

MENTAL HEALTH CHATBOT



TODAY'S AGENDA:

This project implements a Retrieval-Augmented Generation (RAG) chatbot designed to answer user questions using information from real-world websites and documents in the domain of mental health.

- **Phase 1: Choose a Domain and Use Case**
- **Phase 2: Collect Data**
- **Phase 3: Preprocess and Chunk the Text**
- **Phase 4: Embed the Chunks**
- **Phase 5: Create a Vector Store**
- **Phase 6: Build the RAG System**
- **Phase 7: Build a Chat Interface**



PHASE 1: CHOOSE A DOMAIN AND USE CASE

- **Domain:** Mental Health
- **Use Case:** The chatbot answers questions related to mental health topics, such as anxiety, depression, and available treatments. This is valuable for raising awareness and providing accessible information to users.



PHASE 2: COLLECT DATA

- Description: Reliable websites and documents about mental health were scraped using tools like trafiletura.
- Output: A .csv file containing over 20,000 words of content.



PHASE 2: COLLECT DATA



```
import trafilatura
import csv

# List of URLs to scrape
urls = [
    "https://www.who.int/news-room/fact-sheets/detail/suicide",
    "https://www.who.int/news-room/questions-and-answers/item/stress",
    "https://www.who.int/news-room/questions-and-answers/item/suicide",
]

data = []
total_word_count = 0

for url in urls:
    downloaded = trafilatura.fetch_url(url)
    result = trafilatura.extract(downloaded)
    if result:
        word_count = len(result.split())
        total_word_count += word_count
        print(f"\n{url} - {word_count} words")
        data.append((url, result))

print(f"\n✓ Total Word Count: {total_word_count} words")

# Save to CSV
with open("data\mental_health_data.csv", "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(["source_url", "text"])
    writer.writerows(data)
```

PHASE 3: PREPROCESS AND CHUNK THE TEXT

- **Description:** The collected text was cleaned to remove HTML, punctuation, and irrelevant content. It was then split into smaller chunks of 200–500 words for efficient processing.
- **Output:** A cleaned and chunked dataset ready for embedding.



PHASE 3: PREPROCESS AND CHUNK THE TEXT



```
import pandas as pd
import re
import math

# Load your scraped dataset
df = pd.read_csv("data\mental_health_data.csv")

def clean_text(text):
    text = re.sub(r'\s+', ' ', text) # Normalize whitespace
    text = re.sub(r'[^\x00-\x7F]+', '', text) # Remove non-ASCII
    text = re.sub(r'\n+', ' ', text) # Remove newlines
    return text.strip()

def chunk_text(text, chunk_size=300):
    words = text.split()
    return [ ' '.join(words[i:i+chunk_size]) for i in range(0, len(words), chunk_size) ]

# Process each row
chunks = []
for idx, row in df.iterrows():
    cleaned = clean_text(row['text'])
    chunked = chunk_text(cleaned, chunk_size=300)
    for chunk in chunked:
        chunks.append((row['source_url'], chunk))

# Save chunked data
chunked_df = pd.DataFrame(chunks, columns=['source_url', 'text_chunk'])
chunked_df.to_csv("data\cleaned_chunked_data.csv", index=False)

print(f"✓ Saved {len(chunks)} chunks to 'cleaned_chunked_data.csv'")
```

PHASE 4: EMBED THE CHUNKS

- **Description:** A sentence embedding model (all-MiniLM-L6-v2) was used to convert text chunks into numerical vectors, enabling semantic understanding.
- **Output:** A set of vector representations for the text chunks.



