Retrieval-Augmented Generation (RAG) for ROS Q&A Across Four Sub-Domains

By - Adithya Srinivasa Raghavan - as20373, Srikrishna Kumarasamy - sk11595

Abstract

In this report, we present an AI-powered system utilizing Retrieval-Augmented Generation (RAG) to facilitate question-and-answering (Q&A) for the Robot Operating System (ROS) framework. The system looks at four subdomains of the Robot Operating System: **ROS2 robotics middleware**, **Nav2 navigation**, **MoveIt2 motion planning**, and **Gazebo simulation**. By using the Llama 3.1 large language model; the project integrates “Docker” for containerization, data scraping for knowledge base creation, MongoDB for storage of the scraped data, Qdrant for vector similarity search,, and Gradio for a user-friendly interface. We aim to use this RAG system to bridge gaps in roboticists' understanding of these subdomains by providing context-aware, accurate responses to their queries. This report outlines the methodology, technical architecture, and outcomes of our RAG system.

Table of Contents

1. Introduction

As robotics grows increasingly complex, developers are encountering significant learning curves and challenges in mastering robotic middleware, navigation algorithms, motion planning techniques, and simulation environments. The Robot Operating System (ROS), though a widely adopted robotics framework, includes extensive documentation and resources, in which it can be difficult finding precise answers.

This project leverages **Retrieval-Augmented Generation (RAG)** to create an intelligent Q&A system similar to a chatbot engineered for the ROS framework. Unlike traditional knowledge bases, which rely on static text search, RAG systems combine neural retrievers and generative models to provide contextually accurate and nuanced answers. By targeting four critical ROS subdomains, this system addresses a variety of technical queries, from middleware APIs to motion planning algorithms.

Objectives

1. Create a scalable and modular RAG pipeline.
2. Perform data ingestion using web scraping from various data sources.
3. Use MongoDB for document storage and to saved chunked data.
4. Use Qdrant for vectorization of stored data and efficient vector similarity search
5. Provide a robust and user-friendly interface via Gradio.

2. Background

2.1 ROS and ROS Subdomains

2.1.1 ROS2 Robotics Middleware

ROS provides middleware that facilitates real-time processing, distributed computing, and communication as well as in creating robotics applications. Usual queries/questions for developers on this subdomain tend to focus on API usage, node configuration, and DDS protocols.

2.1.2 Nav2 Navigation

Robot navigation is made easier with the Nav2 framework. It provides mechanisms for covering path planning, obstacle avoidance, and localization. Usual queries/questions for developers on this domain include tuning planner parameters or understanding costmaps.

2.1.3 MoveIt2 Motion Planning

MoveIt2 is the motion planning library for ROS2, widely used for robotic arm kinematics and collision checking. Users often ask about algorithm selection or hardware interface integration.

2.1.4 Gazebo Simulation

Robot models and algorithms are tested in virtual worlds using Gazebo, a simulation environment. This subdomain's queries frequently concern model spawning, sensor setups, and physics engines.

2.2 Retrieval-Augmented Generation (RAG)

RAG can retrieve pertinent information and create logical responses by combining information retrieval with generative models such as GPT. For complicated, domain-specific Q&A based chatbots, RAG can work very well.

---

3. System Design

3.1 Architecture Overview

The RAG system makes use of ZenML to create a comprehensive data processing pipeline that includes modules for data collection, cleaning and chunking and vectorization using MongoDB and Qdrant. For end-to-end processing requirements, this architecture guarantees scalability and efficient data flow.:

1. **Knowledge Base Construction**: Data scraping to collect resources through various sources such as GitHub, Websites and YouTube.

