Assistance Chatbot Exercise - Retrieval-Augmented Generation

Table of Contents

GEN AI PARTNERS

Use Case

An assistant generative AI agent to provide a community of users with guidance, precedents, and training for their day-to-day roles. The assistant will answer questions related to its own training or procedural documentation.

Exercise Instructions

1. Your goal is to build the back end of a successful chatbot Assistant using generative AI. You do not need to build the front end, although if it's easier to demonstrate, feel free to build it. Alternatively, you can show the success of your agent in your IDE of choice.
2. You will be provided with full guidance on how to complete this exercise within the upcoming sections. The intent is to help provide you with enough instruction or link-referenced options to make decisions and learn how to proceed.
3. You will use Microsoft Azure as your cloud service provider. Please create a free account to build your solution.
4. Please use the langchain framework and modules to build your chatbot.
5. Helpful links in the Implementation Details section provide you educational material to help you with the exercise.
6. The Sample Repositories to Reference can alternatively provide a jump start as well, although if utilized you'll need to determine how / where to customize the repo code to employ the strategies defined.
7. You can alternatively utilize code in the sample repositories to augment something that you personally build through the other reference links.
8. You may select the content that you will source for this exercise; please consider the target audience for the content covered by your generative AI assistant / agent, and explain why you chose the audience and related content.
9. Discuss strategies and implementation alternatives that you believe could have also worked, or even worked better for creating a helpful assistant.
10. Please ask questions as often as needed.
11. Feel free to use commercially available LLMs to help you close gaps in code. It's now expected that you utilize LLMs to improve your efficiency.

Strategies to Employ

Implementing a Retrieval Augmented Generation (RAG) approach to optimize search and retrieval must consider the characteristics of both:

- Source Content, and
- User Queries

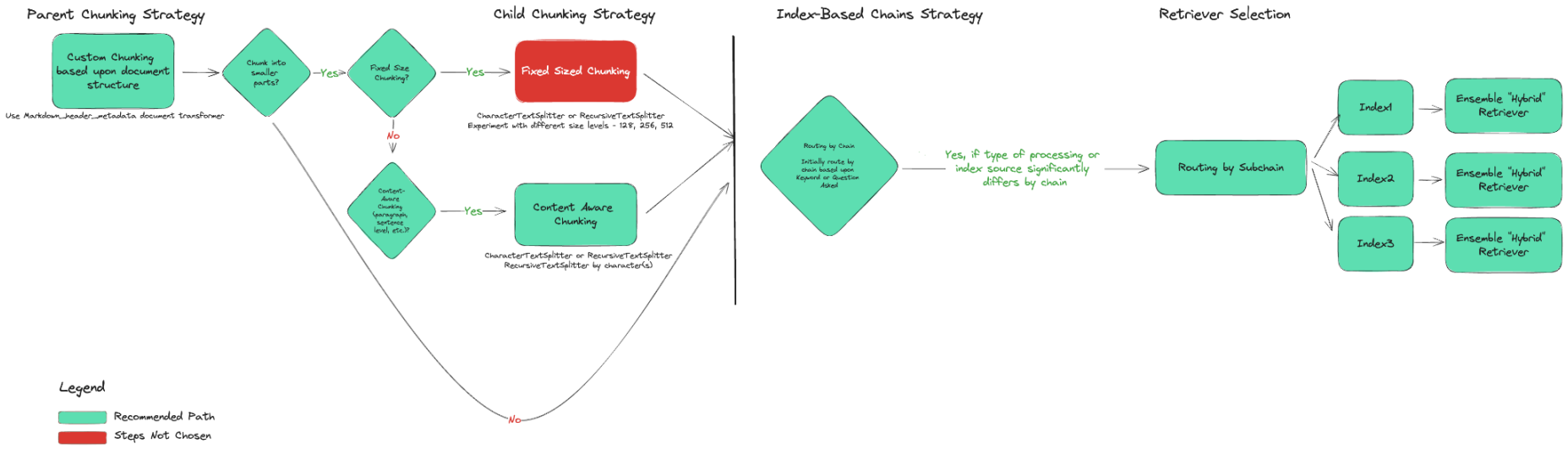
to determine the proper chunking, indexing, and retrieval strategies.

Below, we examine the specifics for each related to the above Use Case.

Category	Characteristics	Strategies to Employ
Source Content	**The Source Content** possesses a hierarchical structure, such as as: • Category • Subcategory • Chapter	**Chunk documents** aligned with the structure - either at the **section level** or the **paragraph level**. **Generate a list of searchable keywords** representing each chunk. **Attach metadata to each chunk** to represent **searchable keyword combinations** related to the chunk, along with a Title field and a link to the chapter (*i.e. parent document*).

