

A REPORT
ON
RAG CHATBOT

BY

Pranav Siby

2022A7PS0356G

AT

Jio Platforms Ltd.

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(June,2024)

**A REPORT
ON
RAG CHATBOT**

BY

Pranav Siby

2022A7PS0356G

Computer Science

Prepared in partial fulfillment of the Practice
School-I Course Nos.
BITS C221/BITS C231/BITS C241

AT

Jio Platforms Ltd.

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(June, 2024)

Acknowledgements

I want to extend my heartfelt thanks to Mr. Aryan Koul, my project advisor, whose unwavering support and guidance were instrumental throughout this project. I am also deeply grateful to Mr. Apparao Mulpuri, my project mentor, for his invaluable assistance and encouragement.

Additionally, I would like to acknowledge and thank my faculty mentors, Dr. Prasanta Kumar Das, Dr. Apurba Das, and Dr. Sukanta Mondal, for their continuous support and mentoring during my internship.

My sincere appreciation goes to my family and friends for their unwavering support throughout this journey.

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI
(RAJASTHAN) Practice School Division**

Station: Jio Platforms Ltd.

Duration: May 18 - July 25, 2024

Date of Submission: 21/06/2024

Title of the Project: RAG Chatbot

**ID No./Name(s)/
Discipline(s)/of the
student(s):**

Pranav Siby

**Name(s) and designation(s)
of the expert(s):**

Mr. Apparao
Mulpuri -
Technical
Architect
(AI&ML)

**Name(s) of the
PS Faculty:**

Dr. Prasanta
Kumar Das,

Dr. Sukanta
Mondal,

Dr. Apurba Das

Key Words: Legacy Search, LLM, RAG

Project Areas: Natural Language Processing, Machine Learning

Abstract:

In the evolving landscape of natural language processing and conversational AI, the ability to efficiently retrieve relevant information from extensive databases is crucial for the performance of Retrieval-Augmented Generation (RAG) chatbots. This project focused on the implementation and enhancement of legacy search methodologies, specifically lexical search (string-matching) and TF-IDF (Term Frequency-Inverse Document Frequency) based search, to optimize the information retrieval capabilities of an RAG chatbot.

The project involved integrating a legacy search framework into the RAG chatbot's architecture to improve its response accuracy and relevance. Lexical search techniques, which match query terms directly with indexed terms in the database, were employed to provide quick, straightforward retrieval. In addition, a TF-IDF based search approach was incorporated, which calculates the importance of terms in documents relative to the entire dataset. This dual approach facilitated the efficient retrieval of both common and contextually significant information, depending on the application.

The integration of these legacy search techniques demonstrated significant improvements in the RAG chatbot's performance, particularly in terms of response relevance and speed. This project not only underscores the efficacy of combining lexical search and TF-IDF methodologies in modern AI systems but also provides a foundation for future enhancements in information retrieval for RAG-based conversational agents.

