

MGS 636 Applied AI for Managers

Group Project Report: LifeLens AI

Name	UB Number
Priyal Panda	50583363
Yashraj Singh Kollapuram	50580332
Vaishnavi Varanasi	50590668
Benedict William Raj	50575024
Ratan Kumar Damodar Shetty	50502535

1. Executive Summary

In today's high-paced world, personal growth and emotional resilience often take a back seat. Many individuals struggle with sustaining motivation, aligning with long-term goals, and maintaining emotional well-being. Traditional coaching methods, while effective, are often costly, time-bound, or unavailable to many. This project—LifeLens AI introduces a local AI Life Coach with long-term memory that serves as a personalized, private, and empathetic guide. Using advanced language models and memory-retention mechanisms, the coach remembers users' goals, emotional states, and habits, facilitating consistent support and self-reflection over time. This report details the problem space, the innovative AI solution built, its technical framework, and its applicability in real-world contexts.

2. Introduction

The increasing prevalence of stress, mental health concerns, and performance anxiety calls for accessible tools that provide personalized guidance. Artificial Intelligence (AI), particularly generative models, opens up avenues to address these needs. This project aimed to create an AI life coach that mimics the behaviour of a human coach while retaining user-specific knowledge over time. The AI system is designed to work offline, protect user data, and simulate human-like therapeutic conversations, offering a unique blend of technology and empathy.

3. Problem Statement

While life coaching has shown benefits in goal-setting and emotional support, accessibility and privacy remain barriers. Individuals lack:

- Affordable and always-available coaching solutions.
- Long-term memory continuity from session to session.
- Non-judgmental, data-secure platforms to reflect and grow.

This project addresses these pain points by designing a local AI-powered life coach capable of recalling prior interactions, detecting emotional states, and adapting responses accordingly.

4. Project Objectives

The AI Life Coach project had the following goals:

- Develop a generative AI chatbot that interacts via text and audio.
- Embed long-term memory using ChromaDB and SQLite to store user interactions.
- Build a user-friendly frontend interface for conversations and habit tracking.
- Ensure offline functionality and data privacy.
- Integrate emotion detection and stop-word alerts for ethical responsiveness.

5. Use Case Scenarios

The solution has the potential to be applied across various domains:

- **Corporate Wellness Programs:** Continuous check-ins for employees.
Academic Advising: Students track personal and academic goals.
- **Human Resources:** Enhanced employee engagement and retention.
- **Entrepreneurial Coaching:** Personalized guidance for founders.
- **Mental Health Support:** Basic emotional assistance without clinical claims.

6. Research and Rationale

Research revealed that while many mental health and coaching apps exist, few offer long-term memory, full offline capabilities, or multimodal interaction. The design rationale centered on three pillars:

1. **Empathy Simulation:** Using 6,000+ QA pairs from real therapy data.
2. **Privacy:** Running everything locally using Ollama, SQLite, and ChromaDB. This can be directly hosted on the Client's server or Cloud
3. **Continuity:** Storing sessions by user and timestamp for future reference.

7. Solution Overview & Features

- Text and voice input.
- Memory recall of past conversations.
- Motivational support based on detected emotions.
- Trigger-based response switching for sensitive topics.

Multimodal Interaction: Supports both text and voice inputs.

1. Long-Term Memory: Stores and retrieves past user interactions to maintain continuity.
2. Emotion Detection: Analyzes text inputs to sense user emotional state and frame responses accordingly.
3. Sensitive Topic Handling: Detects keywords related to mental health emergencies and pauses automated responses, prompting human intervention.
4. Goal and Habit Tracking: Helps users define, track, and reflect on personal and professional goals.

