RAG Chatbot

Raghav Khanna

2025-06-13

Table of contents

List of Figures

Preface

This report has been prepared as a part of our full-time Data Science internship at Sabudh Foundation. The rationale behind this report is to document the development of a RAG chatbot.

In the age of cutting-edge technologies, chatbots aid in information retrieval without any need for human intervention. The idea for this project stemmed from our curiosity in learning how machines can comprehend human languages, extract related information from some knowledge base, and process it all to generate appropriate responses. Over the course of this internship, our team delved deeper into the intricate world of chatbots and RAG to come up with our own model that dazzles bright.

We would like to extend a warm notion of thanks to our project mentor, Mr. Partha Sarathi Mukherjee, for walking us through this journey with utmost sincerity and dedication. His resilience and determination towards his field truly inspires us all. We are also grateful to our course instructors for teaching us with such conviction and patience as well as our fellow batchmates for always keeping spirits in the high.

This project has really been a great learning experience and has prompted us to think beneath the surface to see how beautifully connected the technical aspects of a system are.

Abstract

The main goal of this project is to come up with a Retrieval-Augmented Generation (RAG) Chatbot. A simple chatbot is a software application that simulates human conversation through speech or text and can interact with users in natural languages to facilitate smooth communication.

Chatbots provide strategic business advantage by saving human time and efforts spent in customer support and communication. Business organizations use them for scaled, personalized, and improved communications for a better overall user experience.

However, a simple chatbot is limited in its ability to produce very technical or domain-grounded answers. In addition, they are also susceptible to hallucinations, a situation where a chatbot gives out responses that are false or not grounded in reality. To overcomes these challenges, we have Retrieval-Augmented Generation as an add-on to simple chatbots to boost their performance.

A Retrieval-Augmented Generation Chatbot performs retrieval (retrieving relevant information from a credible source), augmentation (feeding the retrieved information into the Large Language Model along with the user input message), and generation (generating a response based on the retrieved information).

Carrying out these functionalities, the chatbot learns along the way, storing information in its data stores and referring to them continually, facilitating reusability of information and better retrieval.

All in all, a RAG chatbot has an upper edge in the contemporary era.

1 Introduction to Project

Chatbots are increasingly finding their way into e-commerce and e-services, as their implementation opens up promising opportunities to improve customer service. A chatbot is a software application that can imitate human conversation, allowing users to interact with it over text or speech. RAG is an add-on to a chatbot that helps it retrieve information, add it to the user's input, and generate a response.

1.1 Overview

RAG stands for "Retrieval-Augmented Generation" - a technology that combines information retrieval with generation. A RAG chatbot essentially process the user's query, retrieves related information from some knowledge base, adds it to the query, and generates an appropriate response. It is basically an extension of a traditional chatbot that allows it to back its generted responses in order to build credibility and add gravitas to the response.

1.2 Existing System

Opposed to RAG chatbots are:-

- **Rule-based chatbots**: These operate by adhering to a predefined set of rules that dictate how to respond accordingly. For this, RASA framework was prominently used to sustain a rules engine that would take care of generating responses as per the rules. These are very limited in their functionality because of being confined to a set of rules and not being able to expand beyond them.
- **Generative chatbots without RAG**: These rely on the data they were trained on to generate responses without any retrieval. For this, no RAG framework is required. It retains the functionality of generation but without retrieval and augmentation. They may be subject to hallucinations, a situation in which a chatbot gives information that is false and is not grounded in reaity.

RAG is a recent technology and has been gaining traction.

1.3 Objectives of Project

The objective of the project is to build a RAG chatbot that is trained on some data fetched dynamically from the web that is stored in the database to serve as the knowledge base for the chatbot. It ought to serve the following purposes:-

- Enhanced user support
- Improved efficiency
- Reduced costs
- Automation of customer support

> **Note**
>
> Studies show that self- disclosure and empathy aptitudes should be given, in order to enhance the pleasure of the dialogue by building a closer relationship between the two interlocutors, which in turn stands for a successful interaction and thus for a satisfactory service performance. In summary, the user's willingness to interact depends to a large extent on the "attitude" of the chatbot and on the "feelings" the latter transmits. Additionally, in order to interact in a more authentic way in the eyes of the user, chatbots should possess attributes such as a trustworthy personality, active listening, prompt responding and a socially oriented interaction style, e.g. through the use of emojis or modern idioms in their messages. Misischia, Poecze, and Strauss (2022)

2 Pre-Processing and Exploratory Data Analysis

Once the dataset is got, there is a need to preprocess it in order to make it suitable for machine learning. This includes:

- Cleaning the data
- Dropping irrelevant features
- Handling missing values
- Encoding categorical variables
- Scaling numerical features

> **Note**
>
> This step is crucial since the accuracy of a model will depend on the quality of the data.

