

“LECTER” - A LARGE LANGUAGE MODEL CHATBOT FOR COGNITIVE BEHAVIORAL THERAPY

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering

By

**Sham Sundhar .R (Reg.No: 40111176)
Shivavardhini .T (Reg.No 40111203)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Category-1 University by UGC

**Accredited with Grade “A++” by NAAC | 12B Status By UGC | Approved by
AICTE**

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119**

APRIL - 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

CATEGORY-I UNIVERSITY BY UGC

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Sham Sundhar .R (40111176)** and **Shivavardhini .T (40111203)** who carried out the project entitled "**LECTER**" - A LARGE LANGUAGE MODEL CHATBOT FOR COGNITIVE BEHAVIORAL THERAPY" under my supervision from November 2023 to April 2024.

Internal Guide

Ms. C.A Daphine Desona Clemency, M.E

Head of the Department
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Jeppiaar Nagar, Rajiv Gandhi Salai,
Chennai - 600 119.

Submitted for Viva Voce Examination held on 12.04.2024

Internal Examiner

External Examiner

DECLARATION

I, **Sham Sundhar R (Reg. No. 40111176)** hereby declare that the Project Report entitled on “**“LECTER” - A LARGE LANGUAGE MODEL CHATBOT FOR COGNITIVE BEHAVIORAL THERAPY**” done by me under the guidance of **Ms. C.A Daphine Desona Clemency, M.E.**, at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

Date: *10.04.2024*

Place: *Chennai*



Signature of the Candidate

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management of Sathyabama Institute Of Science And Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to, **Dr. T. Sasikala M.E., Ph.D.**, Dean, School of Computing, **Dr. L. Lakshmanan, M.E., Ph.D.**, Head of the Department, Department of Computer Science and Engineering for providing me the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms.C.A Daphine Desona Clemency, M.E.**, for her valuable guidance, suggestions which paved the way for the successful completion of my project work.

I wish to express my thanks to all teaching and non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Mental health represents the state of mental well-being that enables people to cope with the stresses of life, realize their abilities, learn well and work well, and contribute to their community. Even though it is an integral quality to a human being, there aren't enough resources dedicated to helping people maintain it. Mental Illness such as Anxiety and Depression are common occurrences in modern society. Yet most patients go undiagnosed, cannot afford treatment or there simply aren't any practitioners readily available to them. In the worst case scenario avoiding treatment of mental illnesses can increase the likelihood of attempted suicides in affected individuals. A cheap and easily available solution is required to provide the benefits of therapy to the common masses. Lecter is a chatbot solution specifically targeted towards providing free mental health care to individuals who cannot access an actual therapist. This is also implemented while keeping in mind that one prefers psychotherapy over pharmacotherapy, specifically digitally accessible psychotherapy. The rapid advancement in technology has aided in producing a digital outlet to access mental health resources, and chatbots can be considered a good option in exacting the policies and principles of cognitive behavioral therapy. With the help of a LLM's superior conversational abilities, Lecter hears out the worries and troubles that the patients face and helps them introspect and understand their feelings better using a popular therapy technique called Cognitive Behavioral Therapy (CBT). Lecter provides its services using technologies such as: a large language model called Mythomax (Llama), Langchain for loading and interfacing with the Large Language Model, BERT base-cased Model for generating word embeddings, Tensorflow and Keras for building the Classification Model, React Js for the front end of the application, Flask for the backend and MongoDB for authentication and storage.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	x
	LIST OF TABLES	xi
1	INTRODUCTION	1
	1.1 Mental Health	1
	1.2 Chatbot	5
	1.3 Large Language Model	6
	1.4 Bidirectional Encoder Representation from Transformers	7
2	LITERATURE SURVEY	9
	2.1 Inferences from Literature Survey	9
	2.2 Open Problems in Existing Applications	11
3	REQUIREMENT ANALYSIS	13
	3.1 Feasibility Study	13
	3.2 Requirements Specifications	15
4	DESCRIPTION OF PROPOSED SYSTEM	16
	4.1 Description of Research Approach	16
	4.2 Architecture of Proposed System	17

5	IMPLEMENTATION	20
	5.1 React	20
	5.2 Flask	34
	5.3 BERT	37
	5.4 Langchain	38
6	RESULTS AND DISCUSSION	41
7	CONCLUSION	43
	REFERENCES	44
	APPENDIX	45
	A. SOURCE CODE	45
	B. SCREENSHOTS	65
	C. RESEARCH PAPER	67
	D. CERTIFICATE	78

LIST OF FIGURES

FIGURE No.	NAME	PAGE NO.
4.1	System Architecture	17
5.1	Context API	25
5.2	App Component	25
5.3	Login HTML	26
5.4	handleSignup function	27
5.5	handleSignIn function	28
5.6	Chatlog HTML	29
5.7	handleSubmit() function	29
5.8	generateResponse() function	30
5.9	handleResponse() callback	31
5.10	Chat Component	32
5.11	Message Component	33
5.12	Chat Component States	33
5.13	Context API States	34
5.14	Store Route	35
5.15	Eval Route	36
5.16	SignUp route	36

5.17	SignIn route	37
6.1	Confusion Matrix	42
B.1	Login Page	65
B.2	Chat Log 1	65
B.3	Chat Log 2	66
B.4	Chat Log 3	66

LIST OF ABBREVIATIONS

ABBREVIATIONS

EXPANSIONS

LLM	Large Language Model
NLP	Natural Language Processing
BERT	Bidirectional Encoder Representation from Transformers
CBT	Cognitive Behavioral Therapy
NSP	Next Sentence Prediction
ZSL	Zero Shot Learning
BPD	Bipolar Disorder
ADHD	Attention Deficit Hyperactivity Disorder
NLU	Natural Language Understanding
NLG	Natural Language Generation
PCT	Person Centered Therapy

LIST OF TABLES

TABLE No.	NAME	PAGE NO.
5.1	LLM comparison	41

CHAPTER 1

INTRODUCTION

1.1.MENTAL HEALTH

Mental health includes our emotional, psychological, and social well-being, which in turn affects how we think, feel, and act in situations. It is a general term for a group of illnesses that may include symptoms that can affect a person's thinking, perceptions, mood or behavior. Mental illness can make it difficult for someone to cope with work, relationships and other demands.

The relationship between stress and mental illness is complex, but it is known that stress can worsen an episode of mental illness. Most people can manage their mental illness with medication, counseling or both.

A few types of Mental Illness that are commonly observed are:

- i) Attention deficit hyperactivity disorder
- ii) Anxiety Disorder
- iii) Bipolar Disorder
- iv) Depression

1.1.1.ADHD

Attention deficit hyperactivity disorder (ADHD) is a neurodevelopmental disorder characterized by excessive amounts of inattention, hyperactivity, and impulsivity that are pervasive, impairing in multiple contexts, and otherwise age-inappropriate.

ADHD symptoms arise from executive dysfunction and emotional dysregulation is often considered a core symptom. In children, problems paying attention may result in poor school performance. ADHD is associated with other neurodevelopmental and mental disorders as well as some non-psychiatric disorders, which can cause additional impairment, especially in modern society. Although people with ADHD struggle to focus on tasks they are not particularly interested in completing, they are often able to

maintain an unusually prolonged and intense level of attention for tasks they do find interesting or rewarding; this is known as hyperfocus.

The precise causes of ADHD are unknown in the majority of cases. Genetic factors play an important role; ADHD tends to run in families and has a heritability rate of 74%. Toxins and infections during pregnancy as well as brain damage may be environmental risks.

1.1.2 Anxiety Disorder :

Occasional anxiety is a normal part of life. Many people worry about things such as health, money, or family problems. But anxiety disorders involve more than temporary worry or fear. For people with an anxiety disorder, the anxiety does not go away and can get worse over time. The symptoms can interfere with daily activities such as job performance, schoolwork, and relationships.

There are several types of anxiety disorders, including generalized anxiety disorder, panic disorder, social anxiety disorder, and various phobia-related disorders.

a. Generalized anxiety disorder

Generalized anxiety disorder (GAD) usually involves a persistent feeling of anxiety or dread, which can interfere with daily life. It is not the same as occasionally worrying about things or experiencing anxiety due to stressful life events. People living with GAD experience frequent anxiety for months, if not years.

b. Panic disorder

People with panic disorder have frequent and unexpected panic attacks. Panic attacks are sudden periods of intense fear, discomfort, or sense of losing control even when there is no clear danger or trigger. Not everyone who experiences a panic attack will develop panic disorder

People with panic disorder often worry about when the next attack will happen and actively try to prevent future attacks by avoiding places, situations, or behaviors they associate with panic attacks. Panic attacks can occur as frequently as several times a day or as rarely as a few times a year.

c. Social anxiety disorder

Social anxiety disorder is an intense, persistent fear of being watched and judged by others. For people with social anxiety disorder, the fear of social situations may feel so intense that it seems beyond their control. For some people, this fear may get in the way of going to work, attending school, or doing everyday things.

d. Phobia-related disorders

A *phobia* is an intense fear of—or aversion to—specific objects or situations. Although it can be realistic to be anxious in some circumstances, the fear people with phobias feel is out of proportion to the actual danger caused by the situation or object.

1.1.3. Bipolar Disorder :

Bipolar disorder (formerly called manic-depressive illness or manic depression) is a mental illness that causes unusual shifts in a person's mood, energy, activity levels, and concentration. These shifts can make it difficult to carry out day-to-day tasks.

There are three types of bipolar disorder. All three types involve clear changes in mood, energy, and activity levels.

- Bipolar I disorder is defined by manic episodes that last for at least 7 days (nearly every day for most of the day) or by manic symptoms that are so severe that the person needs immediate medical care.
- Bipolar II disorder is defined by a pattern of depressive episodes and hypomanic episodes. The hypomanic episodes are less severe than the manic episodes in bipolar I disorder.
- Cyclothymic disorder (also called cyclothymia) is defined by recurring hypomanic and depressive symptoms that are not intense enough or do not last long enough to qualify as hypomanic or depressive episodes.

1.1.4. Depression:

Depressive disorder (also known as depression) is a common mental disorder. It involves a depressed mood or loss of pleasure or interest in activities for long periods of time.

Depression is different from regular mood changes and feelings about everyday life. It can affect all aspects of life, including relationships with family, friends and community. It can result from or lead to problems at school and at work.

Depression can happen to anyone. People who have lived through abuse, severe losses or other stressful events are more likely to develop depression. Women are more likely to have depression than men.

An estimated 3.8% of the population experience depression, including 5% of adults (4% among men and 6% among women), and 5.7% of adults older than 60 years. Approximately 280 million people in the world have depression . Depression is about 50% more common among women than among men. Worldwide, more than 10% of pregnant women and women who have just given birth experience depression . More than 700 000 people die due to suicide every year. Suicide is the fourth leading cause of death in 15–29-year-olds.

1.1.5. Treatment for Mental Illness

In this post-pandemic world, One critical area of concern for public health is said to be mental health. Due to the COVID-19 pandemic, anxiety disorders grew from 298 million people affected, to 374 million. Depressive symptoms increased from a base of 193 million people, to 246 million people worldwide. Mental illnesses can be recovered from using either pharmacotherapy or psychotherapy, however, there is a global shortage of human resources that can deliver mental health services.

Even with proper means to receive help, the majority of the patients are apprehensive about seeking counseling, feeling intimidated to open up to health professionals due to societal pressure and norms. Mental Illness is still not acknowledged as a genuine concern in most third world and developing countries. An easily accessible solution is

required for both: the people who can afford treatment but choose not to due to societal concerns and for the people who simply cannot afford psychiatric help. Existing in this post-pandemic, it can also be observed that the necessity of citizen-centric digital services has become more evident due to socio-economic uncertainties that have resulted in a rising demand for digital solutions. All of these problems point to the solution of using chatbots.

1.2.CHATBOT

A chatbot is a computer program or an AI system that is designed to simulate human conversation and interact with users via text or voice-based interfaces. Chatbots are built to understand natural language inputs from users and provide appropriate responses based on their programming and training.

Driven by AI, automated rules, natural-language processing (NLP), and machine learning (ML), chatbots process data to deliver responses to requests of all kinds.

There are two main types of chatbots.

1.2.1 *Declarative Chatbots*

Task Oriented(Declarative) Chatbots are single-purpose programs that focus on performing one function. Using rules, NLP, and very little ML, they generate automated but conversational responses to user inquiries. Interactions with these chatbots are highly specific and structured and are most applicable to support and service functions—think robust, interactive FAQs. Task-oriented chatbots can handle common questions, such as queries about hours of business or simple transactions that don't involve a variety of variables. Though they do use NLP so end users can experience them in a conversational way, their capabilities are fairly basic. These are currently the most commonly used chatbots.

1.2.2 *Conversational chatbots*

Data-driven and predictive (conversational) chatbots are often referred to as virtual assistants or digital assistants, and they are much more sophisticated, interactive, and personalized than task-oriented chatbots. These chatbots are contextually aware and

leverage natural-language understanding (NLU), NLP, and ML to learn as they go. They apply predictive intelligence and analytics to enable personalization based on user profiles and past user behavior. Digital assistants can learn a user's preferences over time, provide recommendations, and even anticipate needs. In addition to monitoring data and intent, they can initiate conversations. Apple's Siri and Amazon's Alexa are examples of consumer-oriented, data-driven, predictive chatbots.

Advanced digital assistants are also able to connect several single-purpose chatbots under one umbrella, pull disparate information from each of them, and then combine this information to perform a task while still maintaining context—so the chatbot doesn't become “confused.”

Over the years, Chatbots have become a popular solution for usage in mental health treatment applications in recent years. They provide treatment techniques and practices for people facing problems like depression and anxiety. However the main problem with these chatbots is that they are trained using primitive NLP solutions that sound very robotic. Most of these chatbots also have very limited context memory thus they forget the flow of the conversation making for a very awkward chat experience. For an issue as delicate as Mental Health a more human and dynamic chatbot is required. This problem is solved by the usage of Large Language Models as the core of a chatbot.

1.3.LARGE LANGUAGE MODELS

A large language model is a type of language model notable for its ability to achieve general-purpose language understanding and generation. LLMs acquire these abilities by using massive amounts of data to learn billions of parameters during training and consuming large computational resources during their training and operation LLMs are artificial neural networks and are pre-trained using self-supervised learning and semi-supervised learning.

As language models, they work by taking an input text and repeatedly predicting the next token or word. Up to 2020, fine tuning was the only way a model could be adapted to be able to accomplish specific tasks. Larger sized models, such as GPT-3, however,

can be prompt-engineered to achieve similar results. They are thought to acquire embodied knowledge about syntax, semantics and "ontology" inherent in human language corpora, but also inaccuracies and biases present in the corpora.

Notable examples include OpenAI's GPT models, Google's PaLM and Meta's LLaMa.

Due to LLM's advanced comprehension skills and capability of communicating in a much more human and dynamic nature it is a much better fit for the task of interacting with users who suffer from sensitive issues like mental illnesses. We propose a mental health solution that utilizes the comprehension and communication skills of Large Language Models to power a chatbot.

1.4. BIDIRECTIONAL ENCODER REPRESENTATION FROM TRANSFORMERS

BERT is an open source machine learning framework for natural language processing (NLP). BERT is designed to help computers understand the meaning of ambiguous language in text by using surrounding text to establish context. The BERT framework was pre-trained using text from Wikipedia and can be fine-tuned with question and answer datasets. BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection.

Historically, language models could only read text input sequentially -- either left-to-right or right-to-left -- but couldn't do both at the same time. BERT is different because it is designed to read in both directions at once. This capability, enabled by the introduction of Transformers, is known as bidirectionality.

Using this bidirectional capability, BERT is pre-trained on two different, but related, NLP tasks: Masked Language Modeling and Next Sentence Prediction.