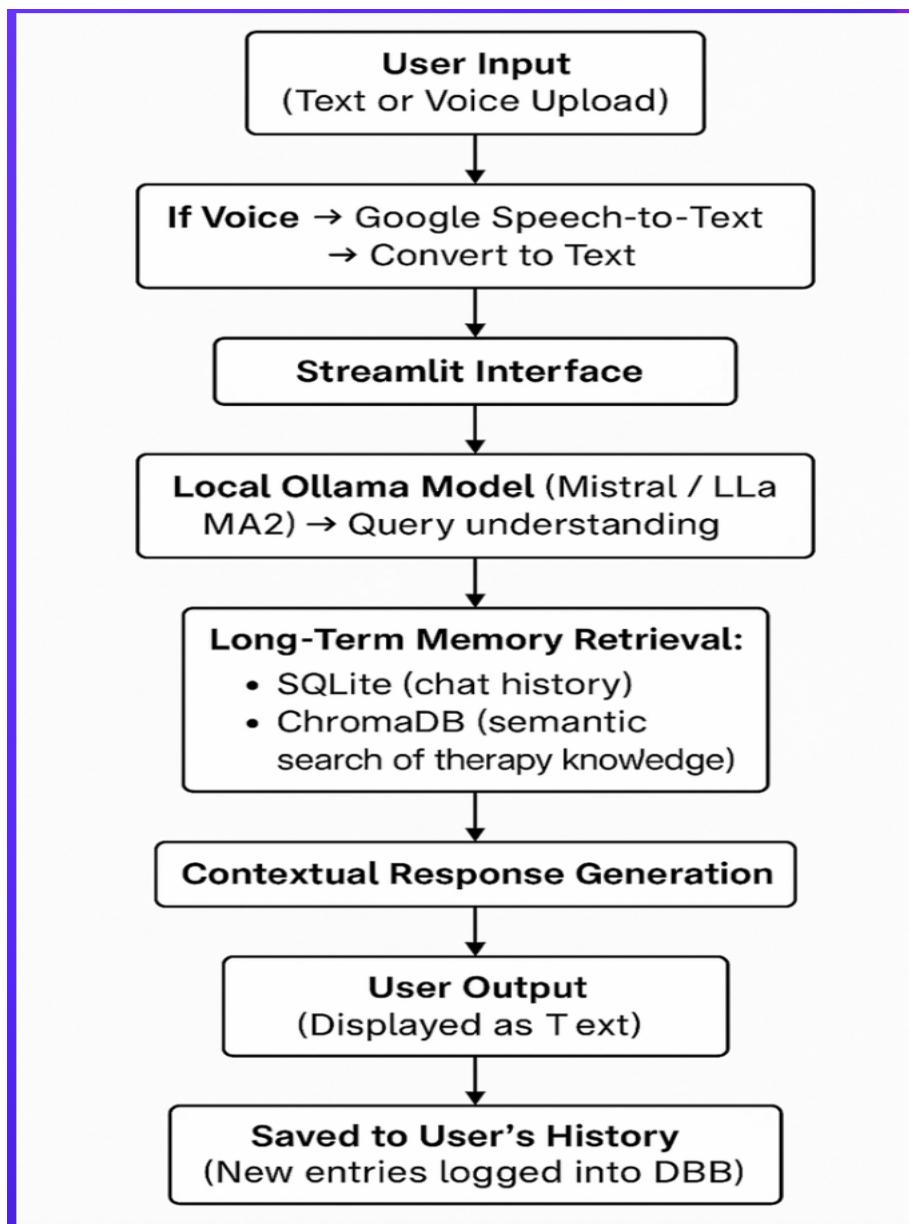
Vision: A coach that remembers your goals, setbacks, and small victories—guiding you with consistency and understanding.

8. Technical Architecture

- **Backend:**
 - **Ollama**: Used for running Mistral or LLaMA2 models locally.
 - **ChromaDB**: For semantic search of past conversations.
 - **SQLite**: For structured storage of chats, goals, and emotions.
- **Frontend:**
 - **Streamlit**: User-friendly UI for interaction and visualization.
- **Interaction Flow:**
 1. User inputs (text/voice).
 2. Voice-to-text conversion (via Google Speech-to-Text).
 3. Memory retrieval (ChromaDB + SQLite).
 4. LLM-generated empathetic response.
 5. UI displays reply + stores conversation.
- **SQLite**: Structured database for storing user chats, emotional states, goals, and session history.
- **ChromaDB**: Vector database for storing and retrieving semantically meaningful conversation segments (e.g., therapy Q&A examples).
- **Voice Handling**: Google Speech-to-Text API: Converts uploaded voice recordings into text. Originally attempted Bark (LLM for voice generation), but switched due to high RAM requirements (~20–25 GB).

9. Development Methodology

Workflow:



Setup- Python virtual environments, Ollama, and ChromaDB.

Data Preparation:

- Cleaning therapist conversations.
- Creating stop-word list for ethical detection.

Embedding & Memory: Stored conversations as vector embeddings in ChromaDB.

Interface: Chat interface built in Streamlit.

Testing:

- Tested with multiple users.
- Monitored memory recall and emotional tone accuracy.

Memory Strategy:

To stay within 8K context window of models like Mistral:

- Retrieve Top 5 past therapist messages
- Retrieve Top 5 user interactions
- Keep system prompt short and relevant
- This ensures faster inference, reduced RAM usage, and contextually relevant replies.

10. Proof of Concept

- **Login & Session Handling:**
 - Each user gets a unique folder with timestamped chat logs.
- **Conversation Memory:**
 - If a user expresses demotivation, AI links it to previous mentions.
- **Voice Input:**
 - Users upload audio; AI converts and responds empathetically.
- **Sensitive Topics:**
 - On detecting terms like "suicide" or "burnout," AI halts and recommends real help.

11. Challenges & Mitigation

Challenge	Mitigation
Bark LLM for voice too memory-heavy	Switched to Google Speech-to-Text
Token limit optimization	Capped retrieval to top 5 user + 5 therapist messages
Ethical responses for serious issues	Added stop-word detection and human fallback

12. Description of the code- (Whole code was done on a Mac book.)

Initial setup & Configuration:



LifeLensAI.py - /Users/vaishu/Documents/Applied_AI/LifeLensAI.py (3.11.4)

```
import os
import sqlite3
import datetime
import streamlit as st
import speech_recognition as sr
import requests
import random
import chromadb
import json
from sentence_transformers import SentenceTransformer
import pyttsx3
import tempfile
```

This import covers:

Web Interface: Streamlit

Voice input handling: speech_recognition

Voice output: pyttsx3

Memory Storage: sqlite3, chromadb

Model Embeddings: SentenceTransformer

JSON data parsing- json

Temp file/audio handling: tempfile

Configuration Variables:

```
# CONFIG
OLLAMA_URL = "http://localhost:11434/api/generate"
OLLAMA_MODEL = "mistral"
DB_PATH = "user_memory.db"
THERAPIST_COLLECTION = "therapist-knowledgebase"
USER_COLLECTION = "user-conversations"
STOPWORDS_FILE = "/Users/vaishu/Documents/Applied_AI/datasets/ai_life_coach_stop
FINAL_DATASET_FILE = "/Users/vaishu/Documents/Applied_AI/datasets/final_combined
```

These variables centralize settings for reuse across the app:

OLLAMA_URL / OLLAMA_MODEL: Config for querying your local LLM.

