CSAI 422: Laboratory Assignment 6

Lab Session: April 7, 2025 **Due Date**: April 21, 2025 at 23:59 **Time Estimate**: 3-4 hours **Points**: 100

Building RAG Systems with Open-Source Tools and Local Vector Storage

Overview

In this lab assignment, you will implement a Retrieval-Augmented Generation (RAG) system using open-source components with local vector storage. RAG systems enhance LLM capabilities by retrieving relevant information from external knowledge sources before generating responses. You will build a complete RAG pipeline, experiment with different embedding models and retrieval strategies, and evaluate your system's performance.

Learning Objectives

- Build a complete RAG pipeline using open-source components
- Implement document processing, chunking, and embedding generation
- Configure and use local vector stores for efficient similarity search
- Design effective retrieval strategies for different types of queries
- Apply prompt engineering techniques to improve RAG performance
- Evaluate and compare different RAG configurations

Prerequisites

- Python 3.10+
- Basic understanding of vector embeddings and semantic search
- Familiarity with LLMs and prompt engineering
- Access to LLM API (OpenAI, Anthropic, etc.) for the generation component

Part 1: Setting Up the RAG Environment (20 points)

Task 1.1: Environment Setup Set up your development environment with the necessary dependencies: - LangChain for orchestration - Sentence Transformers for embeddings - FAISS for vector storage - Document loaders for PDF, DOCX, and TXT files - Integration with an LLM API of your choice

Task 1.2: Create a Document Corpus Create a document corpus for testing your RAG system:

1. Download the provided sample documents from the course materials (PDFs, Word documents, and text files related to AI research papers and concepts)
2. Alternatively, collect 5-10 documents of your choice on a specific domain (e.g., AI, medicine, law, literature)

3. Organize these documents in a 'documents' folder in your project directory

Task 1.3: Implement Basic Document Loading Implement a function to load documents from different file formats (PDF, DOCX, TXT), preserving metadata and handling potential errors.

Part 2: Document Processing and Embedding (25 points)

Task 2.1: Implement Document Splitting Implement a function to split documents into appropriate chunks for retrieval: - Create a text splitter that handles different document types - Experiment with different chunk sizes (small, medium, large) - Implement chunk overlap to prevent information loss at boundaries - Preserve document metadata in the chunks

Task 2.2: Implement Document Embedding Implement embedding generation for document chunks: - Use at least two different embedding models from the Sentence Transformers library - Create a FAISS vector store to efficiently store and search embeddings - Implement saving and loading of vector stores to/from disk

Part 3: Implementing Different Retrieval Strategies (25 points)

Task 3.1: Basic Similarity Search Implement basic similarity search to retrieve relevant documents: - Create a function that takes a query and returns the most similar documents - Display metadata and content snippets for retrieved documents - Experiment with different retrieval parameters (k value)

Task 3.2: Advanced Retrieval Strategies Implement at least one advanced retrieval strategy: - Maximum Marginal Relevance (MMR) for diversity in results - Metadata filtering based on document properties - Hybrid search combining keyword and semantic search

Part 4: Integrating with LLMs for Generation (20 points)

Task 4.1: Implement RAG Prompt Templates Create effective prompt templates for your RAG system: - Design a standard prompt template for general queries - Create specialized templates for different types of questions - Implement prompt components that effectively utilize retrieved context

Task 4.2: Implement the RAG Pipeline Integrate retrieval with generation: - Connect your retrieval system to an LLM of your choice - Implement the full RAG chain using LangChain - Create a query interface for testing your system - Handle edge cases (no relevant documents found, etc.)

Part 5: Evaluation and Performance Analysis (20 points)

Task 5.1: Implement Evaluation Metrics Implement functions to evaluate your RAG system: - Create metrics for retrieval performance (precision, recall, F1) - Implement evaluation for answer quality - Create a testing framework with sample queries and expected results

Task 5.2: Compare Different RAG Configurations Create a function to compare different configurations of your RAG system: - Test different embedding models - Compare different retrieval strategies - Evaluate different prompt templates - Analyze trade-offs between different configurations

Bonus Challenge: Advanced RAG Techniques (10 points)

Implement one or more of the following advanced RAG techniques:

1. **Query Rewriting**: Use an LLM to rewrite the user's query to improve retrieval performance
2. **Post-Retrieval Filtering**: Implement a filter that removes irrelevant documents after retrieval
3. **Hierarchical Retrieval**: Implement a two-stage retrieval process with coarse-grained and fine-grained steps
4. **Self-Correcting RAG**: Implement a system that evaluates its own answers and regenerates them if they're inadequate

Submission Requirements

Submit your work via a GitHub repository containing:

1. Your complete Python code files with all implementations
2. A README.md file containing:
   - Setup instructions for running your code
   - A detailed description of your RAG system's architecture
   - Results from your experiments with different retrieval strategies
   - Evaluation metrics for different configurations
   - Analysis of strengths and weaknesses of your approach
   - Any challenges you encountered and how you addressed them
3. The document corpus you used (or instructions for downloading it)
4. Any configuration files or saved model states

Submit the GitHub repository URL on the course submission portal by the deadline. Make sure your repository is public or that you have added the course instructors as collaborators.

Grading Criteria

- Part 1 (Environment Setup): 20 points
- Part 2 (Document Processing and Embedding): 25 points

- Part 3 (Retrieval Strategies): 25 points
- Part 4 (LLM Integration): 20 points
- Part 5 (Evaluation): 20 points
- Bonus Challenge: 10 points
- Code quality, documentation, and analysis will be considered in the evaluation

References

- "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" - Lewis et al.
- LangChain Documentation: https://python.langchain.com/docs/get_started/introduction
- Sentence Transformers Documentation: https://www.sbert.net/
- FAISS Documentation: https://github.com/facebookresearch/faiss
- "The Illustrated RAG: Retrieval-Augmented Generation" (blog post)
- Chapter 9: LLM Workflows from the textbook