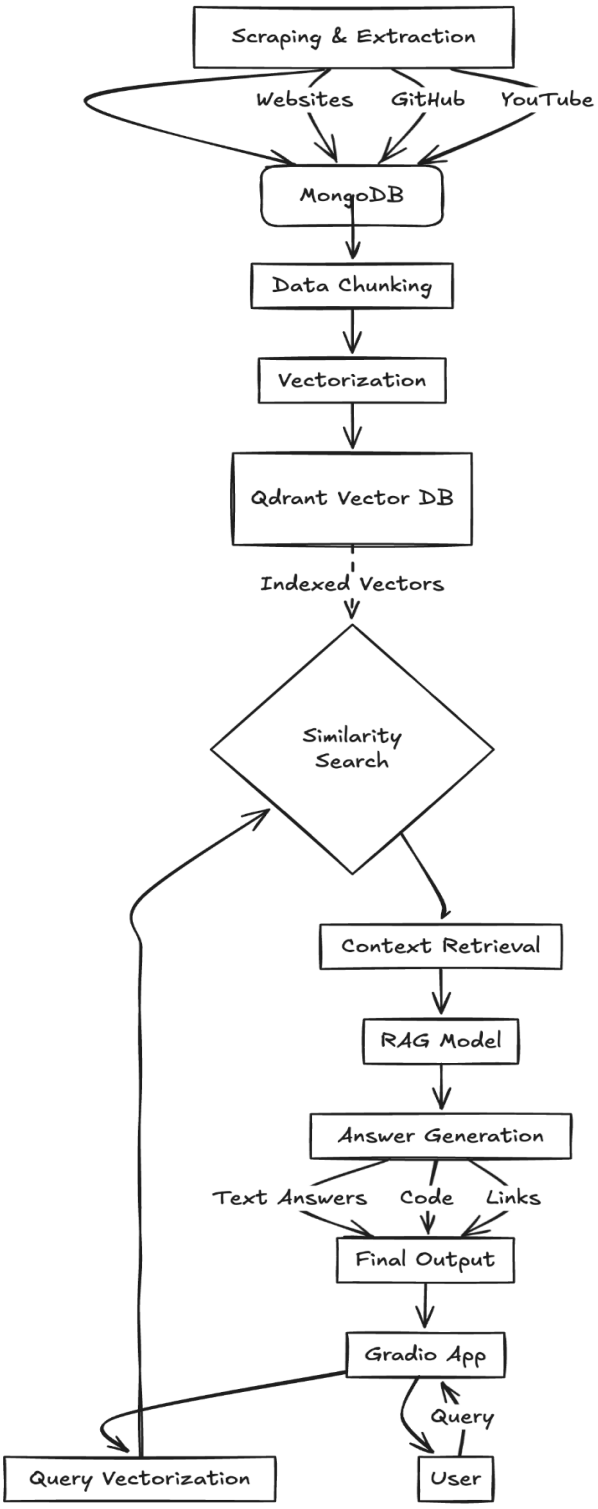
2. **Data Storage**: MongoDB for data storage including chunked data and Qdrant for vector indexing.

3. **Retrieval**: Retrieval using Qdrant for similarity search.

4. **Fine Tuning:** A Llama 3.1-8B model was chosen and fine-tuned using input-context-response triplets on the collected data

