Assignment 3: Agentic RAG System with Azure OpenAI & LangGraph

---

1. Background

An **Agentic Retrieval-Augmented Generation (RAG)** system combines:

- A **Knowledge Base (KB)** indexed in a vector database,
- An **LLM** (for generating answers),
- A **self-critique loop** (to check completeness),
- A **refinement step** (to fill missing gaps).

This setup ensures the system can answer **general questions** and also **leverage what's in the vector database** to provide citation-backed responses.

---

2. Problem Statement

> **"Build an Agentic RAG system on Azure using LangGraph that retrieves up to 5 KB snippets, generates an answer with Azure GPT-4 mini, critiques it, and when required, refines it with one additional snippet from Pinecone or Weaviate (trial cloud instance). The system should log outputs and support observability with MLflow."**

Dataset: self_critique_loop_dataset.json

Dataset and a pinecone starter notebook is also avaialbe on LMS in Assignment 3 files.

3. Detailed Tasks

1. **Preprocessing & Indexing**
    - Load KB JSON (~30 entries).
    - Use **Azure Embeddings**: `text-embedding-3-small`.
    - Store vectors in **Pinecone** or **Weaviate trial cloud instance** (no local hosting).
2. **LangGraph Workflow** Define 4 nodes:
    - **Retriever Node**: fetch top-5 snippets.
    - **LLM Answer Node**: use **Azure GPT-4 mini** (temperature=0) to generate initial answer with `[KBxxx]` citations.
    - **Self-Critique Node**: Gemini checks if answer is COMPLETE or needs REFINE.

   - **Refinement Node**: if REFINE, retrieve 1 more snippet and regenerate answer.

   Decision logic:

   - If COMPLETE → return initial answer.
   - If REFINE → return refined answer.

3. **Tracing & Observability**

   - Integrate **MLflow** to log runs, retrieved snippets, model outputs, critique results, and final answers.

---

4. Tools & Tech

- **Azure GPT-4 mini** (deployed in Azure OpenAI)
- **Azure Embeddings** (`text-embedding-3-small`)
- **Vector DB**: Pinecone (trial) or Weaviate (trial)
- **LangGraph** for pipeline wiring
- **MLflow** for observability
- **Python 3.10**
- Suggested packages:

```
langgraph
azure-ai-inference
pinecone-client   # or weaviate-client
mlflow
pydantic
```

---

5. Sample Queries

1. *"What are best practices for caching?"*
2. *"How should I set up CI/CD pipelines?"*
3. *"What are performance tuning tips?"*
4. *"How do I version my APIs?"*
5. *"What should I consider for error handling?"*

---

6. Deliverables

Submit either:

- **Jupyter Notebook** with all steps, OR
- **ZIP folder** with:
  - `index_kb.py` (embeddings + vector DB indexing)
  - `agentic_rag_azure.py` (LangGraph workflow + MLflow logging)
  - `requirements.txt`

---

7. Notes

- Always cite snippets `[KBxxx]`.
- Keep flow simple: 1 critique, max 1 refinement.
- Use trial vector DB instances (Pinecone/Weaviate portal).
- Temperature=0 for deterministic outputs.