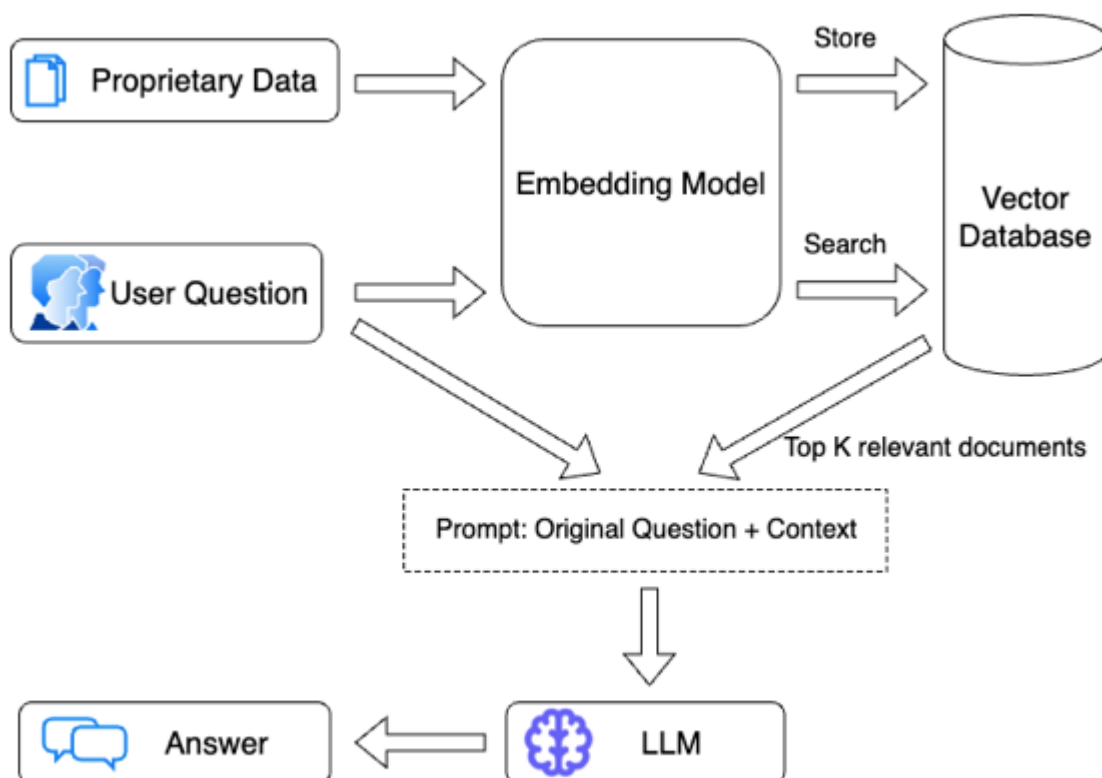
Table of Contents

Introduction	7
Pre-Midsem	8
Task 1: Keyword Extraction	8
Task 2: Gradio UI	9
Task 3: Vectorized Lexical Search with TF-IDF	10
Task 4: Integrating Both Forms of Lexical Search	11
Task 5: Returning Documents Matching the Query Keywords	11
Post-Midsem	12
Task 6: Local LLM Inference Using Ollama	12
Task 7: Making a Sample Dataset	13
Task 8: Ensuring Referential Integrity in the Databases	13
Task 9: Connecting the Database to the LLM	14
Task 10: Perfecting the Responses with Prompt Engineering	14
Glossary	15

Introduction:

Jio is developing an RAG-based (Retrieval-Augmented Generation) assistant chatbot to answer user queries on a wide variety of subjects. In an RAG system, a database is stored from which documents relating to the user's query are searched. The related documents are found by doing 2 processes of searching- Lexical search (string matching) and Semantic matching (meaning-based matching). To get the documents matching the user prompt, we first do lexical search, i.e. find the top K most relevant documents which have the most keywords in common with user prompt, and then do semantic search, which involves finding the documents from the database which are most similar in meaning to the user prompt, out of the documents identified first by lexical search.

Then, the most relevant document thus found is passed to an LLM for further processing (using the chat context etc) which then outputs the end result to the user. In this pipeline, my first task was to do two types of lexical search: Keyword-Based (string matching) as well as vector-based (TF-IDF search).



PRE-MIDSEM

Task-1: Keyword extraction

The first task I was assigned was to extract relevant keywords from the user's query, for further processing. To do this, I tried many python libraries like rake, spacy, and yake. Finally, I settled on using the yake library. My plan for user extraction from the query was to first extract keywords from the RAG database, and then search for those extracted keywords from the user's query.

The yake library uses statistical methods to analyse the database and extract keywords along with Scores, where the lower the score, the more important that keyword is. After getting the keywords from The database, I sorted them in increasing order of score i.e. increasing order of relevance.

The keywords were then saved to a json file along with their scores, for easy lookup later.

Note that json arrays automatically get converted to python dictionaries if written in the appropriate syntax.

Now I had gotten the most important keywords: next was to match those with the user query.

First, I removed stopwords from the user's query in order to quickly get rid of redundant words.

Then, I made a dictionary out of the keywords by loading the json file that I created, and searched each ngram of the query in the dictionary.

