GenAI Research Assistant Hiring Assignment

NL2SQL E-Commerce Query Generator

Objective: Build an end-to-end Natural Language to Query (NL2Query) system that translates natural language questions into safe, executable SQL and Cypher queries over an ecommerce dataset using open-source LLMs, retrieval-augmented generation (RAG), and an **agentic orchestration** framework.

Note: Single purpose LLM will be considered as incomplete assignment

1. **Pick any Agentic framework**
    a. Phidata (preferred)
    b. Agno
    c. Langchain
2. **Database Setup**
    a. **Use Kaggle E-Commerce Dataset:** Ecommerce Dataset for Analysis
    b. **What to deliver**:
        i. Setup a knowledge base of 5-8 tables with Neo4j (graph database)
        ii. Relationship (can be hand-drawn/exported from neo4j)
3. **Vector Database Setup & Prompt Engineering**
    a. **Model Selection:** Choose any model which is freely available
    b. **Prompt Engineering (Required):**
        i. Dynamic schema injection (show relevant tables/columns)
        ii. 10 few-shot examples (NL question → SQL query) stored in vector database, choose any one → Weaviate, Pinecode, FAISS and retrieve top-5 from vector database and dynamically ingest in the prompt
        iii. Clear instructions for SQL generation
    c. **What to deliver:**
        i. Working model integration with code
        ii. Prompt template with schema injection

iii. Comparison of 2 different prompt strategies

d. **Questions to answer:**
   i. Why did you choose this model?
   ii. How does schema context size affect results?

4. Query Validation & Execution

a. Validation Pipeline:

i. Syntax validation
ii. Table/column existence - Verify against schema
iii. Safety checks - Block DROP, DELETE, UPDATE, TRUNCATE
iv. Query execution - Execute and return results
v. Self-correction: If query fails, send error back to LLM for retry

b. **What to deliver:**
   i. Validation module with error handling
   ii. Example of 3-5 validation catches (syntax error, wrong table, etc.)

c. **Questions to answer:**
   i. What percentage of queries are valid on the first attempt?
   ii. How do you prevent SQL injection?

5. Testing & Evaluation

a. **Test Suite:** Create **15-20 test queries** across difficulty levels easy, medium and difficult:

b. Evaluation Metrics
   i. **Execution success rate** (% queries that run without errors)
   ii. **Average generation time** (milliseconds)
   iii. **Average execution time** (milliseconds)

c. **What to deliver:**
   i. Test results table (query, success/fail, error type)
   ii. Performance metrics summary

6. API & Docker Deployment

a. Generate and execute SQL

```
POST /api/nl2sql
{
  "question": "Show total sales by category",
  "execute": true
}
```

b. Get schema info

```
GET /api/schema
```

c. Validate SQL

```
POST /api/validate
{
  "sql_query": "SELECT * FROM products"
}
```

Docker Setup (Optional)

docker-compose.yml with 3 services:

```
services:
  database:
    image: postgres:15
    environment:
      POSTGRES_DB: ecommerce
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - ./data/init.sql:/docker-entrypoint-initdb.d/init.sql

  ollama:
    image: ollama/ollama:latest
    volumes:
      - ollama-data:/root/.ollama

  api:
    build: ./api
    ports:
      - "8000:8000"
    environment:
      DATABASE_URL: postgresql://user:password@database:5432/ecommerce
      OLLAMA_URL: http://ollama:11434
```

```
      MODEL_NAME: codellama:7b
    depends_on:
      - database
      - ollama

volumes:
  ollama-data:
```

What to deliver:

1. Working FastAPI with 3 endpoints
2. Docker setup that runs with `docker-compose up`
3. API documentation (can use FastAPI auto-docs at /docs)

Deliverables Checklist

1. GitHub Repository

```
nl2sql-ecommerce/
├── data/
│   ├── schema.sql          # Database schema
│   ├── seed_data.sql       # Sample data
│   └── relationship_diagram.png       # relationship diagram
├── nl2sql/
│   ├── generator.py        # NL2SQL generation
│   ├── validator.py        # Query validation
│   └── executor.py         # Query execution
├── api/
│   ├── main.py             # FastAPI app
│   └── Dockerfile
├── tests/
│   ├── test_queries.json   # 15-20 test cases
│   └── test_results.md     # Evaluation results
├── notebooks/
│   └── evaluation.py    # Testing & analysis
├── docker-compose.yml
├── requirements.txt
├── .env.example
└── README.md
```

2. README.md

1. Project overview
2. Setup instructions: `docker-compose up`
3. API usage examples (curl commands)
4. Model used and why
5. Test results summary
6. Known limitations

3. Technical Report (1-2 pages)

File: `REPORT.md`

Sections:

1. **Database** - Dataset choice, schema description
2. **Model & Prompting** - Model selection, prompt strategy
3. **Validation** - Safety checks implemented
4. **Evaluation Results** - Success rate, examples, failures
5. **Challenges** - Top 3 issues and solutions

4. Demo (Optional but Recommended)

1. 3-5 minute video OR
2. 10 screenshots showing system in action

Submission Instructions

Email with subject: "GenAI RA - NL2SQL Assignment - [Your Name]"

Include:

1. GitHub repository URL (public)
2. Brief summary (3-4 sentences)
3. Setup verification: "Tested `docker-compose up` on clean machine"
4. Time spent on assignment

Key Questions to Answer

Answer these in your code comments or report:

1. Why is NL2SQL harder than general text generation?
2. Why is RAG-style prompting better than fine-tuning for this task?
3. What percentage of queries succeed on the first attempt?
4. What's the most common failure type?
5. How would you scale this to 50+ tables?

Good luck! Focus on getting a working end-to-end system over perfection.