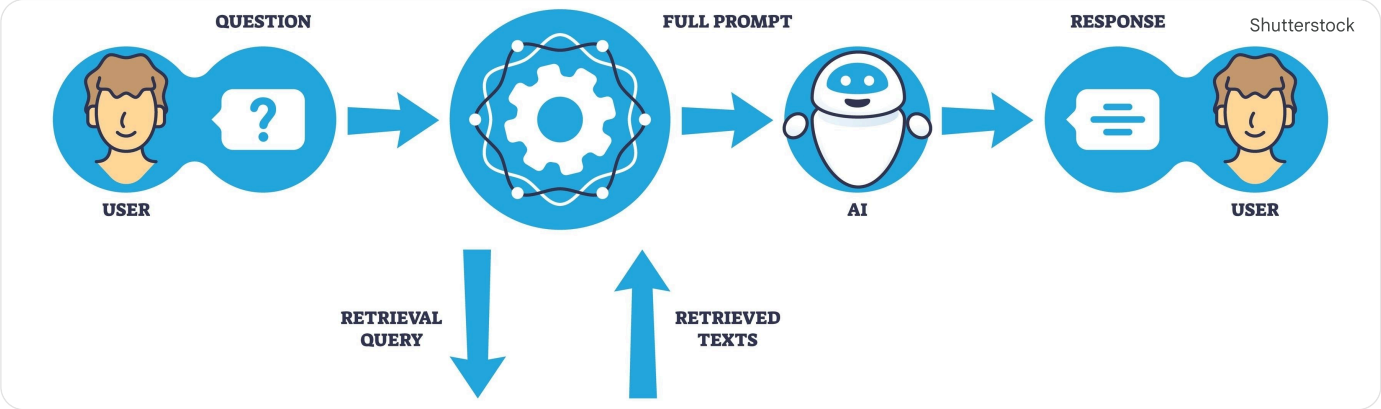
# Ollama + Qdrant RAG Application

This project implements a Retrieval-Augmented Generation (RAG) system using **FastAPI** as the backend API, **LlamaIndex** as the orchestration framework, **Ollama** for hosting open-source LLMs and embeddings, and **Qdrant** as the high-performance vector store. It includes a minimal HTML dashboard for easy interaction and status monitoring.




## 🚀 Key Features

- **Modular Architecture:** Clean separation between API ( `api.py` ), Ingestion Logic ( `ingest.py` ), and Configuration ( `config.yaml` ).
- **Open-Source Stack:** Leverages **Ollama** for local, efficient LLM and Embedding model hosting.
- **High-Performance Vector Store:** Uses **Qdrant** for vector storage and semantic search.
- **FastAPI Backend:** Provides RESTful endpoints for RAG querying, ingestion, and status checks.
- **Web UI:** A simple, responsive `index.html` dashboard to query the RAG system and trigger data ingestion via the API.
- **Dynamic Configuration:** All service URLs, models, and RAG parameters are managed through `config.yaml` .

## 🛠️ Prerequisites

To run this application, you need the following installed locally:

1. **Python 3.10+**
2. **Git** (for cloning)
3. **Docker** (for running Qdrant)
4. **Ollama** (the application for serving LLMs/Embeddings)

### Ollama Model Setup

After installing Ollama, you must pull the models defined in your `config.yaml` .

```
# Verify Ollama is running and list available models
ollama list

# Pull the required models (as per config.yaml)
ollama pull nomic-embed-text
ollama pull gemma3:1b-it-qat

# Optional model you mentioned:
ollama pull embeddinggemma
```

## ⚙️ Configuration

All major settings are controlled by the `config.yaml` file.