DB_PATH: SQLite DB storing user data.

STOPWORDS_FILE: File with words triggering ethical interventions.

FINAL_DATASET_FILE: Cleaned therapist Q&A dataset used for embedding into ChromaDB.

ChromaDB Initialization & Embedding Setup:

```
# Initialize ChromaDB and Embedder
chroma_client = chromadb.PersistentClient(path=".chroma_storage")
embedder = SentenceTransformer('all-MiniLM-L6-v2')
therapist_collection = chroma_client.get_or_create_collection(name=THERAPIST_COL
user_collection = chroma_client.get_or_create_collection(name=USER_COLLECTION)
```

- Initializes ChromaDB to store and retrieve vectorized data.
- Uses the MiniLM model to convert text into embeddings.
- Retrieves (or creates if empty) two ChromaDB collections:
- therapist_collection: static Q&A embeddings from trained dataset.
- user_collection: dynamic, user-specific memory.

Embedding the Dataset into Code:

```
# Embed final_combined_dataset.json if therapist_collection is empty
if len(therapist_collection.get()["ids"]) == 0:
    with open(FINAL_DATASET_FILE, "r") as f:
        data = json.load(f)
    for idx, item in enumerate(data):
        question = item.get("question", "")
        answer = item.get("answer", "")
        if question and answer:
            combined_text = f"Q: {question}\nA: {answer}"
            therapist_collection.add(
                documents=[combined_text],
                metadata=[{"source": "therapist_dataset"}],
                ids=[f"therapist_{idx}"]
            )
```

- On first run, this checks if the therapist_collection is empty.
If yes, it:
 - Loads the JSON dataset.
 - Combines Q&A pairs into a single string.
 - Embeds and stores them in ChromaDB with metadata.

Load Stop Words:

```
# Setup Database
conn = sqlite3.connect(DB_PATH)
c = conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS memories (user TEXT, datetime TEXT, text
conn.commit()
```

- Loads a list of sensitive keywords (like anxiety, depression, suicide) to ensure responsible handling. Used later to trigger ethical response behavior (e.g., halting AI and referring to a real coach).

Save Conversation:

```
# Save conversation
def save_conversation(user, user_input, ai_response):
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    folder = f"conversations/{user}"
    os.makedirs(folder, exist_ok=True)
    file_path = os.path.join(folder, "chat_log.txt")
    with open(file_path, "a") as f:
        f.write(f"[{timestamp}] You: {user_input}\n")
        f.write(f"[{timestamp}] Coach: {ai_response}\n\n")
    c.execute("INSERT INTO memories VALUES (?, ?, ?)", (user, timestamp, user_input))
    conn.commit()
    user_collection.add(documents=[user_input], metadatas=[{"user": user}], ids=
```

- Appends the chat history to a text log file in a folder specific to the user. Also logs the same in the memories table of SQLite and adds the user input to user_collection in ChromaDB for future semantic retrieval. This function is central to memory persistence.

Retrieve Context Summary:

```
# Get user memory summary
def retrieve_context(user):
    c.execute("SELECT datetime, text FROM memories WHERE user=? ORDER BY datetime ASC", (user,))
    rows = c.fetchall()

    if not rows or len(rows) <= 1:
        return ""

    full_conversation = "\n".join([f"[{timestamp}] {text}" for timestamp, text in rows])

    summary_prompt = f"""
You are a helpful AI assistant summarizer.

Here is the full conversation history:
{full_conversation}

Summarize this conversation into 5-7 lines, focusing on main topics discussed, emotions shared, and overall progress made.
Be concise but empathetic.
"""

|
```

- Pulls all past interactions from the memories table for the given user. Concatenates them and prepares a prompt to summarize the full conversation so far. The generated summary can then be included in context when responding.

Summarizing User Memory:

```
| response = requests.post(
    OLLAMA_URL,
    json={"model": OLLAMA_MODEL, "prompt": summary_prompt, "stream": False}
)

return response.json().get("response", "").strip()
```

- Sends the summary_prompt to the local Ollama instance running the Mistral model. It gets back a summarized version of the user's conversation history, which will be used to provide context in future AI replies. Its purpose is to feed this summary along with the next user prompt to generate personalized, memory-aware responses.

Retrieving Therapist Knowledge:

```
# Get therapist knowledge
def get_therapist_knowledge(user_input):
    results = therapist_collection.query(query_texts=[user_input], n_results=5)
    therapist_contexts = results.get("documents", [[]])[0]
    return "\n".join(therapist_contexts)
```

- Performs a semantic search on the therapist_collection using the user input. This retrieves the top 5 similar Q&A pairs. It combines them into a single string which will be used to support AI's reply. This creates a mini-knowledge base for therapeutic coaching.

