EmpathyBot: Sentiment-Driven Chat with Advanced RAG and Few-Shot Prompt Engineering - Project Documentation

To: Employee
From: Mohamed Osama, Senior Technical Lead
Date: September 10, 2025
Subject: 6-Hour Task - EmpathyBot Prototype Development

Dear Employee,

You are tasked with building a functional prototype of **EmpathyBot**, a chatbot that detects user emotions and responds empathetically, within **6 hours**. This high-priority demo showcases AI-driven emotional intelligence for applications like mental health support or customer service. Your focus is a simple, working prototype built in Google Colab with a web-based chat interface, offering Streamlit or FastAPI as deployment options.

Given the tight timeline, use pre-trained models and datasets to minimize setup time. Ensure responses are safe, positive, and include a disclaimer: "I'm not a therapist; please seek professional help for serious issues." Do not store user inputs without consent. Use Git (repo: `empathybot-[your-initials]`) for version control and provide a brief README with setup instructions. Deliver a live demo link and a 2-3 minute video walkthrough by the end of the day.

This document is your guide. Follow the steps, test incrementally, and reach out immediately for blockers. Let's make this demo impactful!

---

1. Project Overview

1.1 What is EmpathyBot?

EmpathyBot is a chatbot that analyzes a user's emotions from text inputs and responds with kind, contextually appropriate replies. It uses **advanced Retrieval-Augmented Generation (RAG)** combined with **few-shot prompt engineering** to select and frame empathetic responses from a large corpus. For example, if a user writes, "I'm feeling really low," the bot might respond, "I'm so sorry you're feeling this way. Want to talk about what's going on?" The prototype will feature a simple web interface for real-time chat.

1.2 Goals

- Build a prototype that accurately detects emotions and delivers empathetic responses.
- Create a web-based chat interface using Streamlit or a RESTful API using FastAPI.
- Ensure responses are safe, ethical, and varied to feel human-like.
- Success metrics:
    - Correctly identify emotions in 8 out of 10 test messages.
    - Responses align with the user's mood (80% relevance via manual review).
    - A live demo handles 3-5 sample conversations smoothly.

1.3 Why This Project?

EmpathyBot demonstrates emotionally intelligent AI for mental health or customer support applications. Using advanced RAG ensures scalable, context-aware responses, while few-shot prompt engineering enhances reply quality without heavy model training. Google Colab enables fast, free development, and Streamlit/FastAPI provides flexible deployment options for a polished demo.

1.4 Scope

- **Included**: Emotion detection with a pre-trained model, advanced RAG with a large response corpus, few-shot prompt engineering, and a Streamlit or FastAPI interface.
- **Excluded**: Training new models, multi-turn conversation memory, or complex features like voice input.
- **Assumptions**: You have basic Python skills, and Colab's free tier is sufficient.

---

2. Tools and Resources

2.1 Software

- **Platform**: Google Colab (free, cloud-based environment).
- **Tools**:
    - Hugging Face Transformers: For pre-trained emotion detection models.
    - Sentence-Transformers: For text embeddings in RAG.
    - FAISS: For efficient response retrieval.
    - Streamlit: For a web-based chat interface.
    - FastAPI/Uvicorn: For an optional RESTful API endpoint.
    - Pandas/NLTK: For data handling and text preprocessing.
- **Version Control**: Git (GitHub repo).

2.2 Data

- **Emotion Detection Dataset**: Use the Hugging Face dataset `emotion` (from `tweet_eval`), containing ~50,000 labeled tweets for emotions like happiness, sadness, anger, etc. Test with 10 sample inputs (e.g., "I'm overjoyed!" → happiness, "I'm so upset" → sadness).
- **RAG Corpus**: Source a large sample (~1,000 empathetic response templates) from the Hugging Face dataset `empathetic_dialogues` (or curate manually if access is limited). Structure as a JSON file with columns: [emotion, template]. Examples:
    - Happiness: "That's wonderful! What's sparking your joy today?"
    - Sadness: "I'm really sorry you're feeling down. Can I listen to what's on your mind?"
    - Neutral: "Thanks for sharing! Want to tell me more?"
- **Text Cleanup**: Convert inputs to lowercase, remove URLs, emojis, and special characters.

2.3 Potential Pre-Trained Models

- **Sentiment/Emotion Detection** (choose one based on performance in tests):
    - `distilbert-base-uncased-finetuned-sst-2-english`: Lightweight, pre-trained for positive/negative sentiment (Hugging Face).
    - `bhadresh-savani/distilbert-base-uncased-emotion`: Fine-tuned for granular emotions (e.g., joy, sadness, anger).
    - `nlptown/bert-base-multilingual-uncased-sentiment`: Supports broader sentiment ranges (e.g., 1-5 stars).
- **RAG Embeddings**:
    - `sentence-transformers/all-MiniLM-L6-v2`: Fast, lightweight model for creating text embeddings.