2.1 Dataset Collection

This involves actually fetching the data to be used. This is the data we are going to work with subsequently. Here, we are getting our data as follows:-

1. Hitting the Serper API with the user query to get related URLs.
2. Scraping those URLs to get the text and store it.

This scraped text is essentially the data for our model.

2.2 Data Pre-processing

Some issues to consider:

- Since we were getting data in the form of texts scraped from links, some of the texts contained characters in unreadable format as well as other invalid and/or irrelavant characters.
- Noisy data was making its way into the data store.
- Larger documents dominated. There was a need to have a uniform size of texts.
- Sometimes vaguely related data made its way into the dataset.

2.3 Exploratory Data Analysis and Visualizations

Some of the steps are:

- **Data Analysis**: Data analysis refers to deriving insights from data to understand the structure, patterns, semantics, etc. of data. This step can help identify outliers and anomalies in the data. Using this, we found out invalid text in the data. In the context of our model, after scraping articles from links, we checked that some of those article contained text in unreadable format. Therefore, to get at the root cause of it, we had to analyze the scraped data and the source of such data.
- **Data Filtering and Manipulation**: Data filtering and manipulation refers to filtering data with noise and then modifying it to get it in the desired form. Using this, we removed noise from our data and retained only the relevant information. We modified our scraping functionality to successively filter out unwanted data so that we only retain useful data.
- **Feature Selection**: Feature selection is the process of selecting the most important variables that will be used in the machine learning model. This step can help you identify which variables are most predictive and which variables can be ignored. We had to choose which tags or elements were most appropriate when extracting information from the web. Using this, we could lay out a scheme as to which features to select and how.
- **Statistical Analysis**: Statistical analysis involves applying statistical methods to the data to identify patterns, trends, and relationships. This step can help you identify correlations between variables and understand the distribution of the data. It includes ascertaining the statistical significance of features.
- **Data Visualization**: Data visualization is the process of representing data in some visual or graphical form so as to better understand it and gain insights from it easily. It also allows for a better communication of concerns to the associated stakeholders, emphasizing the importance of the steps taken during the entire process.

3 Methodology

3.1 Platform and Machine Configurations used

- **Milvus Vector Database**: To store vector embeddings of chunks.
- **MongoDB Atlas**: To store documents as well as chunks.
- **Docker Compose**: To support Milvus connection setup.
- **Serper API**: To extract data from the web.

3.2 Frameworks used

- **Langchain**: Aids in building a Large Language Model.
- **Gradio**: To build the user interface.
- **Sentence Transformers**: Used for embedding strings.

3.3 Python Libraries used

- os
- sys
- datetime
- uuid
- asyncio
- typing
- pymilvus
- pymongo
- tqdm
- dotenv
- langchain
- langchain_google_genai
- requests
- traceback
- sentence_transformers
- re
- bs4
- gradio

3.4 Functionalities used

- API requests
- Web scraping
- HTML boilerplate cleaning
- Text cleaning
- Database storage and handling
- Chunking
- Vector embedding
- Semantic search and retrieval
- Large Language Models
- Prompt Engineering
- Conversation Buffer Memory
- User Interface

3.5 Retrieval-Augmented Generation

The heart of our project lies in the Retrieval-Augmented Generation (RAG) framework, a sophisticated algorithm that enables the deconstruction of the language model into discrete, interchangeable components. This framework integrates a retriever model that sources relevant context and a generator model that synthesizes the retrieved infor-mation into coherent responses. Pichai (2023)

It carries out:

- **Retrieval**: Extraction of relevant information.
- **Augmentation**: Addition of the information to use it as context for the language model.
- **Generation**: Actual generation of response.