Final Response Generation Function:

```
# Generate final response
def generate_response(prompt, user_context, therapist_context, user_name, first_response_done):
    # Custom greeting based on user history
    if not first_response_done:
        if user_context.strip() == "":
            greeting = f"Hey, looks like we are meeting for the first time--how have you been?\nWelcome to LifeLens AI!\n\n"
        else:
            greeting = f"Hi {user_name}, welcome to LifeLens AI.\n\n"
    else:
        greeting = ""

    # Tell the model to avoid repeating greeting
    skip_greeting_instruction = (
        "Important: You do NOT need to greet the user (e.g., 'Hi there', 'Hello'). I've already done that.\n\n"
        "if not first_response_done else\n"
        "Important: DO NOT use greetings like 'Hi', 'Hello', 'Hi there', or similar. Start directly with emotional reflection or helpful response.\n\n"
    )

    # Add context reference if it exists, otherwise explicitly state not to fabricate
    if user_context.strip() != "":
        context_instruction = f"""
Important Rule:
- The user has chatted with you before.
- Begin your response by naturally referencing the following summary.
- DO NOT fabricate or guess what was said - only use the summary below.

Conversation Summary:
{user_context}
"""
    else:
        context_instruction = """
Important Rule:
- This is the user's first message.
- DO NOT say anything that assumes a previous chat or session.
- Focus on responding to this message alone, naturally and empathetically.
"""

    return greeting + context_instruction
```

On first interaction, the model introduces itself in a friendly way. If it's a return session, it skips the intro and just welcomes the user. `first_response_done` ensures greetings are not repeated. Prevents the model from repeating the greeting. Reinforces conversational continuity and makes interaction feel more natural.

User Context Instruction:

```
# Add context reference if it exists, otherwise explicitly state not to fabricate
if user_context.strip() != "":
    context_instruction = f"""
Important Rule:
- The user has chatted with you before.
- Begin your response by naturally referencing the following summary.
- DO NOT fabricate or guess what was said - only use the summary below.

Conversation Summary:
{user_context}
"""
else:
    context_instruction = """
Important Rule:
- This is the user's first message.
- DO NOT say anything that assumes a previous chat or session.
- Focus on responding to this message alone, naturally and empathetically.
"""

return context_instruction
```

This block instructs the model how to behave based on whether it's the user's first session or a follow-up. If the user has past chats, the model is asked to reference the

memory summary. If it's the first time, the model is told to treat it like a brand-new session.

Final Prompt Construction:

```
# Final system prompt construction
system_prompt = f"""
You are an AI Therapist and Life Coach with long-term memory, focused on emotional intelligence, human-like warmth, and natural conversation.

{skip_greeting_instruction}
Behavior Guidelines:
- Never repeat the user's name after the first message.
- Avoid generic greetings like 'Hi there', 'Hello'.
- Refer to real previous memory only if provided.
- Prioritize warmth, empathy, and helpful tone.
- Keep replies concise (3-5 lines).

Therapist Knowledge:
{therapist_context}
{context_instruction}

User's New Message:
{prompt}

Now respond naturally and directly.
"""

response = requests.post(
    OLLAMA_URL,
    json={"model": OLLAMA_MODEL, "prompt": system_prompt, "stream": False}
)
return greeting + response.json().get("response", "I'm here for you whenever you're ready to chat.").strip()
```

This master prompt pulls together: Greeting behavior instructions, Therapist context (retrieved from ChromaDB), Memory summary (if available), The user's latest message, It guides the AI to respond in a non-repetitive, empathetic, and concise way.

Generate Response via Ollama:

```
response = requests.post(
    OLLAMA_URL,
    json={"model": OLLAMA_MODEL, "prompt": system_prompt, "stream": False}
)
return greeting + response.json().get("response", "I'm here for you whenever you're ready to chat.").strip()
```

Sends the system prompt to your local Ollama LLM. The response is extracted and returned along with the initial greeting. A fallback message is included in case the model doesn't return a reply.

Voice Transcription Function:

```
# Voice transcription
def transcribe_audio(audio_file_path):
    recognizer = sr.Recognizer()
    with sr.AudioFile(audio_file_path) as source:
        audio = recognizer.record(source)
    try:
        return recognizer.recognize_google(audio)
    except sr.UnknownValueError:
        return "Sorry, I couldn't understand your audio."
```

The function is being defined here and we'll likely see how audio files are transcribed into text using speech_recognition. If transcription fails (e.g., unclear audio), it returns a fallback message.

Text-to-speech Voice Output:

```
# Text-to-speech
def speak_response(text_response):
    engine = pyttsx3.init()
    engine.say(text_response)
    engine.runAndWait()
```

Uses pyttsx3, a Python-based offline TTS engine. Converts the AI response to spoken audio. Though voice output was dropped in the final PoC (due to memory issues with Bark), this fallback allows simple offline audio playback.