2.4 Hardware

- **Development**: Colab free tier (GPU for inference).
- **Demo**: Host on Streamlit Cloud (free) or a FastAPI server on Render/Heroku (free tiers).

2.5 Dependencies

Install in Colab:

- `transformers`
- `sentence-transformers`
- `faiss-cpu`
- `streamlit`
- `fastapi`

- `uvicorn`
- `pandas`
- `nltk`

---

3. Steps to Build EmpathyBot (4 Hours)

Work in a Colab notebook named `EmpathyBot_Sprint.ipynb`. Organize code into clear sections for easy maintenance.

3.1 Step 1: Setup and Data Preparation (30 minutes)

1. Install all required tools in Colab.
2. Load a pre-trained emotion detection model from Hugging Face (e.g., `bhadresh-savani/distilbert-base-uncased-emotion`).
3. Download a large sample (~1,000 templates) from the `empathetic_dialogues` dataset or manually create a JSON file (`corpus.json`) with responses for happiness, sadness, anger, and neutral emotions.
4. Test the emotion model with 10 sample messages to ensure it detects emotions accurately.

3.2 Step 2: Advanced RAG System (1 hour)

1. Use Sentence-Transformers (`all-MiniLM-L6-v2`) to convert the 1,000 response templates into embeddings for fast searching.
2. Set up FAISS to store and search these embeddings efficiently.
3. Create a retrieval function that:
    - Matches the user's input to the detected emotion.
    - Retrieves the top 3 relevant responses from the corpus, prioritizing those aligned with the detected emotion.

3.3 Step 3: Few-Shot Prompt Engineering (1 hour)

1. Build a pipeline that:
    - Takes a user's message.
    - Detects its emotion using the pre-trained model.
    - Retrieves 3 matching response templates via RAG.
    - Uses few-shot prompting to frame the response naturally (e.g., combine templates with a prompt like: "The user is feeling [emotion]. Respond empathetically: [template]").
2. Add a disclaimer to every response.

3. Test the pipeline in Colab with a text-based loop (e.g., enter a message, see the bot's reply).

3.4 Step 4: Web Interface or API (1.5 hours)

Choose **one** deployment option:

- **Option 1: Streamlit Web Interface**
    - Create `app.py` with a chat interface.
    - Include a text input box, display chat history, and load the emotion model and RAG system.
    - Test locally in Colab using ngrok for a public URL.
- **Option 2: FastAPI Endpoint**
    - Create `main.py` with a RESTful API (e.g., POST `/chat` accepting JSON input like `{"message": "I'm sad"}`).
    - Return JSON with the bot's response and detected emotion.
    - Test locally with a tool like Postman or curl.

---

4. Testing and Deployment (1 Hour)

4.1 Testing (30 minutes)

- **Emotion Detection**: Test 10 messages (e.g., "I'm thrilled" → happiness, "I hate today" → sadness). Aim for 8/10 correct.
- **RAG Responses**: Check that retrieved responses match the emotion (manual review for 80% relevance).
- **Interface/API**: Run 3 sample chats (Streamlit) or API calls (FastAPI) to confirm smooth operation.

4.2 Deployment (30 minutes)

- Push code to a GitHub repo.
- For Streamlit: Upload `app.py`, `corpus.json`, and requirements to Streamlit Cloud; share the public URL.
- For FastAPI: Deploy to Render or Heroku; test the endpoint with a sample request.
- If deployment fails, use Colab's ngrok URL as a backup.

---

5. Risks and Solutions

- **Time Constraints**: Use pre-trained models and a subset of the `empathetic_dialogues` dataset to save time.
- **Colab Limits**: Switch to CPU if GPU crashes; restart the session if needed.
- **Inappropriate Responses**: Review the corpus to ensure all templates are kind and safe. For risky inputs (e.g., self-harm mentions), respond with a helpline suggestion.

---

6. Deliverables

1. **GitHub Repo**: Include `EmpathyBot_Sprint.ipynb`, `app.py` or `main.py`, `corpus.json`, `requirements.txt`, and a README with setup/run instructions.
2. **Demo**: A live Streamlit/FastAPI URL + a 2-3 minute video showing 3 sample chats or API calls.
3. **Notes**: In the README, add a brief section on what worked, challenges, and suggestions (e.g., “Completed in 5.5 hours; emotion detection solid, but corpus could be larger”).

---

7. Timeline (6 Hours)

- **0:00-0:30**: Set up Colab, load dataset and models.
- **0:30-1:30**: Build advanced RAG system.
- **1:30-2:30**: Implement few-shot prompt engineering and chat pipeline.
- **2:30-4:00**: Create and test Streamlit or FastAPI interface.
- **4:00-5:00**: Test all components.
- **5:00-6:00**: Deploy, record video, finalize README.

---

8. Support

Message me immediately for any issues. Refer to Hugging Face, Streamlit, and FastAPI online guides. This demo is high-visibility—focus on a clean, working prototype. You’ve got this!

Best,
Mohamed Osama
Senior Technical Lead