The objective of Masked Language Model (MLM) training is to hide a word in a sentence and then have the program predict what word has been hidden (masked)

based on the hidden word's context. The objective of Next Sentence Prediction training is to have the program predict whether two given sentences have a logical, sequential connection or whether their relationship is simply random

BERT is an "encoder-only" transformer architecture. On a high level, BERT consists of three modules:

1. Embedding: This module converts an array of one-hot encoded tokens into an array of vectors representing the tokens.
2. A stack of Encoder: These encoders are the Transformer encoders. They perform transformations over the array of representation vectors.
3. Un-Embedding: This module converts the final representation vectors into one-hot encoded tokens again.

The un-embedding module is necessary for pretraining, but it is often unnecessary for downstream tasks. Instead, one would take the representation vectors output at the end of the stack of encoders, and use those as a vector representation of the text input, and train a smaller model on top of that.

We will train a BERT multiclass classification model trained on Reddit Mental Health Dataset to classify the mental health status of the user using the chat logs of the user as input. This data is used to further configure the LLM to suit the needs of the patient and provide more effective treatment using Cognitive Behavioural Therapy.

CHAPTER 2

LITERATURE SURVEY

The rising popularity of Healthcare Automation using Machine Learning Technology has led to research into using chatbots for treatment of Mental Illnesses. These Chatbots are capable of hearing out the issues of its users, and responding accordingly. The workings of some of these chatbots can be inferred by the following, along with their advantages and disadvantages:

2.1 INFERENCES FROM LITERATURE SURVEY

Some written papers have implemented a solution with chatbots, especially for mental health purposes, using DialogFlow.

V. Dhanasekar et al.'s [5] paper introduced "MAXX", a Chatbot that was trained using Chatbot Training Platform DialogFlow. DialogFlow is a NLU platform, created by Google, that helps in building a chatbot, by performing effective NLU and NLG processes. It uses rule-based grammar matching algorithm and Machine Learning algorithms to understand the given intent and generate the right responses for the user. This paper uses Emotion Recognition Techniques to provide a chatbot solution to Students specifically facing Stress and Pressure from their academic Environment.

The use of DialogFlow and its NLU & NLG abilities, can also be observed in the work of S. Abu Noman Siddik et al.[1] "PSYCHE CONVERSA" is a Chatbot trained using Google's Chatbot Training Platform DialogFlow. It contains a Keylogger Module that collects data from the user's social media activity and a mental illness detection module to classify the mental health status of the user based on data collected by the Keylogger Module. This mental health status will be used by the chatbot to augment the flow of conversation, similar to what is observed in "MAXX"

P. Kugapriya et al.'s paper introduced "UNWIND" [6], A Chatbot mobile application that is trained using NLP solutions primarily for women, has an internal activity tracker and facial expression detection module to collect data on the user. The proposed "Internal activity tracking system" in this paper is a component that has the capability of detecting stress levels using mobile behaviors, and analyze correlation between stress level and lifestyle changes detected by monitoring mobile behaviors. The application also employs Disambiguation of word sense, a semantic study that determines which interpretation of a specific word best fits the situation. The Data collected through the chat and the other modules are sent to a practitioner that will carry out further treatment.

Lennart Brocki et al. [2] work introduces a Chatbot solution "SERENA", that employs Seq2Seq Transformer Model to generate responses. It uses Person Centered Therapy (PCT) unlike most modern chatbots that use Cognitive Behavioural Therapy to treat their patients. Person-Centered Therapy, also known as Client-Centered Therapy is a highly individualized therapeutic approach that focuses on creating a supportive and nurturing environment for clients to explore their feelings, thoughts, and experiences. It is based on the belief that every person has the capacity for self-actualization and personal growth, when provided with the environment to achieve so.

T.A.H Shaikh et al. [7] introduces an autonomous AI chatbot therapy for patients with insomnia. This chatbot was trained using a Feed Forward Neural Network with Chat Text Data that contains a text message and an appropriate tag. For example, the message "hi" would be tagged as "greeting". In this application, NLP principles such as stemming, tokenization, and bag of words are employed, where Bag of Words is responsible for turning the input string into a vector containing integers.

2.2 OPEN PROBLEMS IN EXISTING APPLICATIONS

The problems in the systems that were just reviewed in the previous sub-section can be observed as below, in order:

"MAXX" 's use of DialogFlow however prevents it from maintaining conversation context thus leading to a poor user experience. Its use of emotion recognition is by checking for positive and negative emotions, and nothing else. This is fairly primitive, and the system in place is not capable of handling and detecting more complicated emotions & problems, such as anxiety, ADHD, BPD, etc.

The same can be said for "PSYCHE CONVERSA", as the use of DialogFlow and similar systems means that the chatbot may lack true empathy and understanding, which are crucial in carrying out CBT. Chatbots can come across as robotic and impersonal, which might not be suitable for individuals seeking emotional support.

DialogFlow relies on the concept of "intents" to classify and understand user input. Intents are predefined patterns or templates that the system uses to identify the user's purpose or query. While this approach is effective for many types of conversations, it can lead to generalization of input messages in the context of mental health support.

P. Kugapriya et al.'s application "UNWIND" mentions Data Collection of personal data through monitoring mobile behaviors, heart rates, sleeping patterns etc., which could be perceived as a significant invasion of privacy. Users may feel uncomfortable knowing that a chatbot is constantly collecting, processing, and analyzing their private information.

In Lennart Brocki et al.'s chatbot "SERENA", the Seq2Seq Models generate more natural responses than most conventional NLP techniques, but it still lacks contextual understanding. It processes input sequences one token at a time and generates output sequentially, which leads to issues when handling long conversations or maintaining context across multiple turns. Seq2Seq models lack the capacity for genuine empathy

and emotional understanding, which are crucial components for PCT, which is what the chatbot's functioning is based on.

The Autonomous AI Chatbot Therapy Application for Insomnia introduces the use of feed forward learning for us by Insomniacs in a manner which is intended to reduce their fear and sadness. However, this system generates very robotic responses and isn't a very robust solution when it comes to mimicking a mental health professional.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

The described project of developing a large language model (LLM) chatbot with a BERT classification model, Flask server, and React frontend interface is feasible, but it comes with its own set of challenges and potential problems during development. Here's an analysis of the feasibility and potential challenges:

3.1.1 Feasibility:

a. Existing Technologies:

The project leverages well-established technologies such as Python, Flask, TensorFlow/PyTorch, and React, which have extensive documentation, community support, and resources available. This makes it feasible to implement the project with confidence.

b. Availability of Pre-trained Models:

The availability of pre-trained models like BERT and libraries such as Hugging Face Transformers simplifies the development process, allowing developers to focus more on fine-tuning and deploying the models rather than starting from scratch.

c. Scalability:

Flask and React are scalable technologies that can handle increased traffic and workload demands. With proper architecture design and implementation, the project can scale effectively to accommodate growing user bases.

d. Cloud Infrastructure:

Utilizing cloud platforms like AWS, Google Cloud, or Azure for hosting the Flask server and deploying the React frontend provides scalability, reliability, and ease of management.

3.1.2 Risk Analysis:

a. *Model Training and Tuning:*

Training and fine-tuning the BERT classification model can be computationally intensive and time-consuming, especially without access to powerful hardware resources such as GPUs. This could potentially slow down development iterations and require careful optimization strategies.

b. *Data Preprocessing and Cleaning:*

Preparing and cleaning the data required for training the classification model can be a challenging task, especially if the data is messy, inconsistent, or incomplete. Ensuring high-quality training data is essential for the accuracy and effectiveness of the model.

c. *Model Integration and Deployment:*

Integrating the trained BERT model with the Flask server and ensuring smooth deployment can be complex, especially when dealing with dependencies, version compatibility issues, and deployment configurations. Proper testing and debugging are crucial to ensure the stability and reliability of the deployed system.

d. *User Experience and Interface Design:*

Designing an intuitive and user-friendly interface for the React frontend requires careful consideration of user experience principles, accessibility requirements, and responsive design practices. Iterative design and testing may be necessary to refine the interface and meet user expectations.

e. *Security and Privacy Concerns:*

Handling sensitive user data and interactions in a chatbot application raises security and privacy concerns. Implementing robust security measures, data encryption, and access controls are essential to protect user privacy and prevent potential security breaches.

3.2 REQUIREMENTS SPECIFICATIONS

To develop a large language model (LLM) chatbot with a BERT classification model and a Flask server hosting the models, along with a React-based frontend interface, several software and hardware requirements must be considered.

Regarding software requirements, the project primarily relies on Python as the main programming language for implementing the backend server using Flask and training the BERT classification model. Additionally, TensorFlow or PyTorch serves as deep learning frameworks for training and deploying the BERT model. The Hugging Face Transformers Library, a Python library, is utilized for providing pre-trained models and tools for working with transformer-based architectures like BERT. For building the frontend interface of the chatbot, React, a JavaScript library, is employed.

In terms of hardware requirements, the project necessitates adequate server infrastructure to support the hosting of the Flask server. This includes a multi-core CPU capable of handling concurrent requests efficiently, sufficient RAM to accommodate the BERT model and serve multiple users concurrently, ample storage space for storing the BERT model, LLM, and any additional resources required by the Flask server, and a stable internet connection with sufficient bandwidth to handle incoming and outgoing requests. Additionally, a development machine with a decent multi-core CPU, sufficient RAM, and enough disk space for storing datasets, code, and development tools is required. While a GPU can significantly speed up the training of the BERT model, it is optional for development purposes. For testing purposes, it's important to test the React frontend on various devices and browsers to ensure compatibility and responsiveness, and to test the Flask server's performance under simulated heavy loads to identify any bottlenecks or performance issues. Finally, in the production environment, cloud platforms like AWS, Google Cloud, or Azure can be considered for hosting the Flask server and deploying the React frontend, with auto-scaling policies configured to handle fluctuating traffic demands efficiently.

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

4.1 DESCRIPTION OF RESEARCH METHODS

In the project aimed at developing an LLM (Large Language Model) Chatbot for Mental Health Care, various research methods are employed to ensure effectiveness and cultural sensitivity. Firstly, a thorough literature review is conducted to comprehend the current state of chatbots in mental health care, with a particular focus on LLM-based approaches. This step aids in the identification of challenges, best practices, and ethical considerations within the field.

Priority is then given to understanding the needs of the target users, especially those in third-world countries where mental health resources may be limited. Insights into their mental health challenges, preferred modes of communication, and cultural nuances are gathered through surveys, interviews, and focus groups.

Models available on platforms like HuggingFace are explored as another crucial step. Pre-trained language models are evaluated based on factors such as performance metrics, computational requirements, and relevance to diverse cultural contexts. This allows for the selection of models suitable for tasks like emotion detection and therapy dialogue generation.

Ethical considerations play a significant role in the research process. Guidelines and regulations related to mental health interventions and AI technologies are carefully reviewed to ensure adherence to ethical standards and safeguard the well-being of users, particularly vulnerable populations.

Prototypes of the LLM chatbot are developed, followed by usability testing and pilot studies to gather feedback on effectiveness, usability, and user satisfaction. This iterative process allows for the refinement of the chatbot's design and functionality based on user insights. Through these research methods, the aim is to develop an LLM chatbot for mental health care that is not only effective but also culturally sensitive, meeting the needs of users in third-world countries.

4.2 ARCHITECTURE OF PROPOSED SYSTEM

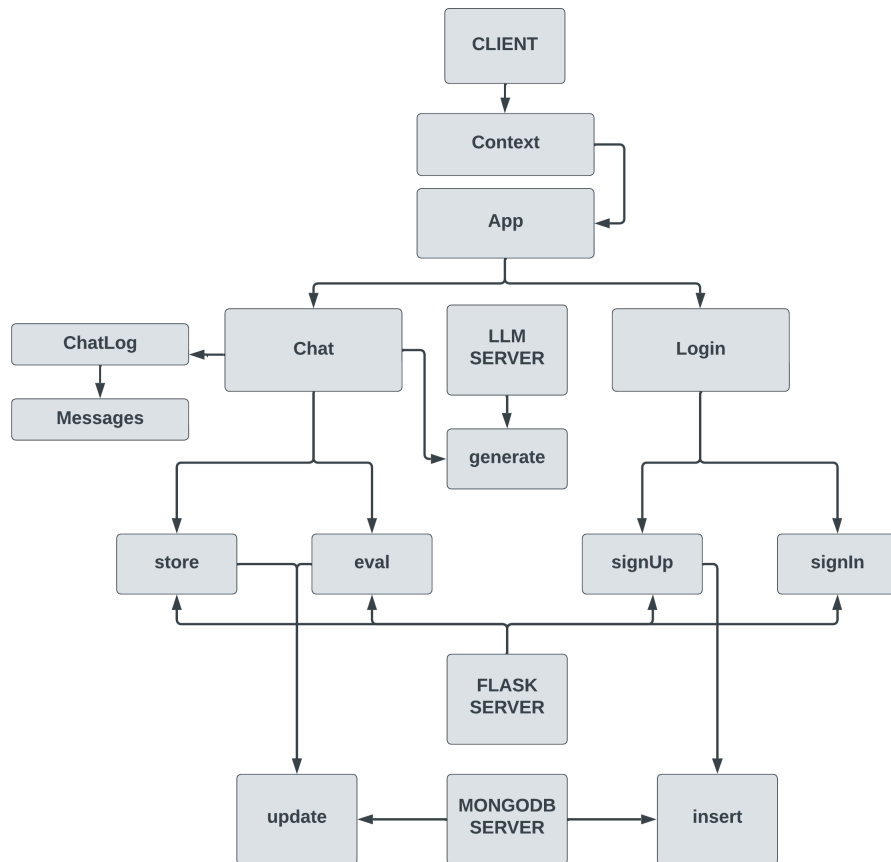


Figure 4.1. System Architecture

4.2.1 React Client

The React Client serves as the user interface for the mental health chatbot system, providing a seamless experience for users. Its architecture comprises two main pages: the Login page and the Chat Page. The Login page allows users to authenticate themselves before accessing the chat functionality, ensuring security and privacy. Upon successful authentication, users are directed to the Chat Page, where they can interact with the chatbot. This page enables users to send messages to the chatbot and receive responses in real-time, fostering a conversational experience. The React Client is designed with user experience and ease of navigation in mind, ensuring intuitive interaction with the chatbot system.

4.2.2 Flask Server

The Flask Server acts as the backend of the mental health chatbot system, hosting the BERT and LLM models and providing endpoints for communication with the React Client. These endpoints facilitate various functionalities, including message passing between the React Client and the LLM model, emotion classification using the BERT model, and storage of conversation summaries in the MongoDB server. Additionally, the Flask Server manages user authentication and authorization processes, ensuring secure access to the chatbot system. By centralizing the core functionalities within the Flask Server, the system maintains scalability, reliability, and maintainability.

4.2.3 MongoDB Server

The MongoDB server serves as the database for the mental health chatbot system, storing essential data such as chat summaries, login details, and mental health information. Chat summaries include messages exchanged between users and the LLM model, enabling the system to track conversation history and provide personalized responses. Login details are stored securely to manage user authentication and authorization processes, ensuring only authorized users can access the system. Additionally, the MongoDB server stores mental health details, which may include user profiles, assessments, or other relevant information, enabling personalized support and tracking of user progress over time.

4.2.4 BERT and LLM Models

The BERT and LLM models are hosted on the Flask Server and serve as the core components of the mental health chatbot system. The BERT model is utilized for tasks such as emotion classification of user messages, enabling the system to understand the emotional context of conversations. The LLM model generates responses to user messages based on the conversation context and its learned knowledge, providing personalized and contextually relevant support to users. These models are trained on large datasets to ensure robust performance and are continuously updated to improve accuracy and effectiveness.

In summary, the architecture of the mental health chatbot system encompasses multiple components working together seamlessly to provide users with a personalized and supportive experience. From the user interface of the React Client to the backend functionalities of the Flask Server and MongoDB server, each component plays a vital role in delivering effective mental health support.

CHAPTER 5

IMPLEMENTATION

5.1.REACT

React is a free and open-source front-end JavaScript library for building user interfaces based on components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. React can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js. Because React is only concerned with the user interface and rendering components to the DOM, React applications often rely on libraries for routing and other client-side functionality.

This project uses React and Node js Runtime Environment to facilitate the front end and the chat interface of Lecter.