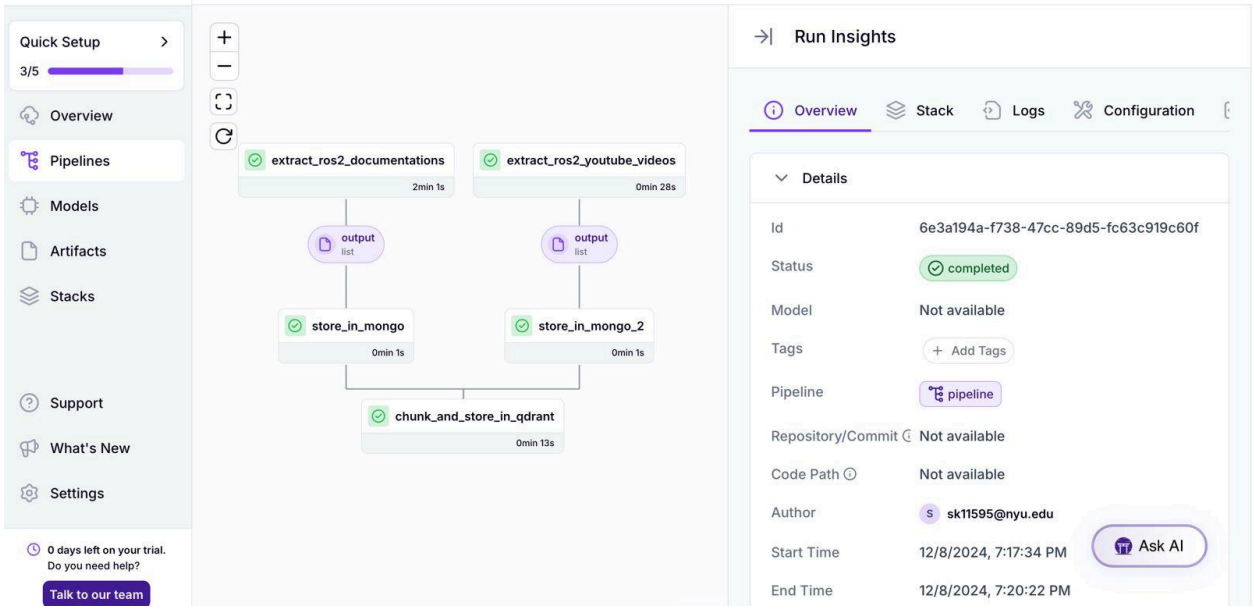
5. **Generation**: Answer generation using a fine-tuned generative model.

6. **User Interface**: A Gradio-powered web interface for interactive Q&A through which answers can be generated as text or code or a combination of both with relevant links provided.