3.6 Workflow

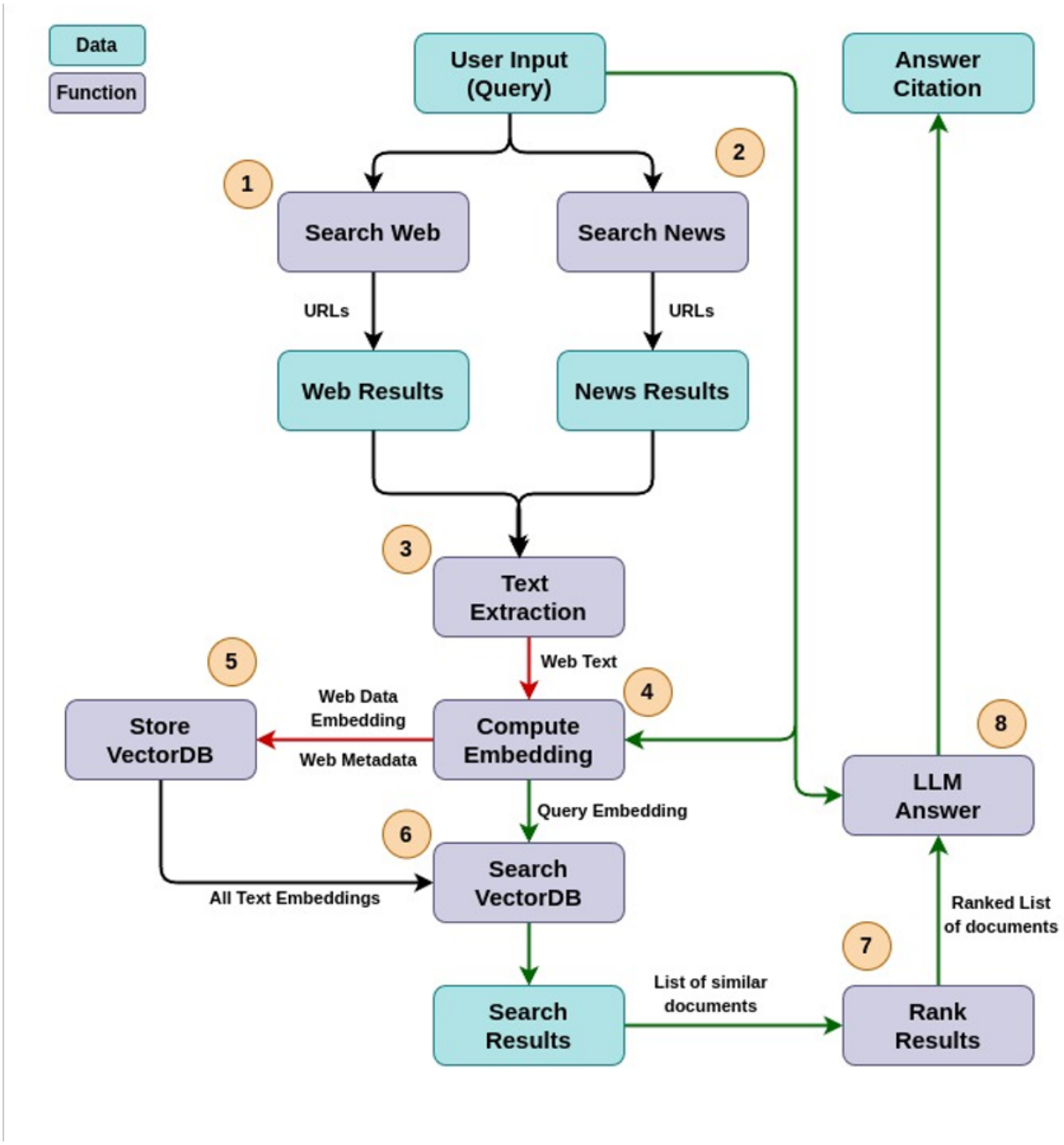

Figure 3.1: Workflow

1. The user input query hits the Serper web and news API endpoints. It extracts all the associated links.
2. The links are scraped to get text content only. The scraped content is stored in MongoDB Atlas database in a specific schema.

