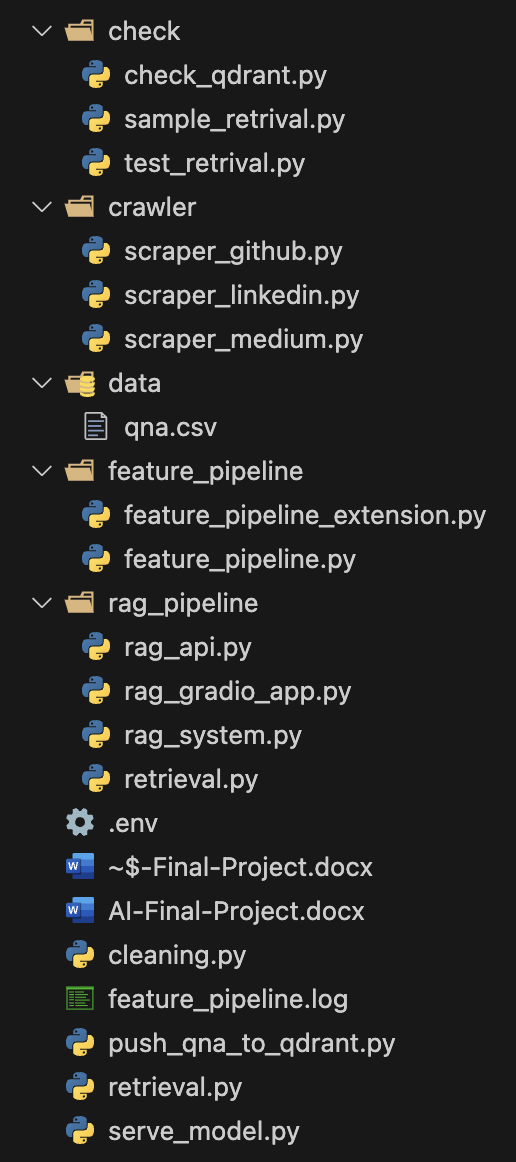
**CS-GY 6613 Final Project**

* **Project Overview:**
  + Briefly introduce the concept of Retrieval Augmented Generation (RAG) systems.
  + Explain the core idea: combining retrieval-based and generation-based models to create a system that answers domain-specific questions using both large-scale information retrieval and language generation.
  + Outline the domain of your project: ROS2 robotics, navigation stack, and the various subdomains like ROS2 middleware, Nav2 navigation, motion planning, and simulation.
* **Objective:**
  + The goal of this project is to build a proof-of-concept (PoC) RAG system that can answer detailed questions related to ROS2 robotics subdomains using a mix of document retrieval and model-based text generation.
  + Mention that the report will focus on the milestones, challenges, architecture, and implementation of the system.

**2. System Architecture**

* **Structure:**

****

* **System Overview:**
  + Provide an overview of the architecture, explaining how different components interact to form the RAG system. Mention each major module:
    - **Crawler/Scraper**: Gathers raw data from GitHub repositories, Medium articles, and LinkedIn profiles related to ROS2 robotics.
    - **Cleaning**: Raw data is cleaned and stored in MongoDB.
    - **Featurization**: Converts the cleaned data into question-answer pairs stored in both MongoDB and Qdrant vector database.
    - **Check**: Checks Qdrant retrieval and connection.
    - **Serving Model**: The fine-tuned Hugging Face model is used to generate responses.
    - **Gradio App**: Allows users to interact with the system by selecting questions from a dropdown menu.
    - **ClearML**: Orchestrates and tracks experiments related to data ingestion and model training.
* **Flow of Data:**
  + Describe the data flow in the system starting from data scraping, cleaning, storing in MongoDB, converting to question-answer pairs, storing in Qdrant, model fine-tuning, and final serving via the Gradio app.
* **Components Diagram:**
  + Include a diagram of your system architecture (you can use the screenshot you attached earlier).

**3. Milestones & Deliverables**

**1. Environment and Tooling Milestone**

* **Objective**: Set up a Docker Compose environment for the development of the RAG system.
* **Deliverables**:
  + Docker Compose file with containers for MongoDB, Qdrant, ClearML, and the app (PyTorch/TF models).
  + Screenshot showing all services running.
* **Setup Details**:
  + Explain how you configured the Docker setup for the necessary components and dependencies.

