ft:gpt-3.5-turbo-0125:personal::9C5x6ovO
Created at	9 Apr 2024, 5:28 pm
Trained tokens	31,73,730
Epochs	3
Batch size	7
LR multiplier	8
Seed	707992467

Checkpoints
ft:gpt-3.5-turbo-0125:personal::9C5x9D8w:ckpt-step-1002
ft:gpt-3.5-turbo-0125:personal::9C5x9MG4:ckpt-step-1503
ft:gpt-3.5-turbo-0125:personal::9C5XAmTw:ckpt-step-1506

7.2 Successful Creation of Fine Tuned Model

7.5 DEPLOYMENT IN CLOUD

AWS (AMAZON WEB SERVICES):

- **Purpose:** AWS provides a scalable and reliable cloud computing platform for deploying and managing applications. It offers a wide range of services that facilitate the deployment and operation of containerized applications.
- **Functionality:** AWS services such as Amazon Elastic Container Service (ECS) or Amazon Elastic Kubernetes Service (EKS) can be used to deploy and manage Docker containers. These services provide features like automatic scaling, load balancing, and monitoring, making it easier to deploy and operate containerized applications on AWS.

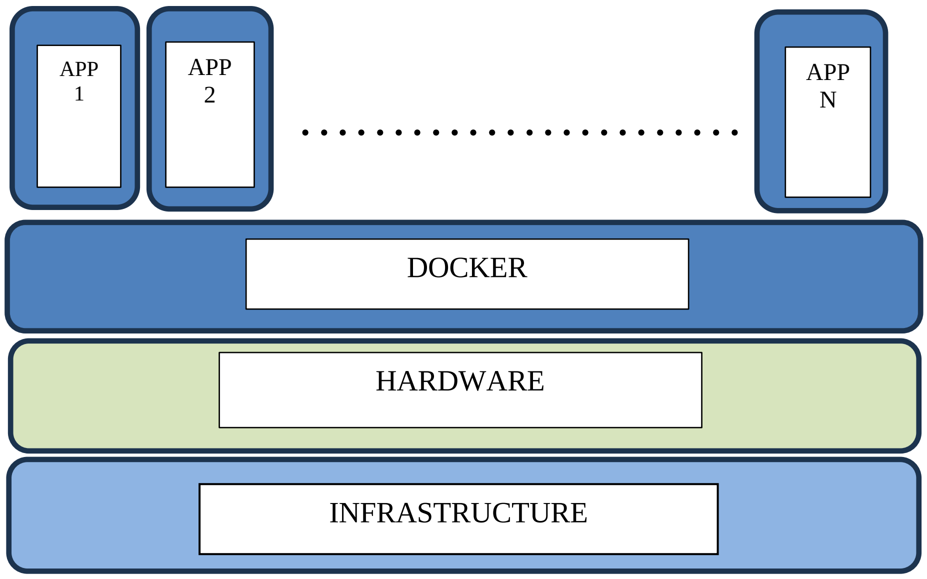
DOCKER CONTAINERS:

- **Purpose:** Docker containers package the chatbot system and its dependencies into a standardized unit that can run on any platform where Docker is installed, ensuring consistency across different environments.
- **Functionality:** Docker containers provide an isolated environment for the chatbot system to run, ensuring that it is not affected by changes or issues in the underlying infrastructure. Docker images are used to create containers, which can then be deployed on AWS using services like ECS or EKS.

DEPLOYMENT PROCESS:

- **Build Docker Image:** The chatbot system and its dependencies are packaged into a Docker image using a Dockerfile.
- **Push Image to Docker Registry:** The Docker image is pushed to a Docker registry, such as Docker Hub or Amazon Elastic Container Registry (ECR), where it can be accessed by AWS services.

- **Create ECS/EKS Cluster:** A cluster is created on ECS or EKS to run the Docker containers. The cluster provides the underlying infrastructure for running and managing the containers.
- **Deploy Containers:** The Docker containers are deployed to the ECS or EKS cluster, where they are started and managed by the platform. The platform handles tasks like scaling, load balancing, and monitoring of the containers.
- **Access the Chatbot:** Once deployed, the chatbot is accessible over the internet, and users can interact with it through the frontend interface.



7.3 Docker deployment

CHAPTER 8
SYSTEM TESTING

In traditional software systems, humans write the logic which interacts with data to produce a desired behavior. Our software tests help ensure that this written logic aligns with the actual expected behavior. A typical software testing suite will include:

- Unit tests which operate on atomic pieces of the codebase and can be run quickly during development,
- Regression tests replicate bugs that we've previously encountered and fixed,
- Integration tests which are typically longer-running tests that observe higher-level behaviors that leverage multiple components in the codebase, and follow conventions such as:
- Don't merge code unless all tests are passing, Always write tests for newly introduced logic when contributing code, When contributing a bug fix, be sure to write a test to capture the bug and prevent future regressions.

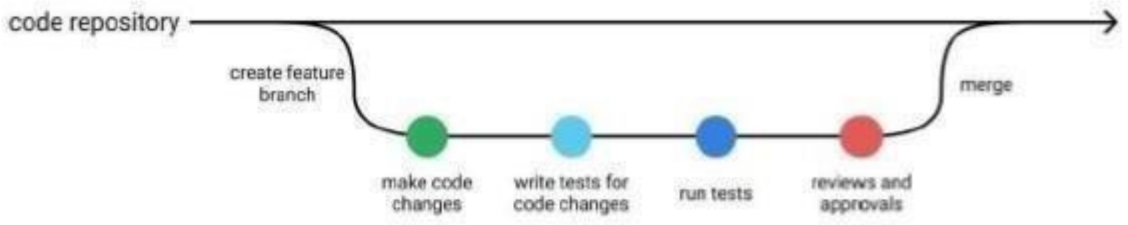


FIGURE 8.1 A Typical Workflow of Software Development

When we run our testing suite against the new code, we'll get a report of the specific behaviors that we've written tests around and verify that our code changes don't affect the expected behavior of the system. If a test fails, we'll know which specific behavior is no longer aligned with our expected output. We can also look at this testing report to get an understanding of how extensive our tests are by looking at metrics such as code coverage.

Let's contrast this with a typical workflow for developing machine learning systems. After training a new model, we'll typically produce an evaluation report including:

- Performance of an established metric on a validation dataset,

model through the inclusion of a new dataset during training. We need more nuanced reports of model behavior to identify such cases, which is exactly where model testing can help.

For machine learning systems, we should be running model evaluation and model tests in parallel.

- Model evaluation covers metrics and plots which summarize performance on a validation or test dataset.
- Model testing involves explicit checks for behaviors that we expect our model to follow. Both of these perspectives are instrumental in building high-quality models.

In practice, most people are doing a combination of the two where evaluation metrics are calculated automatically and some level of model "testing" is done manually through error analysis (i.e. classifying failure modes). Developing model tests for machine learning systems can offer a systematic approach towards error analysis. There're two general classes of model tests that we'll want to write.

- Pre-train tests allow us to identify some bugs early on and short-circuit a training job.
- Post-train tests use the trained model artifact to inspect behaviors for a variety of important scenarios that we define.

PRE-TRAIN TESTS:

There's some test that we can run without needing trained parameters. These tests include:

- Check the shape of your model output and ensure it aligns with the labels in your dataset.
- Make sure a single gradient step on a batch of data yields a decrease in your loss
- Make assertions about your datasets
- Check for label leakage between your training and validation datasets.
- The main goal here is to identify some errors early so we can avoid a wasted training job.

POST-TRAIN TESTS

However, in order for us to be able to understand model behaviors we'll need to test against trained model artifacts. These tests aim to interrogate the logic learned during trainingand provide us with a behavioral report of model performance.

PAPER HIGHLIGHT: Beyond Accuracy: Behavioral Testing of NLP Models with Check List

The authors of the above paper present three different types of model tests that we can use to understand behavioral attributes.

INVARIANCE TESTS

Invariance tests allow us to describe a set of perturbations we should be able to make to the input without affecting the model's output. We can use these perturbations to producepairs of input examples (original and perturbed) and check for consistency in the model predictions. This is closely related to the concept of data augmentation, where we apply perturbations to inputs during training and preserve the original label.