3. The extracted text is then broken down into chunks. These chunks are also stored in MongoDB Atlas database in a specific schema.
4. The embeddings are computed corresponding to these chunks and stored in Milvus vector database.
5. The stored chunks are indexed accordingly to mark that they have been processed once.
6. The user query is then converted into an embedding.
7. The embedding corresponding to the user query is then compared against all embeddings in Milvus vector database.
8. The top k (here, k = 5) embeddings are then elicited and chunks corresponding to these embeddings are obtained.
9. These chunks are then reordered in the context of the user query from most to least relevant via an LLM call.
10. The reordered chunks are then passed to the LLM as context along with the user query to get the final answer.

4 Detailed Module Breakdown

In our project, we have the following modules: - src module - utils module - app module - other miscellaneous files

We shall look at each one of them in detail.

4.1 src Module

The src module helps keep the root project directly clean and free of useless code. It contains the following files:

- **init.py**: Special Python script file that is used to mark a directory as a package so that it can be imported and used in other modules.
- **etl.py**: File to perform extract, transform, and load operations. In our project, we used it to keep all extraction and indexing codes inside it. It contains the following function(s):
  - **extract_all_html(list_of_urls, doc_type)**: Function to extract cleaned text from a list of URLs.
  - **get_all_embeddings(list_of_text: List[str], batch_size: int = 128, model_name=EMBEDDING_MODEL) -> List[List[float]]**: Function to asynchronously create embeddings of a list of texts.
  - **index_documents()**: Function to asynchronously index all unindexed documents in the database.
- **retrieval_engine.py**: File used to retrieve most similar chunks from the database. It contains the following function(s):
  - **search_chunks(query, top_k, article_type = "web")**: Function to retrieve top k semantically similar chunks in terms of their embeddings.

4.2 utils Module

The utils module contains helper functions and classes that can be used across various folders and files in the project to facilitate code reusability. It contains the following files:

- **init.py**: Special Python script file that is used to mark a directory as a package so that it can be imported and used in other modules.

- **search_utils.py**: File for search-related functionalities that is used to hit the Serper API using a query and get the associated links. It contains the following function(s):
  - **search_web(query:str, nb_docs = 10)**: Function to perform a web search using Serper API.
  - **search_news(query:str, nb_docs = 10)**: Function to perform a news search using Serper API.
- **text_utils.py**: File for all text-related functionalities. It contains the following function(s):
  - **chunk_text(document: str, document_id: str) -> List[Dict]**: Function to convert a document into chunks.
  - **clean_text(text)**: Function to remove control characters and excessive whitespace from the given text.
  - **get_html_text(url)**: Function to extract cleaned text and raw HTML from a given link.
  - **get_embedding(chunks: list[str], model_name = "all-MiniLM-L6-v2")**: Function to generate an embedding vector for a given text block.
- **db_utils.py**: File for all database-related functionalities. It contains the following function(s):
  - **insert_in_db_document(document)**: Function to insert each document into the MongoDB Collection (documents_main).
  - **insert_all_documents(list_of_documents)**: Function to dump all documents in MongoDB Atlas collection (documents_main).
  - **insert_in_db_chunks(chunk)**: Function to insert a single chunk into the database (documents_chunks).
  - **insert_all_chunks(list_of_chunks)**: Function to dump all chunks in MongoDB Atlas collection (documents_chunks).
  - **clear_all_chunks(collection)**: Function to clear all the chunks from the MongoDB Atlas database (to be used only when necessary).
  - **clear_all_docs(collection)**: Function to clear all the documents from the MongoDB Atlas database (to be used only when necessary).
  - **create_milvus_collection()**: Function to create Milvus vector database collection.
  - **delete_milvus_collection(collection_name: str)**: Function to delete Milvus vector database collection.
- **LLM_utils.py**: File for all LLM (Large Language Models)-related functionalities. It contains the following function(s):
  - **rank_chunks(query, chunks)**: Function to rank chunks from most relevant to least relevant.
  - **llm_call(query, support_material)**: Function to invoke an LLM call.
  - **llm_response(query)**: Function to make the final call to the LLM to elicit a response.
- **user_interface.py**: File for developing the user interface. It contains the following function(s):
  - **respond(message, chat_history)**: Function to append user message and a placeholder bot response ("Typing…") to the chat history and then replace it with the final response.
  - **launch_chatbot()**: Function to launch the final chatbot interface.