Json dataset format:

```
1 {
2   ("title": "Document 1", "content": "Deep learning is a subset of machine learning."),
3   ("title": "Document 2", "content": "This document discusses the principles of artificial intelligence."),
4   ("title": "Document 3", "content": "Natural language processing is a field of artificial intelligence."),
5   ("title": "Document 4", "content": "This document is about machine learning. Machine learning is fascinating."),
6   ("title": "Document 5", "content": "This document provides an overview of AI and ML."),
7   ("title": "Document 6", "content": "Machine learning algorithms are used in a variety of applications."),
8   ("title": "Document 7", "content": "Artificial intelligence can be categorized into narrow AI and general AI."),
9   ("title": "Document 8", "content": "Supervised learning and unsupervised learning are two types of machine learning."),
10  ("title": "Document 9", "content": "Reinforcement learning is a type of machine learning where agents learn by interacting with the environment."),
11  ("title": "Document 10", "content": "Neural networks are a fundamental concept in deep learning."),
12  ("title": "Document 11", "content": "Deep learning models require large amounts of data to train effectively."),
13  ("title": "Document 12", "content": "This document explores the history of artificial intelligence from its inception to modern applications."),
14  ("title": "Document 13", "content": "Natural language processing techniques include tokenization, stemming, and lemmatization."),
15  ("title": "Document 14", "content": "Machine learning can be applied to predictive analytics in various industries."),
16  ("title": "Document 15", "content": "This document provides a comparison between different machine learning algorithms."),
17  ("title": "Document 16", "content": "Artificial intelligence has the potential to revolutionize healthcare through improved diagnosis and treatment."),
18  ("title": "Document 17", "content": "Unsupervised learning algorithms include clustering and association techniques."),
19  ("title": "Document 18", "content": "This document discusses the ethical considerations in the deployment of artificial intelligence systems."),
20  ("title": "Document 19", "content": "Machine learning models can be prone to bias if not trained on diverse datasets."),
21  ("title": "Document 20", "content": "The field of AI research includes both theoretical and applied studies."),
22  ("title": "Document 21", "content": "This document examines the role of AI in autonomous vehicles."),
23  ("title": "Document 22", "content": "Natural language understanding is a critical component of conversational AI systems."),
24  ("title": "Document 23", "content": "This document covers the basics of supervised learning and its applications."),
25  ("title": "Document 24", "content": "Reinforcement learning can be used to train agents in simulated environments before deployment in the real world."),
26  ("title": "Document 25", "content": "This document explains the concept of transfer learning in machine learning."),
27  ("title": "Document 26", "content": "AI-driven recommendation systems are widely used in e-commerce and content streaming services."),
28  ("title": "Document 27", "content": "This document discusses the challenges and opportunities in deploying AI at scale."),
29  ("title": "Document 28", "content": "Machine learning can be used to enhance cybersecurity by detecting anomalies."),
30  ("title": "Document 29", "content": "This document provides an introduction to generative adversarial networks (GANs)."),
31  ("title": "Document 30", "content": "AI can assist in personalizing educational content for students."),
32  ("title": "Document 31", "content": "This document explores the use of AI in financial services for fraud detection."),
33  ("title": "Document 32", "content": "Machine learning algorithms can improve supply chain management through better demand forecasting."),
34  ("title": "Document 33", "content": "This document discusses the application of deep learning in image recognition.")
35 }
```

Extracted Keywords with Scores:

```
1 {
2   "machine learning": 0.0025543590410716513,
3   "document": 0.004287570598181134,
4   "document discusses": 0.004503513211056773,
5   "learning": 0.00527099911437142,
6   "document explores": 0.008012910948752826,
7   "machine": 0.0097388680778213,
8   "machine learning models": 0.013460885741519425,
9   "discusses": 0.0212101723241637,
10  "machine learning algorithms": 0.022443297884943324,
11  "learning models": 0.028702383751224147,
12  "learning algorithms": 0.034454100920296694,
13  "explores": 0.037063741519289206,
14  "deep learning": 0.045868018885467944,
15  "reinforcement learning": 0.051354526838488755,
16  "artificial intelligence": 0.05280602519900503,
17  "systems": 0.06063840445648878,
18  "artificial": 0.07204164938372212,
19  "improve": 0.07269414643809063,
20  "algorithms": 0.07576841173599081,
21  "intelligence": 0.07645302437037622,
22  "models": 0.08655923037358347,
23  "applications": 0.0922680176118074,
24  "document covers": 0.0951936097733466,
25 }
```