DIRECTIONAL EXPECTATION TESTS

Directional expectation tests, on the other hand, allow us to define a set of perturbations to the input which should have a predictable effect on the model output.

MINIMUM FUNCTIONALITY TESTS (AKA DATA UNIT TESTS)

Just as software unit tests aim to isolate and test atomic components in your codebase, data unit tests allow us to quantify model performance for specific cases found in your data.

This allows you to identify critical scenarios where prediction errors lead to high consequences. You may also decide to write data unit tests for failure modes that you uncover during error analysis; this allows you to "automate" searching for such errors in future models.

Snorkel also has introduced a very similar approach through their concept of slicing functions. These are programmatic functions which allow us to identify subsets of a dataset which meet certain criteria. For example, you might write a slicing function to identify sentences less than 5 words to evaluate how the model performs on short pieces of text.

ORGANIZING TESTS

In traditional software tests, we typically organize our tests to mirror the structure of the code repository. However, this approach doesn't translate well to machine learning models since our logic is structured by the parameters of the model

MODEL DEVELOPMENT PIPELINE

Putting this all together, we can revise our diagram of the model development process to include pre-train and post-train tests. These tests outputs can be displayed alongside model evaluation reports for review during the last step in the pipeline. Depending on the nature of your model training, you may choose to automatically approve models provided that they meet some specified criteria.

1. **Frontend Interface (Python Gradio):**

   - **Functional Testing:** We verified that the interface displayed correctly and was interactive.
   - **Usability Testing:** We evaluated the interface for ease of use and user-friendliness.
   - **Compatibility Testing:** We tested the interface on different browsers and devices.
   - **Performance Testing:** We checked the interface's response time and efficiency.

2. **Backend (Python):**

   - **Functional Testing:** We ensured that the backend correctly received user inputs from the frontend.
   - **Integration Testing:** We verified that the backend communicated effectively with the GPT-3.5 Turbo model.
   - **Error Handling Testing:** We tested how the backend handled errors, such as invalid inputs or communication failures.

3. **OpenAI's GPT-3.5 Turbo Model:**

   - **Functional Testing:** We confirmed that the model generated appropriate responses to user inputs.
   - **Quality Assurance Testing:** We evaluated the quality of the responses in terms of relevance and empathy.
   - **Performance Testing:** We measured the response time of the model under different loads.

4. **Deployment on AWS using Docker:**

   - **Deployment Testing:** We ensured that the chatbot could be deployed successfully on AWS using Docker containers.
   - **Scalability Testing:** We tested the system's ability to handle increased loads by adding more Docker containers.
   - **Security Testing:** We verified that the deployed system was secure against common vulnerabilities.

5. **Integration Testing:**

   - **End-to-End Testing:** We tested the entire system flow from user input to response to ensure seamless integration of all components.
   - **Data Flow Testing:** We verified that data flowed correctly between the frontend, backend, and GPT-3.5 Turbo model.

6. **Performance Testing:**

   - **Load Testing:** We tested the system's performance under varying loads to ensure it could handle multiple users simultaneously.
   - **Stress Testing:** We tested the system's performance under extreme loads to determine its breaking point.

7. **User Acceptance Testing (UAT):**

   - **Beta Testing:** We allowed real users to interact with the chatbot to gather feedback on its usability and effectiveness.
   - **Feedback Analysis:** We analyzed user feedback to identify areas for improvement and made necessary adjustments.

8. **Security Testing:**

   - **Vulnerability Assessment:** We identified and mitigated potential security vulnerabilities in the system.

- **Data Privacy Testing:** We ensured that user data was handled securely and in compliance with relevant regulations.

9. **Scalability and Reliability Testing:**

   - **Scalability Testing:** We tested the system's ability to scale up or down based on demand.
   - **Reliability Testing:** We verified that the system was reliable and available when needed.

10. **Accessibility Testing:**

    - **Accessibility Compliance:** We ensured that the chatbot met accessibility standards for users with disabilities.
    - **Usability Testing:** We evaluated the chatbot's usability for users with different accessibility needs.

CHAPTER 9

CONCLUSION AND FUTURE ENHANCEMENT

In conclusion, the integrated system for the mental health chatbot demonstrates a sophisticated architecture that harmoniously blends frontend, backend, and AI components. The Python Gradio frontend stands out for its user-friendly interface, enabling individuals to interact with the chatbot intuitively.

On the backend, Python orchestrates the seamless communication between the frontend and the GPT-3.5 Turbo model from OpenAI. This model, renowned for its vast dataset and language understanding capabilities, plays a pivotal role in generating responses that are not only accurate but also empathetic, enhancing the user's sense of engagement and support.

The deployment on AWS through Docker containers underscores the system's scalability, reliability, and security. Leveraging AWS's infrastructure ensures that the chatbot can handle varying loads and maintain high availability, critical for providing uninterrupted mental health support.

Overall, this meticulously designed architecture maximizes the strengths of each component to create a chatbot system that offers personalized, effective, and empathetic mental health support, thereby marking a significant milestone in the application of AI for mental wellness.

9.1 LIMITATIONS:

While the proposed chatbot system for mental health treatment offers numerous benefits, it also faces several limitations:

- **Dependency on Data Quality**: The effectiveness of the chatbot heavily relies on the quality and diversity of the data used to train the GPT-3.5 Turbo model.

Biases or gaps in the training data could lead to inaccurate or inappropriate responses.

- **Lack of Emotional Intelligence**: While the chatbot can generate empathetic responses based on patterns in the training data, it lacks true emotional intelligence. It may struggle to understand complex emotions or provide nuanced support.
- **Privacy and Security Concerns**: Storing and processing sensitive mental health information raises privacy and security concerns. Adequate measures must be in place to protect user data and comply with relevant regulations.
- **Limited Scope of Assistance**: The chatbot's support is limited to text-based interactions and may not be suitable for individuals who require more personalized or immediate assistance, such as those in crisis situations.
- **Ethical Considerations**: Using AI in mental health care raises ethical questions regarding autonomy, accountability, and the potential for unintended consequences. Clear guidelines and ethical frameworks are essential to mitigate these risks.

9.2 FUTURE ENHANCEMENT

The future enhancement of the chatbot system for mental health treatment could focus on several areas to improve its effectiveness and user experience:

- **Enhanced Emotional Intelligence**: Future iterations could incorporate advanced natural language understanding capabilities to better recognize and respond to complex emotions, improving the chatbot's ability to provide empathetic support.
- **Multimodal Interaction**: Integrating voice, video, and other forms of communication could enhance the user experience and make the chatbot more accessible to individuals with different communication preferences or accessibility needs.

- **Collaboration with Healthcare Professionals**: The chatbot could be integrated into existing healthcare systems to facilitate collaboration between users and healthcare professionals, enabling seamless information sharing and coordinated care.
- **Continuous Learning and Improvement**: Implementing mechanisms for continuous learning and improvement would ensure that the chatbot stays up-to-date with the latest research and best practices in mental health care.
- **Integration with Teletherapy Services**: Integrating the chatbot with teletherapy services could provide a seamless transition between automated support and human-led therapy, offering users a comprehensive mental health care experience.

