ML + LLM Pipeline Orchestration Documentation

Overview:

This documentation outlines a complete orchestration project integrating a Machine Learning (ML) pipeline with a Retrieval-Augmented Generation (RAG) system. The project uses modern tools like Apache Airflow, MLflow, FastAPI, and Docker for seamless automation, model serving, and document-based LLM querying.

Part A: ML Pipeline (Wine Quality Prediction)

Objective: Automate the training, evaluation, logging, and deployment of a classification model for predicting wine quality.

Tech Stack:

- **Airflow**: Workflow orchestration
- **Pandas**: Data preprocessing and feature engineering
- **Scikit-learn**: ML model development
- **MLflow**: Model tracking and experiment logging
- **FastAPI**: REST API for predictions
- **Docker**: Containerization and orchestration

Directory Structure:

```
ml_pipeline/
├── dags/
│   └── ml_pipeline_dag.py
├── scripts/
│   ├── ingest_data.py
│   ├── preprocess.py
│   ├── feature_engineering.py
│   ├── train_model.py
│   ├── evaluate_model.py
│   └── log_to_mlflow.py
├── model_service/
│   ├── Dockerfile
│   ├── main.py
│   └── requirements.txt
├── data/
└── requirements.txt
```

ML Workflow:

1. **Ingest**: Load dataset (winequality-red.csv)
2. **Preprocess**: Clean, normalize and handle missing values
3. **Feature Engineering**: Generate additional features
4. **Train**: Train a classification model (e.g., RandomForest)
5. **Evaluate**: Calculate metrics (accuracy, precision, etc.)
6. **Log to MLflow**: Store model and metadata
7. **Serve**: Expose prediction API via FastAPI

FastAPI ML Inference Test:

```
curl -X POST http://localhost:8000/predict \
  -H "Content-Type: application/json" \
```

-d '{"data": [[7.4, 0.7, 0, 1.9, 0.076, 11, 34, 0.9978, 3.51, 0.56, 9.4]]}'

Response:
{"predictions": [5]}

Part B: RAG LLM Pipeline (Mini POC)

Objective: Retrieve context from PDF documents and answer user queries using embedded vectors and a lightweight transformer model.

Tech Stack:

- **FAISS**: Efficient similarity search over embeddings
- **DistilBERT**: For text embeddings
- **FastAPI**: For serving the RAG system
- **Docker**: For containerization

Directory Structure:

```
rag_pipeline/
├── app/
│   ├── main.py
│   ├── vector_store.py
│   ├── requirements.txt
│   └── Dockerfile
├── docs/
├── embeddings/
└── requirements.txt
```

Workflow:

1. Upload PDF to docs/
2. Split and embed content
3. Store in FAISS index
4. Query via REST API
5. Return LLM-generated answer using context

FastAPI RAG Query Test:

curl -X GET http://localhost:8001/query?q=What is the project about?

Response:

{
"query": "What is the project about?",
"answer": "This project demonstrates a full-cycle ML pipeline and a simple LLM-based RAG system..."
}

Project Setup Instructions

Step-by-Step Setup:

 # Clone Repository
$ git clone <your-repo-url>
$ cd ml_llm_pipeline_template

 # Optional Cleanup
$ docker system prune -af --volumes

 # Build Images
$ docker-compose build --no-cache

 # Start All Services
$ docker-compose up -d

Docker Services and Ports:

Service	Purpose	Port
airflow	DAG execution	http://localhost:8080
mlflow	Model tracking	http://localhost:5000
model_service	ML API	http://localhost:8000/docs
rag_service	RAG API	http://localhost:8001/docs
spark	(Optional) Spark support	—
postgres	Airflow backend	—

Retrain & Register Model:

$ docker-compose exec airflow python /scripts/train_model.py
$ docker-compose restart model_service

Airflow DAG

- **DAG ID**: wine_quality_pipeline
- **Schedule**: Weekly (@weekly)
- Access via: http://localhost:8080

MLflow Tracking

- **Access URL**: http://localhost:5000
- Tracks: Metrics, Parameters, Registered Models, Artifacts

Challenges Faced

- Airflow import/path configurations
- MLflow logging context inside Airflow
- PySpark compatibility in Docker
- Chunking and FAISS index freshness
- Docker volume and port management

Improvements & Scope

- Add PySpark back for scalable preprocessing
- CI/CD with GitHub Actions
- Real-time alerts via email/Slack
- Use Mistral-7B or TinyLLaMA for better RAG response
- Interactive UI for LLM queries

Conclusion

This orchestration project successfully demonstrates how to:

- Automate an ML lifecycle
- Log and manage models with MLflow
- Deploy scalable APIs with FastAPI
- Implement a functional RAG pipeline

With modularity and Dockerization, the system is easily extensible to production-grade workflows.