.Instructions & Guidelines - Evaluation

Task Introduction

The primary goal of this task is to test and evaluate an AI assistant's capabilities in the area of backend development for the following several use cases:

Code Generation, Test Case Generation, Code Explanation, Debugging.

In evaluation, you will be presented with an input prompt and two model responses with the model - mirroring what real users may seek assistance for in relation to backend development.

You are expected to look over the prompt and understand what is being asked, and then review the two responses provided, and evaluate them against 4 criteria that have been specifically tailored to assess the quality of the code generated:

- **Correctness:** The code should execute fully with no errors, meet all requirements specified in the prompt, and must perform its intended functionality
- **Readability:** The code should be easy to read and understand, use meaningful and logical variable and function naming convention, and include relevant and clean comments where necessary to explain the logic flow - note you can also make some allowance in here for easy to follow commentary provided by the LLM to accompany the code
- **Scalability:** The code should be modularised appropriately, use consistent coding standards and style, and have appropriate error handling where relevant
- **Efficiency:** The code should be well optimized for time and space constraints where appropriate.

Step-by-Step Instructions

You will be performing the following sequence of actions in that order.

1. **Response Rating:**
   You will first need to review and evaluate the two AI-generated responses based on the following criteria.

   - **Correctness**
   - **Readability**
   - **Scalability**
   - **Efficiency**

   As **there is code contained in the responses which seeks to address the original prompt, you will be expected to try to execute the code to inform your ratings** (more details of this are provided below)**.** A lot of insight around the ratings will be informed by whether the code works as expected, how robust and efficient it is, etc.

   Having reviewed the output (and validated the output code) you will be expected to then rank the model response against the relevant criteria, in line with the guidance below.

   Note: there is no expectation to fully refactor the code yourself, although very minor edits, formatting changes (e.g. linting), replacing of placeholders, or piecing together of the code response might be necessary to get the code to a stage where you can fairly execute the logic returned by the model

2. **Preference Ranking:**
   After you have provided ratings against the criteria for both responses you will then be asked to pick a preferred response. This will be based on the evaluation of the two responses in the above step

3. **Preference Ranking Justification**
   You will then be provided with a free text field where you can provide in a few sentences justification that supports the criteria rankings you have provided to the two responses and why you picked the preferred response of the two

4. **Prompt Classification (Topic & Type):**

Lastly you will be asked to review the prompt again and classifying, firstly, which of the following topics it is most relevant to:

- **Code Generation**
- **Test Case Generation**
- **Refactoring**
- **Debugging**

And then into which prompt type it fits:

- **Structured**: Prompts that are well-defined with precise instructions and typically include specific requirements or constraints.
- **Messy**: Prompts that are vague or incomplete. They might be ambiguous, have unclear goals, or contain errors. This reflects how real-world users sometimes interact with AI - especially if they are unsure about the technical details or are not entirely certain of the task requirements.

WALKTHROUGH VIDEO:

https://www.loom.com/share/e5d229dd313041e2ad8132ee8d0bd622?sid=01a0037c-d435-4b26-a237-3af9728eb4ee

Rating Guide

You will be expected to classify the responses against the following criteria.

(1) Correctness:

Verify that the code executes fully without errors and meets all the requirements specified in the prompt. Ensure it performs its intended functionality correctly.

Example: If the prompt is to generate a function that calculates the factorial of a number, the code should correctly return the factorial for any given input without throwing errors.

- **3 (Correct)**: The code executes fully, meets all requirements, and performs the intended functionality correctly.
- **2 (Partially Correct)**: The code executes but contains minor errors or does not fully meet all requirements.
- **1 (Incorrect)**: The code does not execute or contains significant errors that prevent it from meeting the prompt requirements.

---

(2) Readability:

Assess whether the response is easy to read and understand. Ensure the code is well-organized, uses clear and descriptive variable names, and includes comments or docstrings where necessary to explain the logic or purpose. Avoid over-complication or redundant details that reduce readability. You can also give some weighting here to how useful any instruction or commentary the model provides surrounding the code, but do be sure to mainly focus on the code itself and the comments within the code.

Example: For a response generating a REST API endpoint, check that the code structure is logical in the order it is delivered, variable names like “user_id” are meaningful, and any complex sections are accompanied by concise comments explaining their purpose.

- **3 (Readable)**: The code is easy to read and understand, uses meaningful names, and includes appropriate comments.
- **2 (Moderately Readable)**: The code is readable but could benefit from better naming conventions or additional comments.
- **1 (Unreadable)**: The code is difficult to read or understand, uses poor naming conventions, and lacks necessary comments.