APPENDIX 1
SOURCE CODE

DATA VALIDATION

```
import json
import tiktoken # for token counting
import numpy as np
from collections import defaultdict
```

// DATA LOADING

```
data_path = "D:\\ChatBot_for_mental_health\\combined_dataset1.jsonl"
dataset = []

with open(data_path, 'r', encoding='utf-8') as f:
    lines = f.read().splitlines()
    for line in lines:
        # Load each line as a separate JSON object
        try:
            json_obj = json.loads(line)
            dataset.append(json_obj)
        except json.JSONDecodeError as e:
            print(f"Error decoding JSON: {e}")

print("Num examples:", len(dataset))
print("First example:")
for message in dataset[0]["messages"]:
    print(message)
```

//FORMAT VALIDATION

```
format_errors = defaultdict(int)

for ex in dataset:
    if not isinstance(ex, dict):
        format_errors["data_type"] += 1
```

```
            continue

        messages = ex.get("messages", None)
        if not messages:
            format_errors["missing_messages_list"] += 1
            continue

        for message in messages:
            if "role" not in message or "content" not in message:
                format_errors["message_missing_key"] += 1


            if any(k not in ("role", "content", "name", "function_call",
    "weight") for k in message):
                format_errors["message_unrecognized_key"] += 1

            if message.get("role", None) not in ("system", "user",
"assistant", "function"):
                format_errors["unrecognized_role"] += 1

            content = message.get("content", None)
            function_call = message.get("function_call", None)

            if (not content and not function_call) or not
isinstance(content, str):
                format_errors["missing_content"] += 1

        if not any(message.get("role", None) == "assistant" for message in
messages):
            format_errors["example_missing_assistant_message"] += 1

if format_errors:
    print("Found errors:")
        for k, v in format_errors.items():
            print(f"{k}: {v}")
else:
        print("No errors found")
```

// TOKEN COUNTING UTILITIES

```
encoding = tiktoken.get_encoding("cl100k_base")

# not exact!
# simplified from https://github.com/openai/openai-
cookbook/blob/main/examples/How_to_count_tokens_with_tiktoken.ipynb
def num_tokens_from_messages(messages, tokens_per_message=3,
tokens_per_name=1):
    num_tokens = 0
    for message in messages:
        num_tokens += tokens_per_message
        for key, value in message.items():
            num_tokens += len(encoding.encode(value))
            if key == "name":
                num_tokens += tokens_per_name
    num_tokens += 3
    return num_tokens

def num_assistant_tokens_from_messages(messages):
    num_tokens = 0
    for message in messages:
        if message["role"] == "assistant":
            num_tokens += len(encoding.encode(message["content"]))
    return num_tokens

def print_distribution(values, name):
    print(f"\n#### Distribution of {name}:")
    print(f"min / max: {min(values)}, {max(values)}")
    print(f"mean / median: {np.mean(values)}, {np.median(values)}")
    print(f"p5 / p95: {np.quantile(values, 0.1)}, {np.quantile(values,
0.9)}")
```

//DATA WARNINGS AND TOKEN COUNTS

```
# Warnings and tokens counts
n_missing_system = 0
n_missing_user = 0
n_messages = []
convo_lens = []
assistant_message_lens = []

for ex in dataset:
    messages = ex["messages"]
    if not any(message["role"] == "system" for message in messages):
        n_missing_system += 1
    if not any(message["role"] == "user" for message in messages):
        n_missing_user += 1
    n_messages.append(len(messages))
    convo_lens.append(num_tokens_from_messages(messages))
    assistant_message_lens.append(num_assistant_tokens_from_messages(me
ssages))

print("Num examples missing system message:", n_missing_system)
print("Num examples missing user message:", n_missing_user)
print_distribution(n_messages, "num_messages_per_example")
print_distribution(convo_lens, "num_total_tokens_per_example")
print_distribution(assistant_message_lens,
"num_assistant_tokens_per_example")
n_too_long = sum(l > 4096 for l in convo_lens)
print(f"\n{n_too_long} examples may be over the 4096 token limit, they
will be truncated during fine-tuning")
```

//COST ESTIMATION

```
# Pricing and default n_epochs estimate
MAX_TOKENS_PER_EXAMPLE = 4096

TARGET_EPOCHS = 3
MIN_TARGET_EXAMPLES = 100
MAX_TARGET_EXAMPLES = 25000



MIN_DEFAULT_EPOCHS = 1
MAX_DEFAULT_EPOCHS = 25

n_epochs = TARGET_EPOCHS
n_train_examples = len(dataset)
if n_train_examples * TARGET_EPOCHS < MIN_TARGET_EXAMPLES:
    n_epochs = min(MAX_DEFAULT_EPOCHS, MIN_TARGET_EXAMPLES //
n_train_examples)
elif n_train_examples * TARGET_EPOCHS > MAX_TARGET_EXAMPLES:
    n_epochs = max(MIN_DEFAULT_EPOCHS, MAX_TARGET_EXAMPLES //
n_train_examples)

n_billing_tokens_in_dataset = sum(min(MAX_TOKENS_PER_EXAMPLE, length)
for length in convo_lens)
print(f"Dataset has ~{n_billing_tokens_in_dataset} tokens that will be
charged for during training")
print(f"By default, you'll train for {n_epochs} epochs on this
dataset")
print(f"By default, you'll be charged for ~{n_epochs *
n_billing_tokens_in_dataset} tokens")
```

DATA UPLOADING:

```
import os
from openai import OpenAI

input_jsonl_file = 'input_data.jsonl'

client = OpenAI(
  api_key=os.environ.get("OPENAI_API_KEY")
)

file = client.files.create(
  file=open(input_jsonl_file, "rb"),
  purpose="fine-tune"
)

print("File has been uploaded to OpenAI with id ", file.id)
```

CREATING FINE TUNING JOB:

```
import os
from openai import OpenAI

input_jsonl_file = 'input_data.jsonl'

client = OpenAI(
  api_key=os.environ.get("OPENAI_API_KEY")
)

file = client.files.create(
  file=open(input_jsonl_file, "rb"),
  purpose="fine-tune"
)
```

```
print("File has been uploaded to OpenAI with id ", file.id)

ft_job = client.fine_tuning.jobs.create(
  training_file=file.id,
  model="gpt-3.5-turbo"
)

print("Fine Tune Job has been created with id ", ft_job.id)
```

//PRINTING LAST N STATUS ABOUT THE JOB

```
events =
client.fine_tuning.jobs.list_events(fine_tuning_job_id=ft_job.id,
limit=10)

print(events)
```

MENTALHEATHCHATBOT.PY

```
import os
import openai
from openai import ChatCompletion
import gradio

client = ChatCompletion(api_key=os.environ.get("OPENAI_API_KEY"))

def api_calling(question, history):
    try:
        s = list(sum(history, ()))
        s.append(question)
        inp = ' '.join(s)
        response = client.create(
            messages=[
                {
                    "role": "user",
                    "content": inp
                }
            ],
            model="ft:gpt-3.5-turbo-0125:personal::9C5X6ovO",
            temperature=0,
            max_tokens=1024,
            n=1,
            stop=None
        )
        message = response.choices[0].message.content
    except Exception as e:
        message = f"An error occurred: {str(e)}"
    return message

def update_chat(new_message, history):
    history = history or []
    output = api_calling(new_message, history)
    history.append((new_message, output))
    return history, None

block = gradio.Blocks(theme=gradio.themes.Monochrome())

with block:
    gradio.Markdown("""<h1><center>Mental Health
Chatbot</center></h1>""")
    chatbot = gradio.Chatbot()
    message = gradio.Textbox(label="Type your message here")
    state = gradio.State()

    message.submit(update_chat, [message, state], [chatbot, state])

block.launch(debug=True)
```

DEPLOYMENT IN CLOUD:

//DOKERFILE TO BUILD THE IMAGE

```
FROM python:3
ARG GRADIO_SERVER_PORT=7860
ENV GRADIO_SERVER_PORT=${GRADIO_SERVER_PORT}
ENV GRADIO_SERVER_NAME=0.0.0.0
ENV OPENAI_API_KEY="[redacted]"
WORKDIR /app
COPY MentalChatBot2.py /app/
RUN pip install openai==0.28 &&\
    pip install gradio
EXPOSE 7860
CMD ["python","./MentalChatBot2.py"]
```

//BUILD DOCKER IMAGE

```
docker build -t mentalhealthchatbot .
```

//RUN THE CONTAINER ON LOCALHOST

```
docker run -p 127.0.0.1:7860:7860 -it mentalhealthchatbot
```

// RETRIEVE AN AUTHENTICATION TOKEN AND AUTHENTICATE YOUR DOCKER CLIENT TO YOUR REGISTRY.

