Generative AI, Assignment # 3

Department of Computer Science
National University of Computer and Emerging Sciences,
Islamabad, Pakistan
Instructor: Dr. Akhtar Jamil

Due Date: April 30, 2025

Instructions

- Each student must submit the following three files packaged into a single ZIP file and named as **ROLLNO_NAME.ZIP**:
  - A Jupyter Notebook (`.ipynb`) or Python script (`.py`) with the complete implementation.
  - A detailed PDF report containing all details of your implementation written in LaTeX using Overleaf, following the Springer's LNCS paper format:
    Springer LNCS Template on Overleaf.
  - A plain text file (`.txt`) containing all the GPT prompts used for each question.
- Ensure that the code is well-structured with proper comments for each function. Include all necessary dependencies to ensure the code runs without errors.
- There is a **grace time of 2 hours** after the submission deadline expires. You must verify that your submissions are correct. Any submission received after this slack time will be considered late, and NO marks will be awarded.

1 Task-1: Multimodal Retrieval-Augmented Generation (RAG) System

1.1 Objectives

This task involves the implementation of a complete Retrieval-Augmented Generation (RAG) pipeline using three provided PDF documents in the data folder

that contain both textual and visual (image-based) information. These documents may include financial statements, bar charts, graphs, and other data-rich visuals.

You are required to extract, preprocess, chunk, embed, store, retrieve, and utilize this information to respond to natural language queries through a ChatGPT-like interface. The system should be capable of handling both text-based and image-based queries and returning contextually relevant responses using an integrated language model.

Additionally, you must test using advanced prompting strategies such as *Chain-of-Thought (CoT) prompting*, *Zero-shot or Few-shot prompting*, and effective *Prompt Engineering* practices to enhance reasoning and output quality. In your report, include the proper prompts composed and their corresponding responses from your implemented pipeline.

1.2 Requirements

- **Data Extraction and Preprocessing**
  - Parse all three provided PDF files.
  - Extract all textual content including financial figures, tabular data, and paragraphs.
  - Extract and analyze image content (charts, plots, bar graphs, etc.) using OCR and image processing tools.
  - Store each extracted component (text/image with captions or labels) as a separate chunk along with metadata.
- **Text and Image Embedding**
  - Convert textual and visual chunks into dense embeddings using appropriate encoders:
    - Text: Sentence-BERT, OpenAI Embeddings, etc.
    - Image: CLIP, BLIP, or any Vision Transformer-based model.
  - Ensure embeddings are stored in a vector database such as FAISS, Chroma, or Weaviate with proper indexing and metadata tagging.
- **Semantic Search and Retrieval**
  - Implement similarity-based search using the vector database.
  - Enable users to input queries either in textual form or by uploading an image.
  - Retrieve and rank the most relevant chunks (text/image) for the given query.
  - Return results in a coherent format with references to source documents.
- **Language Model Integration**

  - Integrate an LLM (open-source such as LLaMA2, Mistral, GPT-J or a proprietary one such as GPT-4).
  - Use retrieved information as context to generate accurate and informative answers.
  - Utilize advanced prompting strategies (CoT, few-shot, etc.) to improve the depth and quality of responses.

- **User Interface**
  - Design a simple web-based ChatGPT-like interface with the following capabilities:
    * A text box for user queries.
    * An image upload option for visual queries.
    * Display of search results: ranked relevant chunks, charts, or interpreted responses.
    * Option to view the source PDF section from which the answer was generated.
  - The interface must be responsive and intuitive.

- **Evaluation and Visualization**
  - Visualize embedding space and search results.
  - Display retrieval hit rates and measure semantic similarity scores.
  - Evaluate final responses using metrics such as:
    * Relevance score (manual or cosine similarity)
    * BLEU / ROUGE for generative quality
    * Query response time

- **Documentation**
  - Provide a detailed technical report (in LaTeX using Springer LNCS format) covering:
    * End-to-end system architecture
    * Component-wise explanation (data processing, embedding, retrieval, LLM generation)
    * Evaluation metrics and results
    * Use of Prompt Engineering and CoT examples
    * Challenges and how they were addressed
  - Submit a prompt log (‘.txt‘) with all prompts used during testing and deployment.
  - Ensure reproducibility of the entire pipeline.

1.3 Expected Outcomes

- A fully functional RAG system capable of handling multimodal queries.
- Demonstration of reasoning and coherence in LLM responses via prompt strategies.
- Visualization of embeddings and retrieval operations.
- A deployable web interface with working semantic search.

Note: The following bonus task will only be evaluated if you have implemented the above mandatory question.

2 Bonus Task: Semantic Product Search and Ranking

2.1 Objectives

This bonus task explores the use of deep learning techniques for semantic product search and ranking. Unlike traditional keyword-based search systems, semantic product search aims to understand the meaning of a user's natural language query and return the most contextually relevant products.

