MID-SEM TEST, OPEN BOOK, FIRST SEMESTER, 2024-2025, BITS Pilani
CS F441 - SELECTED TOPICS IN COMPUTER SCIENCE

DATE: 09th Oct 2024 MAX MARKS: 50 WEIGHTAGE: 25% TIME: 90 mins

---

Implement a LangGraph-based application to evaluate Java code submissions provided by students. This application will be used by teaching assistants (TAs) to assess coding assignments and provide detailed, automated feedback to all the students. This application will use Large Language Model (LLM) APIs to do the bulk of evaluation and grading. The application will consist of several modules, each focusing on a different part of the evaluation process. Your task is to implement these modules into a cohesive workflow using the LangGraph framework, ensuring that the evaluation process is modular, repeatable, and completely automated.

Input files provided to the application:

1. Problem description of the Java assignment/exam (prepared by the Java instructor).
2. Model solution of the Java assignment/exam (prepared by the Java instructor).
3. The rubric of the Java assignment/exam (prepared by the Java instructor). This will contain a step-by-step marking scheme to evaluate the student code.
4. Student code submission for the Java assignment/exam (submitted by the student).
5. For better understanding, look at the sample input files (problem description, model solution, rubric, student code submission).

Application Modules:

1. **Class Extraction Module**:
    - Each student code submission may contain multiple Java classes. Therefore, your application must first parse the student's code submission and extract individual classes.
    - You should also parse the instructor-provided model solution to extract individual classes from the model solution.
    - **You must use an LLM to perform the class extraction.**
    - For simplicity, you can assume that the student submission has the same class names as mentioned in the problem description, model solution and rubric.
2. **Rubric Extraction Module**:
    - Use an LLM to extract the relevant rubric details for each individual Java class so that each class is evaluated only against the appropriate part of the rubric.
3. **Initial Evaluation Module**:
    - Each class extracted from the student submission must be evaluated using an LLM, based on the rubric details and model solution extracted for that class.
    - The LLM will use a prompt that includes the class code, the relevant rubric details and the relevant portion of the model solution to generate an initial evaluation and assign a grade (numeric score) for each criterion.
        i. The evaluation should not only include the numeric score for each criterion but also include detailed comments about the correctness, errors and suggestions for improvement in the student code.
    - Note that in general, the rubric will have multiple scoring or grading criteria for every class. Hence, there will be a sequence of scores awarded for each class depending on the number of grading criteria for that class. Think of this as step-by-step marking within each

class. The evaluation generated by the LLM for every class will have details about each step and how much the student has scored for each step.

4. **Review Evaluation Module**:
    - The initial evaluation performed by the LLM may not be entirely accurate. Hence, the initial evaluation must be reviewed to ensure correctness. This review should be carried out using an LLM.
    - The LLM should review the initial evaluation, make necessary corrections, and provide a final assessment for each Java class in the student code.
5. **Marks Extraction Module**:
    - The final evaluation generated by the LLM must be further processed to compute the total marks. In this regard, you first need to extract the step-by-step marks from the final evaluation for each class.
    - For each class, the LLM will generate a comma-separated list of the marks awarded for that class (based on the number of criteria mentioned in the rubric for that class).
    - You should extract the comma-separated list of marks for all the classes.
6. **Total Marks Calculation Module**:
    - You must implement a tool called **sum_marks** that takes a comma-separated list of marks and returns their sum.
    - Bind this tool to the LLM so that it can use the tool to calculate the total marks for the entire submission.
    - The LLM will be prompted to calculate the total marks using the sum_marks tool.
    - The final output must be saved in a single file named **final_evaluations.txt**. The output will contain the detailed LLM evaluation (only the final evaluation for each class) along with the total marks obtained by the student.

Application Workflow:

1. The entire process must be implemented as a LangGraph-based state graph workflow, where each module is represented as a separate node.
2. Each module (or sub-step of a module) must translate to a node in the workflow. You may use multiple nodes for any module, but at a minimum, each module should have at least one corresponding node in the graph. You can process all the Java classes in a single student code submission within a single LangGraph Node. A separate node for each Java class is not required.
3. You should use the state variables to transfer information from one node to another. Reading and writing into files will not be acceptable for information flow from one node to another.
4. Look at the sample output file (final_evaluations.txt) for better understanding.

You are free to add/modify/remove any nodes/modules as long as you do not contradict any of the information mentioned above.

Deliverables:

1. Write all the code in one file only unless it is necessary to split the code in multiple files.
2. Feel free to use either Jupyter Notebooks (.ipynb) or simple Python files (.py).
1. Ensure that you are asking the user for the LLM API KEY. It should not be hardcoded in the code.
2. You are free to use LLM tools such as ChatGPT for assistance.
3. **Please upload all your conversations with LLM-based tools to Nalanda at the end of exam.**
4. **Please also upload your code on Nalanda once you are finished.**