Code Performance Analyzer - Test Plan and Report

User Story 1.1: As a user, I want to run a simple model that can analyze Python code through a local script so that I can get an initial Big-O complexity estimate.

Testing:

- Ensure safetensors are in the model directory by cloning the model repo
- Run complexity.py with "<code>" as a command-line argument

Expected Results:

- Complexity.py does not return an error
- The result is just an O() notation with no prompt remnants
- The resulting complexity is not an un-standard O() notation or different syntax

User Story 1.2: As a user, I want to see a mocked complexity annotation inside the editor UI so I can preview how the extension will eventually present feedback.

Testing:

- Launch the VSCode Extension Development Host (F5 in VSCode).
- Open a file in the Development Host window.
- Highlight a block of code containing a nested loop.
- Trigger the analysis command (either through the command palette or the right-click context menu).

Expected Results:

- The editor should render a decoration or annotation at the end of the selected line/block of code (generated by the fallback).
- The text should be styled in gray and read "// Complexity: O(n^2)".

User Story 2.1: As a user, I want the extension to return a relatively accurate time-complexity estimate so that the extension is useful.

Testing:

- Run the Python backend with python serve.py
- Start the teacher model using ollama serve
- Insert an appropriate list of python functions into accuracy_checker.py
- Run accuracy_checker.py

Expected Result:

- Our_model_output.txt is present and contains time complexities
- Deepseek_output.txt is present and contains time complexities
- The resulting accuracy score is over 90%

User Story 2.2: As a user, I want the extension to clearly display complexity in an editor annotation, terminal, and sidebar.

Testing:

- Launch the VSCode Extension Development Host (F5 in VSCode).
- Open a file in the Development Host window.
- Highlight a block of code containing a nested loop.
- Trigger the analysis command (either through the command palette or the right-click context menu).

Expected Results:

- The editor should render an annotation at the end of the selected line/block of code (generated by the fallback).
- The text should be styled in gray and read “// Complexity: O(n^2)”.
- The sidebar should also display the complexity analysis in the respective panel.
- The VSCode terminal output should display the raw analysis complexity result.

User Story 3.1: As a user, I want to be able to generate a test script to profile for real performance metrics like execution time and memory usage.

Testing:

- Enter the development container
- Start the local server by running python serve.py
- Compile the extension with npm run compile
- F5 into extension.ts to open the extension environment
- Paste a piece of python code into the generate test input field in the sidebar
- Save the generated test and run it

Expected Result:

- The generated test appears in the sidebar, with or without a complexity hint
- Exporting the test brings up a OS ui to save a file
- Running the test does not crash
- The test provides time and space usage
- The test brings up time and space usage on a graph
- The time complexity matches that provided by Analysis

User Story 3.2: As a user, I want to submit code through the extension and have the locally hosted model instantly return real complexity results, so the extension becomes functional.

Testing:

- Enter the development container
- Start the local server by running python serve.py
- Compile the extension with npm run compile
- F5 into extension.ts to open the extension environment
- Highlight a piece of python code > right click > Analyze Code Complexity

Expected Result:

- A time complexity is returned as an annotation, in the sidebar, and in stdout
- The result does not come from our local fallback
- An analyze request is present in the server log

User Story 4.1: As a user, I want to be able to use the model without having to download it and use my computer's resources.

Testing:

- Launch the local cluster with start_deploy.bat
- Update to the latest image with update_cluster.bat
- F5 extension.ts to open the extension environment
- Highlight a piece of python code > right click > Analyze Code Complexity
- Stop with stop_deploy.bat

Expected Result:

- kubectl get pods shows a pod is running
- Kubectl logs <pod> shows the server has successfully started
- Analyzing complexity returns a result from the model
- Analyzing complexity does not time out
- Stopping deployment does not leave the cluster containers running

User Story 4.2: As a User, I want the interface to highlight inefficient patterns and show a visual performance chart so I can understand and improve my code's behavior over time.

Testing:

- Ensure backend performance API is running with serve.py.
- Launch the VSCode Extension Development Host (F5 in VSCode).
- Open a file in the Development Host window.
- Highlight a block of code containing a nested loop.
- Trigger the analysis command (either through the command palette or the right-click context menu).

Expected Results:

- The sidebar should generate a performance chart displaying: Time complexity trend and runtime metrics relevant to the given complexity
- Chart should update based on the current code.
- Chart should render without requiring a page refresh.