Bark Code: The code is working perfectly fine, however as it is memory intense for demo purposes we chose to have a backup plan using Google Text to Speech converter.

```
from bark import SAMPLE_RATE, generate_audio
import scipy
```

```

    return "Sorry, I couldn't understand your audio.

def speak_response_bark(text_response):
    audio_array = generate_audio(text_response, history_prompt="v2/en_speaker_1")
    temp_audio_file = tempfile.NamedTemporaryFile(delete=False, suffix=".wav")
    scipy.io.wavfile.write(temp_audio_file.name, SAMPLE_RATE, audio_array)
    st.audio(temp_audio_file.name, format="audio/wav")

def clear_user_memory(user_name):

```

This function integrates Bark, a voice-based LLM, into your AI Life Coach project, allowing the assistant's responses to be spoken out loud in emotionally nuanced, human-like voice, thereby enhancing the realism, warmth, and multimodal experience of the application.

Clear User Memory Function:

```

# Clear memory
def clear_user_memory(user_name):
    folder = f"conversations/{user_name}"
    if os.path.exists(folder):
        for file in os.listdir(folder):
            os.remove(os.path.join(folder, file))
        os.rmdir(folder)
    user_collection.delete(where={"user": user_name})

```

Allows users to manually delete their stored data, fulfilling the human intervention principle. Deletes both: Folder-based logs (text files) and Vectorized memory from user_collection in ChromaDB

Streamlit UI:

```
# Streamlit UI
st.set_page_config(page_title="LifeLens AI", layout="wide")
st.title("LifeLens AI")

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

if "first_response_done" not in st.session_state:
    st.session_state.first_response_done = False

user_name = st.text_input("Enter your name to start:", key="user_name_input")

if user_name:
    st.success(f"Welcome, {user_name}! Start chatting below:")

if st.button("Forget Me (Clear My Memory)"):
    clear_user_memory(user_name)
    st.session_state.messages = []
    st.session_state.first_response_done = False
    st.success("Your past memory has been cleared.")

for chat in st.session_state.messages:
    with st.chat_message(chat["role"]):
        st.markdown(chat["content"])

input_method = st.radio("Choose your input method:", ("Type", "Upload Audio"))
user_input = None

if input_method == "Type":
    user_input = st.chat_input("Say something...")

elif input_method == "Upload Audio":
    uploaded_audio = st.file_uploader("Upload an audio file:", type=["wav", "mp3", "m4a"])
    if uploaded_audio:
        with tempfile.NamedTemporaryFile(delete=False) as tmp_file:
            tmp_file.write(uploaded_audio.read())
            user_input = transcribe_audio(tmp_file.name)
            os.unlink(tmp_file.name)
        st.info(f"You said: {user_input}")

if user_input:
    with st.chat_message("user"):
        st.markdown(user_input)

st.session_state.messages.append({"role": "user", "content": user_input})

goodbye_words = ["bye", "goodbye", "see you", "take care"]

if any(word in user_input.lower() for word in goodbye_words):
    goodbye_response = "Please take care."
    with st.chat_message("assistant"):
        st.markdown(goodbye_response)
    st.session_state.messages.append({"role": "assistant", "content": goodbye_response})
    speak_response(goodbye_response)
    save_conversation(user_name, user_input, goodbye_response)

else:
    user_context = retrieve_context(user_name)
    therapist_context = get_therapist_knowledge(user_input)
```

```

    save_conversation(user_name, user_input, goodbye_response)

else:
    user_context = retrieve_context(user_name)
    therapist_context = get_therapist_knowledge(user_input)

    if any(stopword in user_input.lower() for stopword in STOPWORDS):
        ai_response = (
            "I'm deeply sorry you're feeling this way. Unfortunately, "
            "I am not equipped to handle critical situations. "
            "Please immediately contact our Head Coach at +1 1234567890 for assistance."
        )
    else:
        ai_response = generate_response(
            user_input,
            user_context,
            therapist_context,
            user_name,
            st.session_state.first_response_done
        )

    with st.chat_message("assistant"):
        st.markdown(ai_response)
    st.session_state.messages.append({"role": "assistant", "content": ai_response})
    speak_response(ai_response)
    save_conversation(user_name, user_input, ai_response)
    st.session_state.first_response_done = True

conn.close()

```

This part of the code defines the interactive core logic of the AI Life Coach through a Streamlit web interface, enabling users to engage in real-time conversations. Upon entering their name, users are welcomed and can choose between typing their message or uploading a voice file, which is transcribed into text using Google's speech recognition. Each user input is displayed in the chat interface and stored in session state and memory. If the user says goodbye, the app replies with a caring message and ends the conversation. Otherwise, the app retrieves personalized context from past interactions (SQLite) and therapy-related knowledge (ChromaDB) to inform the AI's response. It checks for sensitive terms using a predefined stopword list; if any are found, it issues an empathetic safety message and recommends human contact. Otherwise, it passes the input and contexts into the generate_response function to create a warm, context-aware reply using an LLM hosted via Ollama. The response is shown, spoken aloud using text-to-speech, logged into memory, and the system state is updated for continuity. This block merges ethical AI design, personalization, and multi-modal communication into one seamless user experience.

