



RAG-powered enterprise knowledge retrieval (using LLMs)

05.11.2023

Group - 4

Anoop Singh, Chakri P, Ijya Samajna, Madhusmita sahu, Praveen Kumar
Gudabandi, Raushan Sinha, Saurav, Sreenivas PN, Vani Murarishetty

Problem Statement:

"Time is money. Save both by investing in technology." - Henry Ford

In a large enterprise, employees need to consult a wide range of documents on a daily basis to complete their tasks. These documents include technical documentation, company policies, and other materials in a variety of formats, such as Word, Excel, and PDF.

It can be time-consuming and tedious to keep track of and search through all of these documents, especially when looking for specific information. A conversational search experience in the form of a Q&A interface would be a more intuitive and efficient way for employees to find the information they need.

Conversational search allows employees to ask questions in natural language, and the system will return relevant results from the document corpus. This would save employees a significant amount of time and effort, and would make it easier for them to find the information they need to do their jobs effectively.

Generative large language models (LLMs) are good at interacting in natural language. Since LLMs are not trained on the enterprise specific information, their responses to user queries are prone to hallucinations and confabulations - a situation where the model generates a believable but factually incorrect response, may not be related to the input.

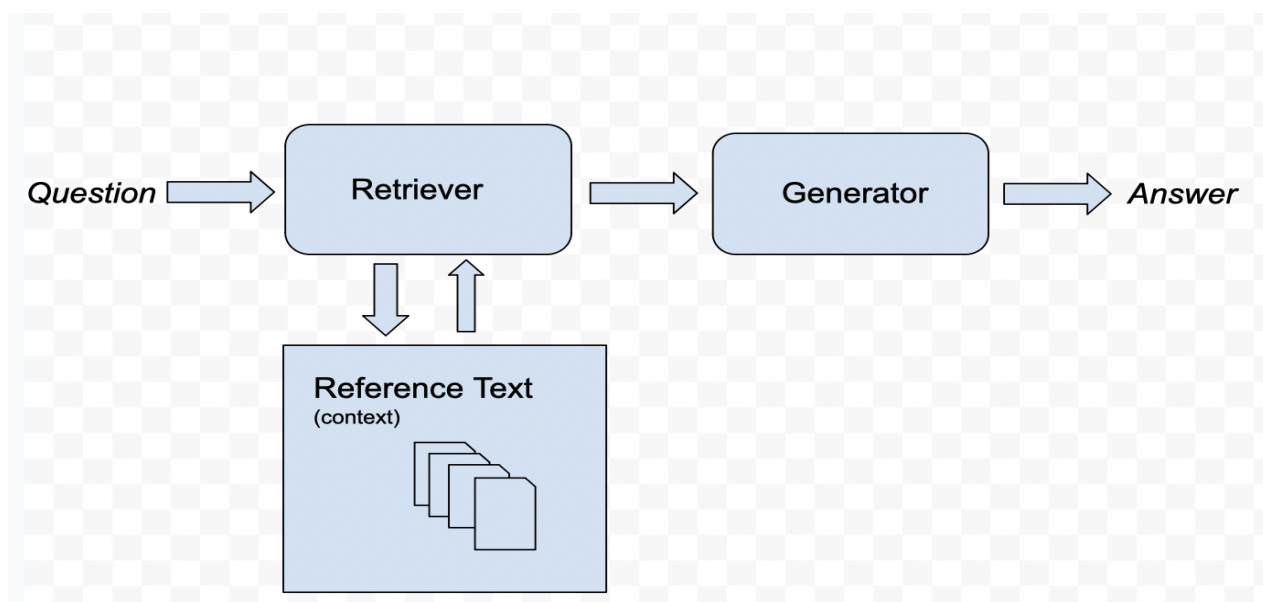
Goals

1. Develop a multi-user Q & A interface for employees of an organization
2. Employees can use it to query any proprietary and company-specific information
3. Any update to the document corpus will show up in the subsequent query responses.

Overview

Retrieval-Augmented Generation (RAG) is a tool designed to pull relevant data from specific sources to be used with large language models. Through the use of these external documents, RAG improves the context for the query, thereby increasing the quality of LLM response. RAG uses two components:

1. The retriever component searches for relevant information in a large dataset of text and code.
2. The generator component uses the retrieved information to generate a response to the user's query.

