Project Overview: RAG for Student Assignment Analysis

Objective:

To develop a system that effectively retrieves student assignments, compares their responses to correct solutions, analyzes code quality, and dynamically grades the submissions using an advanced tech stack.

Components

1. Building a Retriever:

- Data Structuring:

- Structure student assignments in a standardized format to facilitate efficient retrieval. Each assignment should include:

- Metadata: Student ID, assignment ID, submission date, etc.

- Questions: Clearly delineated questions that correspond to the expected solutions.

- Answers: Student responses, including code snippets if applicable.

- Store this structured data in a Chroma DB to enable quick lookups.

- Retrieval Mechanism:

- Implement a retrieval system using LangChain that utilizes semantic search capabilities to match student responses with the correct questions and reference answers.

- Ensure the retriever can handle variations in phrasing and terminology to improve accuracy.

2. Prompt Template for Comparison and Analysis:

- Design a prompt template that will be used to compare student answers against the correct solutions. The prompt should:

- Clearly state the question being answered.

- Include the student's response.

- Provide the reference answer.

- Ask the model to analyze the differences, correctness, and quality of the response.

- Code Analysis:

- For assignments involving code, incorporate specific prompts to evaluate:

- Correctness of the code logic.

- Efficiency and complexity of the code.

- Adherence to best practices (e.g., readability, comments).

3. Dynamic Grading Metric:

- Develop a grading metric that adapts based on various factors:

- Type of Assignment: Different metrics for coding tasks vs. written responses.

- Complexity Level: Adjust grading criteria based on the difficulty of the questions.

- Rubric-Based Grading: Create a rubric that considers accuracy, completeness, clarity, and innovation.

- Use a feedback loop to adjust the grading criteria based on trends observed in student submissions and performance.

Tech Stack

- LangChain: For building the retrieval and generation components, allowing seamless integration of various APIs and models.

- Python: For scripting and implementing the logic behind data handling and grading.

- Grok: Utilize for querying and managing data in a structured manner.

- Colab Pro: For prototyping and testing models, given its powerful GPU capabilities.

- Chroma DB: For efficient storage and retrieval of structured assignment data.

- OpenAI API: For leveraging advanced language models to perform analysis, generate responses, and grade assignments.

Implementation Steps

1. Data Collection:

  - Gather a diverse set of student assignments, ensuring they cover a range of topics and complexity.

2. Data Structuring:

  - Develop a schema for the assignment data and implement a storage solution in Chroma DB.

3. Retriever Development:

  - Build and test the retrieval system using LangChain, ensuring it accurately matches student responses with the correct questions.

4. Prompt Engineering:

  - Create and refine the prompt templates for both answer comparison and code analysis.

5. Grading Logic:

  - Design the dynamic grading system based on the established metrics and rubrics.

6. Testing and Iteration:

  - Conduct extensive testing with sample assignments, iterating on the retriever and grading logic to improve accuracy and reliability.

7. Deployment:

- Deploy the system for use, monitoring its performance and making adjustments as needed based on real-world usage.

Conclusion

This RAG system aims to streamline the grading process, improve the accuracy of evaluations, and provide constructive feedback to students. By leveraging the latest technologies and methodologies, it can adapt to varying assignment types and complexities, ensuring a fair and comprehensive assessment.