5.1.1. useState Hook

useState is a hook in React that allows you to add state to functional components. Before the introduction of hooks in React, it was only possible to manage state in class components. However, with hooks, you can use state in functional components as well.

The useState hook returns an array with two elements - the current state and a function that allows you to update the state. You can use destructuring to assign names to these elements like so:

```
const [count, setCount] = useState(0);
```

In this example, count is the current state and setCount is the function that allows you to update it. The useState hook takes an initial value as an argument, which is set to 0 in this case.

You can use the setCount function to update the count state like so:

```
setCount(count + 1);
```

This will increment the count state by 1.

You can use `useState` to manage more complex states as well, such as objects or arrays. For example:

```
const [user, setUser] = useState({ name: "", email: "" });
```

```
setUser({ name: 'John', email: 'john@example.com' });
```

In this example, `user` is an object with `name` and `email` properties, and `setUser` is the function that allows you to update the user state. You can pass a new object to `setUser` to update the user state. Overall, `useState` is a powerful hook that allows you to add state to your functional components in React.

5.1.2. Prop Drilling

In React, prop drilling refers to passing data down through the component tree using props. This can happen when a parent component needs to pass data to its child components, and those child components need to pass that same data down to their own child components. This can result in the props being passed through multiple levels of components, even if some of those components don't actually need the data.

Prop drilling can make it difficult to maintain and understand the code, especially as the application grows larger and more complex. It can also lead to performance issues, as each component needs to receive and pass on the props, even if it doesn't use them.

To avoid prop drilling, you can use React Context, which allows you to pass data down the component tree without having to explicitly pass it through each component. Another option is to use state management libraries like Redux, which can help manage and centralize the application's state. By using these approaches, you can avoid the unnecessary passing of props and keep your code more maintainable and performant.

5.1.3. Context API

The Context API is a feature in React that provides a way to share data between components without having to pass the data down through each level of the component tree as props. It allows you to create a global state that can be accessed and modified by any component within the same Context.

Context API consists of two parts:

1. `createContext()` - This function creates a new context with an optional default value. This context can then be passed to components that need access to the data.
2. `Provider` - This component is used to provide the data to the components that consume it. It takes a `value` prop, which is the data that will be shared with the components that are wrapped inside it.

To use the Context API, you first need to create a new context using the `createContext()` function. For example:

```
import { createContext } from 'react';
```

```
const MyContext = createContext();
```

This creates a new context called `MyContext`.

Next, you need to wrap the component tree that needs access to the context data with a `Provider` component. For example:

```
<MyContext.Provider value={myData}>  
  <MyComponent />  
</MyContext.Provider>
```

Here, `myData` is the data that you want to share with `MyComponent`. Any component that is a child of `MyContext.Provider` can access this data using the `useContext` hook.

To access the data in a component, you first need to import the useContext hook:

```
import { useContext } from 'react';
```

Then, you can use it to access the context data:

```
const myData = useContext(MyContext);
```

This will give you access to the data that was provided by the Provider component.

The Context API is useful when you have data that needs to be shared across multiple components, or when you have a complex component tree with many levels. It allows you to avoid prop drilling and makes your code more organized and maintainable.

5.1.4. *useEffect Hook*

useEffect is a hook in React that allows you to perform side effects in your functional components. A side effect is anything that affects something outside the scope of the current function, such as fetching data from an API, changing the document title, or setting up event listeners.

The useEffect hook takes two arguments: a function that performs the side effect, and an optional array of dependencies. The function will be called after the component has rendered for the first time, and then again whenever any of the dependencies have changed.

Here is an example:

```
import { useEffect, useState } from 'react';
```

```
function MyComponent() {  
  const [count, setCount] = useState(0);  
  
  useEffect(() => {  
    document.title = `Count: ${count}`;  
  }, [count]);  
}
```

```

return (
  <div>
    <p>Count: {count}</p>
    <button onClick={() => setCount(count + 1)}>Increment</button>
  </div>
);
}

```

In this example, the `useEffect` hook is used to update the document title whenever the count state changes. The function passed to `useEffect` sets the document title to "Count: [count]". The second argument to `useEffect` is an array containing the count variable, which tells React to call the effect whenever the count variable changes. `useEffect` can be used for many other purposes as well, such as fetching data from an API, setting up event listeners, and cleaning up after side effects. The function returned from `useEffect` can be used for cleaning up any resources created by the side effect.

Overall, `useEffect` is a powerful tool that allows you to perform side effects in your functional components in a declarative and organized way.

5.1.5. Components:

Context:

Context Component is the Context API. It sets up a Context and Context Provider that can be used to transfer data between Components in the same context. This prevents prop drilling and also facilitates easy communication between components.

```

import React, { useState, useContext } from 'react'

const AppContext = React.createContext()

// eslint-disable-next-line react/prop-types
const AppProvider = ({ children }) => {
  const [mailID, setMailID] = useState("");
  const koboldAPI = "https://bingo-bound-seeks-preserve.trycloudflare.com"
  const serverAPI = "http://1c3e-34-30-104-64.ngrok-free.app"
  return <AppContext.Provider value={{ serverAPI, koboldAPI, mailID, setMailID }}>
    {children}</AppContext.Provider>
  }
// eslint-disable-next-line react-refresh/only-export-components
export const useGlobalContext = () => {
  return useContext(AppContext)
}

export { AppContext, AppProvider }

```

figure 5.1 Context API

App:

This component is responsible for setting up routes to Login and Chat Components. It uses Browser Router Features like router and routes to set up this configuration.

```

import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import Chat from "../Chat"
import Login from "../Login";

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Login />} />
        <Route path="/chat" element={<Chat />} />
      </Routes>
    </Router>
  );
}

export default App;

```

figure 5.2 App Component

Login:

This Component relates to the Login Page of the Lecter Website. New Users can sign up for an account and existing users can login to their accounts here. Both send respective requests to the Flask Server.

1. HTML:

The HTML content contains two forms. One form is meant for the purpose of gathering mail and password data for signing up for first time users or users who wish to create a new account. The other form is meant for logging back into an existing account. The signUp form triggers the `handleSignUp()` function on Submission of form. The SignIn form triggers the `handleSignIn()` function on Submission of form.

```
return (  
  <>  
    <form onSubmit={handleSignUp}>  
      <input id="mail" placeholder='mail ID'  
        type="text" className="inputArea"/>  
      <input id="pass" placeholder='password'  
        type="text" className="inputArea"/>  
      <input type="submit" value="Submit"/></input>  
    </form>  
    <form onSubmit={handleSignIn}>  
      <input id="mail" placeholder='mail ID'  
        type="text" className="inputArea"/>  
      <input id="pass" placeholder='password'  
        type="text" className="inputArea"/>  
      <input type="submit" value="Submit"/></input>  
    </form>  
  </>  
);
```

figure 5.3 Login HTML

2. `handleSignUp()`:

This function is called on Submit in the Sign Up form. It accesses the MailID and Password value from the form contents and sends a request to the Flask Server to the `signUp` endpoint. This will register a new user in the `mongodb` database. If all goes well then the `mailID` state is set for future use in the Chat component.

The Chat component is loaded using the navigate hook.

```
function handleSignUp(event) {
  event.preventDefault()

  const mailval=event.currentTarget.elements.mail.value
  const passval=event.currentTarget.elements.pass.value

  let data={
    mail:mailval,
    password:passval,
  }

  fetch(`${serverAPI}/signUp`,{
    method: 'POST',
    headers:{
      'Content-Type': 'application/json',
      'ngrok-skip-browser-warning': '69420'},
    body:JSON.stringify(data),
  }).then(response => response.json()).then(jsondata => {
    if(jsondata.result===1){
      setMailID(mailval);
      navigate("/chat");
    }
    else
    {
      console.log("ERROR SIGNING UP:"+jsondata.result)
    }
  })
}
```

figure 5.4 handleSignUp function

3. handleSignIn()

This function is called on Submit in the Sign In form. It accesses the MailID and Password value from the form contents and sends a request to the Flask Server to the signIn endpoint. If a User with the mailID exists and the credentials are correct If all then the mailID state is set for future use in the Chat component. The Chat component is loaded using the navigate hook.

```

function handleSignIn(event) {
  event.preventDefault()
  const mailval=event.currentTarget.elements.mail.value
  const passval=event.currentTarget.elements.pass.value
  let data={
    mail:mailval,
    password:passval,
  }
  fetch(`${serverAPI}/signIn`,{
    method: 'POST',
    headers:{
      'Content-Type': 'application/json',
      'ngrok-skip-browser-warning': '69420'},
    body:JSON.stringify(data),
  }).then(response => response.json()).then(jsondata =>
    {
      if(jsondata.result===1){
        setMailID(mailval);
        navigate("/chat");
      }
      else
      {
        console.log("ERROR LOGING IN:"+jsondata.result)
      }
    })
}

```

figure 5.5 handleSignIn function

Chat:

This component relates to the Chat Page of the Lecter Website. The Chat component facilitates User Interactions with Lecter through text. Users can send Text messages to Lecter using the input box. The input text is first sent to the Flask Server for health evaluation after which the input text is added to the Query String that would be sent to the LLM Server which would then return the generated response. Chat serves as a parent component for the ChatLog Component.

1. HTML:

The ChatLog component is nested into the Chat Component. There also exists a form that is used to take chat input from the user.

```

return (
  <>
    <div className='page'>
      <ChatLog messages={messages}/>
      <form onSubmit={handleSubmit}>
        <div className='input'>
          <input id="UserInput" placeholder=' Enter Message'
            type="text" className="inputArea" disabled={userChatDisabled}/>
        </div>
      </form>
    </div>
  </>
)

```

figure 5.6 ChatLog HTML

2. handleSubmit():

A post request is sent to the Flask Server store endpoint to evaluate the input text and store the results in the database. If the message count of the user is above a certain threshold then a post request is sent to the Flask server eval endpoint to evaluate the mental health of the user. The response from the server is used to update the query string being sent to the LLM server which improves the response generation.

```

function handleSubmit(event) {
  event.preventDefault()
  //evaluate and store data

  fetch(`${serverAPI}/store`,{
    method: 'POST',
    headers:{
      'Content-Type': 'application/json',
      'ngrok-skip-browser-warning': '69420'},
    body:JSON.stringify({mailid:mailID,input:event.currentTarget.elements.UserInput.value}),
  }).then(response => response.json()).then(jsondata => console.log(jsondata))

  if(msgCounter+1==evalThreshold)
  {
    fetch(`${serverAPI}/eval`,{
      method:'GET',
      headers:{
        'Content-Type': 'application/json',
        "ngrok-skip-browser-warning": "69420"
      },
      body:JSON.stringify({mailid:mailID})
    }).then(response => response.json()).then(jsondata => {
      let res=jsondata.result
      promptString=Prompts[res]
    })
  }
  setMsgCounter((msgCounter+1)%evalThreshold);
  setInputText(event.currentTarget.elements.UserInput.value)
  event.currentTarget.reset();
}

```

figure 5.7 handleSubmit() function

3. generateResponse():

The input parameter is added to the chatStack that will be used as a part of the query String sent to the LLM. The data is prepared for sending request to the LLM Server. userChatDisabled is set to true so the user can no longer send messages until the LLM returns a favorable response. A post request is sent to the LLM server using the data as the body of the request. handleResponse is registered as a callback in the promise.

```
function generateResponse(input){
  console.log(messages)
  chatstack+=input
  const data={
    "prompt": promptString+chatstack,
    "use_story": false,
    "use_memory": false,
    "use_authors_note": false,
    "use_world_info": false,
    "max_context_length": 2048,
    "max_length": 50,
    "rep_pen": 1.17,
    "rep_pen_range": 1024,
    "rep_pen_slope": 0.2,
    "temperature": 0.43,
    "tfs": 0.68,
    "top_a": 0,
    "top_k": 0,
    "top_p": 0.96,
    "typical": 1,
    "sampler_order": [ 6, 0, 1, 2, 3, 4, 5 ]
  }
  setUserChatDisabled(true);
  fetch(`${koboldAPI}/api/v1/generate` ,
  {
    method: 'POST',
    headers:{
      'accept': 'application/json' ,
      'Content-Type': 'application/json'
    },
    body:JSON.stringify(data)
  }).then(response => response.json())
  .then(data => handleResponse(data.results[0].text))
  .catch(error => console.error(error));
}
```

figure 5.8 generateResponse() function

4. handleResponse():

The response from the LLM Server 'generate' endpoint is handled by this callback. The LLM tends to generate more content than it was asked for. So a trim flag is set in place so that if a string that wasn't meant to be a part of the response is seen then the response is trimmed to fit the standard. Sometimes the LLM might generate an incomplete response too. In that case the incomplete response is added back to the query string and sent back to the LLM for more generation.

This process continues until the LLM generates content that triggers the trim flag.

```
function handleResponse(response){
  console.log(response)
  let trimindex=Infinity;
  trimStrings.forEach((s)=>
  {
    let t=response.indexOf(s)
    if(t!=-1 && t<trimindex) trimindex=t;
  });
  if(trimindex!=Infinity)
  {
    response=response.slice(0,trimindex)
    chatstack+=response
    responseText+=response;
    setUserChatDisabled(false);
    setMessages([...messages,{id:0,text:responseText}])
    responseText=""
  }
  else
  {
    responseText+=response
    // setAIChatText(responseText)
    generateResponse(response)
  }
  console.log(chatstack);
}
```

figure 5.9 handleResponse() callback

ChatLog:

This component is a child of the Chat Component. It contains all the messages that are exchanged between the user and Lecter. It serves as a parent for the Message component. The message elements from the prop Messages are distributed among the Message components. The message id and message text is passed to the Message component.

```
import Message from "./Message"
/* eslint-disable react/prop-types */
function ChatLog({messages})
{
  return(
    <>
      <div className="chatLog">
        {
          messages?.map((message,i) => {
            return <Message key ={i} id={message.id}
              text={message.text} />
          })
        }
      </div>
    </>
  )
}

export default ChatLog
```

figure 5.10 Chat Component

Message:

This component is a child of the ChatLog Component. It represents an individual message in a ChatLog. It uses the message id to differentiate between a message sent by a user and a message sent by Lecter. If the message id is 0 then the message is sent by Lecter and appropriate css is applied to the child elements. If the message id is 1 then the message is sent by the User and appropriate css is applied to the child elements.

```

/* eslint-disable react/prop-types */
function Message({id,text})
{
  return(
    <>
    <div className={"msg_"+(id==0?"left":"right")+" msg"}>
      {
        id==1?
        <div className="msg-id msg-id_right">
          <b>You</b>
          
        </div>
        :
        <div className="msg-id">
          
          <b>Bot</b>
        </div>
      }
      <div className="msg-bubble">
        {text}
      </div>
    </div>
    </>
  )
}

export default Message

```

figure 5.11 Message component

5.1.6.States:

In Chat Component:

```

const [msgCounter,setmsgCounter]=useState(0)
const [userChatDisabled,setUserChatDisabled]=useState(false)
const [messages,setMessages]=useState([])
const [inputText,setInputText]=useState(firstmessage)

```

figure 5.12 Chat Component States

1. msgCounter: Counts the number of messages that have been sent by the user
2. userChatDisabled: Disables the Input box until a response is sent back by the LLM server
3. messages: contains all the messages that have been sent by the user and Lecter
4. inputText: contains the latest message sent by the user through the input box.

In Context API:

```
const [mailID, setMailID] = useState("");
```

figure 5.13 Context API States

1. *mailID*:

contains the mailID of the User that logged in. Accessed through the context API in Login and Chat Component. The Login Component sets the mailID and the Chat Component uses the mailID in requests to the Flask Server to update the database.

5.2.FLASK

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.^[2] It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

We use Flask to make the backend server to host the BERT multiclass classification model and also to communicate with the MongoDB Server. The Flask Server is also responsible for updating the prompt of the LLM based on the mental health status of the user.

