CS 3341 FOUNDATIONS OF COMPUTING: LAB 4 LLM CODE ANALYZER

Overview: Design and build a job for the JobSystem that will utilize an LLM to analyze and provide feedback on the error json output produced from c++ code in previous labs via a REST interface.

Repo Link: https://classroom.github.com/a/aXQJLjm4

Due: November 29, 2023 @ 11:59pm.

Code Requirements (50%):

- Build upon the JobSystem, FlowScript language developed in Lab 1, 2 and 3. You may modify, change, update any portion of previous labs to meet the objectives defined in Lab 4. Job System must be loaded as a library as in previous labs. The job must maintain the json/string input/output format set in previous labs.
- Create a job for the JobSystem that will enable the ability to send/receive data to an LLM via a REST interface
    - At a minimum, the job must take IP and prompt for LLM as inputs
    - The job must be configurable to connect to different IP addresses
    - The job must return LLM response and/or any errors that occur from the request back to user
    - The job must be launched from the JobSystem, but it can be written in any language
        - The REST calls can be made from python, JavaScript, etc. But the script and its inputs must be initiated/executed from the JobSystem itself
- A minimum of 2 LLM models must be tested
    - Both models can be located within the GPT4All package and use the same IP address
    - OpenAI models are not a requirement for the lab, but it is possible they could provide better results than the opensource models accessed via GPT4All
    - If an LLM outside of GPT4All or OpenAI is used, you must provide the code (server, etc) required to host/connect to the LLM
- At a minimum the contents of the error json produced from Lab 1 must be passed to the job, along with a prompt to obtain feedback on the code and errors within it.
    - You must create prompts to try and obtain the most useful feedback from the LLM to help resolve the error.
        - Ideally the prompt/LLM produces provides code that resolves the error, as well as description of the resolution.

        - While the above is ideally the goal, depending on the model it may not be possible. Your goal is to get as close to this as possible with at least 2 LLMs
        - You will most likely need to update your prompts based upon the model being used.
    - You may reformat the error json object within the job in any fashion to allow for it to be incorporated within the LLM context and prompt for processing.
    - You may pass more information than the error json to LLM if desired (i.e. source files, etc).
    - You must test at least 3 different error types and json objects
    - Remember you can request the LLM to return its output in any format you want using zero/few shot training within the prompt
- FlowScript is not required to be used within this lab

<u>Report Requirements (50%):</u>

- Document and describe all prompts utilized for each LLM
    - The writeup should be written more as development journal
    - You should show the process utilized in developing prompts
        - Document intermediate results, showing what worked and what didn't
        - As you improve prompts, and information passed to LLM you need to provide reasoning for the changes and show how the change increased/decreased performance
    - The writeup should end with full description of the final prompts and reasoning for the specific wording and methodology of how the prompt works
        - Outputs and write up for each of the error json should be shown and full documented
            - Showing inputs passed
            - formatted prompt and context created for inputs
            - Result from LLM query
            - Final parsed output sent back to user
    - Show the above process for each LLM tested
- Document and discuss your custom job
    - Provide details on how the job is created and executed
    - Provide details on any parsing and setup for LLM context and prompt creation
    - Provide details on any zero/few shot training provided within the prompt/context
    - Provide details on any parsing and processing of LLM results sent back to user

All code in the "Code" folder, raw data files generated in "Data" and your final report in "Report" folder in the git repo. If your project fails the compile action upon final push to git repo, it will receive a 0. Be sure to resolve any issues with the compile action prior to deadline.