PHASE 4: EMBED THE CHUNKS



```
import pandas as pd
import numpy as np
from sentence_transformers import SentenceTransformer

# Load data
df = pd.read_csv("data\cleaned_chunked_data.csv")
texts = df['text_chunk'].tolist()

# Load embedding model
model = SentenceTransformer('all-MiniLM-L6-v2')

# Embed all chunks
embeddings = model.encode(texts, show_progress_bar=True)

# Save embeddings and metadata
np.save("Embeddings\embeddings.npy", embeddings)
df.to_csv("data\chunk_metadata.csv", index=False)

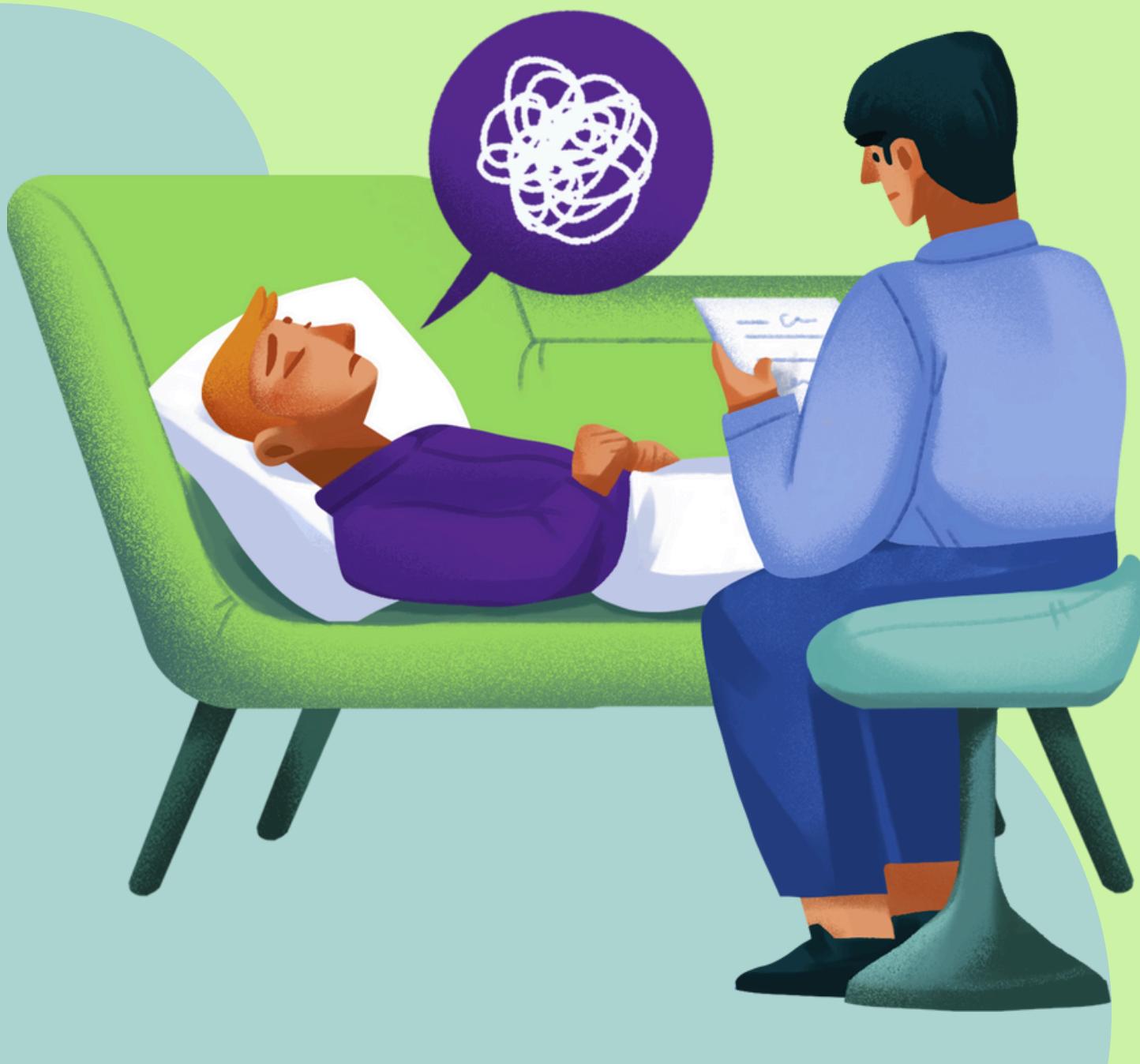
print(f"✓ Saved {len(embeddings)} embeddings to 'embeddings.npy'")
print("✓ Metadata saved to 'chunk_metadata.csv'")
```

PHASE 5: CREATE A VECTOR STORE

- **Description:** The vectors were stored in a FAISS index, allowing efficient retrieval of the most relevant chunks for a given query.
- **Output:** A searchable FAISS index containing the embedded content.



