Assessment 2 - LangGraph and Evaluations in LangSmith

Objective

The objective of this assignment is to become familiar with using LangGraph to create multi-agent solutions and use LangSmith to test the correctness of the output. The assignment will make use of workflow, ReAct and coding patterns.

The task is to take a standard coding benchmark (MBPP) and create an application that writes and tests code. The benchmark defines specification for functions (in plain English).

The application will perform the following steps (not necessarily in this order):

- Take a coding task and generate code
- Take the coding task and generate test cases
- Test the code
- Evaluate test results and fix any failures and retest.

The coding benchmark that will be used also has reference output and tests. So once you have completed the coding task, you will check the results in LangSmith to ensure you're results are correct against the reference test cases and code.

The Coding Bencmark - Mostly Basic Python Problems

The benchmark consists of around 1,000 crowd-sourced Python programming problems, designed to be solvable by entry level programmers, covering programming fundamentals, standard library functionality, and so on. Each problem consists of a task description, code solution and 3 automated test cases.

The benchmark can be downloaded from Huggingface at: nlile/mbpp · Datasets at Hugging Face . The related research paper is here: [2108.07732] Program Synthesis with Large Language Models

It was developed in 2021 but should provide an example of a coding pipeline with evaluation.

There are a few variations of the benchmark. One that is worth considering is the 'sanitized' version, where a subset of the tasks have been improved to be less ambiguous.

Requirements

The assignment can be done in two parts. First develop an application and then setup the test and evaluation environment.

Application Requirements

Use LangGraph to develop a multi agent solution. Coding and testing should be performed by different agents. The graph should recurse between code and test until the tests are completed successfully, fixes defects in the code before retesting.

During this phase you will need a small subset of the benchmark tasks for development (start with one or two). The benchmark is downloaded as a CSV file, so Pandas can be very helpful with manipulating the data, although other approaches are also fine.

It can be very beneficial to use LangSmith at this stage for Observation, looking at the traces to ensure the graph trajectory is as expected and observe the details of the LLM calls and responses. If a particular LLM call is not performing as expected, you can use the Playground to modify the context until it produces the desired result.

Consider creating nodes in your graph to perform some non LLM calls, e.g. for loading/saving data or manipulating the state. This can be a useful way to decompose the solution with one or two extra nodes (this is more by way of suggestion than a requirement).

You should design your graph to perform just one coding task from the benchmark at a time. The graph will be called multiple times for in the evaluation stage below.

During your design and development stage pay attention to the context that you pass to each LLM call. Ensure that only the necessary information is going into the context window for the required task you want the LLM to perform. This will improve reliability and performance. Remember you can use the state to hold any information you want to pass between nodes, you do not need to rely solely on messages.

The tutorial in week 4 and examples in week 5 should provide you all the techniques you require to complete this part of the assignment.

Testing and Evaluation

Once you have a working graph, you need to prepare the dataset and evaluators.

The process of preparing eval datasets can be time consuming. The steps to be performed for this stage are:

- Prepare and create the dataset in LangSmith
- Create the evaluators

- Execute graph and evaluators for each element of the dataset.

The MBPP has the following fields:

- Text (or prompt in sanitized version): contains the coding task
- Code: Reference code
- test_list: List of reference tests

While your application will perform its own coding and testing, you will use the reference code and tests to evaluate the code your graph has produced. You will execute the reference tests to check that your code has the expected behaviour, and you will use the reference code to assess the quality and efficiency of the code your application has produced.

By having a reference set we can enhance our application's process and prompts to meet the standards of the benchmark. The principle is that improving our application with a **training set** will produce improved performance once the application is deployed.

Preparing Dataset

You do not need to load all 1000 tasks into LangSmith for this assessment. 10 to 20 tasks should be plenty to demonstrate the process is working correctly.

When preparing datasets, you need to create inputs and outputs. These will be dictionaries, with each containing the information you need when executing and evaluating the application.

For the input dictionary, you will need the task, but it is also useful to track the task_id. Additional consideration should be given to the evaluation process when preparing the input. Since you are using the provided tests to check the functionality, you really need the function name and parameters to match the test cases. I recommend you take the function 'header' from the reference code and pass it as an input to your application. You need to take this into consideration when creating your application (hopefully you have read this before beginning your development in part 1). This means the input to your application will be the task and the function header as the requirement.

In preparing the outputs dataset, you will also need to get the test_list into a usable form in Python. These almost look like lists, but the separators between elements are line breaks and spaces, rather than commas (a good recommendation I got from copilot was to use the 'ast' library for this).

So in summary, for each task you should have an input dictionary with task, task_id and function header. For the output dictionary you should have code and test_list.

Once you have the data prepared in code use the LangSmith SDK to push the data into LangSmith.

Creating Evaluators

You will create two evaluators. They will perform the following:

- execute the reference tests and provide percentage passed score
- Use LLM as Judge to assess code quality and efficiency. This will be scored out of 10.

The first is an example of using a heuristic evaluator to perform tests where the application is required to pass. The second is an example of an Evaluation, where the application can be incrementally enhanced to improve performance over time. This could continue to be assessed after the application has been deployed (online evaluator).

Each evaluator can receive the inputs, outputs (of the application test) and the reference_outputs.

The first evaluator should take the code output and execute the reference tests to get a percentage pass score (round this to an integer).

The second evaluator should take the code output and reference code and ask an LLM to make an assessment of the code quality, providing the reference code as a standard that it should meet or improve (e.g. it should be at least as efficient as the reference code).

There are different options for the output type of an evaluator, from boolean or integer to a dictionary. In this case use a dictionary for both evaluators. Your dictionary have specific keys that LangSmith recognises. The numerical result should have a key of ‘score’. Also include feedback against the key of ‘comment’. For the test evaluator this should be any exception messages. For the code quality evaluator, it should be the justification of its mark. This is particularly useful for getting suggestions for improvement.

Execute Evaluation

Once the dataset and evaluators are prepared, you can create a simple function that wraps the call to your graph. Ensure the response is a dictionary, which will be stored as your output. The key you use in your outputs will be available to your evaluators.

Finally, call the langsmith client ‘evaluate’ function, passing application function, dataset and list of evaluators.

Slices

It can speed up your development a bit if you create some slices in your dataset. Initially, create a slice with just one or two examples, so you can debug. Once this is working create more slices to test subsets of your dataset, until you are ready to run all the examples. As mentioned before, there is no need to have more than 10 to 20 examples to show everything is working.

Output

Firstly, the objective of the assignment is to build a simple graph and execute an evaluation dataset. You do NOT need to have passed all the examples. You are benchmarking to see how your coder application performs. Many of the tasks are quite ambiguous, so examine failures to see the cause, but it is likely some will fail due to the interpretation of the tasks. If you were using this for real testing, you would likely enhance the task definitions, but no need for this assignment (but feel free to investigate this if you have time).

You should submit working code in the form of Python files or Jupyter work book.

Also submit a document which describes the following:

- Design
- Implementation of graph
- Evidence of successful execution of graph
- Code preparation of dataset
- Code for execution of evaluation (including evalutors)
- Evidence (screenshots) and description of test results in LangSmith

If submitting in Jupyter, markdown cells describing the code will also count towards the documentation.