Retrieval-Augmented Generation (RAG) using Llama3 for the World of Percy Jackson

Rohith Ravindranath
San Francisco, CA, 94087, USA
rohithravin@gmail.com

June 24, 2024

Abstract

Recent advancements in artificial intelligence, specifically in the realms of language models and retrieval-augmented generation (RAG), have facilitated the development of sophisticated applications capable of handling complex question-answering tasks. This project harnesses these cutting-edge technologies to create an advanced question-answering system within the context of the Percy Jackson book series. Utilizing LangChain for seamless orchestration, a Vector Database for efficient embeddings and retrieval, and the Ollama API with Llama3 for robust language modeling, this system exemplifies the integration of retrieval-augmented generation with large language models (RAG+LLM). The end product, a user-friendly demo application built with Gradio, allows for interactive engagement with the RAG system, providing accurate and contextually relevant answers based on the rich content of the Percy Jackson books. This report details the implementation and integration of LangChain, Vector DB, and Ollama (Llama3), showcasing the potential of combining advanced retrieval mechanisms with powerful generative models in creating specialized AI-driven solutions.

Keywords: Retrieval-Augmented Generation (RAG); Large Language Model (LLM); Generative AI; LanChain; Gradio; Vector Database; Embeddings; Llama3; Chroma DB

1 Introduction

Artificial intelligence has seen significant advancements in recent years, particularly in the domains of language models and retrieval-augmented generation (RAG). These advancements have paved the way for the development of sophisticated applications that can handle complex tasks, such as question-answering systems. Traditional question-answering systems have relied heavily on predefined rules and limited datasets, often leading to incomplete or inaccurate responses. However, the integration of RAG has revolutionized these systems by combining the retrieval of relevant information with the generative capabilities of advanced language models, thereby enhancing their accuracy and contextual relevance.

RAG systems operate by first retrieving pertinent information from a large dataset and then using this information to generate coherent and contextually appropriate responses. This approach leverages the strengths of both retrieval and generation, resulting in a more robust and reliable performance. Recent studies have demonstrated the efficacy of RAG in various applications, from conversational agents to educational tools.

This project aims to leverage these technologies to create a powerful question-answering system specifically tailored to the content of the Percy Jackson book series. By utilizing LangChain for orchestration, a Vector Database (Vector DB) for embeddings and retrieval, and the Ollama API with Llama3 for language modeling, this system exemplifies the potential of RAG combined with large language models (RAG+LLM). The use of LangChain ensures seamless interaction between the different components of the system, while the Vector DB facilitates efficient storage and retrieval of embeddings. The Ollama API with Llama3 enhances the generative capabilities, allowing for the production of high-quality and contextually accurate responses.

To demonstrate the practical application of this system, a demo app was developed using Gradio, an interface that allows users to interact with the RAG system in a user-friendly manner. This interactive demo showcases the system's ability to provide precise and contextually relevant answers to questions based on the Percy Jackson books, highlighting the effectiveness of the RAG+LLM approach.

The following sections of this report will delve into the implementation and integration of LangChain, Vector DB, and Ollama (Llama3).

2 Components

2.1 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) represents a sophisticated approach to natural language processing (NLP) that combines the strengths of information retrieval and language generation. Traditional language models generate responses based solely on the input text and their internal parameters, often lacking the ability to access and incorporate external knowledge dynamically. RAG addresses this limitation by integrating a retrieval mechanism that searches for relevant documents or passages from a large corpus, which are then used to inform and enhance the generation process. This dual mechanism allows RAG systems to produce more accurate, contextually relevant, and knowledge-rich responses.

RAG operates through a two-stage process: retrieval and generation. In the retrieval stage, the input query is transformed into a format suitable for retrieval by encoding it using a pre-trained model, such as a BERT-based encoder, to produce a dense vector representation. This dense query vector is then used to search a large corpus of documents, typically stored in a vector database. The search retrieves the most relevant documents or passages, which are also stored as dense vectors for efficient similarity comparison. These retrieved documents are ranked based on their relevance to the query, and the top-k documents are selected as the context for the generation stage.

In the generation stage, the selected documents are combined with the original query to form a contextual input. This input includes both the query and the relevant passages retrieved in the previous stage. A generative model, such as a transformer-based language model (e.g., GPT-3 or Llama3), processes the contextual input to produce a coherent and contextually relevant response. The model leverages the additional information provided by the retrieved documents to enhance the quality and accuracy of its output.

RAG systems are composed of several key components: an encoder that converts the input query into a dense vector representation, a retriever that searches a corpus to find documents similar to the encoded query, and a generator that produces the final response using both the original query and the retrieved documents. This combination allows RAG systems to deliver high-performance results in various applications.

One primary use case of RAG is in question answering systems. RAG excels in knowledge-intensive scenarios where answering questions requires accessing a vast amount of information, such as in scientific research, legal documents, or historical data. By retrieving relevant documents, RAG provides detailed and accurate answers. Additionally, in specialized domains like medical or technical support, RAG can fetch relevant guidelines or documentation, offering precise responses tailored to the domain-specific context.

