Homework 4 Building a RAG Application with Streamlit

Due Date: 12/6/2024 11:59pm

1 Introduction

This module combines the elements of language models that were studied in part 3 of this series. With encoders and generative models, you will create an application that can read PDFs. The application then uses LLMs to answer questions based on the PDFs, otherwise known as retrieval-augmented generation (RAG).

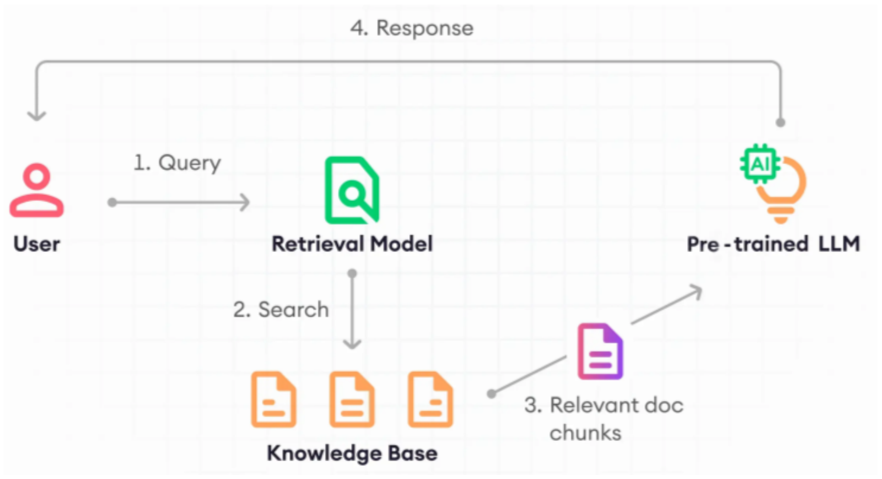

Figure 1: A diagram for retrieval-augmented-generation (RAG)

2 Streamlit Application

Streamlit allows for rapid development of data apps with minimal code. For this first part, you will ensure that your application can run without any heavy lifting by LLMs.

2.1 Setting up Python and Streamlit

1. Install Python locally from the official site (Python 3.10). Choose the installer that best suits your system.
2. Download and install an Integrated Development Environment (IDE). As default, trying installing VSCode at Visual Studio Code (Microsoft).
3. Download our provided folder of starting materials and save it to your preferred location. Open VSCode and then select "File- Open Folder" to open the project.
4. (Recommended) In VSCode, install Python extensions, which will make your code easier to read and provide hints.

5. (Recommended) Open a terminal within VSCode, and set up a virtual Python environment. This is a good practice to avoid package dependency conflicts with other projects.
6. We provided a requirements file, so now just run **pip install-r requirements.txt** to install required packages.
7. In the terminal, do **streamlit run app.py** to start the application. It will not be fully functional until you complete the TODOs in the code.

2.2 Application Structure

The app will have a modular structure, with separate files for the application interface, database management, document encoding, generative model integration, and PDF parsing.

2.2.1 app.py

1. **run_parser**: Use the encoder to compute embeddings for PDF pages found by the parser
2. **get_top_documents_for_RAG**: Use the encoder to compute a query vector and send it to the database.
3. **add_context_to_query**: Add the context retrieved from the database and add it to the chat history.
4. TODO Complete the application initialization of the encoder with Local encoder.
5. TODO Complete the application initialization of the LLM with Mistral.
6. TODO Experiment with prompt design in function add context to query and for SYSTEM PROMPT.

2.2.2 database.py

Manages an in-memory database using Python dictionaries. Stores uploaded documents and their encoded vectors for quick retrieval and comparison.

2.2.3 encoders.py

Handles document encoding using a pre-trained model. Converts uploaded documents into vectors that can be used for semantic similarity searches.

- **SillyEncoder** - a toy encoder that encodes sentences by counting each of the 26 letters of the alphabet.
- **HuggingFaceEncoder** - A sentence-transformers model: It maps sentences and paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search. **_mean_pooling** and **_cls_pooling** are used to collapse token embeddings down to a single vector to represent an entire document.

2.2.4 generative.py

Integrates the generative model, sending context and questions to the model, and receiving generated answers to display in the Streamlit app. The generative models return in "streaming" mode so that the Streamlit app can update visually on the go.

- SillyLLM- a toy generation model that simply counts how many messages there are and echos it back.
- LocalLLM
  - TODO In **generate_ollama_directly**, call **ollama.chat** on the messages to generate a response. Enable streaming.

2.2.5 parsers.py

Extracts text from uploaded PDFs using a naive pdfminer.six page-by-page parse. This is done for you, as parsing is a terrible problem and not the point of this tutorial. If you are interested, LayoutPDFReader seems to be one of the most powerful PDF parsers at time of writing.

3 Integrating LLMs

3.1 Loading and Using Encoders

Load an encoder model, which is BERT, to convert documents into vectors. This will facilitate semantic search across the uploaded documents. Luckily, we are not training the encoder and so using a local model will not be hardware strenuous. Most CPUs should handle it. We don't use the OpenAI API because it will cost a small amount of money (OpenAI's embedding models can embed 9,000 pages for $1.).

TODO Download a model from Sentence-Transformers as they are designed for retrieval. They can be access through huggingface here. For example, **model = AutoModel.from pretrained('sentence-transformers/all-mpnet-base-v2')**. Most of these models use mean pooling, in contrast with the original BERT model we tried before which uses [CLS] pooling.

3.2 Integrating the Generative Model

By using Ollama, it can care of everything downloading, quantization, formatting, installation. It will also automatically handle GPU usage, if you have it. TODO To pursue this assignment with a local model, follow these additional steps:

1. Download and install from Ollama website.
2. Check that the Ollama python library was properly installed from our requirements.txt with **import ollama**. Here is the link of introduction to ollama here.
3. Before running **app.py**, open a second terminal in Vscode (or on your system directly Windows, Linux, Mac) and run **ollama serve**. This will create a ollama server at port 11434. Also, **ollama pull [model_name]** if needed.
4. Complete your **generate_ollama_directly**. Setting **stream=True** modifies functions to return a Python asynchronous generator

4 Questions

1. The application can run with and without PDFs uploaded. Repeat a few questions in each case, what behavior changes? (Provide the question you asked and the PDF you uploaded.)
2. The parser divides PDFs page by page to store in the database. When will it be difficult to answer a user a question?
3. Copy paste your system prompts that you have tried, and describe the behavior of each prompt you tried. (At least try three different prompt.)

5 What to turn in

1. A PDF report that includes:
   (a) A screenshot of your LLM PDF app (please includes the search engine in your screenshot)
   (b) All you answers from the part 4 questions.