PHASE 5: CREATE A VECTOR STORE



```
import faiss
import numpy as np
import pandas as pd

# Load embeddings and metadata
embeddings = np.load("Embeddings\embeddings.npy")
df = pd.read_csv("data/chunk_metadata.csv")

# Create FAISS index
dimension = embeddings.shape[1]
index = faiss.IndexFlatL2(dimension)
index.add(embeddings)

# Save the index
faiss.write_index(index, "models/mental_health_index.faiss")
df.to_csv("data/chunk_metadata.csv", index=False)

print(f"✓ FAISS index built with {index.ntotal} vectors")
```

CLEAN & SAVE THE DATA



```
import pandas as pd

# Load the CSV file with flexible handling of inconsistent columns
df = pd.read_csv("data/chunk_metadata.csv", header=0, dtype=str, on_bad_lines='skip')

# Display the first few rows to inspect the data
print(df.head())

# Fill missing values with a placeholder (e.g., 'N/A')
df.fillna('N/A', inplace=True)

# Rename columns if necessary
df.rename(columns=lambda x: x.strip(), inplace=True)

# Save the cleaned DataFrame to a new CSV file
df.to_csv("data/cleaned_chunk_metadata.csv", index=False)
```

PHASE 6: BUILD THE RAG SYSTEM

- **Description:** The system accepts a user question, embeds it, retrieves the top 3-5 most relevant text chunks, and passes them along with the question to a language model to generate an answer.
- **Output:** A script that processes user questions and returns context-aware answers.



PHASE 6: BUILD THE RAG SYSTEM



```
import os
import sys
import faiss
import pandas as pd
from sentence_transformers import SentenceTransformer
from google.genai import Client
from google.genai.types import GenerateContentConfig

# — CONFIGURATION — #
FAISS_INDEX_PATH      = "models/mental_health_index.faiss"
METADATA_CSV_PATH     = "data/cleaned_chunk_metadata.csv"
EMBEDDING_MODEL_NAME = "all-MiniLM-L6-v2"
GEMINI_API_KEY        = "AIzaSyCMYuQQp4G6MQ6taw2xGpfHtVgtH5AEp1A"
GEMINI_MODEL_NAME    = "gemini-2.0-flash"
TOP_K                 = 5

# — INITIALIZATION — #
# 1. Load FAISS index
if not os.path.exists(FAISS_INDEX_PATH):
    print(f"ERROR: FAISS index not found at {FAISS_INDEX_PATH}", file=sys.stderr)
    sys.exit(1)
index = faiss.read_index(FAISS_INDEX_PATH)

# 2. Load metadata
if not os.path.exists(METADATA_CSV_PATH):
    print(f"ERROR: Metadata CSV not found at {METADATA_CSV_PATH}", file=sys.stderr)
    sys.exit(1)
df_meta = pd.read_csv(METADATA_CSV_PATH, dtype=str)
# Ensure the text column is correctly identified
text_column = "text_chunk"
if text_column not in df_meta.columns:
    print(f"ERROR: Expected column '{text_column}' not in metadata CSV", file=sys.stderr)
    print("Available columns:", df_meta.columns.tolist(), file=sys.stderr)
    sys.exit(1)
metadata_texts = df_meta[text_column].tolist()

# 3. Initialize embedder
embedder = SentenceTransformer(EMBEDDING_MODEL_NAME)

# 4. Initialize Gemini client
client = Client(api_key=GEMINI_API_KEY)
```

PHASE 6: BUILD THE RAG SYSTEM



```
# — HELPERS — #

def check_dimensions():
    """Verify that the FAISS index and embedding model agree on vector size."""
    expected_dim = index.d
    sample_vec = embedder.encode(["test"])
    actual_dim = sample_vec.shape[1]
    print(f"[DEBUG] FAISS index dimension: {expected_dim}; Embedder output dimension: {actual_dim}")
    if expected_dim != actual_dim:
        raise ValueError(f"Dimension mismatch: index expects {expected_dim}, but embedder gives {actual_dim}")

def retrieve_relevant_chunks(query: str, top_k: int = TOP_K):
    """Embed the query and return the top_k most relevant text chunks."""
    try:
        q_emb = embedder.encode([query])
    except Exception as e:
        print(f"[ERROR] Embedding failed for query '{query}': {e}", file=sys.stderr)
        return []

    distances, indices = index.search(q_emb, top_k)
    print(f"[DEBUG] Retrieved indices: {indices[0]}, distances: {distances[0]}")
    chunks = []
    for idx in indices[0]:
        if idx < 0 or idx >= len(metadata_texts):
            print(f"[WARNING] Index {idx} out of metadata range", file=sys.stderr)
            continue
        chunks.append(metadata_texts[idx])
    return chunks
```