4.3 app Module

The app module is the main module that calls the final functions. It is the entry point of the project where the execution starts. It contains the following function(s):

- **ingestion(query, nb_docs = 10)**: Inserts chunks, documents, and embeddings into respective databases followed by indexing them appropriately. It is primarily used for the purpose of training.

> **Note**
>
> It makes a call to the final launch_chatbot() function.

4.4 Other miscellaneous files

There are some other miscellaneous files used for carrying out some other minor functionalities. These include:

- **.gitignore**: File that tells Git which files or folders to ignore in the project i.e. which files are not to be tracked or committed to the repository. We used it to store the .env file that kept all API keys and other sensitive information.
- **.env**: File that contains all the environment variables such as API endpoints, API keys, etc. It has been made to be ignored while committing by the .gitignore file.
- **docker-compose.yml**: File to set up docker specifications so as to set up Milvus vector database.
- **README.md**: File that keeps the whole structure of the repository for better navigability through the project.

5 Results

5.1 User Interface

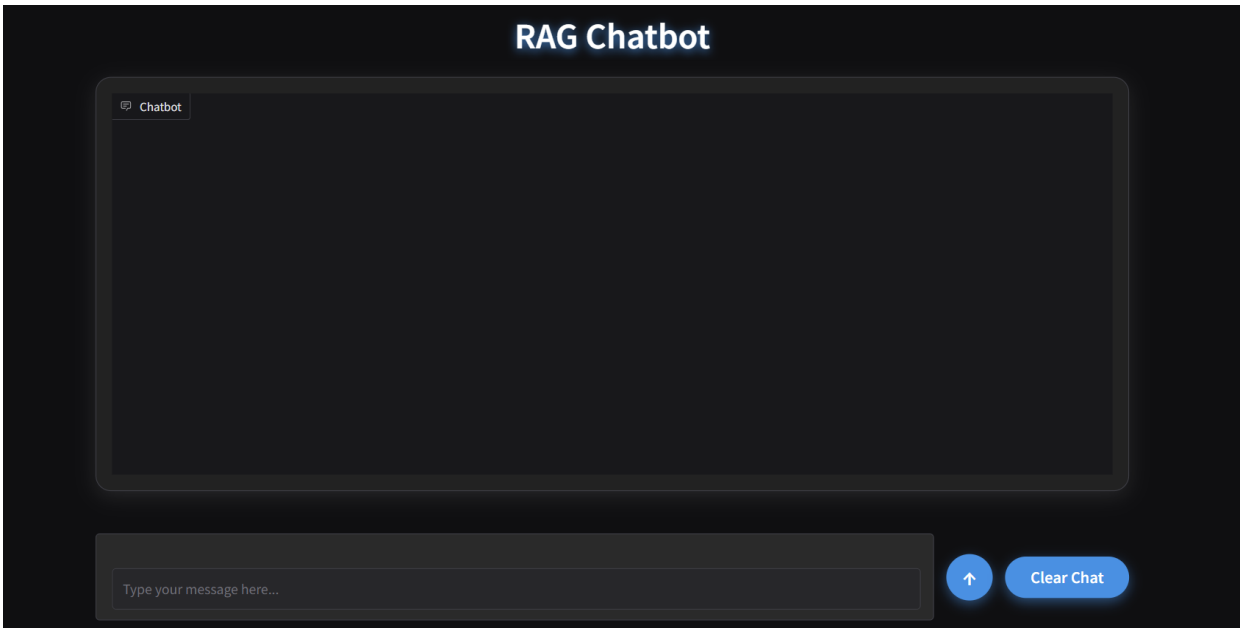

Figure 5.1: User Interface

5.1.1 Components of User Interface

- **Chat window**: This is the workspace where the chatbot's responses to the user's questions will be displayed.
- **Typing area**: This is the field where the user types the message.
- **Send button**: This is used to send the message to the chatbot after typing it.
- **Clear chat button**: This is used to clear chat with the chatbot.

5.2 Example conversation

- User: "Where is India loacted?"
- Assistant: "India is located in South Asia."

- User: “What is its capital?”
- Assistant: “The capital of India is New Delhi.”
- User: “What is its largest city?”
- Assistant: “Mumbai is the largest city in India.”

