Bicycle AI - Internship - 2024

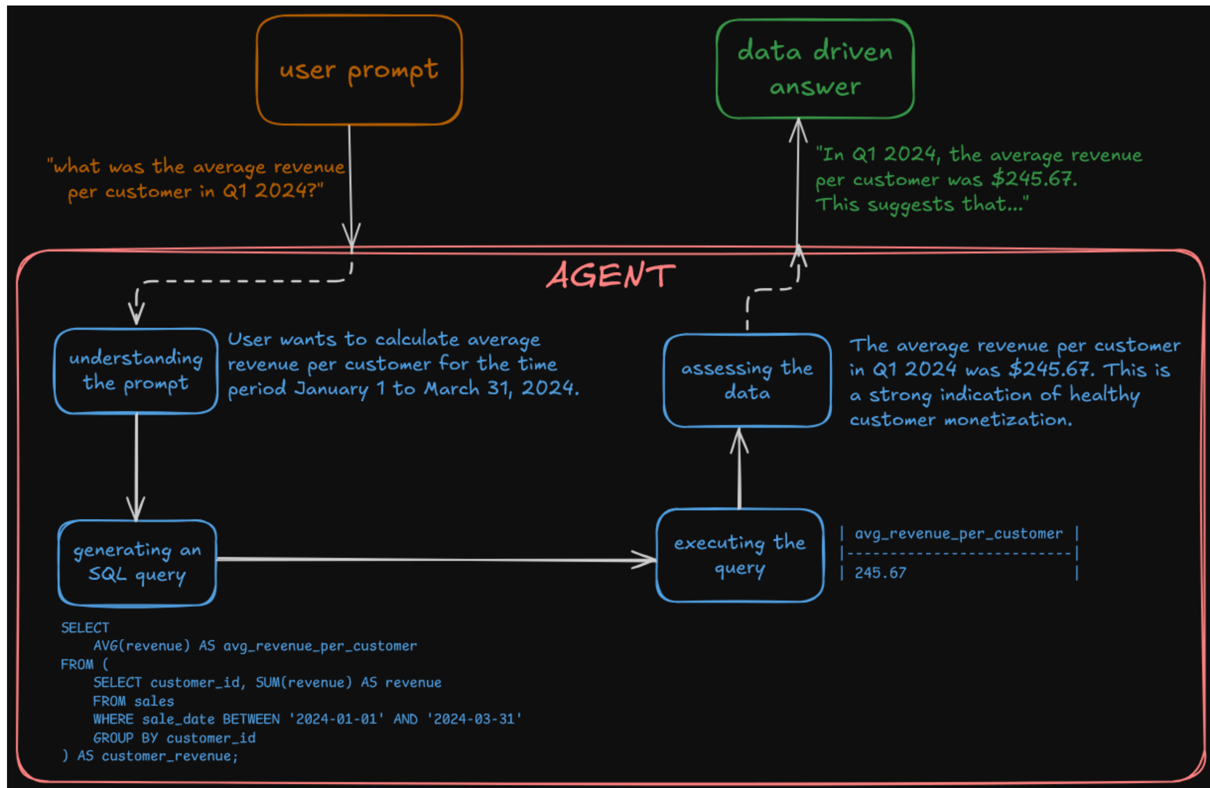
Detail	Information
Company Name	Bicycle AI
Role/Position	AI Engineering Intern
Duration	28th May 2024 - 15th July 2024.
Location	Hyderabad, On-Site
Mentor/Supervisor	Yashaswi Pathak, Rajiv Shivane
Certificate	https://drive.google.com/file/d/1ZMjD_A_s8_QPNIZo8DahAcaeTOS3-53o/view?usp=sharing
Tech Stack	LangChain, LangGraph, Python, LLMs(GPT, Gemini, Claude, Mistral)
Company Website	https://bicycle.ai/

Task:

1. To make an LLM Agent for basic data extraction and analysis(Learning LangChain and LangGraph - Agents)
2. To make a benchmarking system tailor made for the company's AI LLM calls