| Parameter | Description | Default Value | Used By |
|---|---|---|---|

| | | | |
|---|---|---|---|
| OLLAMA_LLM_MODEL | The LLM used for generating the final answer. | gemma3:1b-it-qat | api.py |
| OLLAMA_EMBEDDING_MODEL | The model used for converting text into vectors. | nomic-embed-text | api.py , ingest.py |
| OLLAMA_BASE_URL | The endpoint for your Ollama server. | http://localhost:11434 | api.py , ingest.py |
| QDRANT_HOST | The hostname for the Qdrant instance. | localhost | api.py , ingest.py |
| QDRANT_PORT | The client port for Qdrant API. | 6333 | api.py , ingest.py |
| COLLECTION_NAME | The name of the vector collection in Qdrant. | ai_knowledge_base_recursive | api.py , ingest.py |
| DATA_DIR | Directory where your source documents are located. | data | ingest.py |
| CHUNK_SIZE | Size of text chunks for indexing. | 512 | ingest.py |
| TOP_K_CHUNKS | Number of top documents to retrieve for RAG context. | 3 | api.py |

## 🚀 Local Setup & Execution

### Step 1: Clone Repository & Install Dependencies

```
# Clone the repository (if applicable)
# git clone [your_repo_url]
# cd n26rag-app

# Create and activate a virtual environment
python -m venv venv
source venv/bin/activate  # On Windows, use: venv\Scripts\activate

# Install required Python packages
pip install -r requirements.txt
# Note: You'll need to create a requirements.txt containing:
# fastapi, uvicorn, python-multipart, pydantic, qdrant-client, llama-index, llama-index-llms-ollama, llama-ir
```

### Step 2: Run Qdrant Vector Store

Use the provided Docker command to run Qdrant, ensuring data persistence with a volume mount:

```
docker run -p 6333:6333 -p 6334:6334 -v qdrant_storage:/qdrant/storage qdrant/qdrant
```

Qdrant will be running on `http://localhost:6333` .

### Step 3: Run Ollama Server

Ensure your Ollama server is running in the background:

```
ollama serve
```

Ollama will typically run on `http://localhost:11434` .

### Step 4: Add Documents

Create a directory named `data` and place your documents (e.g., `.txt`, `.pdf`, `.md`) inside it.

```
mkdir data
# e.g., cp my_policy_document.pdf data/
```

### Step 5: Start the FastAPI Application

Run the main API server.

```
python api.py
# Or, if you want to use uvicorn directly: uvicorn api:app --host 0.0.0.0 --port 8000
```

The FastAPI application will be running at `http://localhost:8000`.

## 🌐 Usage

The application provides both a web interface and direct API endpoints.

### 1. Web Dashboard (`index.html`)

Open your web browser and navigate to:

🔗 http://localhost:8000/

The dashboard allows you to:

- **Monitor Status:** See the connectivity of Ollama and Qdrant in real-time.
- **Trigger Ingestion:** Click the "Trigger Document Ingestion" button to load files from the `data/` directory into Qdrant.
- **Query the RAG:** Enter a question in the search box to get an LLM-generated answer grounded by your documents.

### 2. API Endpoints

| Endpoint | Method | Description |
|---|---|---|
| `/` | GET | Serves the `index.html` Web UI. |
| `/status` | GET | Checks the connectivity and current status of Ollama and Qdrant. |
| `/ingest` | POST | Triggers the ingestion process, loading new documents from the `data/` directory and indexing them in Qdrant. |
| `/query` | POST | Performs the RAG query. Accepts a JSON payload with the key `"query"` and returns the LLM's `"answer"` and `"sources"`. |

Example Query using `curl`:

```
curl -X POST "http://localhost:8000/query" \
     -H "Content-Type: application/json" \
     -d '{"query": "Summarize the key findings from the most recent document."}'
```

## 📦 Docker and CI/CD

The following commands are used for building and pushing the Docker image for Continuous Integration/Continuous Deployment (CI/CD) workflows (likely managed via your `.github/workflows/ci.yml`).

**Docker Build and Push**

To create and publish a container image of the FastAPI application:

```
# Build the Docker image
docker build -t gowthambc/n26rag:v1.0 .

# Push the Docker image to the registry (Docker Hub in this case)
docker push gowthambc/n26rag:v1.0
```

This comprehensive `README` should make it very easy for anyone—or for your future self—to understand, set up, and run this RAG application! Let me know if you want to refine any of these sections.