Practical Task

Practical Task: Document Processing and Basic RAG Pipeline

Task Overview

Build a simple document preprocessing and retrieval system that demonstrates core RAG concepts.

sample_texts.txt

Requirements (Complete in 60 minutes)

Part 1: Document Preprocessing (20 minutes)

- Create a Python script that takes a collection of 5-10 text documents (you can use sample articles, FAQs, or documentation)
- Implement text preprocessing including:
  - Text cleaning (remove special characters, normalize whitespace)
  - Sentence/paragraph chunking with overlap
  - Basic metadata extraction (document source, chunk index)

Part 2: Embedding Generation (15 minutes)

- Use a pre-trained embedding model (like `sentence-transformers` or OpenAI embeddings)
- Generate embeddings for each text chunk
- Store embeddings with their corresponding text and metadata in a simple structure (dictionary/JSON)

Part 3: Basic Retrieval System (20 minutes)

- Implement a similarity search function using cosine similarity
- Create a query interface that:
  - Takes a user question as input
  - Converts the question to an embedding
  - Retrieves top 3 most similar document chunks
  - Returns the chunks with their similarity scores

Part 4: Demo & Testing (5 minutes)

- Test with 2-3 sample queries
- Show the retrieved chunks and their relevance scores

Technical Specifications

- **Language**: Python
- **Libraries**: `sentence-transformers`, `numpy`, `sklearn` (for similarity), `json`
- **Input**: Text files or hardcoded sample documents
- **Output**: Console-based query interface showing retrieved chunks

Evaluation Criteria

1. **Preprocessing Quality**: Proper text cleaning and chunking strategy
2. **Embedding Implementation**: Correct use of embedding models
3. **Retrieval Logic**: Functional similarity search with appropriate scoring
4. **Code Structure**: Clean, readable code with basic error handling
5. **Demo**: Working end-to-end pipeline with realistic test cases

Bonus Points (if time permits)

- Add chunk size optimization based on content type
- Implement basic ranking beyond just similarity scores

- Add simple logging for debugging

Deliverables

- Python script(s) with clear comments
- Brief documentation explaining your chunking strategy
- Sample output showing query results