Assignment 3: Simplified Agentic RAG System with Gemini & LangGraph

---

1. Background

Retrieval-Augmented Generation (RAG) combines a knowledge base (KB) with a Large Language Model (LLM) so answers are grounded in factual content. An **agentic RAG** adds a self-review loop: the model critiques its own answer, checks against the KB, and refines if needed.

In this assignment, you will build a **lightweight agentic RAG pipeline** using:

- **Gemini on Google Cloud (Vertex AI)** as the LLM
- **LangGraph** to wire the workflow

---

2. Problem Statement

> **"Build a simplified Agentic RAG system with Gemini + LangGraph that retrieves up to 5 KB snippets, generates an initial answer, critiques it, and when necessary refines the answer by pulling one more snippet, returning a citation-backed response."**

Dataset: self_critique_loop_dataset.json

Note that the dataset and a pinecone starter file is available over LMS.

3. Detailed Tasks

3.1 Preprocessing & Indexing

1. Load the KB JSON (~30 entries).
2. Generate embeddings using **Vertex AI Embeddings (models/gemini-embedding-001)**.
3. Store vectors in a database (Anyone one of: **Pinecone**, **Weaviate**, or **Qdrant**). (recommended is Pinecone)

3.2 Define LangGraph Workflow

Your graph should have 4 main nodes:

1. **Retriever Node (`retrieve_kb`)**

- Input: user question
- Output: top 5 snippets from vector DB

2. **LLM Answer Node (`generate_answer`)**
   - Model: **Gemini 1.5 (Vertex AI)**
   - Input: question + snippets
   - Output: initial answer with citations `[KBxxx]`
3. **Self-Critique Node (`critique_answer`)**
   - Model: Gemini
   - Checks completeness of initial answer vs. snippets
   - Output: `"COMPLETE"` or `"REFINE: <missing keywords>"`
4. **Refinement Node (`refine_answer`)**
   - If refinement needed:
     - Retrieve **1 more snippet** with missing keywords
     - Regenerate answer including this snippet
   - Output: final refined answer

Decision logic:

- If critique = COMPLETE → return initial answer
- If critique = REFINE → return refined answer

---

4. Tools & Tech

- **LangGraph** (Graph API for agentic flows)
- **Gemini on GCP (Vertex AI)** for LLM & embeddings
- **Vector DB**: Pinecone / Weaviate / Qdrant
- **Python 3.10**
- Suggested packages:

```
langgraph
google-cloud-aiplatform
pinecone-client   # or weaviate-client / qdrant-client
pydantic
```

---

5. Testing Queries

Try the pipeline with:

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** showing all steps OR
- **ZIP folder** with:
  - `index_kb.py` (embeddings + vector DB)
  - `agentic_rag_simplified.py` (LangGraph workflow)
  - `requirements.txt`

---

7. Notes

- Keep flow simple (1 critique, 1 refinement at most)
- Always cite snippets `[KBxxx]`
- Use `temperature=0` for consistency
- Log each step: retrieved snippets, initial answer, critique, final answer using loger/mlflow

---