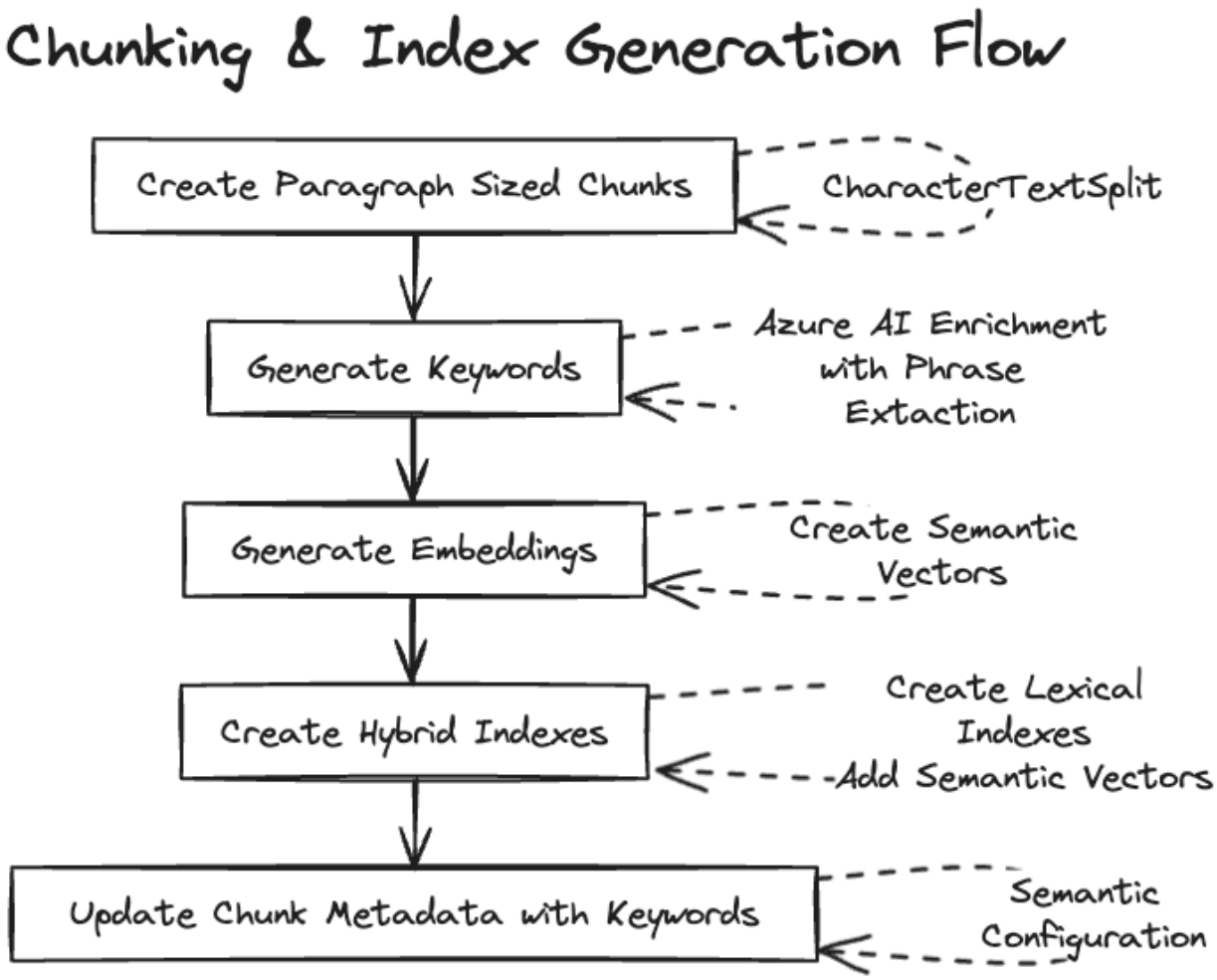
User Queries	More than half of the expected queries will require **content that spans multiple Chapters**, **and may also span multiple Subcategories**.	**Index documents** to employ a mix of indices representing keyword combinations (*representing queried "topics"*) for co-locating otherwise disparate chapter content to support accurate and performant cross-chapter retrieval through index pruning. **Implement the Ensemble Hybrid Retrieval strategy** to drive initially filtered results with BM25-based lexical search, then semantic search. **Note**: utilizing the **hybrid search strategy will auto-invoke Reciprocal Rank Fusion (RRF)** to run parallel queries for the multiple keywords associated with the query, decreasing retrieval latency. **Optionally, also test search semantic re-ranking** to alter the resulting "top k" retrieved results for the model to formulate the answer.

Given the above characteristics spanning both source content and queries, we employ an implementation pattern that invokes the following strategies reflected in green.



Implementation Details

The following flow summarizes the sequencing for chunking and indexing, with references to implementation concepts:



We can further break down the sequenced concepts into more concrete implementation steps, with reference to Microsoft Azure and Langchain 'how to' documentation step details, as applicable:

Step	Instruction	Extended Notes and 'How to' Reference Links
Create Paragraph Sized Chunks	Split text into chunks/passages with a targeted paragraph size. Utilize "\n\n" as the character to split content should avoid splitting a paragraph that includes bullets.	Uniform chunks facilitate fuller vector representation for semantic relevance to specific queries. **Reference Link** Langchain Character Text Splitter
Generate Keywords	Analyze text and identify important topics, entities, ideas for tags	Could use various NLP techniques like topic modeling, named entity recognition, concept extraction. Useful for classification, metadata generation, search optimization. **Reference Links** AI Enrichment to Generate Metadata (using Skillsets) Cognitive Skill - Key Phrase Extraction **-OR -** Docugami for Metadata Extraction
Generate Embeddings	Map chunks into numerical vector space	Represents semantic context. Allows similarity calculations for search matching. **Reference Link** Azure AI Search LangChain vector code sample

Create Hybrid Indexes	Create a lexical index, and then add semantic vector fields to it.	**Reference Links** Create an Index in Azure AI Search Add vector fields to a search index
Update Chunk Metadata with Keywords	Store keywords from text with paragraph chunks as metadata.	Associate metadata with chunks to drive filtering and ranking. **Reference Links** Enable Semantic Ranking Configure Semantic Ranking Azure AI Search LangChain vector code sample

Sample Repositories to Reference for Reusable Code

Azure-search-openai-demo
chat-with-your-data-solution-accelerator