This is a fully functional AI chatbot and knowledge base built with OpenAI, LangChain, and Gradio. It serves as an intelligent personal assistant representing Michael M, a software engineer at Microsoft specializing in Azure Functions and agentic AI. The app includes Retrieval-Augmented Generation (RAG), tool use (e.g., contact logging), evaluation logging, long-term memory, and an admin dashboard.

**🧠 Features**

* 💬 Chatbot with persona + RAG context from personal documents and articles
* 📖 Retrieval of relevant article snippets using Chroma vector store
* 🛠️ Tools for capturing:
  + Unknown questions
  + Interested users (email, name, notes)
* 📊 Evaluation logging with timestamped queries and responses
* 🧵 Per-user threaded memory
* 📂 Dashboard for searching, sorting, and downloading evaluation logs
* ✅ Runs locally or deploys to [Hugging Face Spaces](https://huggingface.co/spaces)

**🚀 Getting Started**

**1. Clone the repository**

git clone https://github.com/yourusername/michaelm-chatbot.git

cd michaelm-chatbot

**2. Create a .env file**

OPENAI\_API\_KEY=your-openai-key

PUSHOVER\_USER=your-pushover-user

PUSHOVER\_TOKEN=your-pushover-token

**3. Install dependencies**

pip install -r requirements.txt

**4. Prepare knowledge base**

Ensure the following exist in the me/ folder:

* linkedin.pdf
* summary.txt
* articles.txt

These are used to build the RAG knowledge base and provide accurate responses.

**5. Run the app locally**

python app.py

**6. Deploy to Hugging Face (optional)**

* Upload all files and make sure the Space is configured as a Gradio app.
* Note: share=True is ignored on Hugging Face.

**🧠 How It Works (Step-by-Step)**

**🔹 Initialization**

* Loads secrets using dotenv
* Loads your personal data files: linkedin.pdf, summary.txt, articles.txt
* Uses Chroma + OpenAIEmbeddings to create a searchable RAG index

**🔹 Tool Functions**

* Describes record\_user\_details and record\_unknown\_question
* Shows how OpenAI tool calls are used in the assistant flow

**🔹 RAG & Prompt Setup**

* User questions trigger retrieval from articles.txt
* Retrieved content is injected into the system prompt along with LinkedIn and summary content

**🔹 Chat Execution**

* Constructs message list with history + current message
* Calls openai.chat.completions.create() with gpt-4o-mini
* Executes tools when needed and appends results
* Logs every interaction to evaluation\_log.jsonl
* Uses user\_threads to track long-term conversation memory by user ID

**🔹 Dashboard UI**

* The Gradio interface has two tabs:
  1. **Chat Interface** – Ask questions and interact with the assistant
  2. **Evaluation Logs** – View logs in a searchable, sortable table with download support

**📊 Evaluation Dashboard**

Accessible via the **"Evaluation Logs"** panel. It shows:

* Timestamp
* User ID
* Question
* Response
* Retrieved context

You can filter, search, sort, and export the logs.

**🛠️ Coming Soon / Optional Enhancements**

* 📈 Charts and analytics from evaluation data
* 🧥 Thread-based UI for better conversation flow
* 🧠 Fine-tuned memory with vector-based user personalization
* 🔍 Filtering/search by keyword, topic, or sentiment

Here's a step-by-step breakdown of how the updated app.py works based on the latest version:

**🔧 1. Environment Setup and Imports**

* **.env loading**: Uses load\_dotenv(override=True) to load environment variables like OpenAI keys and Pushover credentials.
* **Imports**: Includes OpenAI SDK, PDF/Text handling, LangChain components (Chroma, Embeddings), Gradio, and date/time.

**📬 2. Tool Definitions**

* Defines two custom functions:
  + record\_user\_details: Pushes a message to Pushover with the user’s email and optional name/notes.
  + record\_unknown\_question: Pushes unanswerable user questions to Pushover.
* JSON schemas (record\_user\_details\_json, record\_unknown\_question\_json) describe how the LLM should call these tools.

**📚 3. Me Class Initialization**

* **OpenAI Client**: Initializes OpenAI SDK client.
* **Identity**: Sets display name (Michael M).
* **Evaluation Log**: Prepares in-memory and file-based log for interaction evaluation.
* **Load Documents**:
  + Reads me/linkedin.pdf, me/summary.txt, and me/articles.txt.
  + Splits articles.txt into chunks using LangChain's RecursiveCharacterTextSplitter.
  + Embeds and stores chunks in a **Chroma vectorstore** for Retrieval-Augmented Generation (RAG).

**🔍 4. Retrieval-Augmented Generation**

* retrieve\_relevant\_context(query):
  + Performs semantic similarity search in the vectorstore (Chroma) for the given user query.
  + Returns top 4 relevant article segments to enrich the system prompt.

**🧠 5. Tool Call Handling**

* handle\_tool\_call(tool\_calls):
  + Matches tool name to Python function via globals().
  + Deserializes the tool arguments and executes the correct tool function.
  + Packages the tool result back in the OpenAI-compatible response format.

**💬 6. System Prompt Construction**

* system\_prompt(rag\_context):
  + Includes:
    - The persona and context of “Michael M”
    - Summary, LinkedIn profile, and optionally RAG context.
    - Instructions to log unknown questions and prompt for emails.
  + This is injected as the system role in the conversation.

**🧪 7. Evaluation Logging**

* log\_evaluation(entry):
  + Appends interaction metadata to a list and writes to evaluation\_log.jsonl.
  + Each log entry includes:
    - Timestamp
    - Question
    - Response
    - RAG context

**🤖 8. Chat Logic**

* chat(message, history):
  + Prepares the full message list with system prompt, chat history, and user message.
  + Calls the OpenAI Chat API with tool definitions and RAG-enriched prompt.
  + If tool calls are returned, it executes them and appends the results to history.
  + Logs the response to evaluation\_log.jsonl.
  + Returns the assistant’s message text.

**🌐 9. Gradio Chat UI**

* Launches a web-based chat interface using gr.ChatInterface.
  + Uses the me.chat() method as the backend.
  + ssr\_mode=False disables server-side rendering (needed for Hugging Face compatibility).
  + share=True generates a public link (if not hosted elsewhere).

**✅ Summary**

This is a **RAG-enabled**, **tool-augmented**, **evaluated chat app** that:

* Reads your PDF résumé, summary, and articles.
* Answers questions as if it were *you* (Michael M).
* Uses OpenAI GPT-4o-mini with external tool calls.
* Pushes alerts for user interest or unknown queries.
* Logs all interactions for later review or dashboarding.