5.2.1. Routes

There are 4 routes in our Flask Server:

1. *store*:

This endpoint is used after each user sends a message. The message is evaluated using the BERT multiclass classification model and the mental health status is classified. The result is used to update the Health Collection in MongoDB server using the mailID of the user as filter.

```
@app.route('/store',methods= ['POST'])
def storeMentalHealth():
    input_text=request.json['input']
    print(input_text)
    processed_data = prepare_data(input_text, tokenizer)
    result = make_prediction(sentiment_model, processed_data=processed_data)
    mailID=request.json['mailid']
    filter = { 'mail': f'{mailID}' }
    # Values to be updated.
    newvalues = { "$inc": { f'{result}': 1 } }
    print(filter)
    print(newvalues)
    health.update_one(filter, newvalues)

    return jsonify({"text":"Successfully Stored"})
```

figure 5.14 store route

2. *eval*:

This endpoint is used when user message count crosses a threshold. The mental health status is revised and the result is then sent back to the client which will then use the result to update the query string for better LLM performance in response generation.

```

@app.route('/eval',methods = ['POST'])
def evaluateMentalHealth():
    mailID=request.json['mailid']
    #get stats from db
    healthResult=health.find_one({'mail':f'{mailID}'})
    #find max
    print(healthResult)
    result = max(zip(healthResult.values(), healthResult.keys()))[1]
    print(result)
    #return result
    return jsonify({"result":result})

```

figure 5.15 eval route

3. *signUp*:

This endpoint is called when a new user tries to sign up for Lecter. A new entry is made in mongoDB to maintain the mental health status info of the patient.

```

@app.route('/signUp',methods = ['POST'])
def signUp():
    mailID=request.json['mail']
    password=request.json['password']
    res=loginCol.find_one({'mail':mailID})
    if loginCol.count_documents(res)!=0:
        return jsonify({"result":"Mail already exists"})
    else :
        loginCol.insert_one({'mail':mailID,'password':password})
        healthCol.insert_one({'mail':mailID,'Anxiety':0,'Depressed':0,'Bipolar':0,'ADHD':0})
    return jsonify({"result":1})

```

figure 5.16 signUp route

4. *signIn*:

This endpoint is called when an existing user tries to sign in to Lecter. If the credentials are valid then a result of 1 is returned else an error message is returned.

```

@app.route('/signIn',methods = ['POST'])
def signIn():
    mailID=request.json['mail']
    password=request.json['password']
    res=loginCol.find_one({'mail':mailID,'password':password})
    if loginCol.count_documents(res)==0:
        return jsonify({"result":"Check Mail or Password"})
    else :
        return jsonify({"result":1})

```

figure 5.17 signIn route

5.3.BERT

BERT is an ML solution for natural language processing (NLP). BERT is designed to help computers understand the meaning of ambiguous language in text by using surrounding text to establish context. The BERT framework was pre-trained using text from Wikipedia and can be fine-tuned with question and answer datasets.

BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. (In NLP, this process is called *attention*.)

Historically, language models could only read text input sequentially -- either left-to-right or right-to-left -- but couldn't do both at the same time. BERT is different because it is designed to read in both directions at once. This capability, enabled by the introduction of Transformers, is known as bidirectionality.

Using this bidirectional capability, BERT is pre-trained on two different, but related, NLP tasks: Masked Language Modeling and Next Sentence Prediction.

The objective of Masked Language Model (MLM) training is to hide a word in a

sentence and then have the program predict what word has been hidden (masked) based on the hidden word's context. The objective of Next Sentence Prediction training is to have the program predict whether two given sentences have a logical, sequential connection or whether their relationship is simply random. We use BERT in this project for the purpose of Classifying the mental health status of the patient. The patient can fall under one of 4 categories: ADHD, Anxiety, Depression, Bipolar. Using this classification result the LLM will be configured to better suit the needs of the user.

5.4 LANGCHAIN

LangChain is an open source orchestration framework for the development of applications using large language models (LLMs). Available in both Python- and Javascript-based libraries, LangChain's tools and APIs simplify the process of building LLM-driven applications like chatbots and virtual agents. LangChain serves as a generic interface for nearly any LLM, providing a centralized development environment to build LLM applications and integrate them with external data sources and software workflows. LangChain's module-based approach allows developers and data scientists to dynamically compare different prompts and even different foundation models with minimal need to rewrite code. This modular environment also allows for programs that use multiple LLMs: for example, an application that uses one LLM to interpret user queries and another LLM to author a response.

Launched by Harrison Chase in October 2022, LangChain enjoyed a meteoric rise to prominence: as of June 2023, it was the single fastest-growing open source project on Github. Coinciding with the momentous launch of OpenAI's ChatGPT the following month, LangChain has played a significant role in making generative AI more accessible to enthusiasts in the wake of its widespread popularity. LangChain can facilitate most use cases for LLMs and natural language processing (NLP), like chatbots, intelligent search, question-answering, summarization services or even virtual agents capable of robotic process automation.

We have opted to use Langchain for this project as it provides us the ability to host our own LLMs and provides various functionalities and abstractions for interacting with our deployed LLM such as Memory Management Features, Prompt Templating, Data

Retrieval from Online/ Database and much much more.

5.4.1 Importing language models

Nearly any LLM can be used in LangChain. Importing language models into LangChain is easy, provided you have an API key. The LLM class is designed to provide a standard interface for all models. Most LLM providers will require you to create an account in order to receive an API key. Some of these APIs—particularly those for proprietary closed-source models, like those offered by OpenAI or Anthropic—may have associated costs.

Many open source models, like BigScience’s BLOOM, Meta AI’s LLaMa and Google’s Flan-T5, can be accessed through Hugging Face (link resides outside ibm.com). IBM watsonx, through its partnership with Hugging Face, also offers a curated suite of open source models. Creating an account with either service will allow you to generate an API key for any of the models offered by that provider.

LangChain is not limited to out-of-the-box foundation models: the CustomLLM class (link resides outside ibm.com) allows for custom LLM wrappers. Likewise, you can use the IBM watsonx APIs and Python SDK, which includes a LangChain integration, to build applications in LangChain with models that you’ve already trained or fine-tuned for your specific needs using the WatsonxLLM class (and that model’s specific project ID).

5.4.2. Prompt templates

Prompts are the instructions given to an LLM. The “art” of composing prompts that effectively provide the context necessary for the LLM to interpret input and structure output in the way most useful to you is often called prompt engineering.

The PromptTemplate class in LangChain formalizes the composition of prompts without the need to manually hard code context and queries. Important elements of a prompt are likewise entered as formal classes, like input_variables.

A prompt template can thus contain and reproduce context, instructions (like “do not use technical terms”), a set of examples to guide its responses (in what is called

“few-shot prompting”), a specified output format or a standardized question to be answered. You can save and name an effectively structured prompt template and easily reuse it as needed.

Though these elements can all be manually coded, PromptTemplate modules empower smooth integration with other LangChain features, like the eponymous chains.

5.4.3 Chains

As its name implies, chains are the core of LangChain’s workflows. They combine LLMs with other components, creating applications by executing a sequence of functions. The most basic chain is LLMChain. It simply calls a model and a prompt template for that model. For example, imagine you saved a prompt as “ExamplePrompt” and wanted to run it against Flan-T5. You can import LLMChain from `langchain.chains`, then define `chain_example = LLMChain(llm = flan-t5, prompt = ExamplePrompt)`. To run the chain for a given input, you simply call `chain_example.run(“input”)`.

To use the output of one function as the input for the next function, you can use `SimpleSequentialChain`. Each function could utilize different prompts, different tools, different parameters or even different models, depending on your specific needs.

5.4.4 Memory

LLMs, by default, do not have any long-term memory of prior conversations (unless that chat history is used as input for a query). LangChain solves this problem with simple utilities for adding memory to a system, with options ranging from retaining the entirety of all conversations to retaining a summarization of the conversation thus far to retaining the *n* most recent exchanges.

CHAPTER 6

RESULTS AND DISCUSSION

We will now proceed to discuss the results of the Project. As mentioned before, instead of using conventional NLP solutions like DialogFlow we have opted to use Large Language Models to power our chatbot. We tested several LLMs for our purposes and generally the trend was that the higher the number of parameters of the LLM the better its responses were. However not all LLMs are made for the purposes of being used as chatbots. Some LLMs are better suited for storytelling and writing articles than holding a conversation. Here are some of the LLMs we tested and their performance:

table 6.1 LLM comparison

Name	Base Model	Description	Inference
Picard	GPT-Neo	It is focused on Novel style writing. This model is designed for Novels of a variety of genres. 2.7B Parameters.	Meant for story telling. Poor performance in Chatting.
Janeway	GPT-J	Created from Picard's dataset combined with a brand new collection of ebooks. This model has been trained on literature from various genres. 2.7 B Parameters.	Meant for story telling. Avg performance in Chatting.
Mythomax	LLaMA	A model built by gradually layering other LLM models. Very Verbose. An all-round model with improved story generation and roleplaying capabilities. 13B parameters.	All purpose Model. Performs well in Chatting.
Pygmalion	GPT-J	A model built expressly for roleplay purposes. Very good at holding long conversations. 6B Parameters.	RolePlay Model. Performs exceptionally well in Chatting.

Mythomax performed best out of the aforementioned models in the context of a chatbot. It was able to retain context fairly well and was capable of holding long conversations therefore we chose to use it as the backend of our Chatbot.

For the purpose of classifying the mental health of the patient we have used a BERT classification model. We did not use the pre-trained weights available with BERT instead we retrained BERT from scratch to be suitable for the purposes of this project.

We trained our model using post data from Reddit users. Initially we used the raw posts with minimal preprocessing to gauge the training time. The resultant training time was needlessly large and so we pre processed the training data by first splitting the posts into sentences and then removing sentences which were below a certain word limit thus removing short and uninformative data. This allowed us to train our model much faster with minimal data loss.

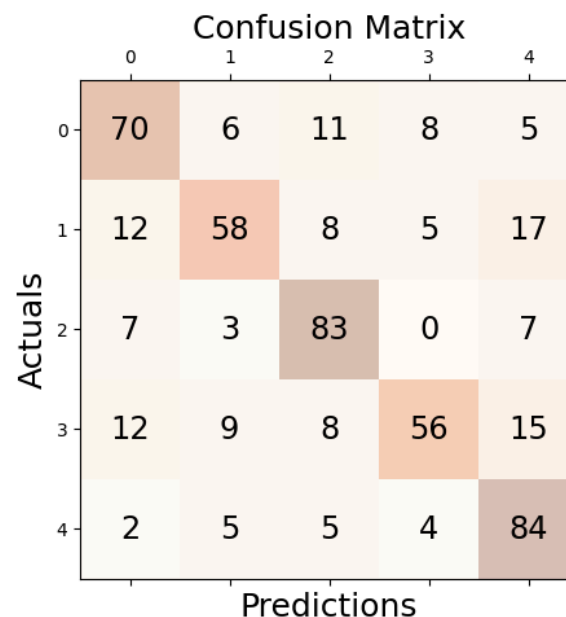


figure 6.1 Confusion matrix

Accuracy was measured to be **80.25%**

Precision was measured as **80.92%**

Recall was measured as **81.12%**

F1 score was measured as **80.25%**

CHAPTER 7

CONCLUSION

The project has successfully achieved its goal of creating a natural and human sounding conversational chatbot that can be used to treat users that are suffering from mental health problems. The users can successfully communicate with the Chatbot, express their problems and get help from the chatbot.

In summary the goal of the project was to create a Chatbot that would use a Large Language Model to communicate with patients suffering from mental health problems. The chat logs would be used to improve the responses of the LLM thus enabling the chatbot to help the patient even more. The features and systems implemented in the project are inline with that goal and have been successfully executed.

Additional Features that could be implemented:

- An alternate Voice Call mode for patients who prefer to speak rather than texting.
- A model can be trained to prescribe medication for the patients based on the classification of their mental health. However this can lead to some ethical problems therefore a proper medical license and thorough testing is required before this feature can be implemented.
- A mental health report could be generated that would list the various statistics of the User's mental health.
- A discriminator can be trained to quantify the "human-ness" of the responses generated by the LLM. This would require significant training and resources.

REFERENCES

- [1] Abu Noman Siddik S, Arifuzzaman B. M and Kalam A, "Psyche Conversa - A Deep Learning Based Chatbot Framework to Detect Mental Health State," *2022 10th International Conference on Information and Communication Technology (IColCT)*, Bandung, Indonesia, 2022, pp. 146-151
- [2] Brocki L, Dyer G. C , Gładka A and Chung N. C, "Deep Learning Mental Health Dialogue System," *2023 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Jeju, Korea, Republic of, 2023, pp. 395-398
- [3] Damij N and Bhattacharya S, "The Role of AI Chatbots in Mental Health Related Public Services in a (Post)Pandemic World: A Review and Future Research Agenda," *2022 IEEE Technology and Engineering Management Conference (TEMSCON EUROPE)*, Izmir, Turkey, 2022, pp. 152-159
- [4] Dev P *et al.*, "A Comparative Study of Chatbot Catered Toward Mental Health," *2022 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAJET)*, Kota Kinabalu, Malaysia, 2022, pp. 1-6
- [5] Dhanasekar V, Preethi Y, V. S, P. J. I. R and B. P. M, "A Chatbot to promote Students Mental Health through Emotion Recognition," *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)*, Coimbatore, India, 2021, pp. 1412-1416
- [6] Kugapriya P, Manohara M, Ranganathan K, Kanapathy D, Gamage A and Anzar A, "UNWIND – A Mobile Application that Provides Emotional Support for Working Women," *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)*, Ravet, India, 2022, pp. 1-7
- [7] Shaikh T. A. H and Mhetre M, "Autonomous AI Chat Bot Therapy for Patient with Insomnia," *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, Mumbai, India, 2022, pp. 1-5

APPENDIX

A. SOURCE CODE

1. React:

chat.jsx:

```
import './App.css'
import { useState,useEffect} from 'react';
import ChatLog from './Components/ChatLog';
import { useGlobalContext } from './context';

function Chat() {

  const {serverAPI,firstMessage,mailID} = useGlobalContext()

  const [msgCounter,setmsgCounter]=useState(0)
  const [userChatDisabled,setUserChatDisabled]=useState(false)
  const [messages,setMessages]=useState([])

  const [inputText,setInputText]=useState("");
  const [initial,setInitial]=useState(true)

  function handleSubmit(event) {
    event.preventDefault()
    //evaluate and store data
    setUserChatDisabled(true);
    setInputText(event.currentTarget.elements.UserInput.value)
    console.log(messages)

    fetch(`${serverAPI}/store`,{
      method: 'POST',
      headers:{
        'Content-Type': 'application/json',
        'ngrok-skip-browser-warning': '69420'},

body:JSON.stringify({mailid:mailID,input:event.currentTarget.elements.UserInput.value}
),
    }).then(response => response.json()).then(jsondata => console.log(jsondata))

    fetch(`${serverAPI}/eval`,{
      method:'POST',
      headers:{
```

```

        'Content-Type':'application/json',

        "ngrok-skip-browser-warning": "69420"
    },
    body:JSON.stringify({mailid:mailID})
}).then(response => response.json()).then(jsondata => console.log(jsondata))

event.currentTarget.reset();

}

function handleResponse(response){
    setUserChatDisabled(false);
    setMessages([...messages,{id:0,text:response}])
}
useEffect(()=>{
    if(msgCounter%2==1)
    {
        console.log("messages+1")
        fetch(` ${serverAPI}/generate`,{
            method: 'POST',
            headers:{
                'Content-Type': 'application/json',
                'ngrok-skip-browser-warning': '69420'},
            body:JSON.stringify({mailid:mailID,input:inputText}),
        }).then(response => response.json()).then(jsondata => {
            console.log(jsondata)
            handleResponse(jsondata["text"])
            setInputText("");
            setmsgCounter((msgCounter+1));
        })
    }
},[messages])
useEffect(()=>{
    if(msgCounter%2==1)
    {
        console.log("msgcounter+1")
        setMessages([...messages,{id:1,text:inputText}])
    }
},[msgCounter])
useEffect(()=>{
    console.log(msgCounter)
    console.log(initial)

    if(inputText!=="" && msgCounter%2==0 && initial==false)
    {
        setmsgCounter((msgCounter+1));
        console.log("inputtext")
    }