PHASE 6: BUILD THE RAG SYSTEM



```
def generate_response(question: str, context: str) -> str:  
    """Call Gemini API, passing in the question and retrieved context."""  
    prompt = f"Question: {question}\nContext: {context}\nAnswer:"  
    print(f"[DEBUG] Sending prompt to Gemini:\n{prompt}\n")  
  
    try:  
        resp = client.models.generate_content(  
            model=GEMINI_MODEL_NAME,  
            contents=[prompt],  
            config=GenerateContentConfig(response_modalities=["Text"])  
    )  
    except Exception as e:  
        return f"ERROR: API call failed: {e}"  
  
    # Debug-print entire response object  
    print(f"[DEBUG] Raw Gemini response:\n{resp}\n")  
  
    # Attempt to extract generated text  
    text = None  
    if hasattr(resp, "generated_content") and resp.generated_content:  
        text = resp.generated_content[0].text  
    elif hasattr(resp, "generations") and resp.generations:  
        text = resp.generations[0].text  
    elif hasattr(resp, "candidates") and resp.candidates:  
        # Newer response format: candidates list  
        candidate = resp.candidates[0]  
        if hasattr(candidate, 'content') and candidate.content and candidate.content.parts:  
            text = candidate.content.parts[0].text  
  
    if not text:  
        return "⚠ Could not parse Gemini response format."  
    return text.strip()
```

PHASE 6: BUILD THE RAG SYSTEM



```
def chat_loop():
    """Main REPL for the RAG chatbot."""
    check_dimensions()
    print("Welcome to your RAG-powered chatbot! (type 'exit' to quit)")
    while True:
        question = input("You: ").strip()
        if question.lower() in {"exit", "quit", "bye"}:
            print("Goodbye!")
            break

        chunks = retrieve_relevant_chunks(question)
        if not chunks:
            print("Bot: ! I couldn't find relevant context to answer that.")
            continue

        context = "\n\n".join(chunks)
        answer = generate_response(question, context)
        print(f"Bot: {answer}\n")

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

PHASE 6: BUILD THE RAG SYSTEM



You: What are the main signs that someone might be experiencing depression?

Bot: Based on the provided text, the main signs that someone might be experiencing depression are:

- **Depressed mood:** Feeling sad, irritable, or empty.
- **Loss of interest or pleasure:** A significant decrease in enjoyment of activities that were once pleasurable.
- **Changes in appetite or weight:** Significant weight loss or gain when not dieting, or a decrease or increase in appetite nearly every day.
- **Disrupted sleep:** Insomnia (difficulty sleeping) or hypersomnia (sleeping too much).
- **Feeling very tired or low in energy:** Fatigue or loss of energy nearly every day.
- **Poor concentration:** Difficulty focusing, remembering details, or making decisions.
- **Feelings of excessive guilt or low self-worth:** Feeling worthless or having excessive or inappropriate guilt.
- **Hopelessness about the future:** A pessimistic outlook and a feeling that things will not improve.
- **Thoughts about dying or suicide:** Suicidal ideation, with or without a specific plan, or a suicide attempt.
- **Withdrawing from people and activities:** Isolating oneself from friends, family, and social events.
- **Feeling numb or like nothing matters:** A sense of emotional detachment or apathy.
- **Having unexplained aches and pains:** Physical symptoms that do not have a clear medical cause.
- **Severe mood swings:** Experiencing extreme shifts in mood that cause problems in relationships.

PHASE 7: BUILD A CHAT INTERFACE

- **Description:** A user-friendly interface was created using Streamlit, allowing users to interact with the chatbot by typing questions and receiving responses.
- **Output:** A fully functional chatbot interface.



PHASE 7: BUILD A CHAT INTERFACE



```
import os
import sys
import faiss
import pandas as pd
from sentence_transformers import SentenceTransformer
from google.genai import Client
from google.genai.types import GenerateContentConfig
from google.genai.errors import ServerError
import streamlit as st

# — STREAMLIT PAGE CONFIG — #
st.set_page_config(page_title="🧠 Mental Health Chatbot", layout="centered")

# — CONFIGURATION — #
FAISS_INDEX_PATH      = "models/mental_health_index.faiss"
METADATA_CSV_PATH     = "data/cleaned_chunk_metadata.csv"
EMBEDDING_MODEL_NAME = "all-MiniLM-L6-v2"
GEMINI_API_KEY        = "AIzaSyCMYuQQp4G6MQ6taw2xGpfHtVgtH5AEp1A"
GEMINI_MODEL_NAME    = "gemini-2.0-flash"
TOP_K                 = 5