You are required to develop a deep learning-based solution that accepts a query, retrieves a set of candidate products from a product catalog (based on their titles and descriptions), and ranks them according to semantic relevance. The model should be trained on a dataset consisting of query-product pairs and deployed as a web application that serves search results to users in real-time.

Your solution must incorporate a modern text representation method (e.g., word embeddings, BERT) to encode the query and product information and learn relevance scores. The final output must present ranked product results that are most semantically aligned with the user query.

Deployment Requirement: Create a web interface with a simple text box where the user can enter a query. The query should be passed to the trained model, which returns and displays ranked products based on relevance. Use both `product_title` and `product_description` fields in training and inference.

2.2 Requirements

- Load the dataset and combine the `product_title` and `product_description` columns for product representation.
- Apply text preprocessing steps: convert to lowercase, remove stop words, apply lemmatization or stemming, and remove special characters.
- Transform text data into numerical representations using:

  - TF-IDF
  - Word embeddings (e.g., Word2Vec, GloVe, FastText)
  - Pretrained models (e.g., BERT, GPT)
- Split the dataset into training, validation, and test sets (e.g., 70%-15%-15%).
- Train a suitable deep learning model to learn relevance between queries and products.
- Fine-tune the model's hyperparameters to optimize performance.
- Visualize training and validation loss curves.
- Evaluate the model using appropriate ranking metrics such as NDCG, MAP, Precision@K, Recall@K, and F1@K.
- Deploy the model in a web application that takes a query and returns a ranked list of relevant products.
- Prepare a detailed technical report explaining:
  - Model architecture and design choices
  - Training strategy and preprocessing steps
  - Evaluation methodology and results (including performance table)
  - Challenges encountered and how they were addressed

Recommendation: Consider using transformer-based models such as BERT for encoding queries and product descriptions to improve semantic understanding.

2.3 Dataset

The Amazon Shopping Queries Dataset contains query-product pairs labeled for semantic relevance. Each product is described using a `product_title` and `product_description`.

- Dataset Link: `https://github.com/amazon-science/esci-data/tree/main/shopping_queries_dataset`
- Dataset Description: `https://github.com/amazon-science/esci-data`

Tools and Deployment Recommendations

These are the recommended tools that you can work with, but not mandatory for implementation of your Task-1.

Recommended Tools for Multimodal RAG Implementation

To implement the Multimodal Retrieval-Augmented Generation (RAG) assignment, students are expected to utilize a range of tools spanning document parsing, image processing, embedding generation, vector storage, LLM integration, and interface development. The following categories outline the recommended toolchain:

1. Document and Image Processing

- **PDF Parsing:** `PyMuPDF`, `pdfminer.six`, `Unstructured`
- **Image Extraction:** `pdf2image` for rendering PDF pages as images
- **OCR (Image to Text):** `Tesseract OCR`, `EasyOCR`, `PaddleOCR` for extracting text from charts and figures

2. Embedding Generation

- **Text Embeddings:** `SentenceTransformers` (e.g., `all-MiniLM-L6-v2`), `OpenAI Embeddings`
- **Image Embeddings:** `CLIP` (Contrastive Language–Image Pretraining), `BLIP`, or Vision Transformer-based encoders

3. Vector Storage

- **Local Solutions:** `FAISS`, `Qdrant, Chroma`
- **Cloud Services:** `Pinecone`, `Weaviate`

4. Language Models (LLMs)

Local Deployment:

- `LLaMA 2` (7B/13B/70B)
- `Mistral`, `GPT-NeoX`
- `Ollama`: A tool to run open models like `LLaMA 2`, `Mistral`, `Phi-2` locally with minimal setup
- `LM Studio`: GUI to load and interact with models locally

Cloud APIs:

- `OpenAI GPT-4`: Requires API key (limited free credits available)

- `Anthropic Claude`: Paid tier with limited trial
- `OpenRouter.ai`: Gateway to several LLMs with a mix of free and paid options

5. **Prompt Engineering and Reasoning Tools**
   - **Chain-of-Thought (CoT)** prompting for multi-step reasoning
   - **Few-shot and Zero-shot** prompting strategies
   - `LangChain`: Framework for chaining LLM and vector store operations
   - `LlamaIndex`: Interface for ingesting and indexing documents into RAG pipelines
   - `PromptLayer`: For logging, debugging, and managing prompt performance
6. **Interface and Application Development**
   - `Streamlit`, `Gradio` for rapid UI/UX development
   - Required features include:
     - Text input for queries
     - Image upload support for image-based search
     - Display of retrieved chunks (text/images) with source context
     - Generated answers using the LLM

Evaluation Guidelines

Retrieval Quality:

- Precision@K, Recall@K, Mean Average Precision (MAP)

Generation Quality:

- BLEU, ROUGE scores
- Human evaluation of relevance, coherence, and correctness

System Metrics:

- Query response latency
- Embedding generation and storage efficiency

Visualization (Optional but Encouraged):

- TSNE/PCA plots for embedding space
- Charts for training curves, memory usage, query hit rates