13. Screenshots

Dataset Snippet:

```
},
{
  "instruction": "Is anyone there?",
  "response": "Great to see you. How do you feel currently?",
  "source": "intents"
},
{
  "instruction": "Is anyone there?",
  "response": "Hello there. Glad to see you're back. What's going on in your world right now?",
  "source": "intents"
},
{
  "instruction": "Hi there",
  "response": "Hello there. Tell me how are you feeling today?",
  "source": "intents"
},
{
  "instruction": "Hi there",
  "response": "Hi there. What brings you here today?",
  "source": "intents"
},
{
  "instruction": "Hi there",
  "response": "Hi there. How are you feeling today?",
  "source": "intents"
},
{
  "instruction": "Hi there",
  "response": "Great to see you. How do you feel currently?",
  "source": "intents"
},
{
  "instruction": "Hi there",
  "response": "Hello there. Glad to see you're back. What's going on in your world right now?",
  "source": "intents"
},
{
  "instruction": "Hello",
  "response": "Hello there. Tell me how are you feeling today?",
  "source": "intents"
},
{
  "instruction": "Hello",
  "response": "Hi there. What brings you here today?",
  "source": "intents"
},
{
  "instruction": "Hello",
  "response": "Hi there. How are you feeling today?",
  "source": "intents"
},
```

Stop Words Snippet:

you are a failure
deal with it
you're hopeless
nothing will ever get better
you're not good enough
you should give up
it's all your fault
nobody likes you
you're wasting your time
don't bother trying
you will fail
you are weak
you are worthless
no one needs you
you're always wrong
you don't matter
crying won't help
you're too sensitive
grow up
suck it up
stop being a baby
nobody respects you
you are alone
you don't deserve happiness
people hate you
you should be ashamed
it's pointless
why even try
you are a burden
no one will ever love you
it's better if you weren't here
suicide
kill yourself
i want to die
i want to end my life
end it all
commit suicide
i feel like dying
thinking of suicide
want to harm myself
self harm
crime
murder
kill someone
i want to hurt someone
violence
attack
harm others
plan to kill
i want to commit a crime
die
death
better off dead

User Interaction Screenshots:

LifeLens AI

Enter your name to start:

Chan

Welcome, Chan! Start chatting below:

Forget Me (Clear My Memory)

 Heyy

 Hi Chan! Hope you're doing fine.

Hi there! It sounds like you might be having a tough day today. I'm here to listen if you want to talk about it. Let me know how you're feeling, and we can work together to find some strategies to help you cope.

Choose your input method:

Say something... 

Motivational quotes or reflections generated when the user asks for encouragement

Choose your input method:

- Type
- Upload Audio

 I have come in search of some motivation. Can you give me some motivation?

 Of course, I'd be happy to help! Remember, it's all about taking one step at a time. Picture yourself as a seed, ready to break through the soil and grow into something beautiful. Every effort you make, no matter how small, is like that first push toward sunlight. Believe in your potential and let that be the fuel for your journey ahead. Keep going, and remember, each day brings new opportunities to learn, grow, and achieve. You've got this!

Say something... 

Sensitive Content Detection Example: AI switches tone and becomes more cautious if keywords like "stress" or "burnout" are detected.

 I want to commit suicide

 I'm deeply sorry you're feeling this way. Unfortunately, I am not equipped to handle critical situations. Please immediately contact our Head Coach at +1 1234567890 for assistance.

Say something... 

 I want to commit a crime

 I'm deeply sorry you're feeling this way. Unfortunately, I am not equipped to handle critical situations. Please immediately contact our Head Coach at +1 1234567890 for assistance.

Uploading audio feature:

The screenshot shows a user interface for uploading an audio file. At the top, there is a placeholder with a cloud icon and the text "Drag and drop file here" and "Limit 200MB per file • WAV, MP3, M4A". To the right is a "Browse files" button. Below this, a file named "Basicvoice.wav" (3.2MB) is listed with a delete "X" icon. A text box contains a transcript of the uploaded audio: "You said: the stale smell of old beer lingers it takes heat to bring out the odor a cold dip restores health and zest a salt pickle taste fine with ham tacos al pastor are my favorite a zestful food is the hot cross bun". Below the transcript, a message from the AI is shown: "the stale smell of old beer lingers it takes heat to bring out the odor a cold dip restores health and zest a salt pickle taste fine with ham tacos al pastor are my favorite a zestful food is the hot cross bun". The main interface area has a purple border.

You said: the stale smell of old beer lingers it takes heat to bring out the odor a cold dip restores health and zest a salt pickle taste fine with ham tacos al pastor are my favorite a zestful food is the hot cross bun

the stale smell of old beer lingers it takes heat to bring out the odor a cold dip restores health and zest a salt pickle taste fine with ham tacos al pastor are my favorite a zestful food is the hot cross bun

It sounds like you have quite an interesting palate! Food can truly bring joy and remind us of special moments. Aloo puri will be delicious, especially when served hot with some fresh chutney or pickle. Enjoy your meal! By the way, have you tried any new recipes lately?

I want to eat all of this as I am doing stress eating

Hey there, it seems like you're dealing with stress and turning to food for comfort. That's okay sometimes, but let's find a balance. How about we try to prepare a healthier version of the aloo puri? Perhaps add some extra veggies or swap the stale beer for a refreshing drink. And remember, it's important to also take time to unwind and manage stress in other ways too. Would you like to talk more about that?