3.2 Constructed ZenML Pipeline, Container and Other Services

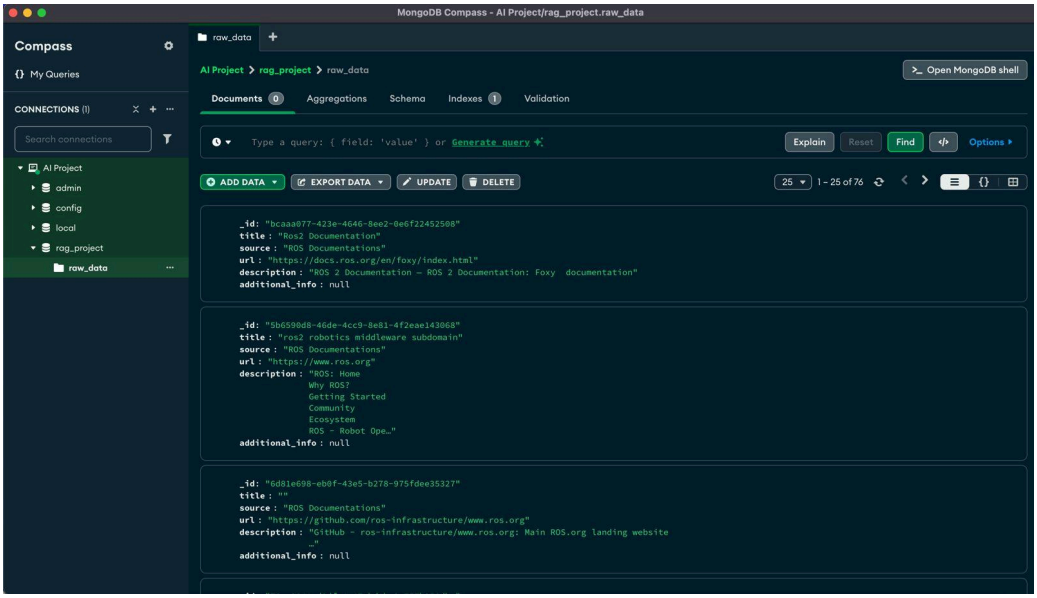
A. ZenML Pipeline



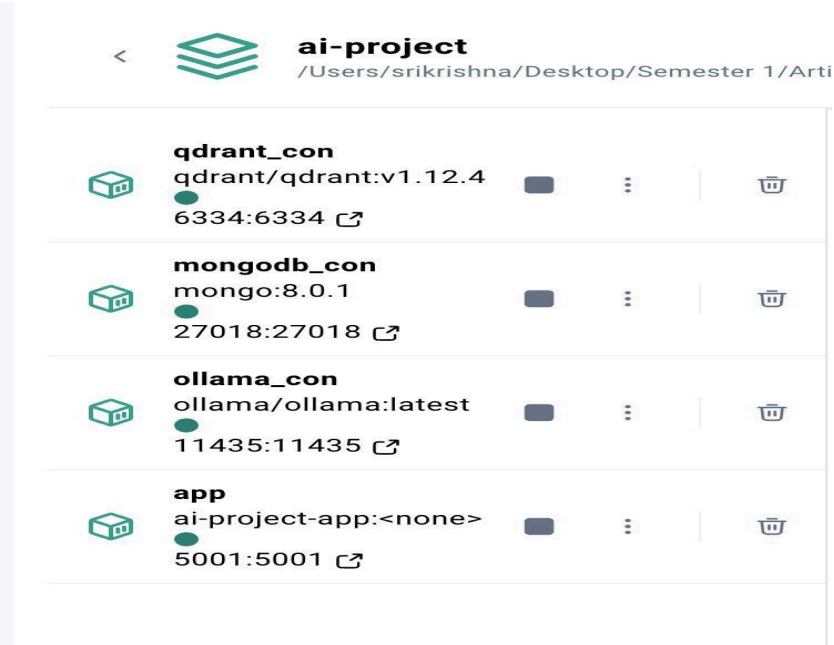
B. Docker PS

```
srikrishna@Mac-11028 AI-Project % docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS          PORTS                                        NAMES
2c6525e333f3   ai-project-app       "bash -c 'zenml logi…"   57 seconds ago   Up 56 seconds   0.0.0.0:5001->5001/tcp                       app
b4022eacb394   ollama/ollama:latest "/bin/ollama serve"      5 minutes ago    Up 57 seconds   11434/tcp, 0.0.0.0:11435->11435/tcp          ollama_con
9b28325c3f8e   qdrant/qdrant:v1.12.4 "./entrypoint.sh"       9 minutes ago    Up 57 seconds   6333/tcp, 0.0.0.0:6334->6334/tcp             qdrant_con
c2fd0cca9e77   mongo:8.0.1          "docker-entrypoint.s…"   9 minutes ago    Up 57 seconds   27017/tcp, 0.0.0.0:27018->27018/tcp          mongodb_con
b6baaec3d9ba   qdrant/qdrant        "./entrypoint.sh"        9 hours ago      Up 2 hours      0.0.0.0:6333->6333/tcp, 6334/tcp             eloquent_merkle
4d0740f06630   mongo                "docker-entrypoint.s…"   9 hours ago      Up 2 hours      0.0.0.0:27017->27017/tcp                     mongodb
```

C. Mongo DB



D. Docker Containers



3.3 Docker Compose

Containerization ensures modularity, scalability, and cross-platform compatibility. The Docker Compose configuration orchestrates services, including:

- **Scraper Service**: Automates data scraping.
- **Database Service**: Runs MongoDB for document storage.
- **Vector Index Service**: Hosts Qdrant for similarity search.
- **API Service**: Executes the RAG pipeline via `app.py`.
- **UI Service**: Hosts the Gradio Q&A interface.

---

4. Implementation

4.1 Data Scraping

Sources

Data was scraped from GitHub repositories, websites which have ROS related information and documentation as well as from YouTube. This ensures a comprehensive knowledge base spanning the 4 sub-domains namely, middleware configurations, navigation, motion planning and simulation nuances.