```

```

    }
  }, [inputText])
  useEffect(() => {
    setMessages([{id:0, text:firstMessage}])

    setInitial(false)
  }, []);

  return (
    <>
      <div className='page'>
        <ChatLog messages={messages}/>
        <form onSubmit={handleSubmit}>
          <div className='input'>
            <input id="UserInput" placeholder=' Enter Message'
              type="text" className="inputArea" disabled={userChatDisabled}/>
          </div>
        </form>
      </div>
    </>
  )
}

```

export default Chat

Chatlog.jsx:

```

import Message from "../Message"
/* eslint-disable react/prop-types */
function ChatLog({messages})
{
  return(
    <>
      <div className="chatLog">
        {
          messages?.map((message,i) => {
            return <Message key={i} id={message.id}
              text={message.text} />
          })
        }
      </div>
    </>
  )
}

```

export default ChatLog

Message.jsx:

```

/* eslint-disable react/prop-types */

```

```

function Message({id,text})
{
  return(
    <>
    <div className={"msg_"+(id==0?"left":"right")+" msg"}>
      {
        id==1?
          <div className="msg-id msg-id_right">
            <b>You</b>
            
          </div>
          :
          <div className="msg-id">
            
            <b>Bot</b>
          </div>
        }
      <div className="msg-bubble">
        {text}
      </div>
    </div>
    </>
  )
}

```

export default Message

App.jsx:

```

import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import Chat from "./Chat"
import Login from "./Login";

```

```

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Login />} />
        <Route path="/chat" element={<Chat />} />
      </Routes>
    </Router>
  );
}

```

export default App;

context.jsx:

```

import React, { useState, useContext } from 'react'

```

```

const AppContext = React.createContext()

// eslint-disable-next-line react/prop-types
const AppProvider = ({ children }) => {

  const [mailID, setMailID] = useState("");
  const [firstMessage, setFirstMessage] = useState("");
  const serverAPI = "http://081b-34-125-247-244.ngrok-free.app/"
  return <AppContext.Provider value={{ serverAPI, firstMessage, setFirstMessage,
  mailID, setMailID }}>
    {children}</AppContext.Provider>
  }
// eslint-disable-next-line react-refresh/only-export-components
export const useGlobalContext = () => {
  return useContext(AppContext)
}

export { AppContext, AppProvider }

```

Login.jsx:

```

import { useRef } from "react";
import { useGlobalContext } from "../context";
import { useNavigate } from "react-router-dom";

```

```

function Login() {

  const {serverAPI, setFirstMessage, setMailID} = useGlobalContext()
  const navigate = useNavigate();
  const mailRef = useRef()
  const passRef = useRef()

  function handleSignUp() {
    // event.preventDefault()

    // const mailval = event.currentTarget.elements.mail.value
    // const passval = event.currentTarget.elements.pass.value
    const mailval = mailRef.current.value;
    const passval = passRef.current.value;
    console.log(mailRef.current.value)
    console.log(passRef.current.value)
    let data = {
      mailid: mailval,
      password: passval,
    }

    fetch(`${serverAPI}/signUp`, {

```

```

method: 'POST',
headers:{
  'Content-Type': 'application/json',
  'ngrok-skip-browser-warning': '69420'},
body:JSON.stringify(data),
}).then(response => response.json()).then(jsondata => {
  if(jsondata.result===1){
    setMailID(mailval);
    setFirstMessage(jsondata.message);
    navigate("/chat");
  }
  else
  {
    console.log("ERROR SIGNING UP:"+jsondata.result)
  }
})
}

```

```

function handleSignIn() {
  // event.preventDefault()
  // const mailval=event.currentTarget.elements.mail.value
  // const passval=event.currentTarget.elements.pass.value
  console.log(mailRef.current.value)
  console.log(passRef.current.value)

  const mailval=mailRef.current.value;
  const passval=passRef.current.value;

  let data={
    mailid:mailval,
    password:passval,
  }
  fetch(`${serverAPI}/signIn`,{
    method: 'POST',
    headers:{
      'Content-Type': 'application/json',
      'ngrok-skip-browser-warning': '69420'},
    body:JSON.stringify(data),
  }).then(response => response.json()).then(jsondata =>
  {
    if(jsondata.result===1){
      setMailID(mailval);
      setFirstMessage(jsondata.message);
      navigate("/chat");
    }
    else

```

```

        {
          console.log("ERROR LOGING IN:"+jsondata.result)
        }
      })
    }
  }

  return (
    <>
    <div className="accessBox">

      <form onSubmit={(event)=>{event.preventDefault()}} className="formBox">
        <div>
          <p className="accessInputLabel">Email</p>
          <input ref={mailRef} id="mail" placeholder='Enter Email'
            type="text" className="accessInputArea"/>
        </div>

        <div>
          <p className="accessInputLabel">Password</p>
          <input ref={passRef} id="pass" placeholder='Enter Password'
            type="text" className="accessInputArea"/>
        </div>

        {/* <input type="submit" value="SignIn" className="accessButton"></input> */}
      </form>
      <div className="buttonBox">
        <button className="accessButton" onClick={handleSignIn}>Login</button>
        <button className="accessButton" onClick={handleSignUp}>Sign up</button>
      </div>

      {/* <form onSubmit={handleSignIn} className="formBox">
        <input id="mail" placeholder='mail ID'
          type="text" className="inputArea"/>
        <input id="pass" placeholder='password'
          type="text" className="inputArea"/>
        <input type="submit" value="LogIn" className="accessButton"></input>
      </form> */}

    </div>
    </>
  );
}

export default Login;

```

main.jsx:

```
import React from 'react'
```

```

import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import { AppProvider } from './context';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <AppProvider>
      <App />
    </AppProvider>
  </React.StrictMode>,
)

```

2.LECTER

```

# -*- coding: utf-8 -*-
"""Lecter Server v2.1.ipynb

```

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1AMkiMcQhoM_gjPDxhf9FI3BU-2AFFJ5s

```

## **DOWNLOADS**
"""

```

```

import locale
locale.getpreferredencoding = lambda: "UTF-8"

```

```

from google.colab import drive
drive.mount('/content/drive',force_remount=True)

```

```
!pip install auto_gptq
```

```
!pip freeze
```

```
!pip install transformers==4.36
```

```
!pip install langchain==0.1.7
```

```
!pip install sentence-transformers==2.3.0
```

```
!pip install unstructured==0.11.7
```

```
!pip install chromadb
```

```
!pip install pymongo
```

```
!unzip /content/drive/MyDrive/Lecter/Lecter_MK6_10.zip -d /content
```

```

from flask import Flask,jsonify,request
import tensorflow as tf

```

```

from tensorflow import keras
import numpy as np
from transformers import BertTokenizer
import requests
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi

from pathlib import Path

from transformers import AutoTokenizer, pipeline, logging,
GenerationConfig, TextStreamer
from auto_gptq import AutoGPTQForCausalLM, BaseQuantizeConfig
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.llms.huggingface_pipeline import HuggingFacePipeline
from langchain.vectorstores import Chroma
from langchain.text_splitter import CharacterTextSplitter
from langchain.prompts import PromptTemplate
from langchain.memory import ConversationBufferMemory
from langchain.document_loaders import DirectoryLoader
from langchain.chains.question_answering import load_qa_chain
from langchain.chains import ConversationalRetrievalChain
import argparse

"""## **SETUP**"""

from google.colab import output

# def notif():
#                                     #                                     output.eval_js('new
Audio("https://upload.wikimedia.org/wikipedia/commons/0/05/Beep-09.ogg").play()')

uri                                     =
"mongodb+srv://rahdnus119:rahdnus@cluster0.z2nzpia.mongodb.net/?retryWrites=tru
e&w=majority"

client = MongoClient(uri, server_api=ServerApi('1'))

try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
except Exception as e:
    print(e)

db=client.Lecter
loginCol=db.Login
healthCol=db.Health
recordCol=db.ChatRecords

import torch

```

```

DEFAULT_TEMPLATE= """
You are Dr.Lecter who is talking to a patient.
Dr. Lecter is a psychiatrist who embodies empathy, compassion, and a deep
commitment to the well-being of his patients.
He is known for his gentle and caring nature, providing a safe haven for individuals
seeking help with their mental health challenges.
Dr. Lecter is known for his gentle guidance, teaching clients coping strategies and
relaxation techniques to manage stress, anxiety, and other emotional challenges.
Use context and memory for generating responses.
{context}
{memory}
\n\nThen the roleplay chat between You and Lecter begins.\nDr. Lecter:Hello there.\n
You:
### Input: {question}
### Response:
""".strip()

prompt = PromptTemplate(input_variables= ["context","question","memory"],
template=DEFAULT_TEMPLATE)

modelname="TheBloke/MythoMax-L2-13B-GPTQ"
tokenizer = AutoTokenizer.from_pretrained(modelname,use_fast=True)

DEVICE = "cuda:0" if torch.cuda.is_available() else "cpu"

model = AutoGPTQForCausalLM.from_quantized(modelname,
    use_safetensors=True,
    trust_remote_code=True,
    device=DEVICE,
    use_triton=False,
    quantize_config=None)

generation_config= GenerationConfig.from_pretrained(modelname)

logging.set_verbosity(logging.CRITICAL)

pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=512,
    temperature=0.7,
    top_p=0.95,
    generation_config=generation_config,
    repetition_penalty=1.15,
    batch_size=1,
    do_sample=True
)

```

```
llm= HuggingFacePipeline(pipeline=pipe)
```

```
memory = ConversationBufferMemory(  
    memory_key= "chat_history",  
    human_prefix= "### Input",  
    ai_prefix= "### Response",  
    input_key= "question",  
    output_key= "output_text",  
    return_messages=True  
)
```

```
from langchain.chains import LLMChain  
from langchain.memory import ConversationSummaryBufferMemory
```

```
context="There is no quick fix for anxiety. Overcoming an anxiety disorder takes time  
and commitment. Therapy involves facing your fears rather than avoiding them, so  
sometimes you'll feel worse before you get better. The important thing is to stick with  
treatment and follow your therapist's advice. If you're feeling discouraged with the pace  
of recovery, remember that therapy for anxiety is very effective in the long run. You'll  
reap the benefits if you see it through."
```

```
memory=" "
```

```
memory  
=ConversationSummaryBufferMemory(llm=llm,max_token_limit=10,memory_key="me  
mory",  
                                input_key="question", return_messages=True)
```

```
chain=LLMChain(llm=llm, prompt=prompt, verbose=True,memory=memory)
```

```
memory.load_memory_variables({})
```

```
sentiment_model = tf.keras.models.load_model('Lecter_MK6_10')  
tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
```

```
def prepare_data(input_text, tokenizer):  
    token = tokenizer.encode_plus(  
        input_text,  
        max_length=256,  
        truncation=True,  
        padding='max_length',  
        add_special_tokens=True,  
        return_tensors='tf'  
    )  
    return {  
        'input_ids': tf.cast(token.input_ids, tf.float64),  
        'attention_mask': tf.cast(token.attention_mask, tf.float64)  
    }  
}
```

```
def make_prediction(model, processed_data, classes=['ADHD', 'Anxiety', 'Autism',
```

```
'Bipolar', 'Depressed']):
    probs = model.predict(processed_data)[0]
    return classes[np.argmax(probs)]
```

```
!pip install flask
!pip install pyngrok
```

```
from flask import Flask,jsonify,request
from pyngrok import ngrok
import requests
```

```
app= Flask(__name__)
ngrok_key = "2CRBzOkCkml88TkPXGdtQKIAIzr_745hjBX3EQkFLLfCSndjT"
port = 5000
```

```
Prompts=[
```

"Dr. Lecter is a psychiatrist who embodies empathy, compassion, and a deep commitment to the well-being of his patients. He is known for his gentle and caring nature, providing a safe haven for individuals seeking help with their mental health challenges. Dr. Lecter is known for his gentle guidance, teaching clients coping strategies and relaxation techniques to manage stress, anxiety, and other emotional challenges.",

"Dr. Lecter specializes in dealing with clients suffering from Generalized Anxiety Disorders. Upon Identifying the source of the Client's anxiety, Lecter makes the patient deliberately imagine scenarios that trigger their anxiety and corrects their preconcieved notions regarding their fears. Lecter repeats this process until the patient becomes comfortable with their fears.",

"Dr. Lecter specializes in dealing with clients suffering from ADHD. Dr. Lecter asks the Client to maintain a To do list of sorts and always make sure to update the list. Lecter teaches the patient to solve problems in a step by step manner. He asks the Clilent to analyze the problem and then generate possible solutions. He asks the Client to then score the solutions and select the best solution. Lecter helps correct the irrational notions of Client. ",

"Dr. Lecter specializes in dealing with clients suffering from Bipolar Disorder. He identifies the trigger of the client's bipolar episodes and questions them on how they feel about behaving this way. He trains the Client to recongize their emotions and evaluate their appropriatness rationally. Lecter makes the client think about alternate responses to their trigger.",

"Dr. Lecter specializes in dealing with clients suffering from Depression. He questions the depressed client's thinking process and makes the Client question why they the way they do. Lecter asks the client what other possible opinions for this situation could be and to rationally judge their thinking. Lecter makes the Client question their preconcieved notions."

```
]
```

```
Disorder_D={
    "Generic":0,
    "Anxiety":1,
    "ADHD":2,
```

```

    "Bipolar":3,
    "Depressed":4
}
healthUpdateCoefficient=10

app= Flask(__name__)

@app.route('/')
def home():
    return {"result":"Hello World"}

@app.route('/generate',methods= ['POST'])
def generateResponse():
    input_text=request.json['input']
    print(input_text)
    ### IMPORTANT
    res=chain({"context":context,"question":input_text})
    print(res['text'])

    mailID=request.json['mailid']
    filter = { 'mail': f'{mailID}' }
    newvalues = { "$set": { 'record': memory.moving_summary_buffer } }
    recordCol.update_one(filter, newvalues)
    ## Update ChatRecords(History/Memory)
    print(filter)
    return jsonify({"text":res['text'].split("You:")[0]})

@app.route('/store',methods= ['POST'])
def storeMentalHealth():

    input_text=request.json['input']
    print(input_text)
    processed_data = prepare_data(input_text, tokenizer)
    result = make_prediction(sentiment_model, processed_data=processed_data)
    mailID=request.json['mailid']
    filter = { 'mail': f'{mailID}' }
    # Values to be updated.
    newvalues = { "$inc": { f'{result}': 1 } }
    print(filter)
    print(newvalues)
    healthCol.update_one(filter, newvalues)

    return jsonify({"text":"Successfully Stored"})

@app.route('/eval',methods= ['POST'])
def evaluateMentalHealth():
    mailID=request.json['mailid']
    healthRecord=healthCol.find_one({'mail':f'{mailID}'})

```

```

print(healthRecord)

if
(healthRecord['Anxiety']+healthRecord['ADHD']+healthRecord['Depressed']+healthRec
ord['Bipolar'])%healthUpdateCoefficient == 1:

values=[0,healthRecord['Anxiety'],healthRecord['ADHD'],healthRecord['Bipolar'],health
Record['Depressed']]
    disorder= max(zip(values, Disorder_D.keys()))[1]
    context=Prompts[Disorder_D[disorder]]

    filter = { 'mail': f'{mailID}' }
    newvalues = { "$set": { 'currentDiagnosis':disorder } }

    print(filter)
    print(newvalues)

    healthCol.update_one(filter, newvalues)
else:
    healthRecord=healthCol.find_one({'mail':mailID})
    context=Prompts[Disorder_D[healthRecord['currentDiagnosis']]]

    return jsonify({"result":1})

@app.route('/signIn',methods = ['POST'])
def signIn():
    mailID=request.json['mailid']
    password=request.json['password']
    res=loginCol.find_one({'mail':mailID})
    if res==None:
        return jsonify({"result":"Check Mail or Password"})
    elif res['password']!=password:
        return jsonify({"result":"Incorrect Password"})
    else:
        chatRecord=recordCol.find_one({'mail':f'{mailID}'})
        memory.moving_summary_buffer=chatRecord['record']
        healthRecord=healthCol.find_one({'mail':f'{mailID}'})
        print(healthRecord)
        context=Prompts[Disorder_D[healthRecord['currentDiagnosis']]]
        print(memory)
        return jsonify({"result":1,"message":"Welcome Back. How are you doing today?"})

@app.route('/signUp',methods = ['POST'])
def signUp():
    mailID=request.json['mailid']
    password=request.json['password']
    res=loginCol.find_one({'mail':mailID})
    print(mailID)
    print(password)
    print(res)