USE THE AWS CLI:

```
aws ecr-public get-login-password --region us-east-1 | docker login --
username AWS --password-stdin public.ecr.aws/c2m1h3t7
```

// TAG YOUR IMAGE SO YOU CAN PUSH THE IMAGE TO THIS REPOSITORY

```
docker tag mentalhealthchatbot:latest
public.ecr.aws/c2m1h3t7/mentalhealthchatbot:latest
```

// RUN THE FOLLOWING COMMAND TO PUSH THIS IMAGE TO YOUR NEWLY CREATED AWS REPOSITORY:

```
docker push public.ecr.aws/c2m1h3t7/mentalhealthchatbot:latest
```

CREATING CONTAINER CLUSTER AND RUNNING TASK IN THE EC2 CONTAINER USING AWS GUI:

1.Create container cluster using Amazon ECS

a) Go to ECS (Elastic Container Service) and lick create cluster.

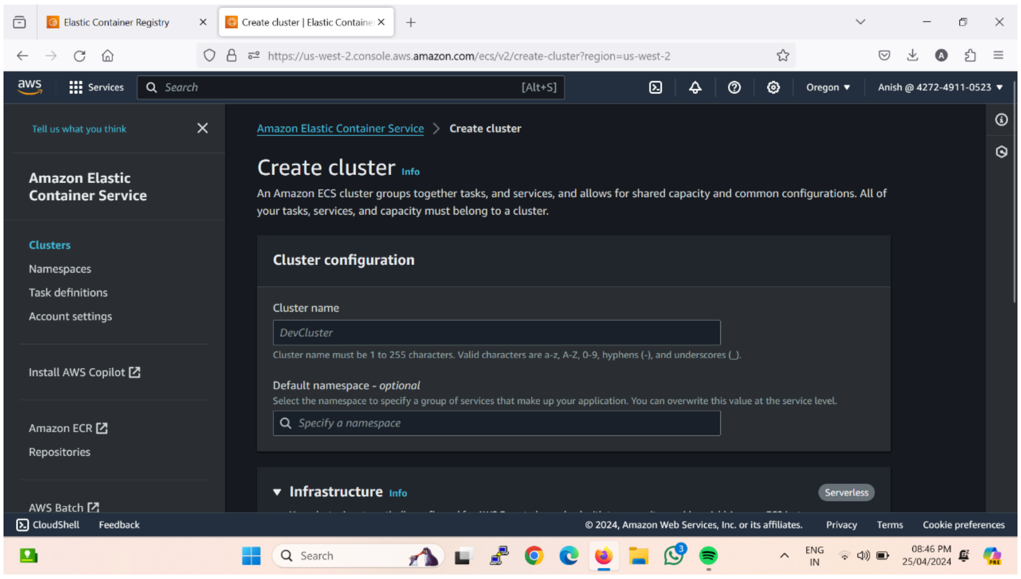
b) Give a name to the cluster.

c) Choose Infrastructure as EC2 instance.

d) Scroll down and give create cluster

e) With the help of CloudFormation it will all the necessary services need for the cluster

(E.g., Auto Scaling Group).



https://us-west-2.console.aws.amazon.com/ecs/v2/create-cluster?region=us-west-2

Amazon Elastic Container Service > Create cluster

Create cluster Info

An Amazon ECS cluster groups together tasks, and services, and allows for shared capacity and common configurations. All of your tasks, services, and capacity must belong to a cluster.

Cluster configuration

Cluster name

DevCluster

Cluster name must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Default namespace - *optional*

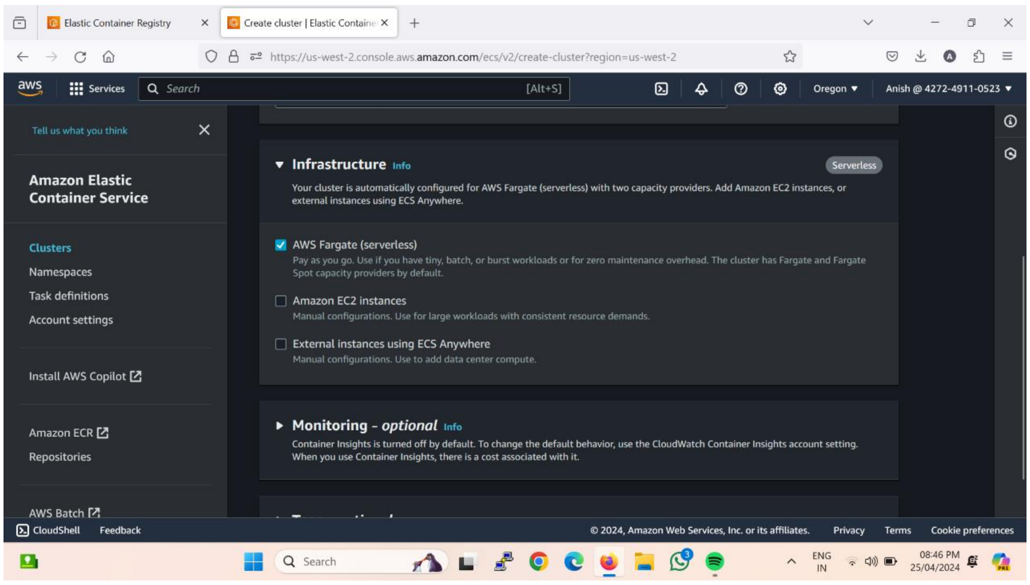
Select the namespace to specify a group of services that make up your application. You can overwrite this value at the service level.

Specify a namespace

Infrastructure Info

Serverless

S1-CREATING CLUSTER



Infrastructure Info — Serverless

Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers. Add Amazon EC2 instances, or external instances using ECS Anywhere.

- [✓] AWS Fargate (serverless)
  Pay as you go. Use if you have tiny, batch, or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.
- [ ] Amazon EC2 instances
  Manual configurations. Use for large workloads with consistent resource demands.
- [ ] External instances using ECS Anywhere
  Manual configurations. Use to add data center compute.

Monitoring - *optional* Info

Container Insights is turned off by default. To change the default behavior, use the CloudWatch Container Insights account setting. When you use Container Insights, there is a cost associated with it.

© 2024, Amazon Web Services, Inc. or its affiliates.

S2-CREATING CLUSTER

2.Creating Task Definition

a) It will take around 5mins to create the cluster, in the mean time we can create Task definition for running a task in the container.

b) Go to Task Definition from the side menu of ECS.

c) Name the Task Definition.

d) Configure the infrastructure.

   a. Choose EC2 instance.

   b. Give operating system as Linux/x86_64

   c. Networking mode as bridge/default (default will be always bridge only).

   d. Enter task size as .5 CPU and .5 memory, this the space required by the container.

   e. Give Task execution role as ec2executionrule which will be automatically created.

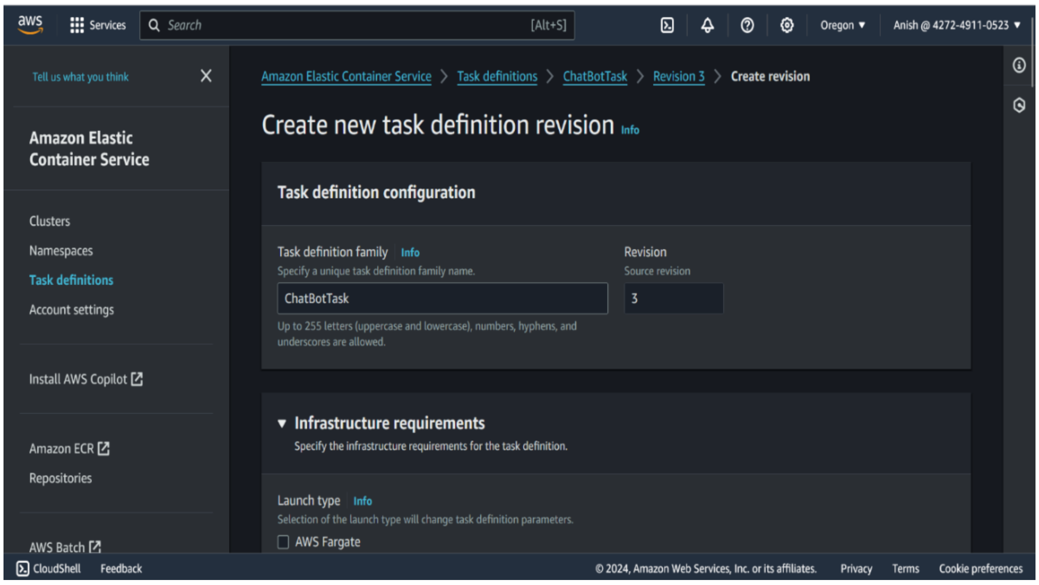
e) Configure the networks.

   a. Name the container, use only small letters.

b. Copy the image uri and paste in the Image URI section and mark the container as essential container.
c. Map Host port and container port to 7860 to 7860 over tcp protocol, this port is used to access the container.
d. Choose App protocol as HTTP so that we can access the container through browser.

f) Now scroll down to extreme bottom and click create.

g) This will create a new task which is ready to run on the EC2 Instance.