RAG also significantly enhances conversational agents. In customer support applications, RAG can retrieve relevant information from FAQs, user manuals, and support tickets to provide accurate and contextually appropriate assistance to users. This improvement extends to content generation tasks, such as report writing and creative writing, where RAG can incorporate diverse pieces of information into stories or articles, enhancing the richness and depth of the content.

2.2 Vector Databases and Embeddings

2.2.1 Introduction to Vector Databases and Embeddings

A vector database is a specialized type of database designed to store and retrieve data in the form of high-dimensional vectors. These vectors, often called embeddings, are numerical representations of data points, such as words, sentences, images, or even entire documents. Embeddings capture the semantic meaning and contextual information of the data, enabling efficient similarity search and retrieval operations. By converting data into vectors, complex relationships and patterns can be uncovered and utilized in various machine learning and NLP applications.

Embeddings are typically generated using machine learning models like neural networks. For text data, models such as BERT, GPT, or Llama3 are commonly used to produce embeddings that encapsulate the meaning of words or sentences in a multi-dimensional space. These embeddings can then be stored in a vector database, where they can be quickly searched and retrieved based on their similarity to other vectors. This capability is crucial for tasks that require understanding and utilizing the contextual meaning of data, such as information retrieval, recommendation systems, and semantic search.

2.2.2 Usage of Vector Database and Embeddings in This Project

In this project, a vector database and embeddings are essential components for building an advanced question-answering system based on the Percy Jackson book series. The vector database stores embeddings of the book's content, created by encoding text passages using a pre-trained language model. These embeddings capture the semantic meaning and contextual relationships within the text, enabling efficient and accurate retrieval of relevant information.

When a user submits a query, the system first encodes the query into a dense vector representation. This vector is then used to search the vector database, retrieving the most relevant passages from the books based on their similarity to the query vector. The retrieval component, managed by LangChain, ensures that the top-k relevant documents are selected and passed to the generative model.

The Ollama API with Llama3 then uses these retrieved passages as context to generate a coherent and contextually appropriate response to the user's query. By leveraging embeddings and a vector database, the system can access and utilize the vast and detailed content of the Percy Jackson books, providing precise and informed answers. This process not only enhances the accuracy of the responses but also ensures that the answers are deeply rooted in the rich narrative world of the books.

2.3 LangChain

2.3.1 Introduction to LangChain

LangChain is a versatile framework designed to facilitate the seamless orchestration of complex workflows in natural language processing (NLP) and other machine learning applications. By providing tools and abstractions for chaining together various components and processes, LangChain enables the creation of sophisticated systems that can perform multiple, interdependent tasks efficiently. It is particularly useful in scenarios that require the integration of different models, data sources, and processing stages, ensuring that each component works harmoniously within the overall system.

2.3.2 Usage of LangChain in This Project

In this project, LangChain plays a critical role in orchestrating the different components required to build an advanced question-answering system based on the Percy Jackson book series. The project leverages LangChain to coordinate the interactions between the retrieval and generation stages, ensuring that each part of the process contributes effectively to the overall system's performance.

Specifically, LangChain manages the workflow by first using an encoder to transform user queries into dense vector representations. These vectors are then used by the retrieval component to search a Vector Database (Vector DB) containing embeddings of the book's content. LangChain handles the retrieval of the most relevant passages, which are then passed to the generative model provided by the Ollama API with Llama3. This model uses the retrieved context to generate accurate and contextually relevant responses.

By orchestrating these components, LangChain ensures that the system can efficiently handle user queries and provide detailed answers based on the rich narrative of the Percy Jackson books. Additionally, LangChain supports the integration with Gradio, which is used to build a user-friendly demo application. This application allows users to interact with the RAG system seamlessly, showcasing the practical benefits of LangChain in creating a sophisticated, interactive question-answering platform.