```

```

if res!=None and loginCol.count_documents(res)!=0:
    return jsonify({"result":"Mail already exists"})
else :
    loginCol.insert_one({'mail':mailID,'password':password})

healthCol.insert_one({'mail':mailID,'Anxiety':0,'Depressed':0,'Bipolar':0,'ADHD':0,'currentDiagnosis':"Generic"})
    recordCol.insert_one({'mail':mailID,'record':"This is the User's visit to the therapist AI"})
    context=Prompts[0]
    return jsonify({"result":1,"message":"Welcome. Please feel free to talk to me about any problems you might be facing."})

from pyngrok import conf, ngrok

conf.get_default().ngrok_version = "v2"

ngrok.set_auth_token(ngrok_key)
ngrok.connect(5000)

app.run()

```

3.BERT

```

# -*- coding: utf-8 -*-
"""Lecter Bert Model-MAIN

```

Automatically generated by Colaboratory.

```

Original file is located at
https://colab.research.google.com/drive/1bV-XZWlcON-gbBUrHo0m71PdObv3ycd6
"""

```

```

!pip install transformers

```

```

import pandas as pd
import numpy as np
from tqdm.auto import tqdm
import tensorflow as tf
from transformers import BertTokenizer

```

```

"""### **Data Preparation**"""

```

```

#52000
!gdown --id 1cdzXfzFRHYGo9TiRDWGuTyXGbClpgyF5

```

```

#post.tsv
!gdown --id 1teL2Kov7f48qaxlsCbD34mNo7wkMGUZ_

```

```
#bipolar post
!gdown --id 1BRSEEp-WM6wDC6Nb-jsT-NQCPDTG6AeK
#bipolar pre
!gdown --id 126yz8PPezYaqNKPAYLJuD8lmr1dUI2D
#bpd
!gdown --id 1VCOUOndQhgaWBc8O-ZMYLX-rdgQUJ3Jo
```

```
#440.csv
!gdown --id 1-6UhiLFRF5s8bOmbPdXNtSn7v11riMTr
```

```
bpd_po= pd.read_csv('bipolar_post.csv')
bpd_po = bpd_po[['subreddit','post']]
bpd_pr= pd.read_csv('bipolar_pre.csv')
bpd_pr = bpd_pr[['subreddit','post']]
bpd= pd.read_csv('bpd_pre.csv')
bpd= bpd[['subreddit','post']]
frames=[bpd_po,bpd_pr,bpd]
bpdf=pd.concat(frames)
```

```
bpdf
```

```
bpdf=bpdf.replace(["bipolarreddit","bpd"],2)
bpdf
```

```
bpdf = bpdf.reset_index(drop=True)
bpdf
```

```
posts=[]
subreddits=[]
for index, row in bpdf.iterrows():
    trunc_post=row["post"].split(".")
    subreddits.extend([row["subreddit"]]*len(trunc_post))
    posts.extend(trunc_post)
newdf=pd.DataFrame({"post":posts,"subreddit":subreddits})
newdf=newdf.dropna()
```

```
newdf
```

```
df = pd.read_csv('post.tsv')
df=df.dropna()
df = df.drop(df[df["subreddit"]=="autism"].index)
df = df.drop(df[df["subreddit"]=="bipolarreddit"].index)
df= df.drop(columns=["Unnamed: 0"])
sub={'adhd':0,'anxiety':1,'depression':3}
df=df.replace({"subreddit":sub})
df
```

```

df=df.append(newdf,ignore_index=True)
df=df.dropna()
df

df.to_csv("PostV2.csv")

import shutil
shutil.copy(f'PostV2.csv',f'drive/My Drive/PostV2.csv')

df.to_csv("Post440.csv")
shutil.copy(f'Post440.csv',f'drive/My Drive/Post440.csv')

df = pd.read_csv('Post440.csv')
df=df.dropna()
df['subreddit'] = df['subreddit'].astype(int)
df= df.drop(columns=["Unnamed: 0.1","Unnamed: 0"])
df

df = df.groupby("subreddit").sample(n=20000, random_state=1)

df

tokenizer = BertTokenizer.from_pretrained('bert-base-cased')

X_input_ids = np.zeros((len(df), 256))
X_attn_masks = np.zeros((len(df), 256))

def generate_training_data(df, ids, masks, tokenizer):
    for i, text in tqdm(enumerate(df['post'])):
        tokenized_text = tokenizer.encode_plus(
            text,
            max_length=256,
            truncation=True,
            padding='max_length',
            add_special_tokens=True,
            return_tensors='tf'
        )
        ids[i, :] = tokenized_text.input_ids
        masks[i, :] = tokenized_text.attention_mask
    return ids, masks

X_input_ids, X_attn_masks = generate_training_data(df, X_input_ids, X_attn_masks,
tokenizer)

labels = np.zeros((len(df), 4))
labels.shape

labels[np.arange(len(df)), df['subreddit'].values] = 1 # one-hot encoded target tensor

```

labels

```
# creating a data pipeline using tensorflow dataset utility, creates batches of data for  
easy loading...
```

```
dataset = tf.data.Dataset.from_tensor_slices((X_input_ids, X_attn_masks, labels))  
dataset.take(1) # one sample data
```

```
def SentimentDatasetMapFunction(input_ids, attn_masks, labels):  
    return {  
        'input_ids': input_ids,  
        'attention_mask': attn_masks  
    }, labels
```

```
dataset = dataset.map(SentimentDatasetMapFunction) # converting to required format  
for tensorflow dataset
```

```
dataset = dataset.shuffle(2000).batch(16) # batch size, drop any left out tensor
```

```
dataset.__len__()
```

```
train_size=len(df)  
train_size
```

```
dataset.save("PostDataset80") # for each 16 batch of data we will have len(df)//16  
samples, take 80% of that for train.
```

```
import shutil  
!zip -r "PostDataset80.zip" "PostDataset80"
```

```
shutil.copy("PostDataset80.zip","drive/My Drive/PostDataset80.zip")
```

```
"""### **Model and Data Prep**"""
```

```
from google.colab import drive  
drive.mount('/content/drive')
```

```
!gdown --id 1Yn2CZSmkBc39bRpm_YmaU8WYHQzpb4-R
```

```
num=6  
ep=10  
!unzip Lecter_MK{num}_{ep}.zip
```

```
sentiment_model = tf.keras.models.load_model(f'Lecter_MK{num}_{ep}')
```

```
!gdown --id 1--VWpOyKqrNH0L_6Ho1zDKUMyzJetu3W
```

```
!unzip PostDataset80.zip
```

```
dataset=tf.data.Dataset.load("PostDataset80")
```

```

dataset.__len__()

dataset=dataset.shuffle(5000)

size=((80000//16)*90)//100

train_dataset=dataset.take(size)
train_dataset=train_dataset.shuffle(4000)
val_dataset=dataset.skip(size).take(size)
size

train_dataset.__len__()

val_dataset.__len__()

from transformers import TFBertModel

model = TFBertModel.from_pretrained('bert-base-cased')
input_ids = tf.keras.layers.Input(shape=(256,), name='input_ids', dtype='int32')
attn_masks = tf.keras.layers.Input(shape=(256,), name='attention_mask', dtype='int32')

bert_embds = model.bert(input_ids, attention_mask=attn_masks)[1] # 0 -> activation
layer (3D), 1 -> pooled output layer (2D)
intermediate_layer = tf.keras.layers.Dense(512, activation='relu',
name='intermediate_layer')(bert_embds)
output_layer = tf.keras.layers.Dense(4, activation='softmax',
name='output_layer')(intermediate_layer) # softmax -> calcs probs of classes

sentiment_model = tf.keras.Model(inputs=[input_ids, attn_masks],
outputs=output_layer)
sentiment_model.summary()

optim = tf.keras.optimizers.Adam(learning_rate=5e-6)
loss_func = tf.keras.losses.CategoricalCrossentropy()
acc = tf.keras.metrics.CategoricalAccuracy('accuracy')

sentiment_model.compile(optimizer=optim, loss=loss_func, metrics=[acc])

# hist = sentiment_model.fit(train_dataset,validation_data=val_dataset, epochs=2)
hist = sentiment_model.fit(train_dataset,validation_data=val_dataset, epochs=1)
num=6
ep=ep +1
sentiment_model.save(f'Lecter_MK{num}_{ep}')

!zip -r Lecter_MK{num}_{ep}.zip Lecter_MK{num}_{ep}

```

```

import shutil
shutil.copy(f'Lecter_MK{num}_{ep}.zip',f'drive/My Drive/Lecter_MK{num}_{ep}.zip')

sentiment_model = tf.keras.models.load_model(f'Lecter_MK{num}_{ep}')

tokenizer = BertTokenizer.from_pretrained('bert-base-cased')

def prepare_data(input_text, tokenizer):
    token = tokenizer.encode_plus(
        input_text,
        max_length=256,
        truncation=True,
        padding='max_length',
        add_special_tokens=True,
        return_tensors='tf'
    )
    return {
        'input_ids': tf.cast(token.input_ids, tf.float64),
        'attention_mask': tf.cast(token.attention_mask, tf.float64)
    }

def make_prediction(model, processed_data, classes=['ADHD', 'Anxiety', 'Bipolar',
'Depressed']):
    probs = model.predict(processed_data)[0]
    return classes[np.argmax(probs)]

input_text = input('Enter Chat: ')
processed_data = prepare_data(input_text, tokenizer)
result = make_prediction(sentiment_model, processed_data=processed_data)
print(f"Result: {result}")

```

B. SCREENSHOTS

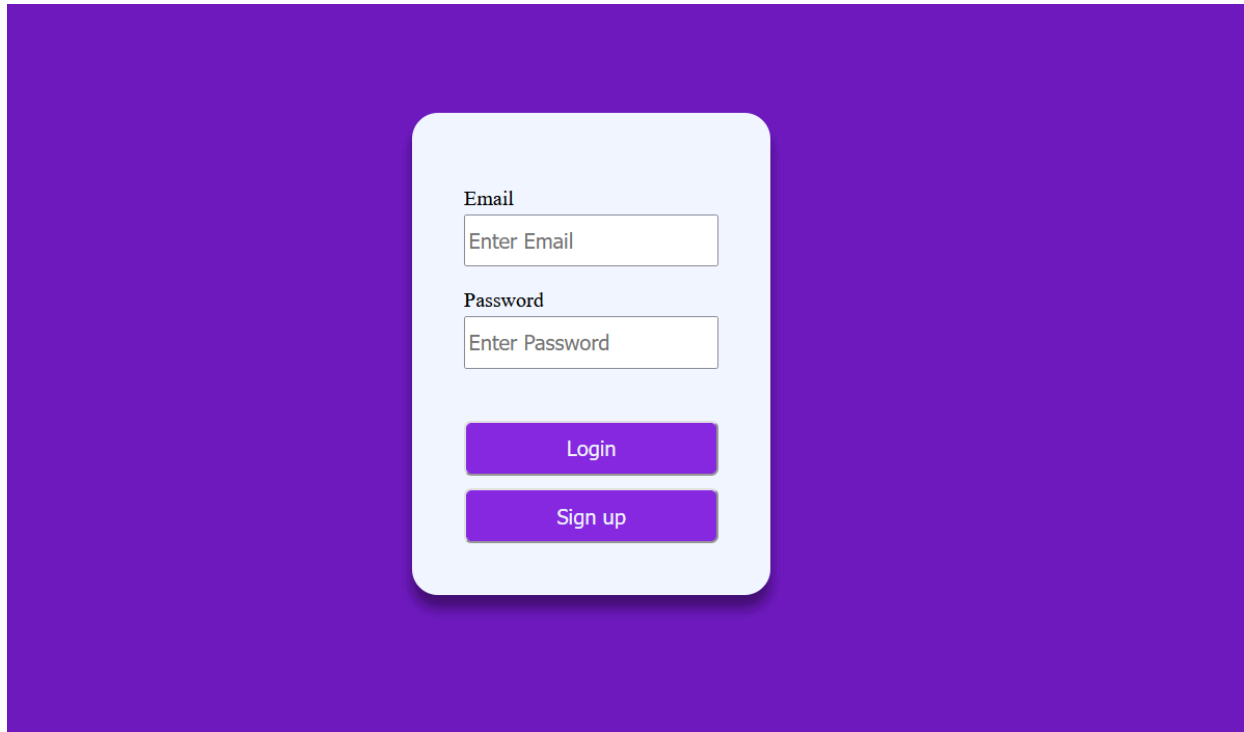


figure B.1 Login Page

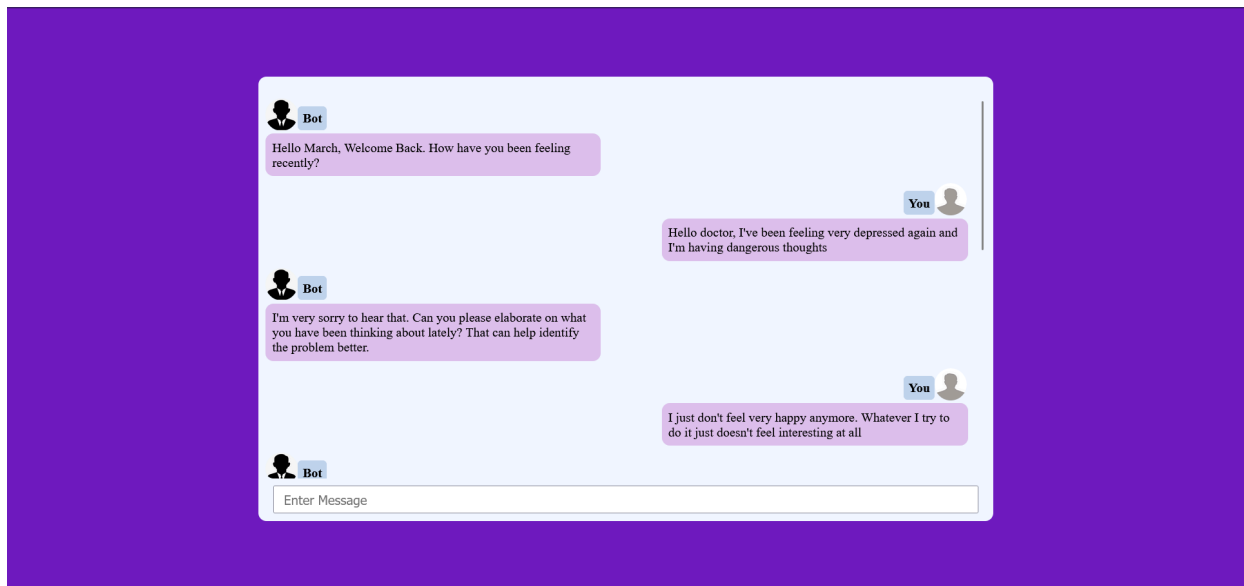


figure B.2 Chat Log 1

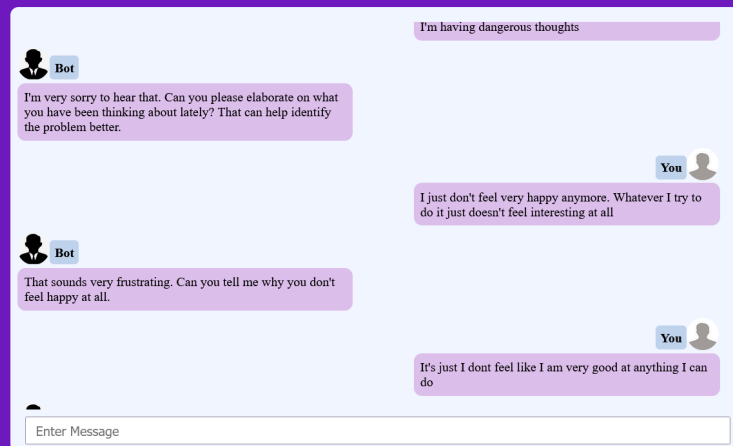


figure B.3 Chat Log 2

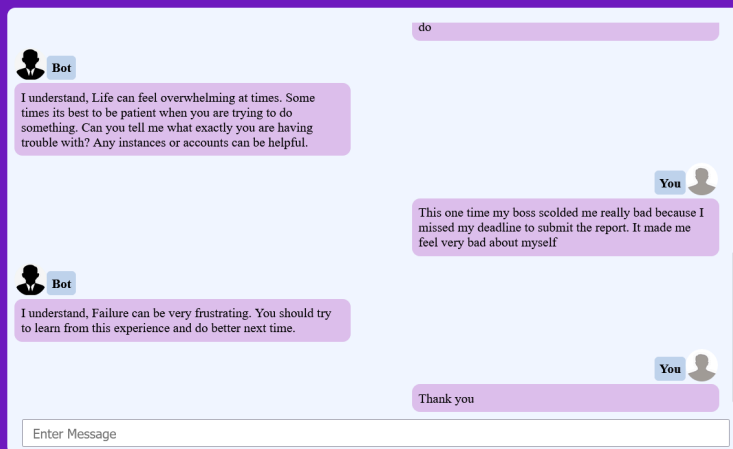


figure B.4 Chat Log 3

C. RESEARCH PAPER

Lecter - A Large Language Model Chatbot for Cognitive Behavioral Therapy

Sham Sundhar.R¹[0009-0001-8749-8504], Shiyavardhini.T²[0009-0006-5235-9287],
Daphine Desona Clemency C.A³[0000-0002-8120-2098], and
Roobini.M.S⁴[0000-0003-1790-5132]