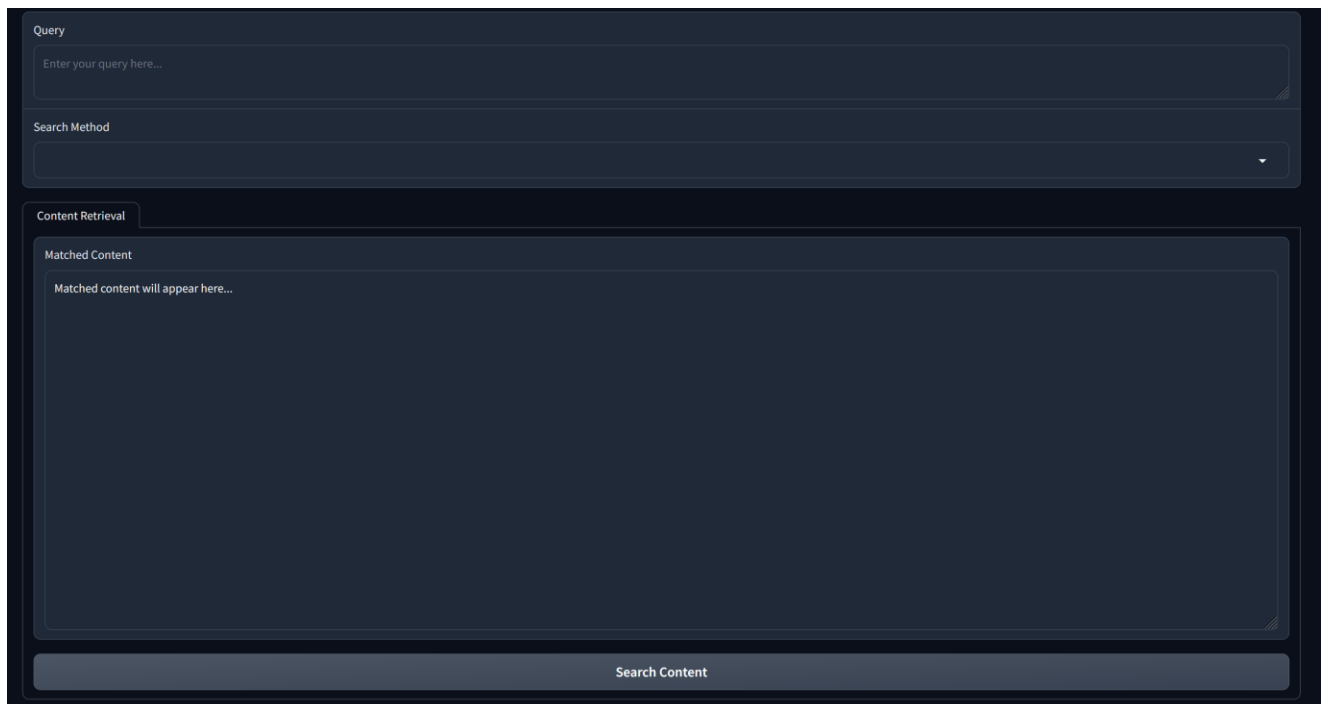

Task 2: Gradio UI

The next task I was assigned was to make a UI for the same using the python library gradio.

It took me a few days to learn how to use gradio, but afterwards it was simple.

Gradio uses simple python commands to generate a local webpage by converting the python code into css, Javascript and HTML instructions which are then used to create a localhost website without any knowledge of web development.

Gradio UI example:



The image displays a Gradio web interface with a dark theme. At the top, there is a 'Query' section with a text input field containing the placeholder 'Enter your query here...'. Below this is a 'Search Method' section with a dropdown menu. The main area is titled 'Content Retrieval' and contains a 'Matched Content' section with a large text area showing the placeholder 'Matched content will appear here...'. At the bottom, there is a 'Search Content' button.

Gradio allows the programmer to insert textboxes, sliders, buttons, and pictures. I used gradio Blocks() objects to create a UI like the one shown here, with a textbox, dropdown window and button.

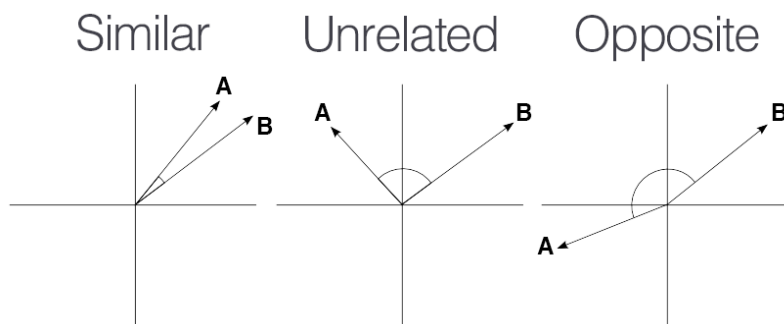
Task 3: Vectorized lexical search with TF-IDF