5.3 Sample usage

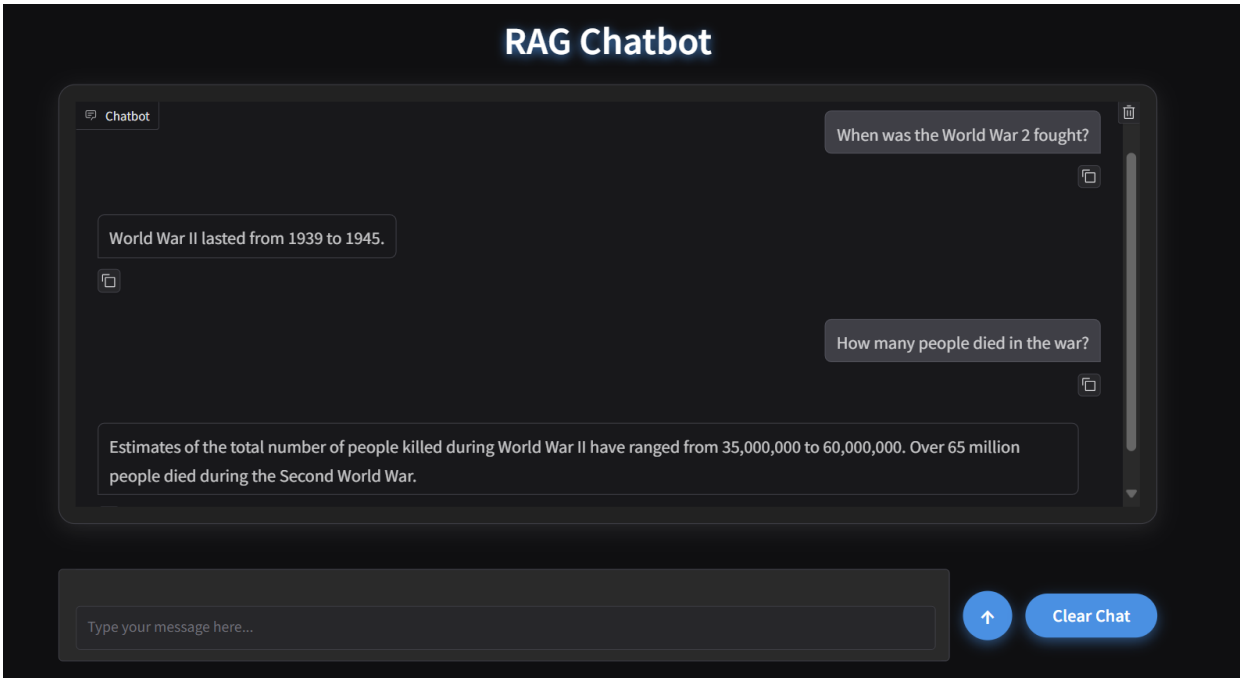

Figure 5.2: Sample usage

6 Conclusion

We have successfully developed a chatbot that combines the strength of information retrieval and augmentation with response generation. The model works well by perfectly integrating these aspects. We explored certain key data preprocessing in the form of manipulating th HTML boilerplates of URLs we were extracting, removing unwanted characters, retaining only the main content, chunking, vector embeddings, semantic search and retrieval, and finally generation using Large Language Models (LLMs). We evaluated the model performance in the end.

This project shows the ever-growing potential of Retrieval Augmented Generation (RAG) and similar technologies that can be useful in leveraging the utility of Large Language Models with a factual corroboration through the used URLs, thereby reducing hallucinations and the risk of spread of false information that might be dangerous.

Like Retrieval Augmented Generation, we have another upcoming technology called Retrieval Interleaved Generation (RIG) that might revolutionize the space of Large Language Models even further, thus strengthening the scope of Artificial Intelligence in the years to come.

References

Misischia, Chiara Valentina, Flora Poecze, and Christine Strauss. 2022. "Chatbots in Customer Service: Their Relevance and Impact on Service Quality." *Procedia Computer Science* 201: 421–28. https://doi.org/https://doi.org/10.1016/j.procs.2022.03.055.

Pichai, Kieran. 2023. "A Retrieval-Augmented Generation Based Large Language Model Benchmarked on a Novel Dataset." *Journal of Student Research* 12 (November). https://doi.org/10.47611/jsrhs.v12i4.6213.