Amazon Elastic Container Service > Task definitions > ChatBotTask > Revision 3 > Create revision

Create new task definition revision Info

Task definition configuration

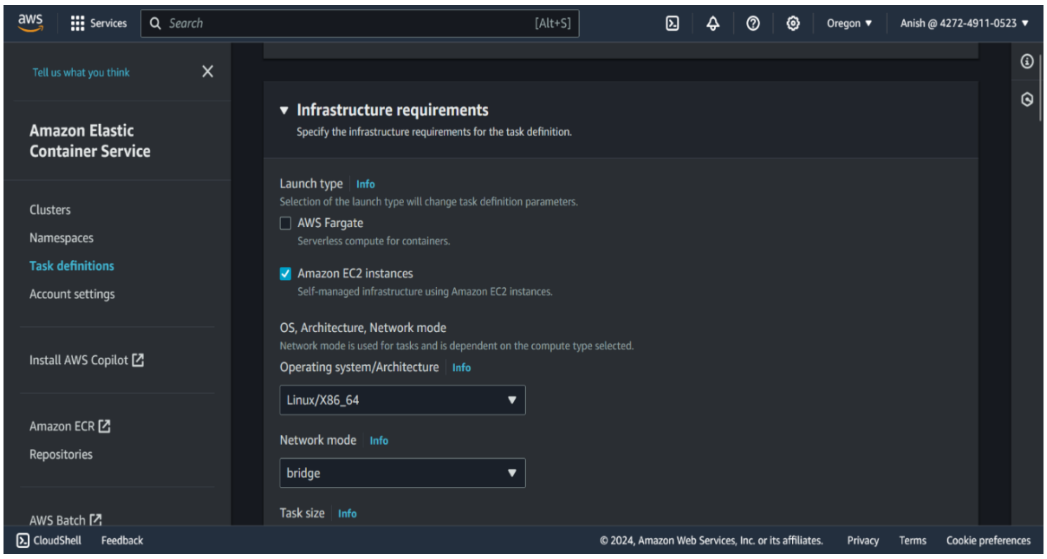
Task definition family Info
Specify a unique task definition family name.
ChatBotTask
Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

Revision
Source revision
3

▼ Infrastructure requirements
Specify the infrastructure requirements for the task definition.

Launch type Info
Selection of the launch type will change task definition parameters.
☐ AWS Fargate

S3-CREATING TASK DEFINITION

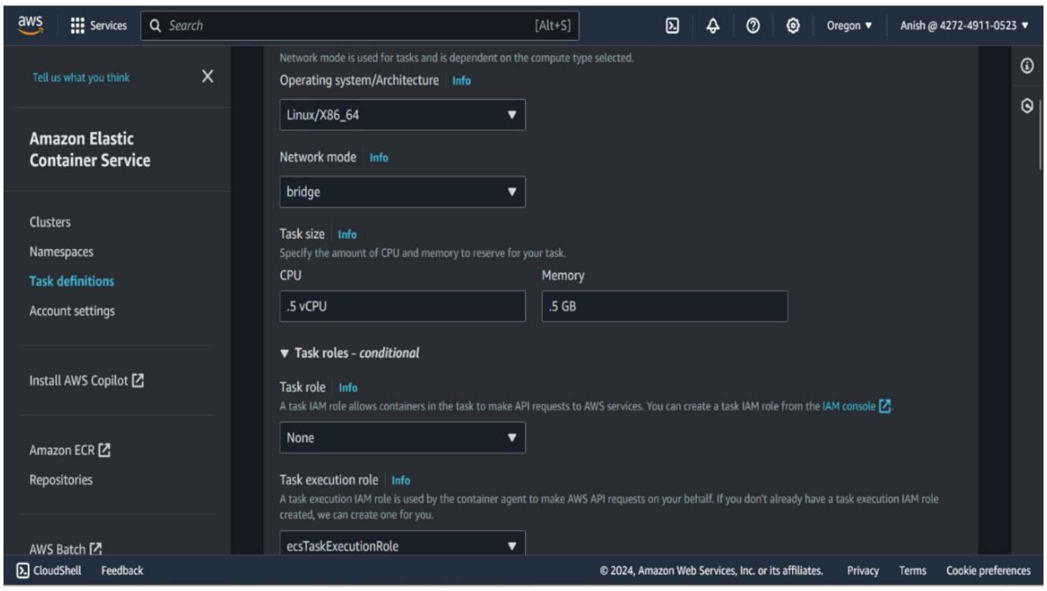


▼ Infrastructure requirements
Specify the infrastructure requirements for the task definition.

Launch type Info
Selection of the launch type will change task definition parameters.
☐ AWS Fargate
Serverless compute for containers.
☑ Amazon EC2 instances
Self-managed infrastructure using Amazon EC2 instances.

OS, Architecture, Network mode
Network mode is used for tasks and is dependent on the compute type selected.
Operating system/Architecture Info
Linux/X86_64
Network mode Info
bridge
Task size Info

S4-CREATING TASK DEFINITION



Network mode is used for tasks and is dependent on the compute type selected.

Operating system/Architecture Info

Linux/X86_64

Network mode Info

bridge

Task size Info

Specify the amount of CPU and memory to reserve for your task.

CPU: .5 vCPU

Memory: .5 GB

▼ Task roles - *conditional*

Task role Info

A task IAM role allows containers in the task to make API requests to AWS services. You can create a task IAM role from the IAM console.

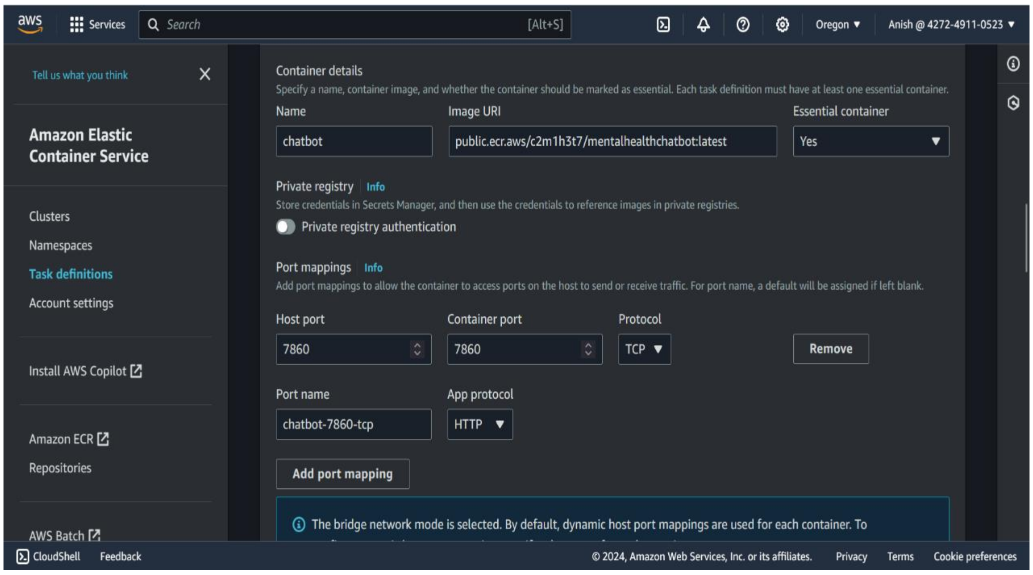
None

Task execution role Info

A task execution IAM role is used by the container agent to make AWS API requests on your behalf. If you don't already have a task execution IAM role created, we can create one for you.

ecsTaskExecutionRole

S5-CREATING TASK DEFINITION



Container details

Specify a name, container image, and whether the container should be marked as essential. Each task definition must have at least one essential container.