After completing the gradio UI for string-matching lexical search, my project mentor asked me to try doing the same with a vectorized approach to test the two and compare them. To do this, I used the tfidf vectorizer from the scikit-learn library which allows the vectorization of a database from strings into numerical form (vectors). TF-IDF stands for Term Frequency-Inverse Document Frequency. It is a statistic computed for each term which quantifies its importance in the dataset. It is computed as follows:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j
 df_i = number of documents containing i
 N = total number of documents

To do the lexical search, first the database is vectorized. Then, the query is also transformed into a vectorized form, and then we search the database of vectors for the one that is closest (in this high-dimensional space) to the query vector. The “closeness” is computed using cosine similarity, also calculated using the scikit-learn library.



$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Task 4: Integrating both forms of lexical search into the same program and UI

Next, I had to integrate and compare the two methods against one another, to see which one was faster and more accurate. I broke each function into parts and separated the code into separate files, each one having a different purpose. This was done following the principle of functional separation of code.

After integrating the two methods, I inserted a dropdown box into the UI where the user has the option to choose between the two methods of keyword extraction from query.

Task 5: Returning documents matching the query keywords

Now that the keywords from the query and database had been extracted, the next task was returning the matching documents from the database which are relevant to the user's query. To do this for string-matching, I made a new Json file containing the keywords present in each document of the database. Then, I did a simple dictionary lookup to find the documents matching the keywords found and returned them. In the case of vectorized lexical search (TF-IDF), I returned the 100 documents with the highest cosine similarity. This was all further integrated into the gradio UI. Then, I was tasked with generating a larger dataset for more accurate time and accuracy comparisons.

Finalized UI demonstration

The screenshot displays a web interface for a search application. At the top, there is a 'Query' input field containing the text 'What is reinforcement learning? Is it similar to machine learning?'. Below this is a 'Search Method' dropdown menu currently set to 'lexical_search'. The main section is titled 'Content Retrieval' and contains a 'Matched Content' box. This box lists search results with their titles and contents. At the bottom of the interface is a 'Search Content' button.

Query

What is reinforcement learning? Is it similar to machine learning?

Search Method

lexical_search

Content Retrieval

Matched Content

Time taken for lexical search: 0.016703367233276367

Title: Document 9
Content:
Reinforcement learning is a type of machine learning where agents learn by interacting with their environment.

Title: Document 1
Content:
Deep learning is a subset of machine learning.

Title: Document 4
Content:
This document is about machine learning. machine learning is fascinating.

Title: Document 6
Content:
Machine learning algorithms are used in a variety of applications.

Title: Document 8
Content:
Supervised learning and unsupervised learning are two types of machine learning.

