Experiments and Evaluation Documentation

Vector Store: Elasticsearch

- As a document storage service I used Elasticsearch. Because I was not asked to use any other vector store for this specific problem I am unable to do practical comparison in terms of vector stores. One important thing to note is that my ultimate solution is using Llamaindex. Llamaindex does not support more complex and advanced retrievers integrated with ElasticSearch (like in this one https://docs.llamaindex.ai/en/stable/examples/vector_stores/chroma_auto_retriever/). Given flexibility on this constraint, one could go with Chroma or Pinecone as well to integrate with Llamaindex.

Experiments

Experiment 1: Retrieval Augmented Generation (RAG) System

Overview:
The first experiment involved implementing a **Retrieval Augmented Generation** (RAG) to develop the QA system. This approach integrated document retrieval with a language generation model to generate responses based on policy documents. To enhance document handling and retrieval, various chunking methods were experimented with, including Character Split, Recursive Character Split, and Semantic Split.

Implementation Details:

- Embedding Model: Utilized the OpenAI embeddings model (text-embedding-3-small) to create embeddings for policy documents. These embeddings were used to facilitate semantic search through the documents, ensuring that the queries could fetch relevant document sections.
- Document Store: Set up an Elasticsearch index (test_index_v2) to store the document embeddings, allowing quick retrieval based on semantic similarity. The index was refreshed to ensure data integrity and up-to-date retrieval capabilities.
- Retrieval: Configured a retriever with a similarity score threshold of 0.5 and set the number of documents to fetch (k=3). This was crucial to balance between retrieving too many irrelevant documents and missing out on relevant content.
- Generator: Integrated with gpt-3.5-turbo-0125 for generating responses based on the context provided by the retrieved documents. Used a pre-defined prompt from the LangChain hub (rlm/rag-prompt) to structure the interaction between the retriever's output and the language model.

Chunking Methods: Tested different methods to segment documents into manageable parts, enhancing retrieval:

- Character Split: Divided documents based on a fixed number of characters.
- Recursive Character Split: Divided documents recursively into smaller chunks whenever a segment exceeded a certain character limit.
- Semantic Split: Segmented documents based on semantic completeness, ensuring each chunk represented a coherent unit of meaning.

Thought Process:

- I have chosen RAG because it combines the precision of document retrieval with the generative capabilities of neural language models, aiming to provide precise and contextually accurate answers.
- Selected Elasticsearch due to its robust full-text search capabilities and the ability to handle large volumes of data efficiently, which is ideal for managing extensive policy documents.
- The decision to experiment with different chunking methods was driven by the need to optimize the retrieval process, making it more efficient and contextually relevant.

Pros and Cons:

- **Pros:**
    - Accuracy: By directly utilizing content from the relevant documents, the system can provide highly accurate and specific information without need for too much modeling.
    - In my manual checks, model handles well in case of totally irrelevant query (‘How is the weather today in Istanbul?’)
    - Adaptable: Different chunking methods allow for flexibility in handling various document sizes and complexities.
    - Can give more detail than Llamaindex and easy to do some prompt engineering while in the augmenting phase.
- **Cons:**
    - Speed OR Latency Issue: Retrieval and generation can be slow, particularly as the document corpus grows, due to the dual-step nature of the RAG system and the additional complexity introduced by document chunking.
    - I sensed that it requires more prompt engineering than Llamaindex to prevent hallucination.

Experiment 2: LlamaIndex with Chunking Variations

Overview:

The second experiment involved the use of LlamaIndex which I believe is tailored for applications that demand efficient data indexing and intelligent search capabilities, particularly suited for policy document handling. This experiment explored various configurations combining two chunking methods (Character Split and Semantic Split) with two structural approaches (using documents or nodes as the basic units for indexing).

Implementation Details:

Data Ingestion and Chunking:

- Character Split: This method was utilized to break down documents into manageable chunks by a fixed number of characters, either retaining these chunks as whole documents or further dividing them into nodes, smaller segments with added metadata.
- Semantic Split: A more sophisticated approach, where documents were divided into nodes based on semantic completeness rather than fixed character counts. This ensured that each unit contained a full idea or concept, which could enhance the relevance of search results and the coherence of answers generated from the nodes or chunks.

Data Indexing:

- Node Creation: Beyond simple text chunks, nodes created in the semantic split approach included metadata such as source document identifiers and contextual tags. This enrichment allows for more nuanced retrieval strategies, as nodes can be indexed and queried based on both their content and their relational metadata.
- Vector Storage: Employing Elasticsearch as a backend, both documents and nodes were converted into vector formats to support semantic querying. This vector store acted as the foundation for the retrieval operations, where search algorithms could quickly identify relevant segments based on query embeddings.

Query Interface:

- Retrieval Mechanics: The system was configured to prioritize the retrieval of data based on its semantic relevance to the query, with parameters set to identify the top relevant nodes or documents.
- Response Synthesis: Following retrieval, a response synthesizer was used to compile the information from the nodes into a coherent answer, ensuring that the output was not only accurate but also user-friendly.