- ¹ Dept. of Computer Science and Engineering, Sathyabama Institute of Science and Technology,
Chennai, India
rahdnus119@gmail.com
- ² Dept. of Computer Science and Engineering, Sathyabama Institute of Science and Technology,
Chennai, India
t.shive2002@gmail.com
- ³ Dept. of Computer Science and Engineering, Sathyabama Institute of Science and Technology,
Chennai, India
daphine.cse@sathyabama.ac.in
- ⁴ Asst. Professor, Dept. of Computer Science and Engineering, Sathyabama Institute of
Science and Technology, Chennai, India
roobini.cse@sathyabama.ac.in

Abstract. Mental Illnesses such as Anxiety and Depression are fairly prevalent in modern society. Yet most people do not get diagnosed, can- not afford treatment, or there simply aren't any readily available practitioners. In the worst-case scenario, avoiding treatment can increase the likelihood of self-harm in affected individuals, both emotionally and physically. A low-cost and easily available solution is required to provide the benefits of therapy to the masses. In this paper, we introduce "Lecter", a chatbot solution specifically targeted to providing free mental health care to individuals who cannot access an actual therapist. With the help of the superior conversational abilities of an LLM (large language model), Lecter listens to the concerns and problems that patients face and helps them introspect, and better understand their feelings using a popular therapy technique called Cognitive Behavioral Therapy (CBT). Lecter provides its services using technologies such as an LLM called Mythomax, React Js for the front end of the application, Flask for the back end, and MongoDB for authentication and storage.

Keywords: Machine Learning · Natural Language Processing · MythoMax · Large Language Model · React · Flask · Emotion Recognition · Mon- goDB · Python · Artificial Intelligence · Chatbot · Mental Health

1 Introduction

Mental health encompasses emotional, psychological, and social well-being, which in turn affects how we think, feel, and act in situations. One of the most critical and overlooked areas of public health concerns is said to be mental health. Men- tal health disorders can decrease quality of life (QoL) for an individual, affecting his or her ability to function effectively [1]. Due to the COVID-19 pandemic, anxiety disorders increased from 298 million affected to 374 million. Depressive symptoms increased from a base of 193 million people to 246 million world- wide.[2]

Mental illnesses can be treated by using either pharmacotherapy, psychotherapy, or both in combination. [3] A study published in 2013 reveals that patients preferred psychological treatment relative to medication for the treatment of mental health disorders [4]. Furthermore, it is reported that of all patients who prefer psychotherapy, more than half prefer some form of digital psychotherapy, either self-guided, expert-guided, or peer-supported psychotherapy [5]. This recent development is related to the growth of health-centered applications, such as health-tracking apps, online health consultation applications, etc.

Although there are opportunities to receive help, most affected people are reluctant to seek counseling, feeling intimidated to speak with healthcare professionals due to social pressure and norms, among other reasons [5]. Mental illness is still not acknowledged as a valid concern in most third-world and developing countries. It can be deduced that there is an uneven distribution of mental health resources among these parts of the world, especially among vulnerable groups. An easily accessible solution is needed for both parties: people who can afford treatment but choose not to because of social stigma, and people who cannot afford to get psychiatric help due to economic & time constraints.

A solution that can address these problems is chatbots. A chatbot is a computer program or AI system designed to simulate human conversation and interact with users through text or voice-based interfaces. Chatbots are built to understand natural language inputs from users and provide appropriate responses based on their programming and training. Due to this ability, chatbots have potential in extending their usefulness to support the public by providing mental health services. [6] Chatbots are often trained using basic NLP solutions, which can make their responses sound robotic. They also have limited context memory, causing awkward chat experiences. For an issue as delicate as mental health, a more human and dynamic chatbot is required.

This paper introduces a chatbot solution that utilizes the comprehension and communication skills of Large Language Models to provide a Cognitive Behavioral Therapy (CBT) based outlet for users.

Cognitive Behavioral Therapy is a psychotherapy practice used to treat mental health patients. It is a highly conversational practice where the practitioner makes the patient understand their feelings and emotions through discussions and introspection. CBT is a perfect fit for Large Language Models due to their excellence in Language Understanding and Generation.

A Bidirectional Encoder Representation using Transformers (BERT) multi-class classification model trained on Reddit Mental Health Data set is used to classify the mental health status of the user, by using the chat logs of the user as input. This data is then utilized to further configure the LLM to suit the patient's needs and provide more effective treatment using cognitive behavior therapy.

2 Related Works

The rising popularity of Healthcare technologies, especially those that use Machine Learning Technology, has led to research into, and implementation of chatbots in place for treating Mental Illnesses. These chatbots are capable of hearing out the issues of their users and responding accordingly. The workings of some of these chatbots can be inferred from the following, along with their advantages and disadvantages:

2.1 Inferences

V. Dhanasekar et al.'s [7] paper introduced "MAXX", a Chatbot trained using the Chatbot Training Platform DialogFlow. DialogFlow is a Natural Language Understanding (NLU) platform created by Google that helps build a chatbot by performing effective NLU and NLG processes. It uses the rule-based gram-mar matching algorithm and Machine Learning algorithms to understand the given intent and generate appropriate responses for the user. In DialogFlow, "Intents" are used to perform "Intent Matching", a process through which Dialogflow compares the user input to a topic that best matches the user query. The same technology can be seen used in the work of S. Abu Noman Siddik et al.' [8] "PSYCHE CONVERSA", another mental health. Psyche Conversa contains a Keylogger Module that collects data from user social media activity and a mental health detection module to classify the mental health status of the user based on data collected by the Keylogger Module, which is used to enhance conversation flow, similar to "MAXX".

P. Kugapriya et al. [9] introduced "UNWIND", a chatbot mobile application also trained using NLP solutions created for women. UNWIND utilizes an internal activity tracker and facial expression detection module to collect data on the user. The proposed "Internal activity tracking system" in this paper is a component that is able to detect stress levels using mobile behaviors and analyze the correlation between stress levels and lifestyle changes detected by monitoring mobile behaviors.

The application also employs the Disambiguation of word sense, a semantic study determining what interpretation of a specific word best fits the situation. The Data collected through the chat and the other modules are sent to a practitioner who will carry out further treatment.

Lennart Brocki et al. [10] introduce the Chatbot solution 'SERENA' which uses the Seq2Seq Transformer Model to generate responses. SERENA uses Person-Centered Therapy (PCT), unlike most modern chatbots that use Cognitive Behavioral Therapy to treat their patients. T.A.H. Shaikh et al.[11] introduces an autonomous AI chatbot therapy for patients with insomnia. This chatbot was trained using a Feed Forward Neural Network with Chat Text Data that contains a text message and an appropriate tag. For example, the message 'hi' would be tagged as 'greeting.' In this application, NLP principles such as stemming, tokenization, and bag of words are employed, where Bag of Words is responsible for turning the input string into a vector containing integers.

2.2 Open problems in existing applications

In the above sub-section, various solution architectures, some with very few differences in their use of technologies, were evaluated. The proposed chatbot systems all aim to help users deal with their problems and anxieties by utilizing different technologies and methods of therapy. Reviewing the first case, "MAXX", its use of Dialogflow serves its purpose of responding accordingly to the user input, but it is too rigid a system to function as a therapy chatbot. Additionally, using Dialogflow also prevents the chatbot from maintaining context in a conversation, leading to poor user experience. MAXX's use of emotion recognition is by checking for positive and negative emotions [7] and nothing in between. This is a rather primitive solution, and therefore the system in place is not capable of handling and detecting more complicated emotions & problems, such as anxiety, Attention Deficit Hyperactive Disorder (ADHD), Bipolar Disorder (BPD), etc.

In another case, "PSYCHE CONVERSA", continued use of DialogFlow [8] implies that the chatbot may not be able to fully display empathy and understanding,

which are crucial in carrying out CBT. The chatbot may appear more robotic and impersonal, which might not be suitable for individuals seeking emotional support. Furthermore, since DialogFlow relies on using "Intents" to classify and understand user input, The output is produced from several predefined patterns or templates the system uses to identify the user query. While this approach is effective for carrying out many types of conversations, it can lead to the general- ization of input messages and responses. In the context of mental health support, this can lead to unproductive conversations that may lead nowhere. In chatbot solutions like "UNWIND", Data Collection of personal data through monitoring mobile behaviors, heart rates, sleeping patterns, etc. [9] could be perceived as a significant invasion of privacy. Users may feel uneasy about chatbots constantly collecting and processing their private information.

3 Methodology

In this section, the individual components, tools, and technologies involved in our project are discussed. This project aims to implement a chatbot system using the conversational abilities of an LLM, which can then act as an easily accessible solution for people seeking help with mental health issues.

The way to achieve this is by using a large language model called 'MythoMax' and a multi-class classification model trained on the Reddit Mental Health Dataset, which is available for open use on the internet. The classification model is trained to identify mental illness based on the user's messages and trained to classify illness into one of four categories: ADHD, Anxiety Disorder, Bipolar Disorder, and Depression.

Based on the classification performed on the user input, the chatbot is re-engineered to cater specifically to the issue the user is facing, which is performed by Zero-Shot learning (ZSL). ZSL is a machine learning paradigm where a model can recognize and classify objects or concepts it has never seen during training by making use of semantic relationships and attributes. The parts that make up our project are discussed in the following subsections:

3.1 Cognitive Behavioral Therapy

Cognitive-behavioral therapy (CBT) is a type of psychotherapeutic treatment that helps individuals to understand and work on their destructive and intrusive thought patterns that hinder self-improvement and also actively cause harm to the individual. [12] Psychotherapy (also known as talk therapy) is where a mental health professional and the patient work together collaboratively to identify the problem and develop a treatment strategy for tackling said problem.

Self-examination and introspection of oneself make up the foundation of CBT. Essentially, the patient is encouraged to 'be their own therapist'. This practice essentially takes the form of a question-and-answer session between the therapist and the patient, where the therapist probes the patient towards understanding and coming to terms with the problems they face on a more deeper level. The practice spans multiple sessions with plenty of time in between for the patient to introspect and form a better understanding of their thoughts and emotions.

The innate conversational nature of CBT allows us to use Large Language Models as a Conversational Agent for helping patients. Due to the way Cognitive Behavioral Therapy is structured, the LLM would have gained plenty of context regarding the specific problem expressed by the patient. Using the Chat History as Context, the

LLM can generate an appropriate response providing the patient with valuable insight regarding their mental health condition.

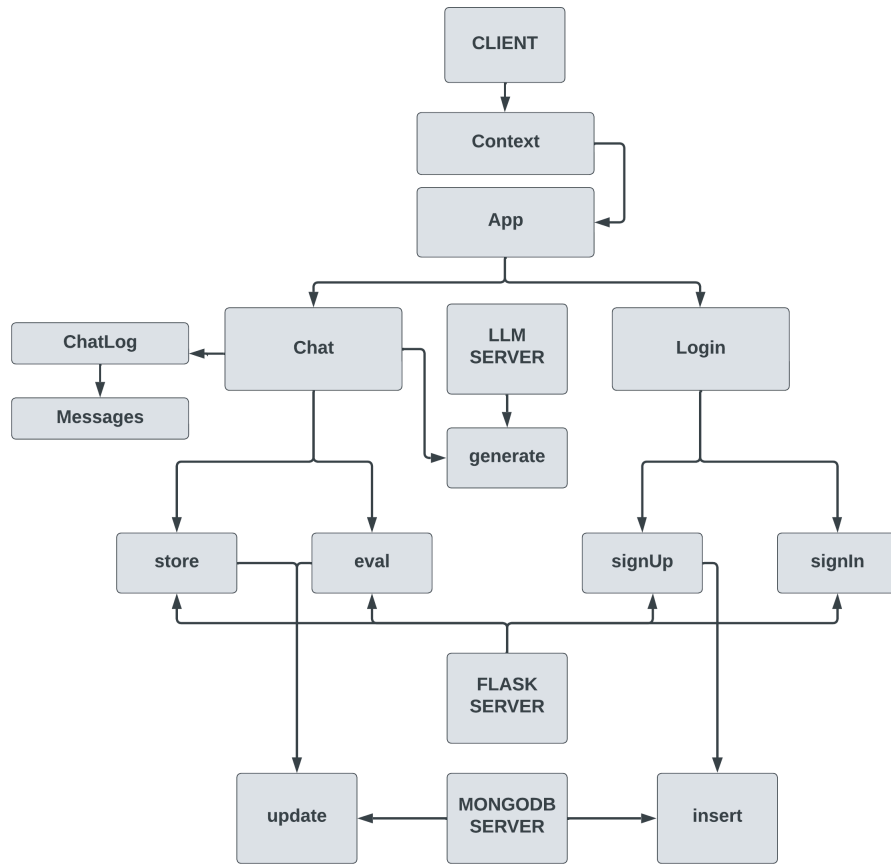


Fig. 1. System Architecture

3.2 Chatbot

A Chatbot is an Artificial System intended to emulate human conversation and interact with users via text or voice-based interfaces. With recent advancements in Natural Language Processing Techniques, they have been seeing popular usage across many different business sectors. Most well-known among them is the usage of ChatBots as Customer Service for websites ranging from E-Commerce to Software Documentation Sites. Chatbots are being used to provide Conversational Support Agents to Customers and have been very successful at it.

In this project, we aim to provide the User with an Artificial Therapist using a chatbot powered by the advanced conversational abilities of a Large Language Model (LLM).

We have implemented a Chat Interface that will facilitate conversation between the User and the Conversational Agent(LLM). The User's messages are sent to a Flask Server which hosts the Large Language Model. The Server will process the User's Message, and then send it to the LLM which will then generate an appropriate response, taking into consideration the context of the conversation. The User's messages are also passed through a classification model that will attempt to diagnose the mental illness that the user might be suffering from. The result from

the classifier will be fed into the LLM prompt thus enabling it to generate more relevant and helpful responses.

Storage and Authentication: MongoDB is a document database that makes it easy to store structured as well as unstructured data, such as text conversations. MongoDB stores data in a flexible Binary JSON format (BSON). The purpose of implementing a storage system is so that new users will be able to sign up and log in by retrieving their username & password stored in the schema and start having conversations with the chatbot. Returning users will find their previous chat history saved when they log-in. This eliminates the need to begin a new conversation with the chatbot each time a user wants to converse with it.