Search Content

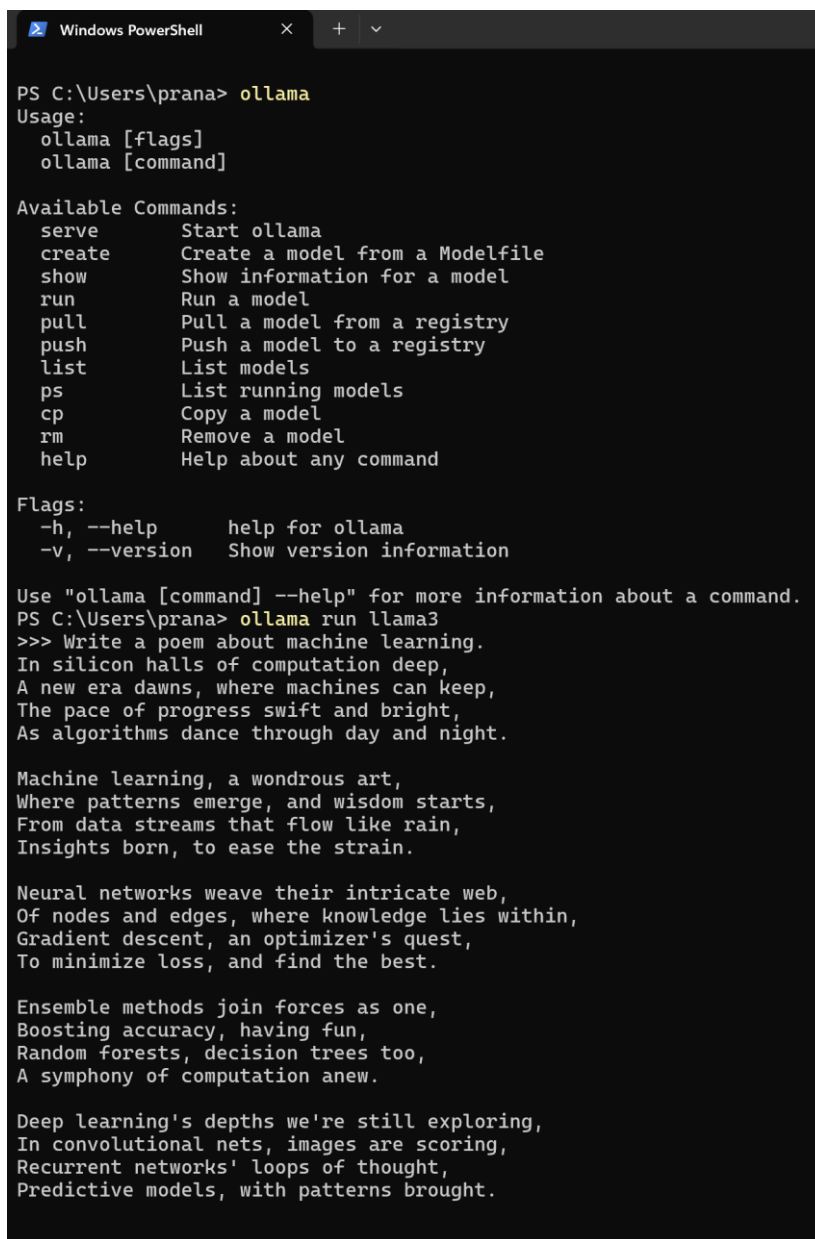
For example, in this query, “machine learning”, “reinforcement learning”, and “learning” were identified as keywords from the query and relevant documents from the database are returned.

POST-MIDSEM

Task-6: Local LLM inference using ollama

Now that the search methods had been completed, my next task was to connect it to an LLM to process the queries and return the final result to the user. For this, I searched the internet and in particular, found a ranking of different LLMs. From this list, I decided that the best choice for the job would be the llama-3 8B param model from Meta AI. It was small, trained on a vast amount of data, yet capable. I used the ollama library to run inference on the model locally, as I was not allowed access to Jio's GPUs.

Running llama3 locally using ollama



```
PS C:\Users\prana> ollama
Usage:
  ollama [flags]
  ollama [command]

Available Commands:
  serve    Start ollama
  create   Create a model from a Modelfile
  show     Show information for a model
  run      Run a model
  pull     Pull a model from a registry
  push     Push a model to a registry
  list     List models
  ps       List running models
  cp       Copy a model
  rm       Remove a model
  help     Help about any command

Flags:
  -h, --help      help for ollama
  -v, --version   Show version information

Use "ollama [command] --help" for more information about a command.
PS C:\Users\prana> ollama run llama3
>>> Write a poem about machine learning.
In silicon halls of computation deep,
A new era dawns, where machines can keep,
The pace of progress swift and bright,
As algorithms dance through day and night.

Machine learning, a wondrous art,
Where patterns emerge, and wisdom starts,
From data streams that flow like rain,
Insights born, to ease the strain.

Neural networks weave their intricate web,
Of nodes and edges, where knowledge lies within,
Gradient descent, an optimizer's quest,
To minimize loss, and find the best.

Ensemble methods join forces as one,
Boosting accuracy, having fun,
Random forests, decision trees too,
A symphony of computation anew.

Deep learning's depths we're still exploring,
In convolutional nets, images are scoring,
Recurrent networks' loops of thought,
Predictive models, with patterns brought.
```

Task 7: Making a sample dataset

My next task was building a sample dataset to test the RAG capabilities. The chatbot being developed was to be used for JioMart, an online shopping service operated by Jio. The purpose of the bot was to act as a shopping assistant which could answer the customer's queries about the products in the shop, by retrieving information from the inventory database.

I was tasked with creating 4 sample databases:

1. Customer data
2. Inventory information
3. Customer service queries asked by users, and the provided answers and solutions.
4. User reviews of the products

I created data for 100 customers, 120 products, reviews for each product, and 2 questions with answers for each product. Then, I edited the lexical search and TF-IDF code from the search portion in order to connect the new database to the old code.