Pros and Cons:

- **Pros:**
  - Pipeline construction with ease
  - Support more complex node relationship architecture I would be trying given more time
  - It handles irrelevant questions like weather in Istanbul well.
  - Precision in Retrieval: By indexing nodes that contain complete semantic units, the system can achieve higher precision in retrieving relevant information, reducing noise and improving user satisfaction.
  - Adaptability: The choice between document and node indexing provides flexibility to configure the system.
- **Cons:**
  - Indexing with nodes takes up a longer time than with documents.
  - Though handling irrelevant answers with appropriate responses, sometimes it provides irrelevant sources for some queries. But I believe it is a solvable problem with more methodical depth or changing some prompts in the source code regarding augmenting the queries.

Evaluation

Overview of My Evaluation Approach:

I mainly used open-source technique provided by LLamaindex (Relevancy Evaluator - LlamaIndex). Evaluating the retrieval-augmented generation system I built with LlamaIndex required a structured approach that could effectively simulate real-world usage without relying on manually labeled data. This method is particularly beneficial for systems dealing with dynamic information where the context and relevance of questions can frequently change.

1. Question or Synthetic Data Generation:

- Purpose: I aimed to generate a dataset of questions automatically from the chunked policy documents to test the system's ability to retrieve and generate relevant answers.
- Method: I used the DataGenerator class from LlamaIndex, which employs the gpt-3.5-turbo model from OpenAI to create questions. This approach ensures that the questions are directly tied to the document's content.

2. Generating Answers and Retrieving Context:

- Process: After generating questions, I used the LlamaIndex Query Engine to retrieve answers and source nodes. This part of the process checks how well the system uses its indexed data to provide accurate responses.
- Utility: This step is crucial as it directly tests the core functionality of the system—its ability to synthesize information in response to queries.

3. Relevancy Evaluation:

- **Objective:** My goal here was to assess if the system's responses and the source nodes used for generating those responses were in alignment with the original queries.
- **Approach:** I employed a Relevancy Evaluator to methodically check if the answers provided were relevant and accurately addressed the queries. This evaluation considered not just the correctness but also the relevance of the responses, which I believe is vital for practical QA applications.
- **Outcome Measurement:** The evaluation produces a straightforward Pass/Fail result for each query-response pair, offering a clear metric to gauge system performance. An answer passes if it effectively answers the question with appropriate context; otherwise, it fails.

Rationale for My Evaluation Criteria:

- The evaluation method I chose is especially suitable for this application because it mimics how the system would be used in real scenarios without the need for creating extensive manual labels. I think this is one of the best ways to evaluate a QA model because data with ground-truth is either impossible to find or very costly to create. Also this approach is more useful than getting top-n MRR or hit scores for this task because the end product here is to give a correct response. Moreover, combining question generation with relevancy evaluation allows for ongoing system improvements and validation against the actual content of the documents.
- By focusing on both the accuracy and the relevance of the responses, as well as how well they relate to the source nodes, I ensure that the answers are not only correct but also contextually suitable. This is crucial in a policy-related QA system where accurate information can significantly influence employee actions and decisions.

Implementation of the Evaluation:

- I implemented the evaluation through a sequence of automated steps, starting with question generation from the policy documents. Then, the system's query engine retrieved answers and context, which were evaluated for relevancy. This process was designed to be repeatable and scalable, allowing for continuous refinement based on the results.
- I created more than 250 questions for testing.

Conclusion and Results:

LlamaIndex and RAG (LangChain) models were assessed based on various chunking methods and the unit of data abstraction used as shown in **Table 1**, which were either documents or nodes. My final solution is the one highlighted with bold.

Model	**Chunking Method**	**Data Unit (Data Abstraction)**	**Accuracy**
LLamaIndex	Character Split (chunk_size=1024, chunk_overlap=200)	Documents	85.0%
		Nodes	90.1%
	Semantic Split (breakpoint_percentile_threshold=95)	Documents	87.6%
		Nodes	**95.7%**
RAG (LangChain)	Character Split (chunk_size=1024, chunk_overlap=200)	Documents	86.3%
		Documents	94.5%
	Semantic Split (breakpoint_percentile_threshold=95)	Documents	95.5%

Table 1: Relevancy Results Based On Pass/Fail Ratio

LlamaIndex Results:

- When using Character Split and treating entire documents as data units, the accuracy was recorded at 84.0%. However, when the same chunking method was

applied but with nodes as the data units, the accuracy improved significantly to 90.1%.
- For Semantic Split, there was a clear superiority of nodes over documents, with the node-based method achieving a remarkable accuracy of 95.7%, compared to 87.6% for documents.

RAG (LangChain) Results:

- Character Split chunking with documents as data units yielded an accuracy of 86.3%. Recursive Character Split on documents saw a substantial increase in accuracy, coming in at 94.5%.
- Semantic Split with documents stood close to the recursive method, with an accuracy of 95.0%.

Conclusion

From the results, it is evident that the Semantic Split method, especially when coupled with nodes as the data unit, offers the highest accuracy in the context of LlamaIndex, reaching an impressive 95.7%. This suggests that a more nuanced and context-aware approach to chunking and indexing provides substantial benefits in terms of the system's ability to return accurate information.

The conclusion I've drawn from these experiments is that while both RAG (LangChain) and LlamaIndex perform effectively, the latter, with its combination of Semantic Split and node-based data abstraction, is particularly powerful. It is this configuration that I have chosen as my final solution and the one currently implemented in the application. The high degree of accuracy achieved with this setup indicates that the system is highly adept at interpreting queries and providing relevant, precise answers.