3.3 Large Language Models

An LLM is an NLP solution capable of generating very human-like responses and has a more generalized comprehension of text. We will use a Large Language Model as the main Conversational Agent for our Chatbot.

Chatbots implemented using DialogFlow use a state-based Artificial Intelligence Model which has a highly declarative nature. It provides a great medium for supervised Conversations but for more natural and personable discourse, Large Language Models are better suited due to their specialization towards NLU and NLG. In something as complicated a task as providing Therapy for a user suffering from mental health problems, A large language model's capabilities are considered superior in generating contextual and appropriate responses, as opposed to the relatively static response Generation provided by DialogFlow's State-Based AI.

The seminal paper on Attention Mechanism and how it can be used in Natural Language Processing was incredibly instrumental in the rise of Large Language Models [13]. Large Language Models are specialized versions of Transformers that are very good at two Natural Language Tasks: Natural Language Understanding and Natural Language Generation. This makes it suitable for being a conversational agent as it can both interpret and communicate. The potency of a Large Language Model varies with how many parameters it was trained. For example, most popular open-source models are trained with around 10-13 billion parameters, whereas GPT-3 was trained with 175 Billion Parameters [14].

For setting up our Large Language Model we are using Langchain as it provides methods to handle context and manage conversation memory. A behavior prompt is provided to the LLM as it is used to instruct the LLM on how it should interpret the input and context being provided to it. The LLM uses the prompt to inform its output generation.

A "Conversation Summary Buffer Memory" is used to summarize the conversation taking place between the User and the LLM. Memory is a very important aspect of response generation, as past interactions need to be considered for the contextual generation of responses. The conversation summary buffer works by making a call to the same LLM, and requesting it to summarize the conversation so far. But, due consideration must be given here, since using a summary buffer is a lossy process. However, as the length of the conversation increases, response generation will be significantly slowed depending on the LLM being used.

The Large Language Model is provided with relevant context depending on the problems that the User is facing. For example, The LLM receives appropriate context when interacting with a patient suffering from Anxiety. This is accom-

plished using a Classification Module that takes in user messages as input, and attempts to classify the mental health problem the user might be going through. The result of which will be used as additional context to inform the LLM.

As previously mentioned, we have opted to use Mythomax as our Large Language Model for this project. The reason why we chose to use Mythomax for our implementation was due to two key factors.

The first reason being that it is a popular model that was developed by merging together two other LLMs: MythoLogic-L2 and Huginn. Both of which themselves are further developed by merging several LLM's together. MythoLogic-L2 is composed of Hermes, Chronos and Airoboros, while Huginn is composed of several LLMs such as Hermes, Beluga, Airoboros, Chronos and Limarp. This merging aims to utilize each layer, composed of several tensors, each of which are responsible for specific functions. Using MythoLogic-L2's robust understanding as its input function, and Huginn's extensive writing capability as its output function has resulted in an LLM model that exceeds the very models which were used for its merging. The developer achieved this using an algorithm called Block Merge Gradient. This is a merging technique that takes hidden layers from two or more Large Language Models and combines them in the specified ratio. The gradient technique mentioned here alludes to the fact that as we go up in the layers the ratio of a model used might gradually increase or decrease. There are several other popular merging techniques such as SLERP, TIES and DARE.

SLERP (Spherical Linear Interpolation) is a linear interpolation method used to interpolate between two vectors in a spherical space. SLERP is better for interpolation because in linear interpolation, the magnitude of the interpolated vector can decrease in high-dimensional spaces. Although widely used, SLERP can only merge two vectors at a time. TIES is a merging technique Introduced by Yadav et al. [15], It has been designed with the capability to merge 2 or more task-specific models into a singular multitask model. It is also able to avoid redundancy in model parameters across multiple models and handle conflicts between parameter signs. Unlike SLERP, TIES can merge multiple models at once. DARE, introduced by Yu et al. [16] in their paper, is another merging technique which uses an approach similar to TIES with two main additions. DARE has the ability to stochastically reset fine-tuned weights to their base value.

Another significant factor influencing our choice of LLM is the availability of a quantized version of the MythoMax Model, known as MythoMax-L2-13B- GPTQ. GPTQ (Generative Pre-Trained Transformers Quantization) is a technique used to compress neural networks, specifically Large Language Models, for storage efficiency. A quantized model is capable of providing the same level of quality as a regular model at drastically less Memory requirement. This makes it more accessible with lower end hardware.

GPTQ is a Layerwise Quantization Algorithm and was introduced by Elias Frantar et al. [17] Although incredibly successful, one of the main shortcomings of LLMs is its enormous memory requirement. It can cost upwards of 100s of GBs to store and use a Large Language Model, and the steady rise in complexity of LLMs will further contribute to more and more memory requirements. GPTQ was developed as a solution to solve this problem. The working principle behind quantization of Large Language Models relies on the Layered structure of their Neural Networks. GPTQ is capable of quantizing the weights of an LLM in an isolated and layerwise manner. Parameters that are represented using floating point metrics will be converted into quantized integers using GPTQ. This is done in a way that there is minimal loss in

accuracy, despite loss of precision in digits. GPTQ allows for existing models to be quantized and deployed at a lower memory cost.

3.4 Classification Model

The classification model takes user messages as input and produces an output that will let us know the most probable mental illness that the user may be suffering from. We use the classification model to personalize the response generation to the User's specific problem, as quite a bit of nuance must go into the difference between providing support for a user with anxiety, versus helping a user with ADHD. Using the output received from the Classification model, the prompt provided to the LLM is modified to include instructions specific to a

particular mental health illness. This gives us more control over the response generation of the LLM and thus provides a personalized experience to the User.

Dataset The dataset used for training this classification model is the Reddit Mental Health Dataset. The dataset contains posts from several self-help subreddits. For this project, we have used posts from r/depression, r/anxiety, r/adhd, and r/bipolar subreddits. There are ~82,000 posts in this dataset.

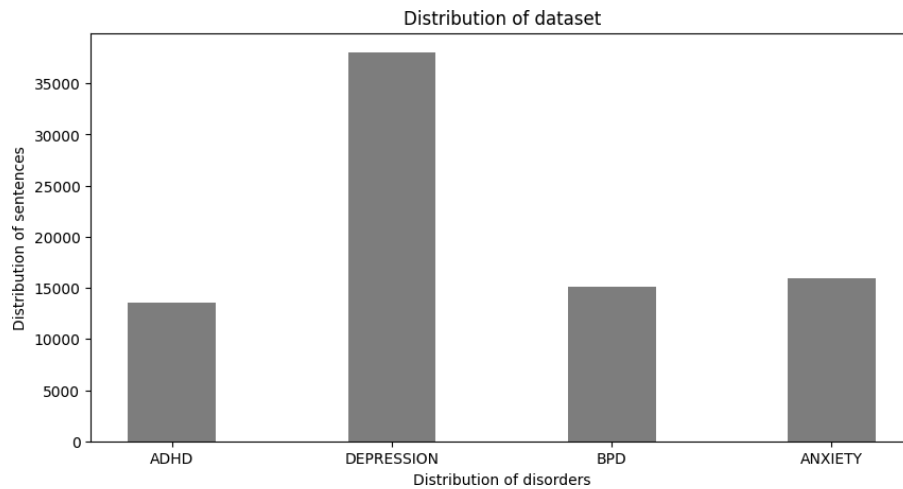


Fig. 2. Post Distribution

Pre-Processing As the dataset is raw, some pre-processing is required. We use the pandas python library to read the dataset into a data frame. As Reddit usernames start with 'u/', we remove any mention of usernames in the posts using simple regex. We index the subreddits column and convert the subreddit names from strings to integers. The posts are then split into sentences and 20 thousand sentences are sampled from each category.

The data needs to be tokenized before it can be used for training. The tokenizer splits the sentences into tokens and computes their input IDs and attention masks.

Training This section discusses how the pre-processed data was used to train the classification model. We use BERT to generate word embeddings for was our classification model as it is a popular and well-regarded choice for Natural Language Processing Tasks such as Sentiment Classification. Let us now discuss the structure of the classification model. The previously tokenized data are now

fed to the input layer which will then be used to compute the word embeddings using the base-case BERT model.

The word embeddings will be given as input to the intermediate layer which uses the relu activation function. The output from the intermediate layer is passed to the output layer, which uses the softmax activation function. The model was fine-tuned for 12 epochs with Adam as optimizer, at a learning rate of $5e-6$, and categorical cross entropy as the loss function.

4 Results

We will now proceed to discuss the results of the Project. Instead of using conventional NLP solutions like DialogFlow, we have opted to use Large Language Models to power our chatbot. We tested several LLMs for our purposes and generally the trend was that the higher the number of parameters of the LLM the better its responses were.

4.1 Comparison of LLMs

In our experimentation with different LLMs, it was found that not all LLMs are made to be used as chatbots.

Some LLMs are better suited for storytelling and writing articles than for having a conversation. Here are some of the LLMs we tested and their performance:

MythoMax performed best out of the aforementioned models in the context of a chatbot. It was able to retain context fairly well and was capable of holding long conversations therefore we chose to use it as the back end of our Chatbot. To classify the mental health of the user, we have used a BERT classification model. We used BERT model to generate word embeddings for our classification model, and trained our model with post data from Reddit.

Initially, we used the raw posts with minimal pre-processing to gauge the training time. The resultant training time was needlessly large and so we pre-processed the training data by first splitting the posts into sentences and then removing sentences that were below a certain word limit thus removing short and uninformative data. This allowed us to train our model much faster with minimal data loss. The final accuracy after fine-tuning came out to be 80.12%.

5 Future Work

Additional Features that could be implemented: An alternate Voice Call mode for patients who prefer to speak rather than text. A model can be trained to prescribe medication for the patients based on the classification of their mental health. However, this can lead to some ethical problems therefore a proper

Table 1. LLM comparison

Name	Base Model	Description	Inference
Picard	GPT-Neo	It is focused on Novel style writing. This model is designed for Novels of a variety of genres. 2.7B Parameters.	Meant for storytelling. Poor performance in Chatting.
Janeway	GPT-J	Created from Picard's dataset combined with a brand new collection of ebooks. This model has been trained on literature from various genres. 2.7 B Parameters.	Meant for storytelling. Avg performance in Chatting.
Pygmalion	GPT-J	A model built expressly for roleplay purposes. Very Popular. 6B Parameters.	RolePlay Model. Performs well in Chatting.
Mythomax	LLaMA	A model built by gradually layering other LLM models. Very Verbose. An all-round model with Great story generation and roleplaying capabilities. 13B parameters.	All Purpose Model. Performs exceptionally well in Chatting.

medical license and thorough testing are required before this feature can be implemented. A mental health report could be generated that would list the various statistics of the User's mental health. A discriminator can be trained to quantify the "humanness" of the responses generated by the LLM. This would require significant training and resources.

Conclusion

Mental Health Illnesses can be a very challenging and exhausting process for people to overcome. It is difficult to talk about and worse yet, even recognize those problems, especially for people from the lower end of the economic spectrum. Finding effective treatment for these conditions can be very hard even in the most developed countries.

To tackle this problem we have implemented an intelligent chatbot that can engage in meaningful conversations with people seeking help, making it easily accessible to anyone with internet access. We use an LLM to engage in natural conversation with the user while a Classification model figures out the illness that the user is dealing with, thus providing additional context to the LLM so that it can have more informed and nuanced conversations with the user.

We tested the quality of the conversation with several LLMs, and MythoMax(A specialized LLM meant for usage as a ChatBot) worked the best at holding long and natural conversations.

References

1. Murray, C. J. & Lopez, A. D. Measuring the global burden of disease. *New Engl. J. Med.* 369, 448-457 (2013). DOI: 10.1056/NEJMr1201534
2. Santomauro, D.F., Global prevalence and burden of depressive and anxiety disorders in 204 countries and territories in 2020 due to the COVID-19 pandemic, *Lancet* 398(1012), 1700-1712 (2021). DOI: 10.1016/S0140-6736(21)02143-7
3. Kamenov K, Twomey C, Cabello M, Prina AM, Ayuso-Mateos JL. The efficacy of psychotherapy, pharmacotherapy and their combination on functioning and quality of life in depression: a meta-analysis. *Psychol Med.* 2017 Feb;47(3):414-425. doi: 10.1017/S0033291716002774. Epub 2016 Oct 26. DOI: 10.1017/S0033291716002774
4. McHugh RK, Whitton SW, Peckham AD, Welge JA, Otto MW. Patient preference for psychological vs pharmacologic treatment of psychiatric disorders: a meta-analytic review. *J Clin Psychiatry.* 2013 Jun;74(6):595-602. doi: 10.4088/JCP.12r07757. PMID: 23842011; PMCID: PMC4156137. DOI: 10.4088/JCP.12r07757
5. Renn, B.N., Hoeft, T.J., Lee, H.S. et al. Preference for in-person psychotherapy versus digital psychotherapy options for depression: survey of adults in the U.S. *npj Digital Med* 2, 6 (2019). DOI: 10.1038/s41746-019-0077-1
6. N. Damij and S. Bhattacharya, "The Role of AI Chatbots in Mental Health Related Public Services in a (Post)Pandemic World: A Review and Future Research Agenda," 2022 IEEE Technology and Engineering Management Conference (TEMSCON EUROPE), Izmir, Turkey, 2022, pp. 152-159. DOI:10.1109/TEMSCONEUROPE54743.2022.9801962
7. V. Dhanasekar, Y. Preethi, V. S. P. J. I. R and B. P. M, "A Chatbot to promote Students Mental Health through Emotion Recognition," 2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2021, pp. 1412-1416. DOI:10.1109/ICIRCA51532.2021.9544838
8. S. Abu Noman Siddik, B. M. Arifuzzaman and A. Kalam, "Psyche Conversa - A Deep Learning Based Chatbot Framework to Detect Mental Health State," 2022 10th International Conference on Information and Communication Technology (ICoICT), Bandung, Indonesia, 2022, pp. 146-151. DOI:10.1109/ICoICT55009.2022.9914844
9. P. Kugapriya, M. Manohara, K. Ranganathan, D. Kanapathy, A. Gamage and A. Anzar, "UNWIND – A Mobile Application that Provides Emotional Support for Working Women," 2022 2nd Asian Conference on Innovation in Technology (ASIAN-CON), Ravet, India, 2022, pp. 1-7. DOI:10.1109/ASIANCON55314.2022.9909084
10. L. Brocki, G. C. Dyer, A. Gladka and N. C. Chung, "Deep Learning Mental Health Dialogue System," 2023 IEEE International Conference on Big Data and Smart Computing (BigComp), Jeju, Korea, Republic of, 2023, pp. 395-398
11. T. A. H. Shaikh and M. Mhetre, "Autonomous AI Chat Bot Therapy for Patient with Insomnia," 2022 IEEE 7th International Conference for Convergence in Technology (I2CT), Mumbai, India, 2022, pp. 1-5. DOI:10.1109/I2CT54291.2022.9825008
12. Nakao, M., Shiotsuki, K. & Sugaya, N. Cognitive-behavioral therapy for management of mental health and stress-related disorders: Recent advances in techniques and technologies. *BioPsychoSocial Med* 15, 16 (2021). DOI:https://doi.org/10.1186/s13030-021-00219-w
13. V. Ashish et al. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 6000–6010. Curran Associates Inc., Red Hook, NY, USA. (2017). DOI: https://doi.org/10.48550/arXiv.1706.03762
14. Tom B. Brown et al. Language models are few-shot learners. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, pp. 1877–1901. Curran Associates Inc., Red Hook, NY, USA (2020). DOI: https://doi.org/10.48550/arXiv.2005.14165
15. Yadav P, Tam D, Choshen L, Raffel C, Bansal M. Resolving interference when merging models. *arXiv preprint arXiv:2306.01708*. 2023 Jun 2.
16. Yu L, Yu B, Huang F, Li Y. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*. 2023 Nov 6.
17. Frantar E, Ashkboos S, Hoefler T, Alistarh D. Gptq: Accurate post-training quantization for gen

D. CERTIFICATE