Task 8: Ensuring referential integrity in the databases

After making the databases, my mentor noticed one crucial problem: The databases did not maintain referential integrity. Referential integrity refers to the feature of a database where each table in it must be consistent with all others. For example, each review must be left by a customer who has bought that product before, each product bought by a customer must actually exist in the inventory database, etc.

To connect the databases and make sure that they were consistent with each other, I wrote a python program that assigned a random number of random products to each user and looked into the inventory file to obtain the details of that product. I also made sure that the reviews were all written by users who had bought that item. Also, my mentor asked me to clean up the database and make the age range of customers more diverse, and add out-of-stock items to the inventory, along with other small changes.

Example of entries in the Inventory table

```
1  [
2    {
3      "product_id": "P0001",
4      "category": "shoes",
5      "available_units": 5,
6      "price_of_each_unit": 89.99,
7      "product_name": "Puma Running Shoes",
8      "description": "Comfortable running shoes for men by Puma."
9    },
10   {
11     "product_id": "P0002",
12     "category": "shoes",
13     "available_units": 3,
14     "price_of_each_unit": 79.99,
15     "product_name": "Adidas UltraBoost",
16     "description": "High-performance running shoes for men by Adidas."
17   },
18   {
19     "product_id": "P0003",
20     "category": "cosmetics",
21     "available_units": 10,
22     "price_of_each_unit": 29.99,
23     "product_name": "L'Oreal Mascara",
24     "description": "Lengthening and volumizing mascara for women."
25   },
26   {
27     "product_id": "P0004",
28     "category": "electronics",
29     "available_units": 20,
30     "price_of_each_unit": 499.99,
31     "product_name": "Apple iPhone 13",
32     "description": "Latest iPhone model for men and women with advanced features."
33   },
34 ]
```

Task 9: Connecting the database to the LLM

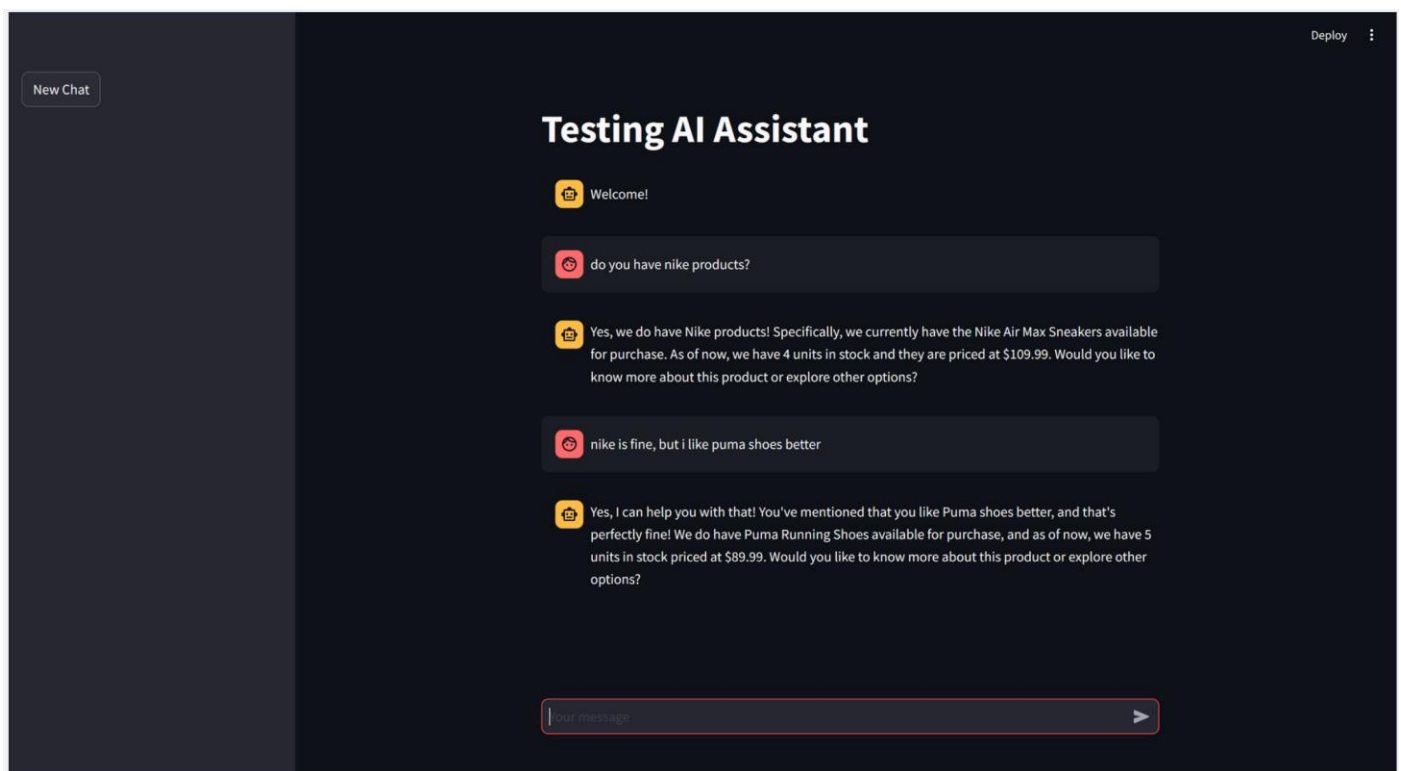
Now that the database was completed, I was to connect it to the LLM and test the accuracy and speed of the output. For this, my mentor provided me with a python template with a streamlit UI function built in, and asked me to add in functions implementing LLM functionality. I used ollama's python library for this task. In order to make sure the model gave responses befitting that of a shopping assistant, I tried different methods but eventually settled on passing a system prompt to the LLM during call-time to tell the LLM how to answer. For example, the chatbot was instructed not to reply to questions unrelated to shopping and to ignore gibberish prompts, etc. This was a quick and easy solution, as opposed to classifying queries into relevant and irrelevant queries using another LLM call or separate classifier neural network.