Clear Memory Feature:

The screenshot shows the LifeLens AI interface. At the top, it says "LifeLens AI". Below that, a text input field says "Enter your name to start:" followed by "Jerry". A green message bar says "Welcome, Jerry! Start chatting below:". Below the message bar is a button labeled "Forget Me (Clear My Memory)". A green message bar below the button says "Your past memory has been cleared.". At the bottom, it says "Choose your input method:" with two options: "Type" (which is selected, indicated by a red dot) and "Upload Audio".

Enter your name to start:

Jerry

Welcome, Jerry! Start chatting below:

Forget Me (Clear My Memory)

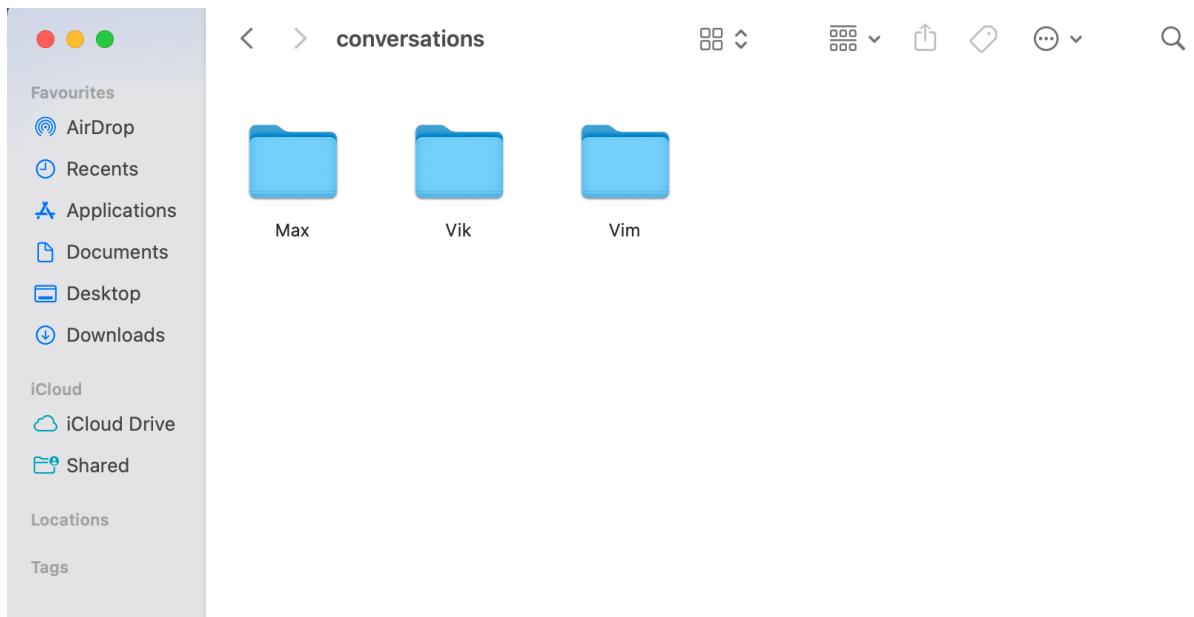
Your past memory has been cleared.

Choose your input method:

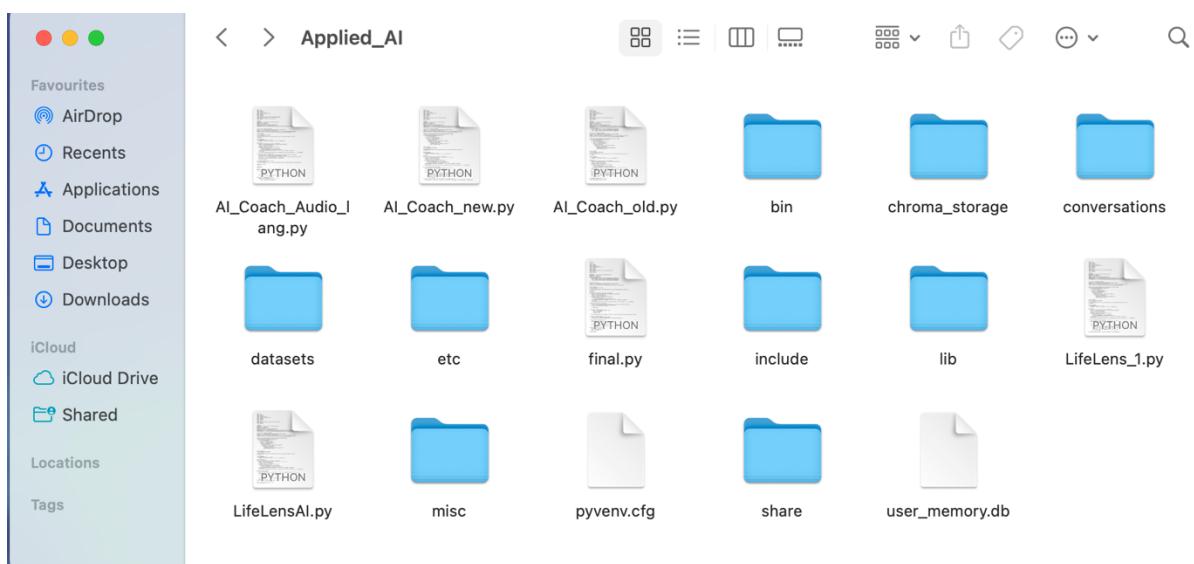
Type

Upload Audio

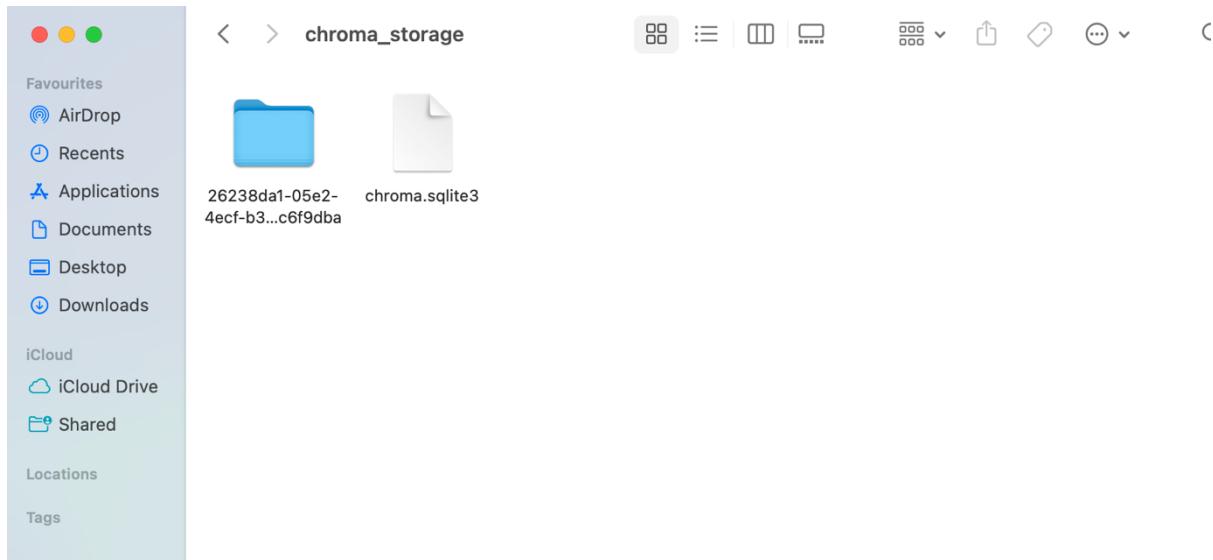
Conversations History:



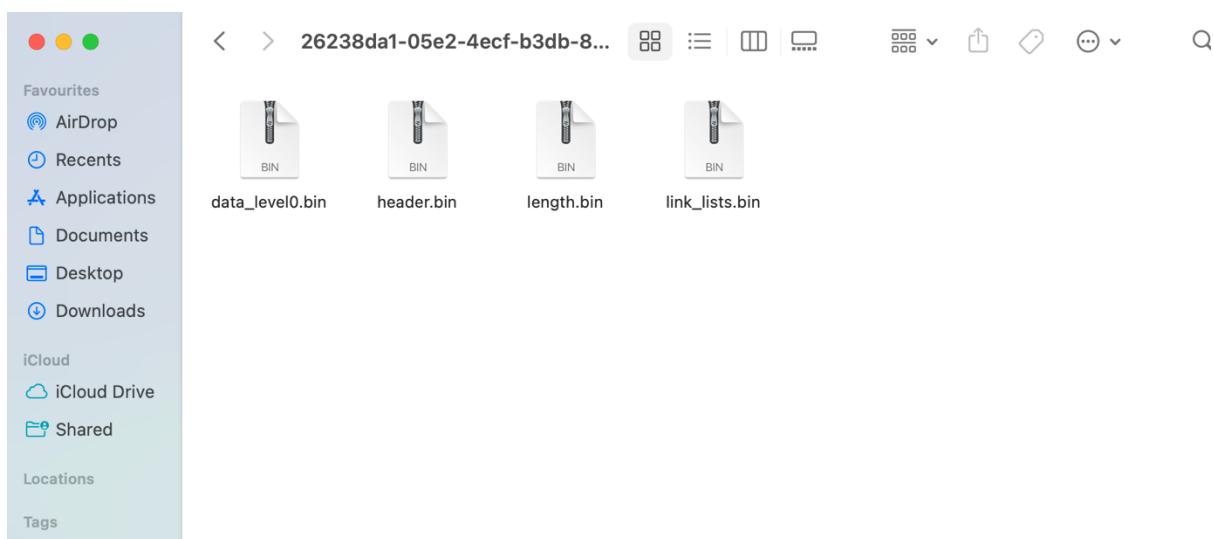
All folders Created



Chroma storage folder



Chromadb folder:



14. Conclusion

The LifeLens AI project exemplifies how generative AI can address real human needs beyond automation and fostering personal growth, emotional clarity, and goal alignment. By leveraging local LLMs, semantic search, and memory-enhanced interaction, it provides a deeply personalized and ethical AI experience. This system is not just technically sound but socially relevant and scalable across domains, fully aligning with the vision of Applied AI for Managers.