RAG Implementation - Technical Interview

Objective

Build a **minimal Retrieval-Augmented Generation (RAG) prototype** that can answer questions using a small corpus of FAQ documents.
You will **complete the TODOs** in the starter file `rag_assessment_partial.py`;

Problem Statement

Build a Python script that:

1. Reads FAQ documents from a directory
2. Processes and embeds the content for semantic search
3. Takes user questions and finds the most relevant information
4. Generates answers using the retrieved context
5. Returns results in a structured JSON format

Expected Workflow

Your solution should follow this general pattern:

1. **Load & Process**: Read FAQ files and break them into manageable chunks
2. **Embed**: Convert text chunks into vector embeddings for similarity search
3. **Query**: Accept user questions and embed them
4. **Retrieve**: Find the most relevant chunks using similarity matching
5. **Generate**: Use an LLM to create answers based on retrieved context
6. **Format**: Return structured output with sources

Requirements

Input

- FAQ documents stored as `.md` files in a `faqs/` directory
- User questions entered via command line input

Output

Your script should output JSON in this format:

json

```JSON
{
  "answer": "Generated answer with source citations",
  "sources": ["filename1.md", "filename2.md"]
}
```

Technical Specifications

- Use OpenAI's API for both embeddings and text generation
- Implement cosine similarity for relevance scoring
- Chunk documents into ~200 character pieces
- Retrieve top 4 most relevant chunks
- Ensure answers cite at least 2 source files
- Use `text-embedding-ada-002` for embeddings
- Use `gpt-3.5-turbo` for answer generation

Provided Resources

Sample FAQ Structure

Your `faqs/` directory will contain files like:

```None
faqs/
├── faq_auth.md
├── faq_employee.md
├── faq_sso.md
```

Required Dependencies

```python
import os
import json
from openai import OpenAI
import numpy as np
from tqdm import tqdm
```

OpenAI Client Setup

```python
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
```

Embedding API Usage Example

```python
# Get embedding for text
response = client.embeddings.create(
    input=["your text here"],
    model="text-embedding-ada-002"
)
embedding = np.array(response.data[0].embedding)
```

Chat Completion Example

```python
# Generate response
response = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": "your prompt"}],
    temperature=0.2
```

```
)
answer = response.choices[0].message.content.strip()
```

Implementation Hints

Text Chunking

Consider how to split longer documents into smaller, searchable pieces while maintaining context.

Similarity Calculation

You'll need to compare query embeddings with document embeddings. Cosine similarity is a good choice:

python

```python
def cosine_sim(a, b):
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))
```

Context Building

When generating answers, provide the LLM with:

- Clear instructions about citing sources
- The retrieved context with source attribution
- The original question

Progress Tracking

Since embedding multiple chunks can take time, the code uses `tqdm` for progress visualization.

Evaluation Criteria

- **Functionality**: Does the system correctly retrieve and generate relevant answers?
- **Code Quality**: Is the implementation clean and well-structured?
- **Source Attribution**: Are sources properly tracked and cited?

Getting Started

Follow these steps to set up your environment and run the technical exercise:

1. **Ensure Homebrew is installed** on your computer.

   If not, install it from: https://brew.sh

2. **Install Python 3.10** (if not already installed):

   ```
   brew install python@3.10
   ```

3. **Create and activate a virtual environment**:

   ```
   python3 -m venv venv

   source venv/bin/activate
   ```

4. **Install the required libraries**:

   ```
   pip install openai numpy tqdm
   ```

5. **Set your OpenAI API key** (for bash/zsh):

   ```
   export OPENAI_API_KEY=your_api_key
   ```

6. **Run the script**:

```
python rag_assessment_partial.py
```

7. **Complete the remaining steps**:

    a. Create the `faqs/` directory and place your markdown files inside.

    b. Implement the core functions in the starter script.

    c. Test the program with sample queries.

    d. Refine the output based on quality and source attribution.

Time Expectation

This exercise should take approximately 2-3 hours to complete, including testing and refinement.