**2. ETL Milestone**

* **Objective**: Implement an ETL pipeline to ingest multiple media sources like GitHub repositories, YouTube videos, and Medium articles.
* **Details**:
  + Discuss the structure of your scrapers (e.g., scraper\_github, scraper\_medium, scraper\_linkedin) and how they fetch data.
  + Describe the cleaning pipeline and how the raw data is stored in MongoDB.
  + Provide examples of scraped data and their storage format.

**3. Featurization Pipeline Milestone**

* **Objective**: Process raw data into question-answer pairs and store them in MongoDB and Qdrant.
* **Details**:
  + Describe how feature\_pipeline.py and related scripts convert raw data into question-answer pairs.
  + Explain the usage of OpenAI API (if available) or fallback to manually created question-answer pairs stored in a qna.csv file.
  + Discuss how the featurized data is stored in MongoDB and pushed into the Qdrant vector database.

**4. Fine-tuning Milestone**

* **Objective**: Fine-tune a model for domain-specific question answering.
* **Details**:
  + Explain your approach to fine-tuning the Hugging Face model (based on GPT-2 or other base models).
  + If you didn’t have access to OpenAI API, explain how you generated data for fine-tuning and used a self-made dataset.

**5. Deploying the App Milestone**

* **Objective**: Develop and deploy a Gradio app to allow users to interact with the RAG system.
* **Details**:
  + Describe the Gradio app’s functionality, including the dropdown interface for users to select questions.
  + Explain how the app connects to the Qdrant vector database to retrieve the top 5 most similar question-answer pairs.
  + Describe how the fine-tuned model is used to generate a coherent and accurate answer.
* **Screenshots**: Provide screenshots of the Gradio app in action, showing the user interface and example interactions.

**4. Detailed Component Explanation**

**1. Crawler/Scraper Modules**

* **Purpose**: Collect domain-specific data from external sources.
* **Components**: scraper\_github, scraper\_medium, scraper\_linkedin.
* **Explanation**: Describe the functionality of each scraper module, the types of data collected, and how it's stored in MongoDB for further processing.

**2. Data Cleaning Module (cleaning.py)**

* **Purpose**: Clean and preprocess raw data before further usage.
* **Process**: Describe the data cleaning steps (e.g., removing irrelevant content, normalizing formats, handling missing data) and the final output that is stored in MongoDB.

**3. Featurization Pipeline**

* **Purpose**: Convert cleaned data into question-answer pairs suitable for the RAG model.
* **Process**: Describe how the featurization is done using OpenAI (if available) or manual data (from qna.csv), and how the question-answer pairs are stored in MongoDB and Qdrant.

**4. Model Serving (serve\_model.py)**

* **Purpose**: Load and serve the fine-tuned model to answer user queries.
* **Details**: Explain how serve\_model.py loads the fine-tuned Hugging Face model and uses it to generate answers based on the question retrieved from the Qdrant vector database.

**5. Gradio App (rag\_pipeline/\_gradio\_app.py)**

* **Purpose**: Allow users to interact with the RAG system through a web interface.
* **Details**: Describe how the Gradio app works, including how it takes input from users, queries the Qdrant vector database, and uses the fine-tuned model to generate answers.

**5. Results and Evaluation**

* **Evaluation of RAG System**:
  + Discuss how well the system performs in terms of answering domain-specific questions.
  + Include any evaluation metrics (e.g., accuracy, precision, recall, or qualitative evaluation).
  + Explain how you measure the utility and specificity of the answers provided by the system to ROS2 developers.
* **Challenges Faced**:
  + Discuss any technical challenges faced during the project (e.g., setting up Docker containers, fine-tuning the model, integration issues).
  + Mention the absence of OpenAI API key and how you worked around this limitation.

**6. Conclusion and Future Work**

* **Summary**: Summarize the work done and the overall success of building a RAG-based question-answering system for ROS2 robotics.
* **Future Enhancements**:
  + Suggest areas for future improvement, such as better fine-tuning of the model, additional data sources, or more refined question-answer pair generation.

**7. References**

* Include any references to academic papers, documentation, or resources you used throughout the project (e.g., Hugging Face documentation, ROS2 tutorials, ClearML, etc.).

**Appendices**

* Include any relevant code snippets, configuration files, or additional screenshots that may help to clarify the report.

**Member Details**

**Pratham Saraf** – ps5218

**Siddh Mandirwala** – sm12505