Name	Image URI	Essential container
chatbot	public.ecr.aws/c2m1h3t7/mentalhealthchatbot:latest	Yes

Private registry Info

Store credentials in Secrets Manager, and then use the credentials to reference images in private registries.

Private registry authentication

Port mappings Info

Add port mappings to allow the container to access ports on the host to send or receive traffic. For port name, a default will be assigned if left blank.

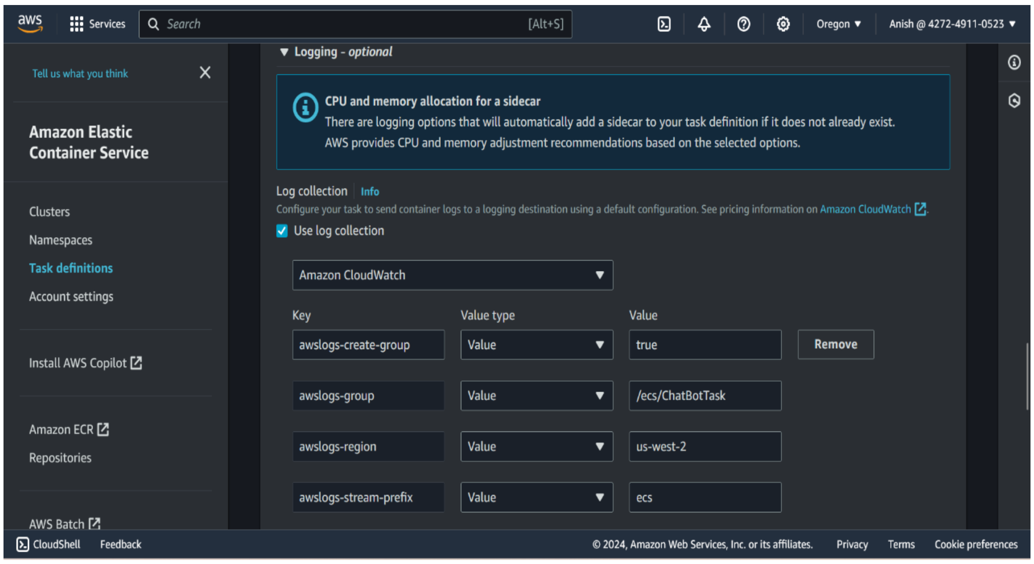
Host port	Container port	Protocol	
7860	7860	TCP	Remove

Port name	App protocol
chatbot-7860-tcp	HTTP

Add port mapping

The bridge network mode is selected. By default, dynamic host port mappings are used for each container. To

S6-CREATING TASK DEFINITION



▼ Logging - *optional*

CPU and memory allocation for a sidecar
There are logging options that will automatically add a sidecar to your task definition if it does not already exist.
AWS provides CPU and memory adjustment recommendations based on the selected options.

Log collection | Info
Configure your task to send container logs to a logging destination using a default configuration. See pricing information on Amazon CloudWatch.
☑ Use log collection

Amazon CloudWatch

Key	Value type	Value	
awslogs-create-group	Value	true	Remove
awslogs-group	Value	/ecs/ChatBotTask	
awslogs-region	Value	us-west-2	
awslogs-stream-prefix	Value	ecs	

S7-CREATING TASK DEFINITION

3.Running a Task on the EC2 container

a) Go to the ECS Cluster and scroll down.

b) Navigate to the Task menu and click create new task.

c) In Environment section choose Launch type and give EC2 instance.

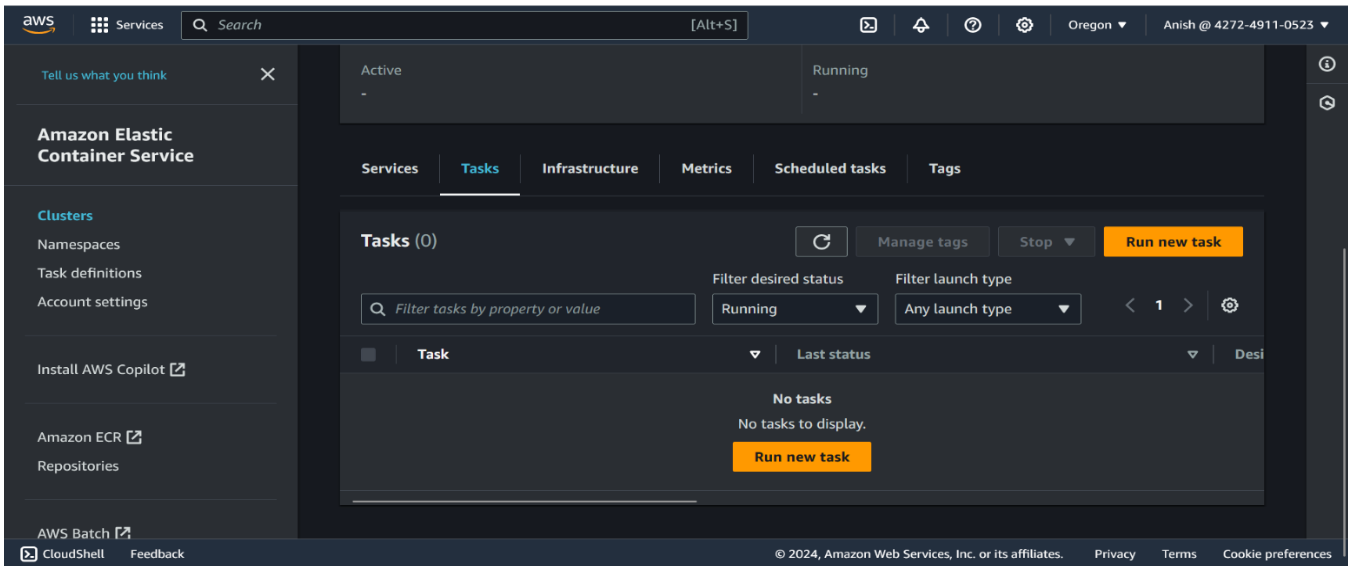
d) Scroll down to Environment Configuration and choose the Task Family name and the latest revision of the task.

e) Give desired Task as one so that it runs one task in the EC2 instance.

f) Scroll down and click create.

g) After creating go to the EC2 Section and copy the IP address of the Instance created by the autoscaling group.

h) Paste the copied address on the browser URL with the port number 7860.



Active: -

Running: -

Tasks (0)

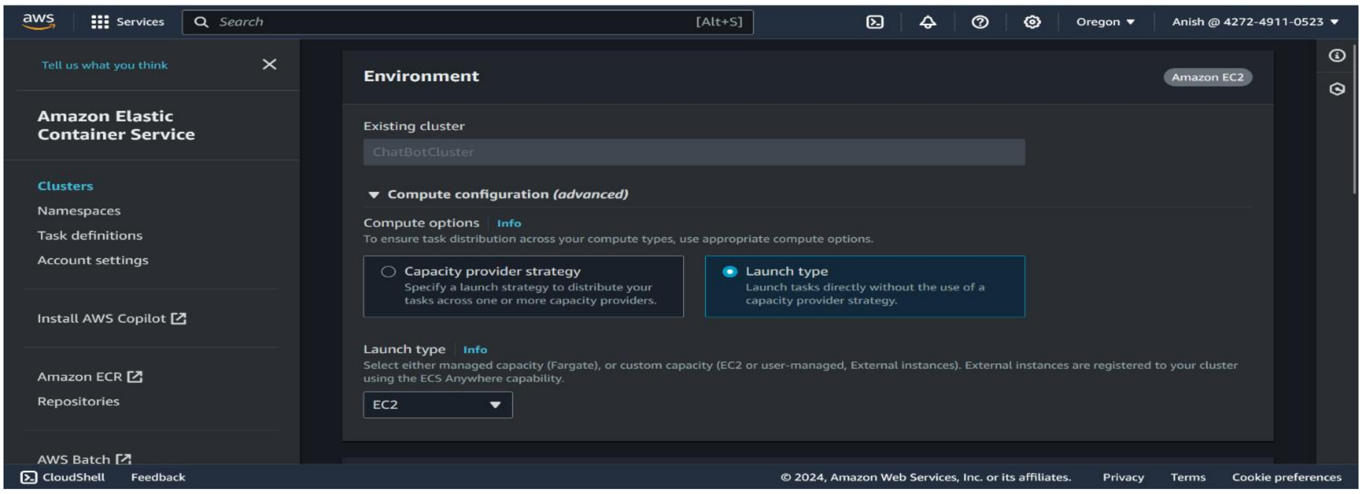
Filter desired status: Running

Filter launch type: Any launch type

No tasks

No tasks to display.