1. LLM Agent for Data Analytics

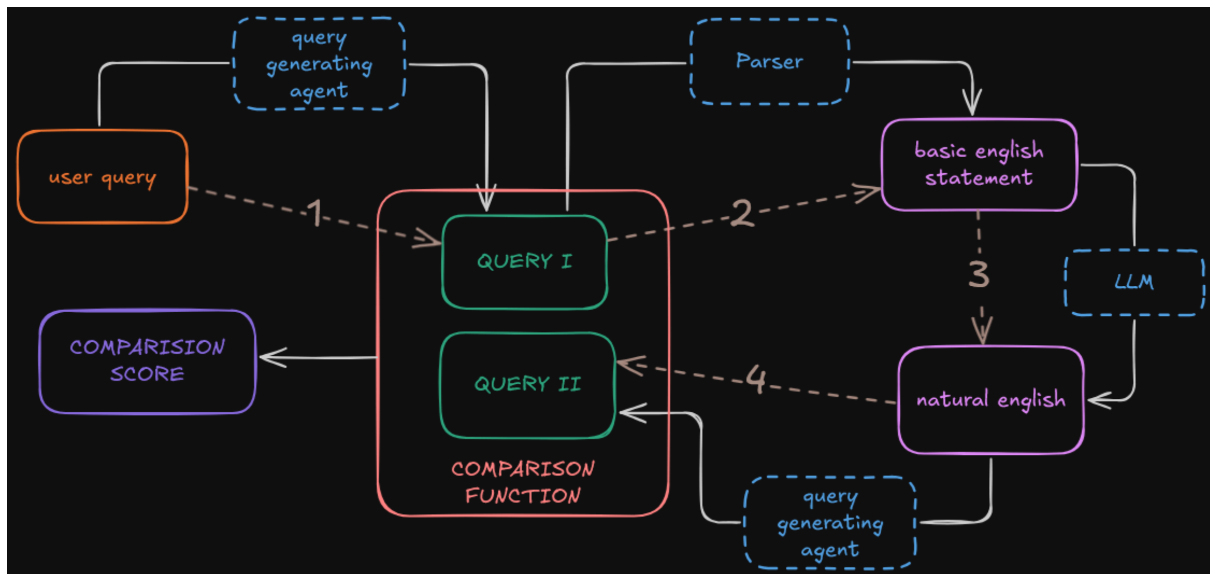
- Built a custom **LLM agent** for data analytics using LangChain, incorporating **ReAct**, **Reflection**, and **Reflexion** concepts. The agent improves the clarity and accuracy of data-driven insights



2. Framework for LLM Performance Benchmarking

- Developed a framework to compare leading LLM models (GPT-3.5, GPT-4, Gemini, and Claude), measuring response quality and speed. This enables straightforward performance evaluation as new LLM versions and vendors enter the market
- The LLM was being used for generating custom server queries to fetch data. I was tasked with evaluating the performance of the LLMs

Framework Ideated and Implemented:

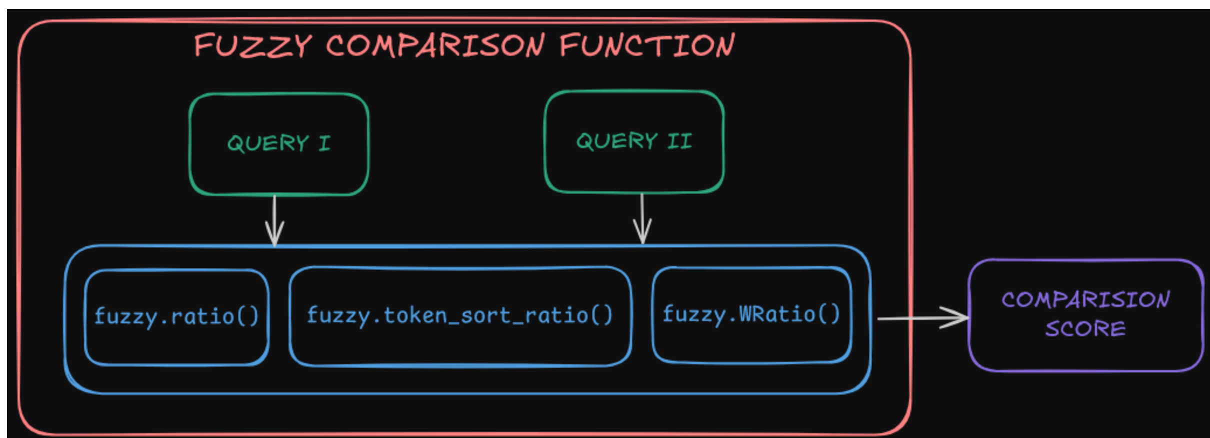


1. **User Query Input**
    - The process starts with a **user query** (natural language input from a user)
    - This query is passed to the **query generating agent**
2. **Generate Query I**
    - The **query generating agent** takes the user query and generates **Query I** (initial structured query)
    - **Query I** is sent into the **Comparison Function** block for benchmarking
3. **Create Basic English Statement**
    - Simultaneously, the user query is passed through a **Parser**, which converts it into a **basic English statement** (simplified representation of the query intent)
4. Generate Natural English with LLM
    - The **basic English statement** is then processed by a **Large Language Model (LLM)**
    - This generates a **natural English** version of the user query with enhanced clarity or reformulated wording
5. Generate Query II
    - This **natural English** statement is fed back into the **query generating agent** to generate **Query II** (alternative structured query)
    - **Query II** is also sent into the **Comparison Function** block

6. Comparison Function
    - The **Comparison Function** receives both **Query I** and **Query II**
    - It performs a comparison between the two queries to assess how close the initial structured query is to the enhanced interpretation
7. Output Score
    - A **comparison score** is generated based on the difference or similarity between **Query I** and **Query II**
    - This score serves as a benchmark to evaluate the accuracy or effectiveness of the original query generation

---

Fuzzy Comparison Function:



`fuzz.ratio(query1, query2)`

- Computes the Levenshtein distance between the two strings and returns a similarity ratio (0–100)

`fuzz.token_sort_ratio(query1, query2)`

- Tokenizes the strings, sorts them alphabetically, and then computes a Levenshtein-based ratio
- Better handles differences in word order

`fuzz.WRatio(query1, query2)`

- A weighted composite score that combines `ratio`, `partial_ratio`, `token_sort_ratio`, and `token_set_ratio`. It tries to choose the most appropriate comparison type based on input length and similarity

- Adjusts strategy based on query differences

Other Methods:

1. `fuzz.partial_ratio(query1, query2)`
   - Compares the shorter string to all substrings of the longer string.
2. `fuzz.token_set_ratio(query1, query2)`
   - Similar to `token_sort_ratio`, but better handles subsets. It compares the intersection, difference, and union of token set

---

Alternative Approaches for Comparison:

1. `sqlparse` or `sqlglot`
   - Parse both queries into abstract syntax trees (ASTs), then compare tree structures.
2. **Embedding-based Comparison (Semantic Similarity)**
   - converting SQL queries to embeddings and calculate cosine similarity

---

Future Steps that CAN be Taken:

1. Fine tuning an LLM/SLM to optimise the output of the code generation, use the same benchmarking system to decide on the correct option.
2. Along with the speed and accuracy include the cost as another factor for deciding the best model

---

Learnings:

1. Process Evaluation: I identified key areas in my system and framework that could enhance benchmarking effectiveness. Moving forward, I am committed to systematic examination of each process step and conducting rigorous assessments
2. Testing Strategy: I recognize the importance of implementing comprehensive testing at every development stage, particularly focusing on edge cases. Building a robust suite of test cases will strengthen the code's reliability
3. Quality Standards: While understanding that perfect query generation accuracy may not be achievable, I would try to maintain high standards

throughout development, which would drive me to create a thorough and detailed methodology