Table of Contents

Introduction

Retrieval Augmented Generation (RAG) is a technique, enhancing the accuracy and reliability of generative AI models by supplementing them with factual information extracted from external sources (Merritt, 2023). This innovative approach not only empowers the model to harness its innate language generation capabilities but also enables it to capably respond to queries tailored to a specific domain.

In this RAG project, the focus was on training Ollama.ai's "mistral" Large Language Model (LLM) using a publicly available food recipes dataset sourced from Kaggle (Mooney, n.d.). Our primary objective revolves around ensuring that the RAG framework is capable of retrieving and generating recipes in response to user queries. This entails the ingestion of recipe data from a PDF file, subsequent processing and storage of this data to facilitate swift retrieval, and the utilisation of the LLM to generate informative responses to user requests.

This RAG system, by streamlining recipe retrieval and generation, is aimed at providing a promising solution to this dilemma.

Installing Ollama

To kickstart the implementation of the RAG system, the initial step is installing and running Ollama on the local environment from *https://ollama.com*. Although initially attempted on Google Colab, **I** encountered issues during response generation, with ollama servers returning 99 and 404 error codes. So, the code was transitioned to Jupyter Notebook and executed on the local system

Domain and Dataset

In our RAG system, the choice of dataset played a crucial role in shaping its effectiveness and applicability. The dataset utilised for this project is known as RecipeNLG, sourced from Kaggle (Mooney, n.d.). This dataset comprises of a staggering 2,231,142 cooking recipes, making it a rich source of culinary knowledge. Originally employed in the research of semi-structured text generation (Bień et al., 2020), the RecipeNLG dataset offers a diverse collection of recipes.

However, given the computational constraints in processing such a vast dataset, a subset of 10,000 recipes was randomly sampled from the RecipeNLG dataset to train the LLM model.

Data Ingestion

After the dataset was sampled, it underwent conversion into a PDF format using R programming language. Subsequently, to facilitate its integration into the Retrieval Augmented Generation (RAG) system, the *UnstructuredPDFLoader* package was employed within the Python environment from LangChain (www.restack.io, n.d.).

The selection of *UnstructuredPDFLoader* over alternative packages was primarily motivated by its commendable ease of use and adaptability within the LangChain framework (www.restack.io, n.d.). Featuring a user-friendly interface, this module simplifies the loading of PDF documents, thereby facilitating seamless data ingestion and subsequent analysis. Its compatibility with the LangChain ecosystem not only ensures a smooth integration process but also enhances the system's flexibility by enabling the loading of diverse file formats.

Chunking and Embedding

With the necessary data integrated, the next phase in constructing a functional Retrieval Augmented Generation (RAG) system is chunking. This process involves segmenting the document into smaller, more manageable sections that sticks to the context window of the large language model (Eteimorde, 2023). While Langchain offers various chunking methodologies, *RecursiveCharacterTextSplitter* emerged as the optimal choice, as recommended by (Eteimorde, 2023).

RecursiveCharacterTextSplitter functions by partitioning the provided text into smaller segments based on a specified chunk size. By fine-tuning the hyperparameters, particularly *chunk size* and *chunk overlap*, the process yielded refined chunks favourable for subsequent processing.

After obtaining the text chunks, the next phase was embedding. For this task, I opted for Ollama.ai's *nomic-embed-text* as the embedding model. According to (Rastogi, 2024), Nomic Embed demonstrates superior performance across a range of embedding dimensions, from 64 to 768. Notably, it outperforms other models particularly at 512 and 768 dimensions, achieving remarkable memory reduction while maintaining performance levels comparable to advanced models. Utilising a BERT (Bidirectional Encoder Representations from Transformers) base with additional optimisations, Nomic Embed is an efficient solution for text embedding (Rastogi, 2024).

Upon embedding the text, the embedded representations are stored in a vector database created using *chromadb*. As highlighted by Dwyer (2023), Chroma stands out as an open-source database renowned for its competence in storing vector embeddings. Noteworthy features of Chroma include its emphasis on ease of use, scalability, and adaptability. Engineered for speed, Chroma facilitates swift retrieval and processing of vector embeddings—an important aspect for the seamless operation of a successful RAG system.

The process resulted in the Nomic Embed model embedding 1098 chunks produced by the *RecursiveCharacterTextSplitter*, with hyper-parameters of chunk size as 7500 characters and chunk overlap as 100 characters.

Prompt design

The prompt design employed in the RAG system is a critical component that enables effective interaction between users and the AI model. Designed to aid the model in generating relevant responses, the prompt template serves as the framework for communication within the system.

At its core, the prompt template sets clear expectations for users by framing their queries within the context of an AI language model assistant. By explicitly stating the task at hand - generating multiple perspectives on the user question to retrieve relevant documents from a vector database - the template ensures that users understand the purpose of their interaction with the system.