# — CACHING INITIALIZATION — #
@st.cache_resource
def load_faiss_index(path):
    if not os.path.exists(path):
        st.error(f"FAISS index not found at {path}")
        st.stop()
    return faiss.read_index(path)

@st.cache_data
def load_metadata(path, text_col="text_chunk"):
    if not os.path.exists(path):
        st.error(f"Metadata CSV not found at {path}")
        st.stop()
    df = pd.read_csv(path, dtype=str)
    if text_col not in df.columns:
        st.error(f"Expected column '{text_col}' not in metadata CSV. Available: {df.columns.tolist()}")
        st.stop()
    return df[text_col].tolist()

@st.cache_resource
def init_embedder(model_name):
    return SentenceTransformer(model_name)

@st.cache_resource
def init_client(api_key):
    return Client(api_key=api_key)
```

PHASE 7: BUILD A CHAT INTERFACE



```
# Load resources
index = load_faiss_index(FAISS_INDEX_PATH)
metadata_texts = load_metadata(METADATA_CSV_PATH)
embedder = init_embedder(EMBEDDING_MODEL_NAME)
client = init_client(GEMINI_API_KEY)

# — HELPERS — #

def check_dimensions():
    expected = index.d
    actual = embedder.encode(["test"]).shape[1]
    if expected != actual:
        st.error(f"Dimension mismatch: index expects {expected}, embedder gives {actual}")
        st.stop()

@st.cache_data
def retrieve_relevant_chunks(query: str, top_k: int = TOP_K):
    q_emb = embedder.encode([query])
    distances, indices = index.search(q_emb, top_k)
    return [metadata_texts[idx] for idx in indices[0] if 0 <= idx < len(metadata_texts)]

@st.cache_data
def generate_response(question: str, context: str) -> str:
    prompt = f"Question: {question}\nContext:\n{context}\nAnswer:"
    try:
        resp = client.models.generate_content(
            model=GEMINI_MODEL_NAME,
            contents=[prompt],
            config=GenerateContentConfig(response_modalities=["Text"])
        )
    except ServerError:
        return "⚠️ The generation service is currently overloaded. Please try again in a moment."
    except Exception as e:
        return f"⚠️ Error calling generation service: {e}"
    # Extract text
    if hasattr(resp, "generated_content") and resp.generated_content:
        return resp.generated_content[0].text.strip()
    if hasattr(resp, "generations") and resp.generations:
        return resp.generations[0].text.strip()
    if hasattr(resp, "candidates") and resp.candidates:
        cand = resp.candidates[0]
        if cand.content and cand.content.parts:
            return cand.content.parts[0].text.strip()
    return "⚠️ Could not parse response."
```

PHASE 7: BUILD A CHAT INTERFACE



```
# — STREAMLIT CHAT UI — #
check_dimensions()

if "messages" not in st.session_state:
    st.session_state.messages = []

st.title("🧠 Mental Health Chatbot")
st.subheader("💬 Ask anything about mental health and get insightful answers!")

for msg in st.session_state.messages:
    if msg["role"] == "user":
        st.chat_message("user").write(msg['content'])
    else:
        # Display error messages differently
        if msg.get("content", "").startswith("⚠️"):
            st.chat_message("assistant").write(f"⚠️ {msg['content']}") # could use st.error
        else:
            st.chat_message("assistant").write(msg['content'])

# Input
if prompt := st.chat_input("💡 Type your question here..."):
    # Display user message
    st.session_state.messages.append({"role": "user", "content": prompt})
    st.chat_message("user").write(prompt)

