RAG System Design Task - Sentence-Level Retrieval for Essay Feedback

Context

You're joining an EdTech startup that provides AI-powered writing feedback to students. Our platform helps students improve their essays by referencing relevant content.

The Challenge: We need to retrieve **specific sentences** from books that are relevant to a student's essay, so our LLM can provide feedback that references concrete examples, concepts, or writing techniques from the course material.

Example Scenario:

- **Student Essay Topic:** "The impact of social media on mental health"
- **Assigned Book:** "Digital Minimalism" by Cal Newport
- **Desired Output:** Specific sentences from the book that relate to points in the student's essay, which the LLM can use to provide contextualized feedback

---

Your Mission

Design a RAG system architecture that can retrieve relevant sentences from books to support essay feedback generation.

Task Requirements

Part 1: System Design Document (60-90 mins)

Write a technical design document addressing:

1.1 Retrieval Granularity Strategy

- **Why sentence-level retrieval?** Discuss tradeoffs vs. paragraph or chunk-level retrieval
- **Chunking approach:** How will you handle sentence extraction while maintaining context?
    - Do you store individual sentences? Sentences with surrounding context?
    - How do you handle sentences that depend on previous context to make sense?
- **Metadata strategy:** What metadata will you store with each sentence? (chapter, page, topic, etc.)

1.2 Embedding & Retrieval Logic

- **Embedding strategy:** What will you embed? (individual sentences, sentence + context, etc.)
- **Query formulation:** How will you convert a student essay into effective retrieval queries?
    - Do you embed the entire essay? Break it into sections? Extract key claims?
- **Retrieval method:** Describe your approach to finding relevant sentences
    - Pure semantic search? Hybrid with keyword matching?
    - How many sentences should be retrieved per essay?

1.3 Quality & Relevance Filtering

- **Relevance scoring:** How will you ensure retrieved sentences are truly relevant?
- **Diversity:** How will you avoid retrieving very similar sentences?
- **Context preservation:** How will you ensure sentences make sense when presented to the LLM?

1.4 Integration with LLM Feedback Generation

- **Prompt design:** How will retrieved sentences be formatted in the LLM prompt?
- **Citation strategy:** How will students see which book sentences were used?
- **Feedback types:** What kinds of feedback can be enhanced by sentence-level retrieval?
    - Supporting arguments with evidence
    - Identifying contradictions
    - Suggesting alternative perspectives
    - Improving writing style

Deliverable: 2-3 page technical design document (PDF or Markdown)

Part 2: Proof of Concept Implementation (90-120 mins)

Implement a **minimal prototype** demonstrating your core retrieval logic. In this part we do not evaluate the accuracy of the model, just how you structure your code.

Requirements:

Input:

- A sample book (Digital Minimalism by Cal Newport.pdf)
- A sample student essay (essay_sample.txt)

Output:

- Top 5-10 relevant sentences from the book
- Relevance scores
- Brief explanation of why each sentence was retrieved

Technical Constraints:

- Python 3.8+
- Use async.
- Requests must be using Pydantic models.
- The embedding model should be deployed locally using vLLM. (run the following command `vllm serve intfloat/e5-small`)
- Simple vector store is fine (Chroma Vector Database - in-memory)
- No need for production-grade infrastructure

```
# Your code should support something like this:

retriever = SentenceRetriever(book_path="sample_book.txt")
results = await retriever
.retrieve(
    student_essay="[essay text]",
    top_k=5
)

for result in results:
    print(f"Sentence: {result.sentence}")
    print(f"Score: {result.score}")
    print(f"Context: {result.context}")  # surrounding sentences if
needed
    print(f"Location: Chapter {result.chapter}, Page {result.page}")
    print("---")
```

Submission Format

Please provide:

1. **Design Document** (PDF or Markdown)
    - 2-3 pages
2. **Code Repository** (GitHub link or ZIP)
    - Python scripts (no notebooks)
    - `requirements.txt`
    - `README.md` with setup instructions
    - Sample output (saved results from running your code)

Time Expectation

- **Estimated time:** 3-4 hours
- **Deadline:** [5-7 days from assignment]

What You DON'T Need to Do

To keep this focused, you don't need to:

- Implement the actual LLM feedback generation (focus on retrieval only)
- Build a user interface
- Set up production infrastructure (Azure, databases, etc.)
- Handle multiple books or complex book parsing

Questions & Assumptions

Feel free to make reasonable assumptions about:

- Book format and structure
- Scale (number of books, students, concurrent users)
- Latency requirements
- Available compute resources

Document your assumptions clearly in your submission.

If you have any clarifying questions, please reach out to joao@markmywords.au We'll respond within 24 hours.