(3) Scalability:

Assess whether the code is modularized appropriately, uses consistent coding standards and style, and has appropriate error handling. This should ensure that the code can be easily extended or modified for future expansion, and to handle increasingly high-load scenarios.

Example: The code should be divided into functions or classes where appropriate, follow consistent style guidelines, and handle potential errors gracefully. Ensure the response is designed to scale effectively for larger inputs or increased demands

- **3 (Scalable)**: The code is well modularized, consistent in style, and includes robust error handling.
- **2 (Moderately Scalable)**: The code is somewhat modular but could be improved in consistency and error handling.
- **1 (Not Scalable)**: The code is not modular, lacks consistency in style, and has inadequate error handling.

---

(4) Efficiency:

Evaluate the optimization of the code in terms of time and space complexity where appropriate. Check if the code performs efficiently under expected workloads.

Example: If the prompt asks for a function to sort a list of integers, ensure that the provided solution uses an efficient algorithm like Timsort rather than a suboptimal one like Bubble Sort.

- **3 (Efficient)**: The code is well optimized and performs efficiently under expected workloads.
- **2 (Moderately Efficient)**: The code is somewhat optimized but has room for improvement.
- **1 (Inefficient)**: The code is not optimized and performs poorly in terms of time and space complexity.

Provision DataBases

The focus of this project is around backend engineering, therefore we have provisioned a MySQL and Postgres database with adequate read only permissions.

In order to enable members of this evaluation team to test outputted code that looks to read or interact with a relevant database, we have provisioned a MySQL and Postgres database with some dummy data within.

In many cases the database details have been provided in the prompt and therefore are already accounted for in the response, but you should always double check this. The relevant details are as follows:

MySQL

- Username: readonly_user (SELECT only permissions)
- Password: labeling_now
- Host: admin.c58u848ggx5b.us-west-1.rds.amazonaws.com
- Database: pagila
- Schema: https://public-bucket-ramy.s3.us-west-1.amazonaws.com/mysql-pagila.png

Postgres

- Username: readonly_user (SELECT only permissions)
- Password: labeling_now
- Host: postgres.c58u848ggx5b.us-west-1.rds.amazonaws.com
- Database: pagila
- Schema: https://public-bucket-ramy.s3.us-west-1.amazonaws.com/postgres-pagila.png

Observing the Data
While the schemas have been provided above, you may find it beneficial to connect with the above credentials in order to fully explore the database and see the content types of data within specific tables.

In the event of this,, it may be helpful to use the likes of DataGrip or a VSCode plugin to view the database(s) in more detail.

Code Execution

When it comes to code execution to validate responses you will be expected to do so in your own IDE

In all cases where code is returned in the response, you will need to **execute the code returned to assess:**
- That it runs as intended and delivers the expected output.
- That there are no errors contained and the code is robust in how it handles error edge-cases.

- That the format is appropriate (i.e. adheres to specific linting guidelines if relevant).

You will leverage the code execution to determine where there are issues or faults with the code, or where optimisation and improvement can be found. **These findings will inform your evaluation of the model response, as per the Ranking Guide** presented below.

In such cases, **you will execute the code as originally returned (note some LLM responses might need reformatting/indenting to work properly).**

Follow these guidelines for the execution of code:

- Software and Environment Installation
  - IDE Recommendation: Use Visual Studio Code (VS Code)
  - Download and Install VS Code if you do not already have it (or other appropriate IDE)
  - Install Python Extension (e.g. in VS Code)
- Python Installation
  - Check if python is installed by running python - - version in the terminal.
  - If you do not have python downloaded, you can download it here
- Create and Build Project Folder
  - Create a new folder on your Desktop and open it in VS Code: File > Open Folder …
  - Recommended: create and activate a virtual environment to avoid installing packages onto your base machine
    - $ Python3 -m venv venv
    - $ source ./venv/bin/activate
- A note on dependencies
  - For the most part in evaluation all the dependencies will be clear from the import statements in the code base returned by the models
  - Be sure to install these using whichever preferred package repository you use (e.g. PyPI, etc.)
  - The one major dependency you might need to install for evaluation is “postgresql” (e.g. “$ brew instal postgresql” or alternative package installer than brew)l
- Create your programs
  - Create appropriately named .py files and paste your code
  - Make minor adjustments to the code base required.
    Note, slightly indentation issues are somewhat expected in the models responses in evaluation (and can be easily fixed with linting)**,** these are less severe than actual errors in the code itself