    # Retrieve context and generate
    chunks = retrieve_relevant_chunks(prompt)
    context = "\n\n".join(chunks)

    # Call generation
    response = generate_response(prompt, context)

    # Store and display
    st.session_state.messages.append({"role": "assistant", "content": response})
    st.chat_message("assistant").write(response)
```

PHASE 7: BUILD A CHAT INTERFACE



Mental Health Chatbot

Ask anything about mental health and get insightful answers!



Tell me about bipolar disorder in 1 paragraph



Bipolar disorder is a mental health condition affecting approximately 40 million people worldwide, characterized by alternating periods of depressive episodes and manic or hypomanic episodes. Depressive episodes involve depressed mood or loss of interest in activities, while manic episodes include elevated mood, increased energy, racing thoughts, and impulsive behaviors, and hypomanic episodes are similar but less intense. Bipolar disorder can significantly impair daily functioning, increase the risk of suicide, and often leads to misdiagnosis or lack of treatment, however, effective treatments exist that combine medicines and psychosocial interventions to help manage symptoms.



Type your question here...



KEY FEATURES

- Custom RAG Implementation: Built without relying on external frameworks like LangChain.
- Domain-Specific Knowledge: Focused on mental health to provide accurate and helpful responses.
- Efficient Retrieval: Uses FAISS for fast and accurate vector-based search.
- User-Friendly Interface: Streamlit-based interface for seamless interaction.



FUTURE ENHANCEMENTS

- Expand the dataset to include more diverse mental health topics.
- Integrate additional language models for improved response quality.
- Add multilingual support to reach a broader audience.



TEST REPORT

- The project includes a `Test_Report.md` file, which contains a detailed test report with sample questions and the chatbot's responses. This report demonstrates the chatbot's capabilities and provides insights into its functionality. It serves as a reference for understanding how the chatbot handles various queries related to mental health.



SAMPLE QUESTIONS AND CHATBOT RESPONSES



Question 1

You: What are the key components of community-based mental health care, and why are they important?

Bot: Based on the provided text, the key components of community-based mental health care are:

1. **Integrated Mental Health Services in General Healthcare:** Mental health services are integrated within general hospitals and primary healthcare settings. This often involves task-sharing, where non-specialist care providers in primary care settings are trained to deliver basic mental health services.
2. **Community Mental Health Services:** This includes a range of services delivered within the community, such as:
 - **Community mental health centers and teams:** Provide specialized mental health services in local communities.
 - **Psychosocial rehabilitation:** Focuses on helping individuals with mental health conditions develop the skills and supports necessary to live fulfilling lives in the community.
 - **Peer support services:** Connect individuals with shared experiences to provide emotional support, practical assistance, and advocacy.
 - **Supported living services:** Offer assistance with housing, daily living skills, and other supports to enable individuals to live independently.
3. **Mental Health Care in Social Services and Non-Health Settings:** This involves integrating mental health services into settings such as:
 - **Child protection services:** Addresses the mental health needs of children and families involved in child welfare systems.
 - **School health services:** Provides mental health support to students within schools.
 - **Prisons:** Offers mental health care to incarcerated individuals.
4. **Innovative ways to diversify and scale up care:** Using non-specialist psychological counselling or digital self-help, for example, to address the mental health treatment gap.

Importance of Community-Based Mental Health Care:

- **Accessibility:** Community-based care is more accessible to individuals than institutional care, as services are located within their communities.
- **Acceptability:** Individuals may be more willing to seek mental health care when it is provided in a familiar and less stigmatizing setting.
- **Prevention of Human Rights Violations:** Community-based care reduces reliance on large institutions, which are often associated with human rights violations such as involuntary admission, seclusion, and restraint.
- **Improved Recovery Outcomes:** Community-based care is associated with better recovery outcomes for individuals with mental health conditions, as it focuses on promoting independence, social inclusion, and participation in meaningful activities.

In summary, community-based mental health care is important because it provides a more accessible, acceptable, and humane approach to mental health care that promotes recovery and protects the rights of individuals with mental health conditions.

SAMPLE QUESTIONS AND CHATBOT RESPONSES



Question 2

You: Can you explain five strategies to promote social support and reduce loneliness among the elderly?

Bot: Based on the provided text, here are five strategies to promote social support and reduce loneliness among the elderly:

- 1. Befriending Initiatives:** These programs pair volunteers with older adults for regular visits or phone calls. This provides companionship, reduces isolation, and can improve mental well-being.