S8-RUNNING TASK



Environment

Existing cluster: ChatBotCluster

Compute configuration *(advanced)*

Compute options

To ensure task distribution across your compute types, use appropriate compute options.

Capacity provider strategy — Specify a launch strategy to distribute your tasks across one or more capacity providers.

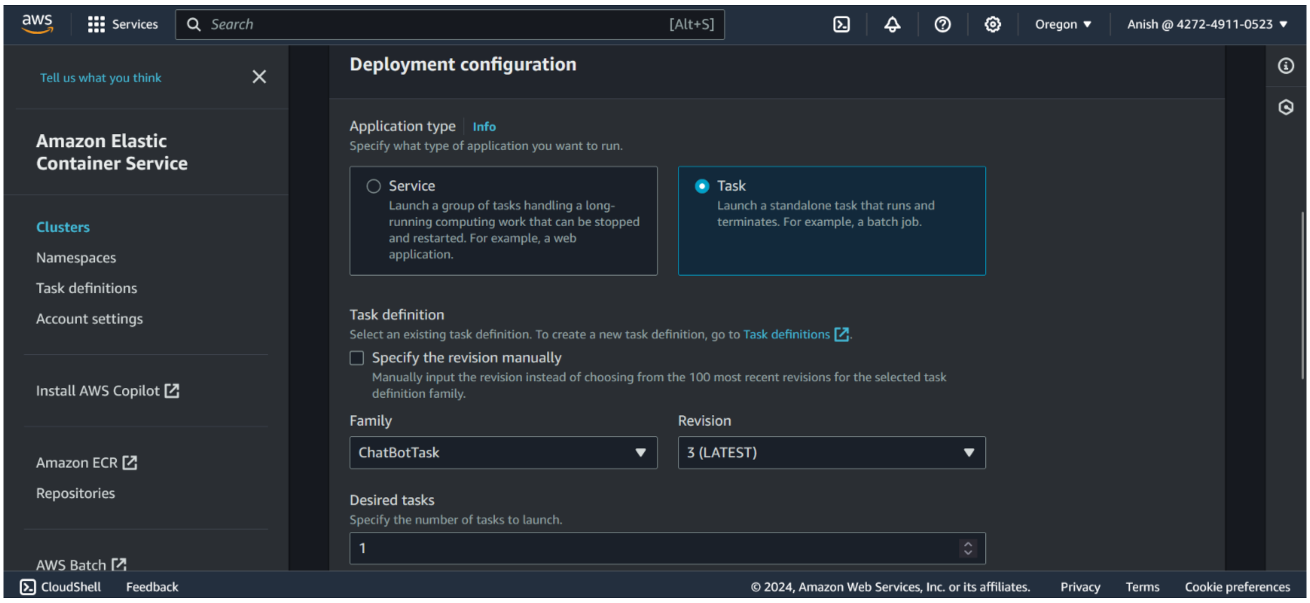
[✓ Launch type] — Launch tasks directly without the use of a capacity provider strategy.

Launch type

Select either managed capacity (Fargate), or custom capacity (EC2 or user-managed, External instances). External instances are registered to your cluster using the ECS Anywhere capability.

EC2

S9-RUNNING TASK



Deployment configuration

Application type Info
Specify what type of application you want to run.

Service
Launch a group of tasks handling a long-running computing work that can be stopped and restarted. For example, a web application.

Task
Launch a standalone task that runs and terminates. For example, a batch job.

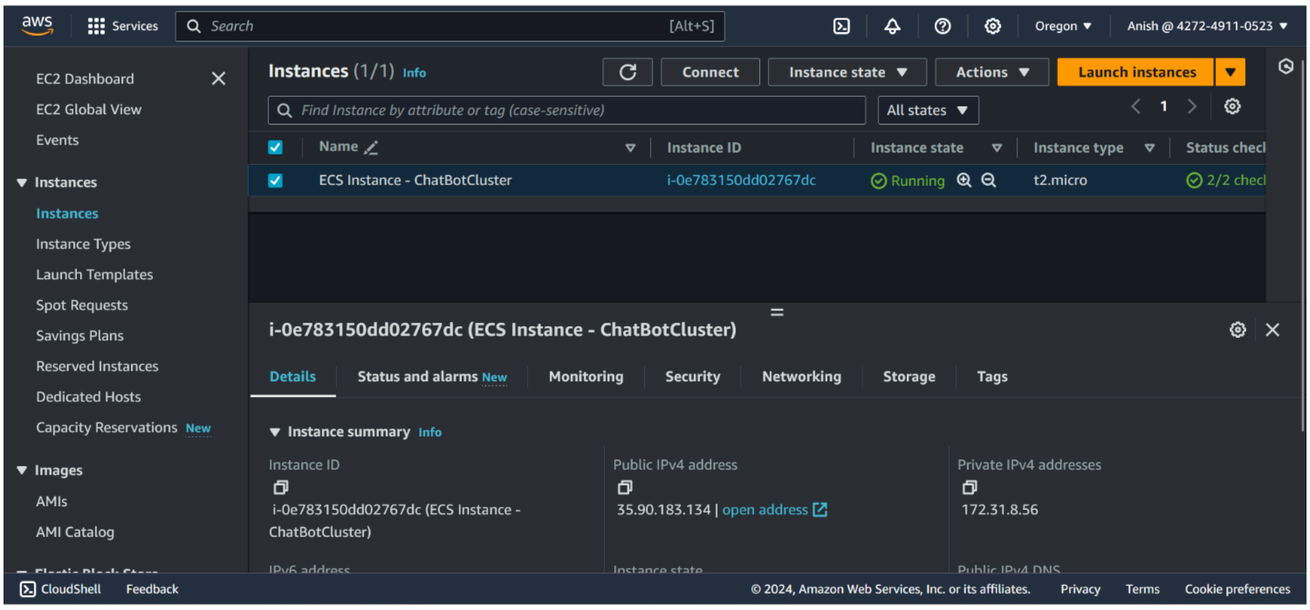
Task definition
Select an existing task definition. To create a new task definition, go to Task definitions.

Specify the revision manually
Manually input the revision instead of choosing from the 100 most recent revisions for the selected task definition family.

Family: ChatBotTask
Revision: 3 (LATEST)

Desired tasks
Specify the number of tasks to launch.
1

S10-RUNNING TASK



Instances (1/1) Info

Name	Instance ID	Instance state	Instance type	Status check
ECS Instance - ChatBotCluster	i-0e783150dd02767dc	Running	t2.micro	2/2 check

i-0e783150dd02767dc (ECS Instance - ChatBotCluster)

