Working Architecture of **Retail-Product Catalog Q&A** product

STEP 1:

All local HTML files, along with some sub-links within the HTML files, are scraped using **Beautiful Soup** with the **`html.parser`** method.

The extracted text is initially in an unstructured format. Then, using **`langchain_core.prompts`** and the open-source LLM model **"llama3-70b-8192"** from CloudGroq inference, the unstructured text is converted into a structured and readable format.

This structured text is then saved as a PDF using the **ReportLab** library.

This process is repeated for all given HTML files and relevant sub-links within them.

Then, all these PDFs are combined into a single PDF of approximately 170 pages.

STEP 2:

Then, using **`langchain.document_loaders`** and **`langchain.text_splitter`** with **`RecursiveCharacterTextSplitter`,** each page in the PDF is made into a single chunk based on the amount of text on each page.

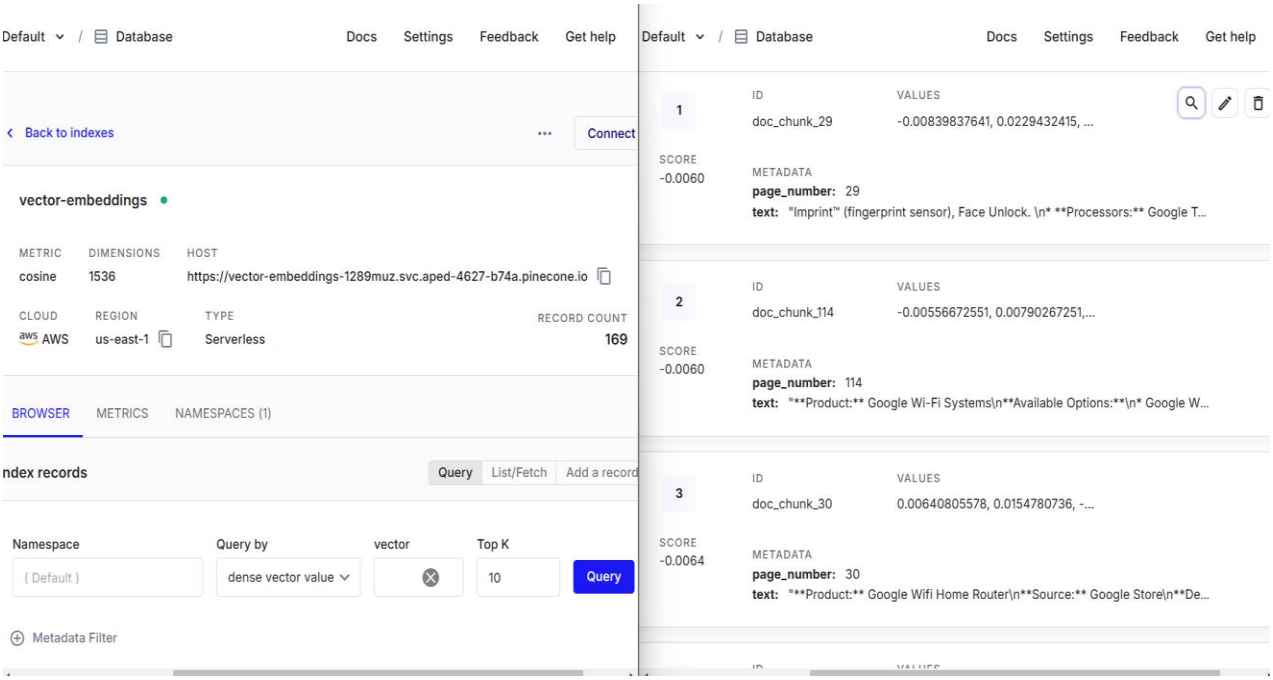
- If the chunk size exceeds a certain limit, each page is further divided into two.

In this way, 170 chunks are created.

STEP 3:

Using **`langchain.embeddings`** **with** **`OpenAIEmbeddings`** (using the default model **`text-embedding-ada-002`**), each chunk (i.e., each page) is vector-embedded with the metadata of each document.

These vector embeddings are then saved in the **Pinecone online database using the Pinecone API.** We chose Pinecone for its cloud accessibility and because it's free of cost (at least for our project).



Cosine similarity is used as the metric for vector embeddings, and the default dimension size of **`OpenAIEmbeddings` is 1536.**

- For each chunk 1536 vector embeddings (numerical values) are created.

Each chunk is saved with **metadata** as shown in the figure above. As you can see, our record count is approximately **170.**

STEP 4: (Completely RAG Process)

The user enters their prompt, which is also vector-embedded using the same embedding metric and model. The vector embedding of the user prompt is then compared with the vector embeddings of chunks in the Pinecone database, retrieving the top results **(with `k = 3`, which can be adjusted). Here, the top 3 documents are retrieved as `k = 3`.**

Using these retrieved documents, a final answer is generated using **OpenAI's LLM 'gtp-4o'** based on the text from the top results in response to the user prompt.

STEP 5:

For seamless interaction with the RAG system, a user interface is created in the style of a chatbot, similar to ChatGPT, using the Streamlit library. We call it '**Pixel AI Bot'.**

- This Chat Bot is hosted using **Streamlit Community**
- Here's the link for you to check it out: chattbott.streamlit.app (copy and paste link)