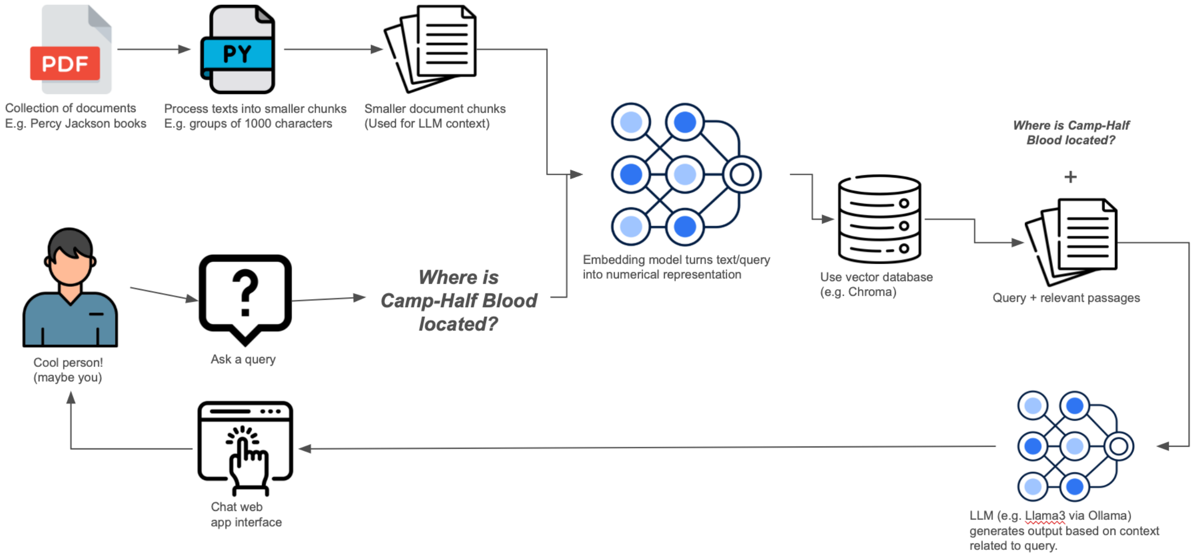
2.4 Ollama and Gradio

Ollama is a state-of-the-art API service designed to provide access to powerful language models, such as Llama3, which are used for a variety of natural language processing (NLP) tasks. These models are capable of understanding and generating human-like text, making them invaluable for applications like question answering, text generation, and conversational agents. Ollama simplifies the integration of advanced language models into applications by providing a robust and scalable API, allowing developers to leverage the capabilities of large language models without needing to manage the underlying infrastructure.

Gradio is a user-friendly interface-building tool that allows developers to create interactive web applications with minimal effort. It is particularly popular in the machine learning community for building demos and prototypes that allow users to interact with models and visualize their outputs in real-time. Gradio supports a wide range of input and output formats, making it versatile for various applications, from image classification to text-based tasks. Its simplicity and ease of use enable quick deployment and sharing of machine learning models with a broader audience.

3 Implementation

This project primarily utilizes Python 3 as the programming language and runs on a MacBook Pro M3. The following sections provide detailed insights into the implementation of specific components. The GitHub repository for this project can be accessed at `https://github.com/rohithravin/percy-jackson-rag/`.


PDF
Collection of documents
E.g. Percy Jackson books
PY
Process texts into smaller chunks
E.g. groups of 1000 characters
Smaller document chunks
(Used for LLM context)
Embedding model turns text/query into numerical representation
Use vector database (e.g. Chroma)
Where is Camp-Half Blood located?
+
Query + relevant passages
Cool person! (maybe you)
Ask a query
Where is Camp-Half Blood located?
Chat web app interface
LLM (e.g. Llama3 via Ollama) generates output based on context related to query.

Above is a flowchart for the implemented RAG!

3.1 Data Collection and Mining

All books within the Percy Jackson universe written by Rick Riordan were utilized for this project. Supplementary books such as The Demigod Files were excluded from the dataset. PDF versions of the books were initially collected and subsequently converted into Markdown format. This decision was driven by the complexity of converting PDFs, which often include graphics and images. Converting them to Markdown files ensured that only the textual content from the books was retained.

Furthermore, the Markdown files were segmented into chunks of 1000 characters with a 500-character overlap. This segmentation approach is standard practice in implementing Retrieval-Augmented Generation (RAG) systems. It enhances efficiency in retrieval, maintains contextual relevance, supports scalability, and optimizes memory usage, thereby improving the overall performance of the system.

3.2 Embeddings and Vector Database

Next, Chroma DB was selected for its suitability as an AI-native open-source embedding database, which aligns well with the requirements of this application. For generating embeddings within this database, we utilized nomic-embed-text from Ollama as our embedding model. This choice was motivated by its accessibility—being free to use and not requiring an API key—which streamlined the integration process.

It's worth noting that replacing this embedding function with more advanced alternatives like OpenAI embeddings or Amazon Bedrock would likely enhance the retrieval accuracy and responsiveness of the language models (LLMs). These alternatives could potentially improve the system's ability to retrieve and generate more precise and contextually relevant responses.

3.3 Connecting to Llama3

Finally, a script was developed to handle user queries in the system. Upon receiving a query from the user, the script embeds it and sends it to our vector database. The script retrieves the top 10 embeddings that are most similar to the query's embedding vector, along with their corresponding text passages. Subsequently, a prompt for Llama3 was constructed, incorporating both the retrieved passage and the original query. This formatted prompt is then passed to the LLM, which generates a response based on the provided context.

Below is the format used for the prompt passed to the LLM:

```
PROMPT_TEMPLATE = """
Answer the question based only on the following context:

{context}

---

Answer the question based on the above context: {question}
"""
```

4 Conclusion

Enjoy the demo video the of RAG in action at https://www.youtube.com/watch?v=d46U_W_QdF4!