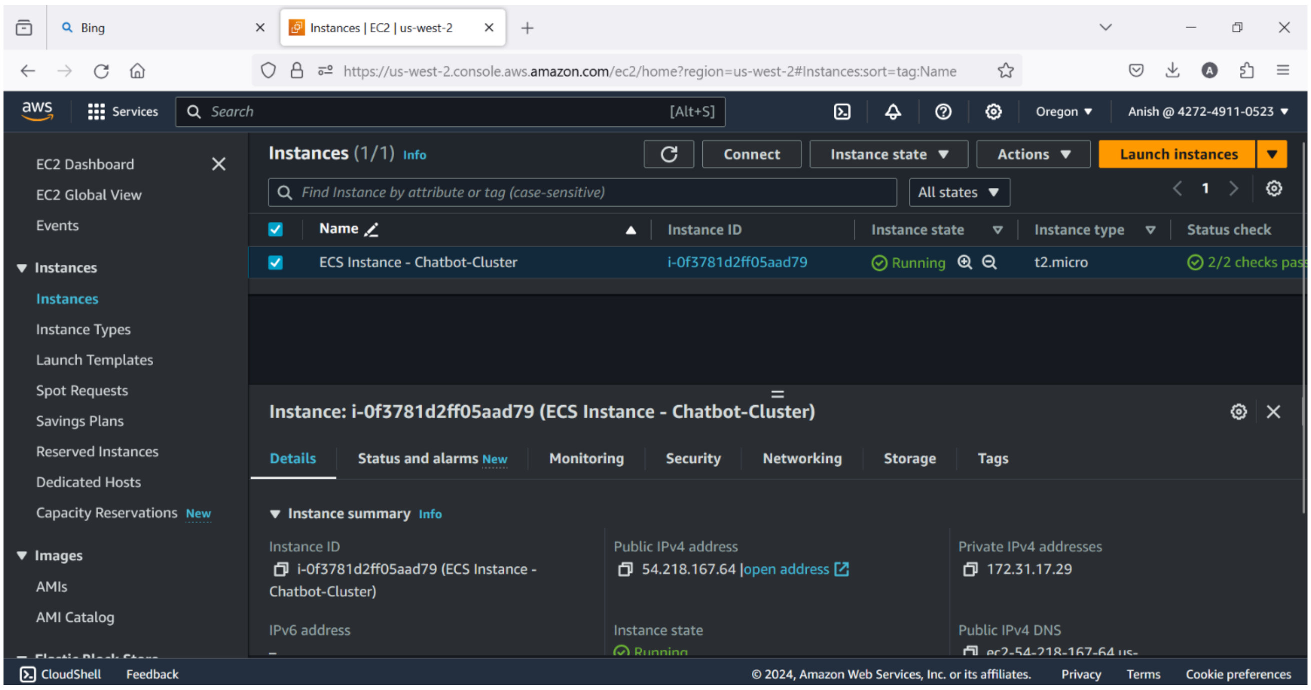
Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

Instance summary Info

Instance ID: i-0e783150dd02767dc (ECS Instance - ChatBotCluster)
Public IPv4 address: 35.90.183.134 | open address
Private IPv4 addresses: 172.31.8.56

S11-CREATING TASK DEFINITION

APPENDIX 2
SCREEN SHOTS



https://us-west-2.console.aws.amazon.com/ec2/home?region=us-west-2#Instances:sort=tag:Name

Instances (1/1) Info

Find Instance by attribute or tag (case-sensitive)

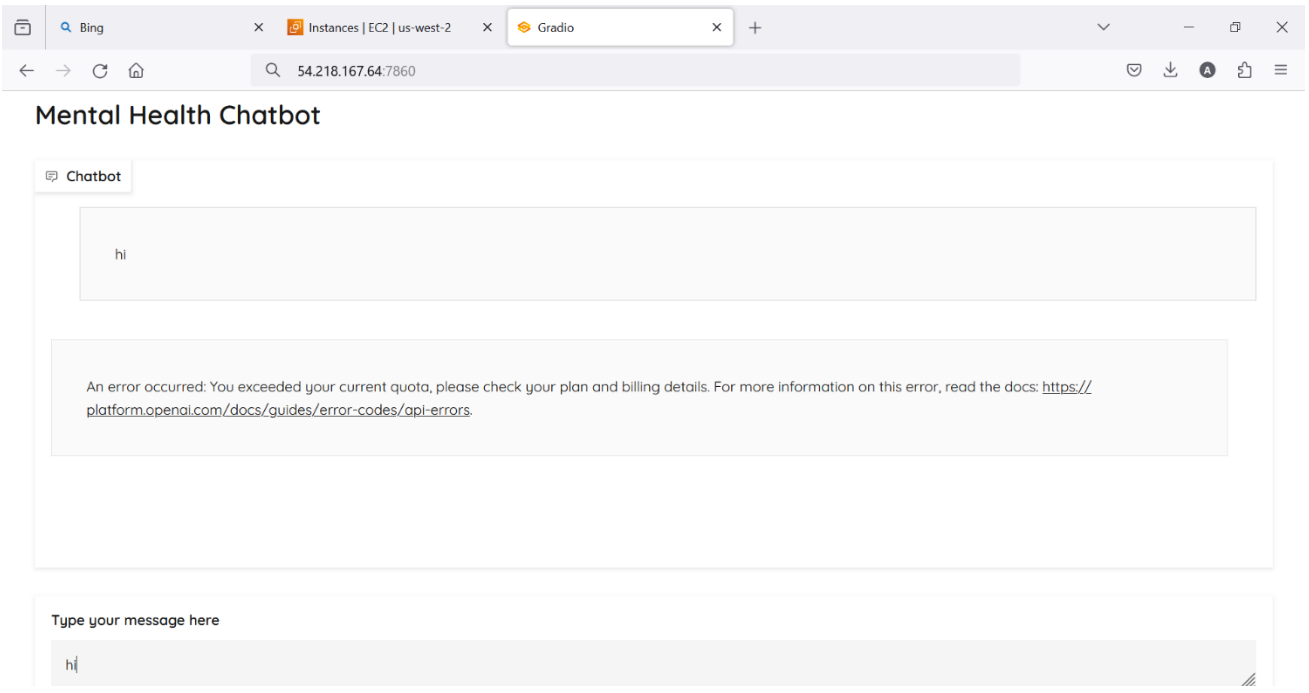
Name	Instance ID	Instance state	Instance type	Status check
ECS Instance - Chatbot-Cluster	i-0f3781d2ff05aad79	Running	t2.micro	2/2 checks pas

Instance: i-0f3781d2ff05aad79 (ECS Instance - Chatbot-Cluster)

Instance summary Info

Instance ID	Public IPv4 address	Private IPv4 addresses
i-0f3781d2ff05aad79 (ECS Instance - Chatbot-Cluster)	54.218.167.64 |open address	172.31.17.29
IPv6 address	**Instance state**	**Public IPv4 DNS**
–	Running	ec2-54-218-167-64.us-

SCREENSHOT 1



Mental Health Chatbot

hi

An error occurred: You exceeded your current quota, please check your plan and billing details. For more information on this error, read the docs: https://platform.openai.com/docs/guides/error-codes/api-errors.

Type your message here

hi

SCREENSHOT 2

REFERENCES

[1] Adamopoulou E, Moussiades L. An overview of chatbot technology. Proceedings of the 16th International Conference on Artificial Intelligence Applications and Innovations; AIAI '20; June 5-7, 2020; Neos Marmaras, Greece. 2020.

[2] Khan S, Rabbani MR. Artificial intelligence and NLP -based chatbot for Islamic banking and finance. Int J Inf Retr Res. 2021.

[3] Cui L, Huang S, Wei F, Tan C, Duan C, Zhou M. SuperAgent: a customer service chatbot for e-commerce websites. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics-System Demonstrations; ALC '17; July 30-August 4, 2017; Vancouver, Canada. 2017.

[4] Winkler R, Söllner M. Unleashing the potential of chatbots in education: a state-of-the-art analysis. Proceedings of the 78th Academy of Management Annual Meeting; AOM '18; August 10-14, 2018; Chicago, IL, USA. 2018.

[5] Bhirud N, Tataale S, Randive S, Nahar S. A literature review on chatbots in healthcare domain. Int J Sci Res. 2019 Jul;8(7):225–31.

[6] Lee YC, Yamashita N, Huang Y. Designing a chatbot as a mediator for promoting deep self-disclosure to a real mental health professional. Proc ACM Hum Comput Interact. 2020 May 29.

[7] Oh YJ, Zhang J, Fang M, Fukuoka Y. A systematic review of artificial intelligence chatbots for promoting physical activity, healthy diet, and weight loss. Int J Behav Nutr Phys Act. 2021 Dec 11.

[8] Lee M, Ackermans S, van As N, Chang H, Lucas E, IJsselsteijn W. Caring for Vincent: a chatbot for self-compassion. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems; CHI '19; May 4-9, 2019; Glasgow, Scotland, UK. 2019.

[9] Abd-Alrazaq AA, Alajlani M, Alalwan AA, Bewick BM, Gardner P, Househ M. An overview of the features of chatbots in mental health: a scoping review. Int J Med Inform. 2019 Dec.