After connecting it, I tested the relevance and quality of the outputs it produced. The user was able to chat with the chatbot in a locally hosted webpage generated using streamlit.

Task 10: Perfecting the responses with prompt engineering, chat context

My final task to completing the chatbot was making sure the chatbot was able to understand context from previous queries. For example, if the user asks for Nike products and then asks for shoes, the chatbot should respond with Nike shoes in particular. The way I implemented this was by adding the past 5 queries to a queue, where the oldest queries are popped out of the queue. This way, recent context was taken into account without taking account every query ever asked. The chat history queue was then passed to the LLM as part of the prompt. Thus, the chatbot was able to remember previous data. However, the TF-IDF search also needed to use the context provided to give more relevant results from the database, so I also edited the user's prompt using an LLM which edited the user's prompt before searching the database with a query and before adding it to the conversation history. Thus, the retrieved documents were able to also become more relevant using chat context.

Completed Chatbot: Sample Conversation



Glossary

1. **LLMs (Large Language Models):** A Large Language Model (LLM) is a type of artificial intelligence (AI) model that has been trained on vast amounts of text data to understand, generate, and manipulate human language. LLMs use deep learning techniques, particularly neural networks, to predict the probability of a word or sequence of words based on context. They are capable of tasks such as text completion, translation, summarization, and conversation generation. Examples of LLMs include GPT (Generative Pre-trained Transformer) models and BERT (Bidirectional Encoder Representations from Transformers).
2. **RAG (Retrieval-Augmented Generation):** Retrieval-Augmented Generation (RAG) is a framework in AI that combines the strengths of retrieval-based and generative models. It involves using a retrieval component to fetch relevant documents or information from a database, which is then used as context by a generative model to produce a response or output. This approach is particularly useful for tasks that require accurate, contextually relevant information, such as question answering or conversational agents.
3. **Stopwords:** Stopwords are common words in a language that are typically filtered out in natural language processing (NLP) tasks because they provide little to no meaningful information in text analysis. Examples of stopwords include words like "the," "is," "in," and "and." Removing stopwords can reduce the amount of data that needs to be processed and helps in focusing on the more significant words that carry the core meaning of the text.
4. **TF-IDF (Term Frequency-Inverse Document Frequency):** TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It is calculated by multiplying two metrics: Term Frequency (TF), which measures how frequently a term appears in a document, and Inverse Document Frequency (IDF), which measures how important a term is by considering how often it appears across all documents in the corpus. High TF-IDF scores indicate that a term is significant in a specific document but not common across the corpus.
5. **NLP (Natural Language Processing):** Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human languages. It involves the development of algorithms and models that enable computers to understand, interpret, and generate human language in a way that is both meaningful and useful. NLP encompasses a wide range of tasks, including text analysis, language translation, sentiment analysis, and conversational AI.