Scraper Implementation

We utilized Python scripts including `BeautifulSoup` for scraping data from websites, documentation as well as YouTube and additionally used `selenium` for dynamic content extraction and to ensure we did not miss out on relevant information.

```
def scrape_page_with_headings(self, url):

    print(f"Scraping URL: {url}")

    response = requests.get(url, timeout=10)

    response = response.text

    soup = BeautifulSoup(response, 'lxml')

    page_content = soup.get_text(separator="\n", strip=True)

    links = []

    for link in soup.find_all('a', href=True):

        href = link['href']

        if href.startswith("http") or href.startswith("/"):

            full_url = href if href.startswith("http") else
    os.path.join(os.path.dirname(url), href)

            if validators.url(full_url):

                links.append({"text": link.get_text(strip=True), "url": full_url})

    return page_content, links
```

4.2 Data Storage and Indexing

MongoDB

The scraped data is stored in MongoDB, enabling efficient retrieval. The data is also chunked to ensure it makes more sense when being consumed.

```
from dataclasses import dataclass

from typing import Optional, Dict, Union

@dataclass

class Content:

   _id: str

   title: str

   source: str

   url: str

   description: Union[str, Dict, None] = None

   additional_info: Optional[str] = None
```

```
class MongoDB:
```

```
  def __init__(self):

      self.client = client = MongoClient('localhost', 27017)

      self.db = client.rag_project

      self.collection = self.db['raw_data']
```

Qdrant - Qdrant was used for dense vector indexing, allowing cosine similarity search for context retrieval.

```
class Qdrant:



  def __init__(self):

      self.encoding_model = SentenceTransformer('all-MiniLM-L6-v2')

      self.client = QdrantClient(host="localhost", port=6333)

      if not self.client.collection_exists("rag_data"):

          self.client.create_collection(

              collection_name="rag_data",

              vectors_config=VectorParams(size=384, distance=Distance.COSINE)
```

4.3 Retrieval-Augmented Generation Pipeline

The retrieval system queries Qdrant to fetch relevant documents based on a user's question which are received through the Gradio App. The data is given to a fine-tuned Llama 3.1 8B model, which synthesizes an answer.

```
async def get_response(query):

   query_vector = model.encode(query)

   results = client.search(

       collection_name="rag_data",

       query_vector=query_vector,

       limit=10

   )

   context = ""

   for index, result in enumerate(results):

       context += f"Title : {result.payload['title']}\nLink :
       {result.payload['url']}\nContext {index} : {result.payload['description']}\n\n"

   response = ""

   async for word in llama.query(prompt_template.format(context, query)):

       response += word

       yield response
```

4.4 Gradio Interface

Gradio provides a simplified user interface which is intuitive to ask questions and view responses. The responses are generated as text or code or a combination of both with relevant links provided.

```
with gr.Blocks() as demo:

   gr.Markdown("# Ros Question Answering System")

   inputs=gr.Dropdown(choices=questions, label="Choose a question")

   output_text = gr.Markdown(label="Streaming Output")

   submit_button = gr.Button("Generate Answer")

   submit_button.click(

       fn=get_response,

       inputs=inputs,

       outputs=output_text

   )
```

4.5 Fine Tuning

The base Llama 3.1-8b model was optimised to manage ROS-related Q&A system. A collection of context-input-response triplets; gathered from domain-specific data was used for training. To improve its comprehension of technical queries and guarantee correctness and

relevance in responses, this base model was provided with the carefully chosen data. For effective inference in Q&A systems, the refined model has been optimised and stored onto Hugging-Face.

---

5. Conclusion

This project demonstrates the potential of Retrieval-Augmented Generation for domain-specific Q&A chatbot like systems. By targeting ROS subdomains, we address a critical need for efficient knowledge access in robotics. The integration of modern tools like MongoDB, QDrant and Gradio ensures a scalable and user-friendly solution.
