Project Overview

This Streamlit-based application simulates a **Motivational Interviewing (MI)** session for HPV vaccine counseling. It allows users (e.g., students or providers) to **practice communication skills** with a realistic virtual patient ("Alex"). The chatbot plays the role of a hesitant but curious patient and gives **feedback at the end** using an MI rubric. The app uses **Groq LLMs** for real-time dialogue and **retrieval-augmented generation (RAG)** to incorporate structured feedback from rubric documents.

Key Libraries and Tools

- **streamlit**: For building the web interface.
- **groq**: For accessing LLMs via the Groq API (Groq is known for extremely fast inference and supports models like LLaMA 3.1).
- **sentence-transformers** + **faiss (Facebook AI Similarity Search)**: To create a semantic search engine over MI rubrics.
- **numpy**, **json**, **os**: For file I/O and array handling.

App Structure and Core Components

1. System Prompt and LLM Setup

A detailed **system prompt (SYSTEM_PROMPT)** defines how the chatbot ("Alex") behaves. It ensures Alex acts like a patient during the conversation and only evaluates the provider at the end. This prompt also contains grading criteria for feedback (Evocation, Acceptance, Collaboration, Compassion, Summary).

The Groq API key is collected from the user, and a Groq() client is initialized to interact with the LLM.

---

2. Streamlit UI Configuration

The Streamlit app is configured with a clean title and welcome markdown. It introduces the MI practice simulation and guides the user to enter their Groq API key. If the key is not entered, the app stops execution.

python
CopyEdit
st.text_input("🔑 Enter your GROQ API Key", type="password")

This ensures API security and prevents unnecessary LLM calls.

---

3. Loading and Embedding the MI Rubric

The application loads multiple .txt files from the hpv_rubrics folder. These contain feedback examples or guidelines. All the files are concatenated into a single document, which is then chunked using a custom split_text() function.

These chunks are embedded using a **sentence transformer** (all-MiniLM-L6-v2) and added to a **FAISS index**, which enables semantic search over the rubric content during feedback generation.

faiss_index = faiss.IndexFlatL2(dimension)
embeddings = embedding_model.encode(knowledge_chunks)
faiss_index.add(np.array(embeddings))

---

4. Conversation State Management

Streamlit's session_state is used to maintain chat history. The conversation begins with a predefined assistant message from "Alex." Each message is rendered using st.chat_message().

if "chat_history" not in st.session_state:

```
st.session_state.chat_history = []
```

This ensures continuity across turns within a single browser session.

---

5. Chat Interface and LLM Turn-taking

The user enters input via st.chat_input(), which gets appended to chat_history.

Before sending the conversation to the model, a lightweight **instructional message** is added to guide the assistant in structuring its responses according to MI best practices (e.g., asking open-ended questions, evoking change talk).

```
turn_instruction = {
   "role": "system",
   "content": "Follow the MI chain-of-thought steps..."
}
```

The messages list is then passed to Groq's LLaMA-3.1-8B-Instant model for generating a new assistant response. The response is added to the chat and displayed.

---

6. End of Session: Generating Feedback with RAG

When the user clicks **"Finish Session & Get Feedback"**, the app:

1. Compiles the full chat transcript.
2. Uses the **semantic search engine to retrieve relevant rubric content.**
3. Constructs a feedback prompt that includes both the transcript and the retrieved MI criteria.

4. Sends the prompt to the LLM to generate an evaluation.

The resulting feedback contains **scores, strengths, and improvement suggestions**, mimicking a trained evaluator's MI assessment.

🌟 Benefits of Using Groq over Hugging Face or OpenAI

- **Latency**: Groq's hardware accelerates inference drastically, with sub-1ms token generation times. This makes the app very responsive.
- **Compatibility**: Supports high-quality open models like LLaMA 3.1, which are on par with proprietary models in performance.
- **Cost-effectiveness**: Using hosted open-weight models (vs. paid APIs like OpenAI) may reduce cost in production environments.

That said, Groq requires API setup and model configuration manually, unlike HuggingFace's hosted inference or OpenAI's plug-and-play design.

Summary and Future Work

To maintain or extend this app:

- You must provide your **Groq API key** or replace the backend with another LLM provider.
- Rubric examples live in **hpv_rubrics/ or ohi_rubrics/**— these can be expanded or modified.

- The core logic is in 3 steps: load rubric + embed → run chat interaction → RAG-based feedback.

- **You can improve this app by:**

    - Logging chat history to database.

    - Adding multi-session storage via Streamlit sidebar using session names/ timestamps.

    - Improve feedback via prompt engineering/ improving the RAG based search engine (using more rubric examples, different sentence transformer for semantic search engine)

    - Try using different versions of Llama or Deepseek (any other open source model APIs in Groq)