Ema Intern Take-Home Challenge

Initial Exploration

Documenation of Approach

Area of Improvement

Submitted By

Diksha Barnwal

IIT Patna

Initial Exploration

Initially, I explored various open-source models and tools to understand query-answer systems. I started with a basic query-answer system using Ollama embeddings for the first set of lecture notes. However, the retrieval accuracy was not satisfactory. This led me to explore other open-source models like HuggingFace models, FLAN-T5, and others. Despite some improvements, the results were still not optimal. Eventually, I found a model that performed better than the previously used LLMs, so I decided to stick with it. I recognized that using a more advanced API, like OpenAI, might yield different and potentially better results.

Approach and Implementation

Here's a clear and well-documented description of the approach I took to create a functional implementation for processing and querying data from web articles and GitHub repositories.

1. Data Ingestion

GitHub Repository Data:

- **Module**: `github`
- **Functionality**: Retrieve README file content from a specified GitHub repository.
- **Reason**: Provides structured and relevant information about the repository's content.

Web URLs Data:

- **Module**: `SeleniumURLLoader` from `langchain`
- **Functionality**: Load dynamic web pages to capture content rendered by JavaScript.
- **Reason**: Allows extraction of content from complex web pages that are not fully static.

2. Data Processing

Markdown to HTML Conversion:

- **Module**: `markdown`, `BeautifulSoup`
- **Functionality**: Convert README content from Markdown to HTML and extract specific sections.
- **Reason**: Enables easy parsing and extraction of relevant sections from the README.

3. Intermediary Representation

Text Splitting:

- **Module**: `RecursiveCharacterTextSplitter` from `langchain`
- **Functionality**: Split long texts into smaller, manageable chunks with overlap.
- **Structured Documents**: I tokenize and split the content into manageable chunks, each stored as a `Document` with associated metadata.
- **Embeddings**: Each document chunk is converted into embeddings using a model, allowing for efficient similarity searches.
- **Reason**: Ensures chunks are contextually complete and improves retrieval efficiency.

4. Embedding Generation

Embedding Model:

- **Module**: `HuggingFaceInstructEmbeddings`
- **Functionality**: Generate embeddings for the text data.
- **Reason**: Provides a semantic representation of the text for efficient similarity search.

5. Question Answering

Language Model:

- **Module**: `transformers`, `HuggingFacePipeline`
- **Functionality**: Generate answers to queries using a pre-trained language model.
- **Reason**: Provides natural language responses to user queries based on the data.

RetrievalQA Setup:

- **Module**: `RetrievalQA` from `langchain`
- **Functionality**: Combine the retriever and language model to answer questions.
- **Reason**: Provides a complete system for query-based responses.
- **Pipeline**: The `pipeline` function simplifies the process of using a pre-trained model for a specific task. Here, the task is "text2text-generation", which means the model will generate text based on the input text.
- **Parameters**:
    - `model` and `tokenizer`: Specify the model and tokenizer to use.
    - `max_length=512`: Limits the maximum length of the generated text to 512 tokens.
    - `temperature=0.2`: Controls the randomness of the text generation. Lower values make the model more deterministic.
    - `do_sample=True`: Enables sampling, allowing for more varied outputs.
    - `top_p=0.9`: Implements nucleus sampling, where the model considers the smallest set of tokens whose cumulative probability is greater than or equal to `0.9`.
    - `repetition_penalty=1.15`: Applies a penalty to repeated tokens to reduce repetitive text in the output.

- **Integrating with LangChain**
- `llm = HuggingFacePipeline(pipeline=pipe)`
- Finally, I wrap the HuggingFace pipeline using the `HuggingFacePipeline` class from LangChain. This allows me to integrate the text generation capabilities of the BART model into the LangChain framework, which facilitates the retrieval and question-answering functionalities of the system.

Vector Store and Retriever:

- **Module**: `Chroma`
- **Functionality**: Store document embeddings and retrieve relevant chunks.
- **Reason**: Efficiently handles similarity search and retrieval of relevant document sections.

Citation System

Display sections from lecture notes to show how conversational answers were constructed, proving there wasn't any hallucination.

Strategies:

1. **Reference Extraction:** Implement a system to extract relevant sections from lecture notes or other references used to construct the answer.
2. **Citation Display:** Develop a method to display these citations alongside the answer, providing transparency.
3. **Source Management:** Maintain a database of sources and references for easy retrieval and citation.

Areas for Improvement

1. **Web Interface with Streamlit:** Develop a user-friendly web interface using Streamlit for easy deployment and interaction.
2. **Enhanced Data Ingestion**: I aim to automate the discovery of new documents and web pages.
3. **Advanced Embedding Models**: I plan to experiment with more advanced models for better performance.
4. **Improved Question Answering**: I will incorporate feedback loops to continuously improve response quality.

This approach ensures a flexible, efficient, and scalable system for processing and querying data from diverse sources.