In this system, a prompt template object was first created using the *PromptTemplate* class. Then we define an input variable where we accept the question to the prompted to the AI model. Then we define the template of the prompt. This is done in 2 steps:

- The introductory text informs the AI assistant of its role and task: **"You are an AI language model assistant. Your task is to generate five different versions of the given user question to retrieve relevant documents from a vector database."**

- Then we explain the purpose of generating multiple versions of the user question, which is to enhance the retrieval process: **"By generating multiple perspectives on the user question, your goal is to help the user overcome some of the limitations of the distance-based similarity search."**

Overall, the prompt design plays a crucial role in facilitating seamless interaction and knowledge discovery within the RAG system.

Setting up Retrieval

This section of code sets up the retriever component within RAG system. Initially, an instance of the *ChatOllama* class is created, specifying the LLM to be used as “mistral”. This initialises the language model (LLM) from Ollama.ai, which will be utilised for generating responses. Then, the *MultiQueryRetriever* is instantiated using the *from_llm* method. This retriever is configured with the vector database, as the source for document retrieval. Additionally, the prompt template *query_prompt* is provided to guide the retrieval process.

Finally, a processing chain is constructed using the defined components: retriever for document retrieval, prompt for guiding the response generation process, LLM for generating responses, and finally parsing the output into a readable format. This chain controls the flow of information and interactions within the RAG system, enabling communication between users and the AI model.

Testing

To thoroughly assess the efficacy of the RAG system, it was essential to evaluate its adherence to the provided context and its inclination to rely on pre-existing knowledge. Five test queries were posed, including one deliberately chosen to lie beyond the system's existing dataset. The system adeptly recognised the lack of information within the provided context for this query, transparently indicating its limitation before drawing from its internal knowledge base to offer a response.

In contrast, for the remaining queries, the system delivered precise and detailed responses consistent with the dataset provided. These responses showcased the system's proficiency in harnessing the available dataset to provide accurate recipe recommendations, thereby demonstrating its capability to operate within the prescribed context while drawing upon its knowledge base when necessary.

Limitations

The foremost limitation of this RAG system lies in its scalability and performance. Had we not downsized the dataset from its original size of over 2 million recipes, the computational demands would have been substantial, presenting a significant cost challenge.

Another notable drawback pertains to the system's contextual understanding. While it proficiently retrieves relevant recipes based on user queries, its capacity to understand nuanced or ambiguous queries may be constrained. Consequently, there is a risk of delivering potentially irrelevant responses.

Moreover, the system's response generation capability is constrained by the dataset available during training. Without access to updated data, it cannot integrate new recipes or adapt to evolving culinary preferences.

Conclusion

The Retrieval Augmented Generation (RAG) system developed in this project demonstrates the potential of integrating large language models with external data sources to provide domain relevant responses. By leveraging the RecipeNLG dataset and the powerful mistral language model from Ollama.ai, the system showcases its capability to provide precise recipe recommendations in response to user queries.

While the current implementation exhibits promising performance within the culinary domain, there remain opportunities for further enhancement and expansion. Addressing scalability concerns and incorporating mechanisms for continuous learning and adaptation could significantly broaden the system's applicability.

Ultimately, this RAG system serves as a testament to the transformative potential of combining state-of-the-art language models with domain-specific knowledge bases. As natural language processing techniques continue to evolve, such systems hold the promise of revolutionising human-machine interactions across diverse domains, encouraging more intuitive and knowledge-driven experiences.

References

- Bień, M., Gilski, M., Maciejewska, M., Taisner, W., Wisniewski, D. and Lawrynowicz, A. (2020). *RecipeNLG: A Cooking Recipes Dataset for Semi-Structured Text Generation*. [online] ACLWeb. doi:https://doi.org/10.18653/v1/2020.inlg-1.4.

- Dwyer, J. (2023). *Exploring Chroma Vector Database Capabilities | Zeet.co*. [online] zeet.co. Available at: https://zeet.co/blog/exploring-chroma-vector-database-capabilities#:~:text=Chroma%20is%20an%20open%2Dsource [Accessed 2 Jun. 2024].

- Eteimorde, Y. (2023). *Understanding LangChain's RecursiveCharacterTextSplitter*. [online] DEV Community. Available at: https://dev.to/eteimz/understanding-langchains-recursivecharactertextsplitter-2846.

- Merritt, R. (2023). *What Is Retrieval-Augmented Generation?* [online] NVIDIA Blog. Available at: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/.

- Mooney, P. (n.d.). *RecipeNLG (cooking recipes dataset)*. [online] www.kaggle.com. Available at: https://www.kaggle.com/datasets/paultimothymooney/recipenlg/suggestions?status=pending&yourSuggestions=true [Accessed 2 Jun. 2024].

- Rastogi, R. (2024). *Papers Explained 110: Nomic Embed*. [online] Medium. Available at:https://ritvik19.medium.com/papers-explained-110-nomic-embed-8ccae819dac2#:~:text=dataset%20using%20MRL.- [Accessed 2 Jun. 2024].

- www.restack.io. (n.d.). *LangChain unstructured PDF loader*. [online] Available at: https://www.restack.io/docs/langchain-knowledge-langchain-unstructured-pdf-loader [Accessed 2 Jun. 2024].