- 2. Community and Support Groups:** Creating or supporting community centers and support groups where older adults can gather, share experiences, and participate in activities together. These groups can cater to specific interests or needs, fostering a sense of belonging and shared identity.
- 3. Leisure and Education Services:** Offering or promoting access to leisure activities (e.g., arts and crafts, book clubs, games) and educational opportunities (e.g., workshops, lectures, classes). These provide structured opportunities for social interaction, cognitive stimulation, and a sense of purpose.
- 4. Volunteering Programmes:** Encouraging and facilitating older adults' participation in volunteer activities. Volunteering provides a sense of purpose, connection to the community, and opportunities for social interaction, thereby combating loneliness and enhancing mental well-being.
- 5. Intergenerational Activities:** Facilitating interactions between older adults and younger generations. This can be achieved through joint activities like mentoring programs, storytelling sessions, or shared community projects. These activities combat ageism, foster mutual understanding, and provide meaningful social connections for both age groups.

SAMPLE QUESTIONS AND CHATBOT RESPONSES



Question 3

You: How does the WHO recommend integrating mental health services into primary health care?

Bot: The WHO recommends integrating mental health services into primary health care through the following:

- **Integration into General Healthcare:** Integrate mental health services into general healthcare, typically in general hospitals and through task-sharing with non-specialist care providers in primary health care.
- **Community-Based Care:** Prioritize community-based mental health care, which is more accessible and acceptable than institutional care. This involves establishing a network of interrelated services:
 - Community mental health centers and teams.
 - Psychosocial rehabilitation services.
 - Peer support services.
 - Supported living services.
- **Task-Sharing:** Training and capacity building for non-specialized healthcare providers (doctors, nurses, and other health workers in non-specialist health settings) to assess and manage mental disorders through programs like the Mental Health Gap Action Programme (mhGAP).
- **Innovative Approaches:** Diversifying and scaling up care for common mental health conditions like depression and anxiety through non-specialist psychological counseling or digital self-help.

SAMPLE QUESTIONS AND CHATBOT RESPONSES



Question 4

You: What are common early warning signs of depression in older adults?

Bot: Based on the provided documents, common early warning signs of depression in older adults include:

- **Changes in eating or sleeping habits:** This could manifest as disrupted sleep (insomnia or oversleeping) or changes in appetite leading to weight loss or gain.
- **Withdrawing from people and activities they enjoy:** Social isolation and loss of interest in hobbies are common indicators.
- **Having low or no energy:** Feeling very tired or low in energy is a hallmark symptom.
- **Feeling numb or like nothing matters:** This reflects a loss of interest or pleasure in life.
- **Feeling helpless or hopeless:** Hopelessness about the future.
- **Excessive guilt or low self-worth.**
- **Thoughts about dying or suicide.**
- **Experiencing unexplained aches and pains.**
- **Feeling unusually confused, forgetful, angry, upset, worried, or scared.**
- **Dire living conditions, poor physical health, or lack of access to quality support and services.**
- **Experiencing abuse.**
- **Cumulative impacts of earlier life experiences and specific stressors related to aging.**
- **Exposure to adversity, significant loss in intrinsic capacity and a decline in functional ability.**
- **Bereavement, or a drop in income or reduced sense of purpose with retirement.**
- **Ageism.**
- **Social isolation and loneliness.**
- **Being a carer of spouses with chronic health conditions.**

SAMPLE QUESTIONS AND CHATBOT RESPONSES



Question 5

You: What are the main signs that someone might be experiencing depression?

Bot: Based on the provided text, the main signs that someone might be experiencing depression are:

- **Depressed mood:** Feeling sad, irritable, or empty.
- **Loss of interest or pleasure:** A significant decrease in enjoyment of activities that were once pleasurable.
- **Changes in appetite or weight:** Significant weight loss or gain when not dieting, or a decrease or increase in appetite nearly every day.
- **Disrupted sleep:** Insomnia (difficulty sleeping) or hypersomnia (sleeping too much).
- **Feeling very tired or low in energy:** Fatigue or loss of energy nearly every day.
- **Poor concentration:** Difficulty focusing, remembering details, or making decisions.
- **Feelings of excessive guilt or low self-worth:** Feeling worthless or having excessive or inappropriate guilt.
- **Hopelessness about the future:** A pessimistic outlook and a feeling that things will not improve.
- **Thoughts about dying or suicide:** Suicidal ideation, with or without a specific plan, or a suicide attempt.
- **Withdrawing from people and activities:** Isolating oneself from friends, family, and social events.
- **Feeling numb or like nothing matters:** A sense of emotional detachment or apathy.
- **Having unexplained aches and pains:** Physical symptoms that do not have a clear medical cause.
- **Severe mood swings:** Experiencing extreme shifts in mood that cause problems in relationships.

TEAM MEMBERS

Fares Hazem (ID: 20221443356)

Ali Ashraf (ID: 2103106)

Ahmed Dawood (ID: 20221454408)

Ahmed Yousri (ID: 2103108)

Khaled Mansour (ID: 20221320444)



ANY QUESTIONS?