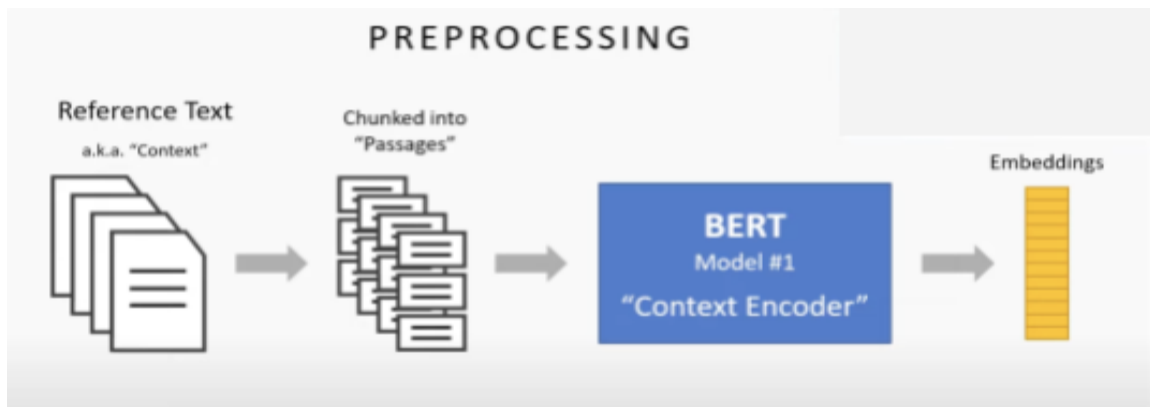


Methodology

A) Data Preparation Step:

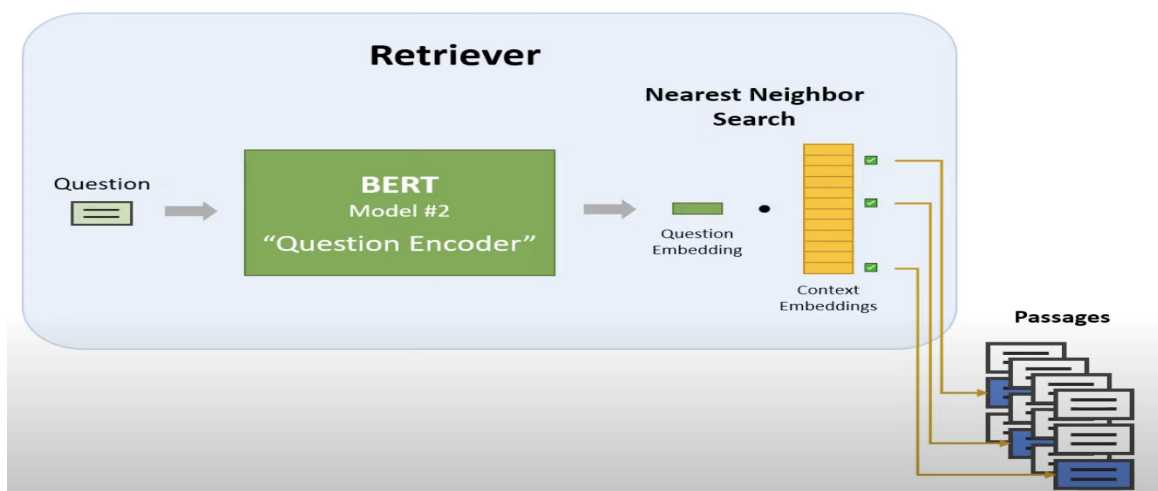
- a) Extract different components like images, tables and texts from the documents
- b) Break down the text passages into chunks
- c) Generate the embeddings of images, tables and text input
- d) Store these embeddings into a vector DB like Pinecone or Amazon OpenSearch

This step can be triggered when there's any update to the document corpus/repo.



B) User Query Processing:

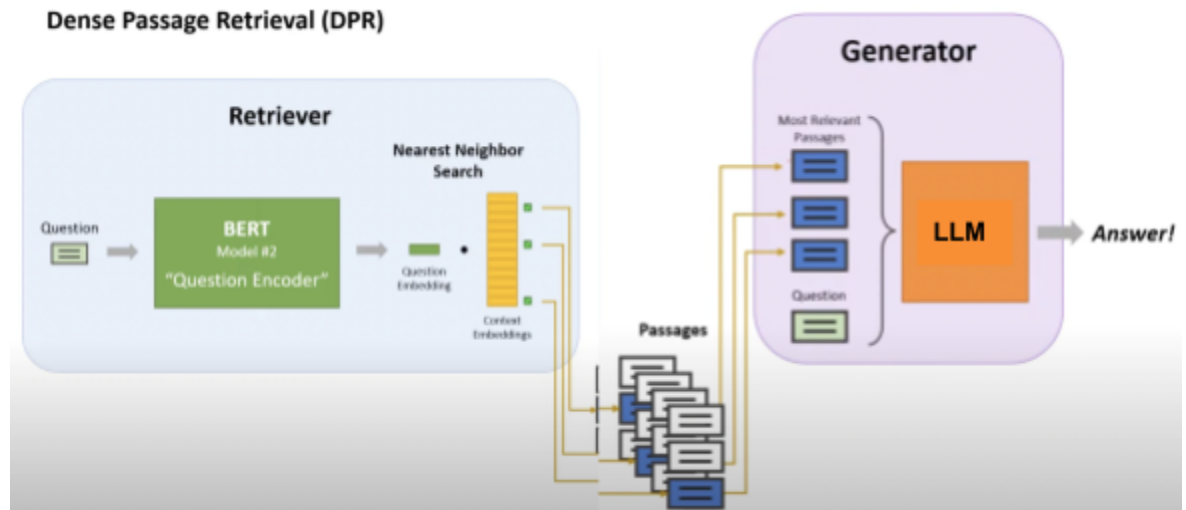
- The user interacts with the Q & A service by entering the query in the test box.
- The query is passed through the BERT encoder to get its vector representation (embedding).
- Use vector search to get k-relevant contexts (passages, images)
- Formulate a context appended user query and pass it on to the Generator



C) Generate Query Response:

- Use a pre-trained LLM from Huggingface library
- Pass on the extracted passages/images in previous step as context, along with the user query
- Process the response from LLM and display it in the UI text box.

- d) List out the sources of the response and add an option for the user to give feedback on the response. This feedback can be used for future enhancements.




MLOPs:

- This project will be developed and deployed using MLOps practices. Any code or data (new document addition/updates) will be driven through a pipeline, tested and automatically deployed in the Cloud.
- The resource usage monitoring will be provided using Prometheus and Grafana.
- Alerts will be sent to admins when resource usage exceeds preset thresholds.
- We envisage collecting user feedback through our UI, logging it and using it to improve our solution as part of our MLOps lifecycle.

Steps:

1. Pre-processing:
 - 1.1. Process text, tables and images from input docs
 - 1.2. Explore the choices for tokenizers
 - 1.3. Explore various vector stores
2. Generate synthetic data to be used for evaluation
3. Retriever:
 - 3.1. Create / Update (when the input docs changes) of vector index
 - 3.2. Process the user query and generate its vector/embedding to be used in vector search
 - 3.3. Explore choices of vector search
 - 3.4. Test the performance of vector search by examining the search results

- 
- 3.5. Explore quantifying and evaluating the performance
 4. Generator:
 - 4.1. Explore choices for LLM to be used as Generator
 - 4.2. With the output of Retriever as context, which is the chunks related to the user query, formulate the context + query input for the LLM
 - 4.3. Post-process the LLM response to produce the Answer to the user query
 - 4.4. List out the sources of the response
 - 4.5. Add an option to capture user's feedback on the response. This feedback can be used for future enhancements.
 - 4.6. Evaluate the end-end performance of this RAG solution
 5. Implement the solution in a collab notebook
 6. Add UI interface and dockerize the app, push it to Huggingface Spaces
 7. Create a CI/CD workflow in GitHub Actions to push the docker image to ECR and deploy the application with EKS/ECS.
 8. Add Monitoring and Alerting mechanism, create a dashboard showing the service status and stats (no.of queries answered type of details, precision, recall)

Dataset:

Generate synthetic dataset as a csv file containing (question, context, answer). This can be used for evaluation of Retriever as well as RAG.

Milestones

I. Week - 1 (Nov 5 to Nov 12)

Finish off exploratory work, like **how-to** for:

1. Using table data and graphs (images) in the documents in the context for generating meaningful response
2. Vector store and vector search mechanisms to be used in final implementation
3. Generate synthetic data to be used for evaluation
4. Performance evaluation
5. LLM to be used in Generator

II. Week - 2 (Nov 13 to Nov 20)

Implement the solution in collab notebook, with Gradio interface

III. Week - 3 (Nov 21 to Nov 30)

1. Work on improvements in performance (as-needed)



2. Dockerize the app
3. Create a CI/CD workflow in GitHub Actions to push the docker image to ECR and deploy the application with EKS/ECS

IV. Week - 4 (Dec 1 to Dec 10)

1. Create monitoring and alerting system using Prometheus and Grafana
2. Create dashboard showing the service stats like precision, recall
3. Work on final presentation for demo