**Explanation of How RAG Model Works**

Initially, the documents are ingested using a data connector (i.e., Reader) present in LlamaHub. These documents can be split into nodes if they are very big. For our example, we will be taking data points that are small, so we can directly proceed to indexing.

The below flowchart represents how a RAG Model works.

A diagram of a dog

Description automatically generated

# **Indexing**

The Index is a data structure that allows fast retrieval of relevant context for a user query. For LlamaIndex, it is the core foundation for retrieval-augmented generation (RAG) use cases.

After loading the data, the text or the unstructured data is embedded, and the resulting embedding vectors are stored in Vector Stores or Vector Databases. The below image gives a basic visual representation of how a part of the text is represented as embedding vectors.

A screenshot of a computer

Description automatically generated

By default, VectorStoreIndex uses an in-memory SimpleVectorStore that is initialized as part of the default storage context. We can also use a custom vector store like PineconeVectorStore, ChromaVectorStore, DeepLakeVectorStore, and other Vector Stores supported by LlamaIndex.

A Vector Store takes care of storing embedded data and performing vector search for the user.

For a better explanation, we'll use a subset of the Airline Travel Information System (ATIS) intent classification dataset. This dataset consists of inquiries coming to airline travel inquiry systems. Here are a few example data points:

1 - which airlines fly from boston to washington dc via other cities

2 - show me the airlines that fly between toronto and denver

3 - show me round trip first class tickets from new york to miami

4 - i'd like the lowest fare from denver to pittsburgh

5 - show me a list of ground transportation at boston airport

6 - show me boston ground transportation

7 - of all airlines which airline has the most arrivals in atlanta

8 - what ground transportation is available in boston

9 - i would like your rates between atlanta and boston on september third

First, we need to use the SimpleVectorStore to turn each article’s text into embeddings.

Depending on the model used, the dimensions vary accordingly. Each dimension stores one additional piece of information about the text, so as the number of dimensions increases, the representational power increases.

Here is an example of the first few dimensions given by the model for "*show me inquiries about boston ground transportation*":

[0.20641953, 0.35582256, 0.6058123, -0.058944624, 0.8949609, 1.2956009, 1.2408538, -0.89241934, -0.56218493, -0.5521631, -0.11521566, 0.9081634, 1.662983, -0.9293592, -2.3170912, 1.177852, 0.35577637, ... ]

To get some visual intuition about this, these numbers are plotted on a heatmap after reducing their dimensionality to 10. The heatmap below shows 10-dimensional embeddings of 9 data points.

A graph with different colored squares

Description automatically generated

# **Querying**

Querying a vector store index involves fetching the top-k most similar Nodes and passing those into the Response Synthesis module. So, during query time the unstructured query is embedded and the ‘most similar’ embedding vectors are retrieved.

So, if we provide an unstructured query that says, "*show me inquiries about boston ground transportation*", embedding vectors that are most similar to the query are fetched as seen in the heatmap below. They are all inquiries about ground transportation in Boston and their embeddings patterns are very similar.

A blue and purple squares

Description automatically generated

So based on the query, the Retrievers fetch the most relevant context from an index based on the similarity between the embedding vectors.

# **Response Synthesis**

LlamaIndex offers different methods of synthesizing a response from a relevant context. There are different types of response synthesis like:

* Refine
* Compact and Refine
* Tree Summarize
* Simple Summarize
* Generation

“Compact and Refine” is the default in LlamaIndex. It first combines text chunks into larger consolidated chunks that more fully utilize the available context window, then refine answers across them.

So, based on the above query, it fetches the most similar nodes shown in the heatmap:

show me a list of ground transportation at boston airport

show me boston ground transportation

what ground transportation is available in boston

Now, the Response Synthesizer reiterates and refines the answer based on the context and gives the following response:

These are the inquiries about Boston ground transportation:

1. show me a list of ground transportation at boston airport
2. show me boston ground transportation
3. what ground transportation is available in boston

# **Chat Engine**

This class takes in a query and returns a Response object. It can make use of **Retrievers** and **Response Synthesizer** modules under the hood. Conceptually, it is a stateful analogy of Query Engine. The Query engine is a generic interface that allows the user to ask questions over the ingested data. A query engine takes in a natural language query and returns a rich response.

A Chat Engine keeps track of the conversation history, it can answer questions with past context in mind.