Perfect Assertions

Customer Goals and Objectives:

The goal of the project is to create prompts, associated unit tests that test the functionality requested in those prompts, and a golden response that implements the functionality requested in the prompt while passing all associated unit tests. The prompts should be difficult - models being tested should fail twice for the prompt to be considered valid.

The customer asked for ambiguous prompts. Thus, problems may require the model to reason alongside the problem statement to identify the most likely interpretation but the request must still be feasible

Important Documents

- Instructions Document

Audit Workflow

1. Briefly review the attempter's instructions
2. Review the seeded prompt. This is synthetically generated along with the test cases.
3. Evaluate the paraphrased prompt (simply referred to as "prompt" sometimes)
    a. The paraphrased prompt should reword the seed to sound more natural and remove any "story" that is not relevant to solving the problem

   b. It is crucial that the essence of the coding problem from the seed is maintained. The request should not change in any way. If you independently write test cases for both the seed and the paraphrased prompt, you should have the same set of test cases.
   c. The prompt should be difficult enough to make the model fail.
4. Evaluate the model failure
   a. The failure needs to be objective and the cause of a faulty implementation of the most likely interpretation of the problem
   b. The initially provided unit tests may not be comprehensive yet but should still result in a model failure
5. Evaluate the model guidance i.e. CB instructions that explain to the model what it got wrong and how to fix it
6. Evaluate the golden unit tests (not the pre-provided ones) for overall quality and alignment with the prompt
   a. You can ignore the pre-provided synthetically generated unit tests
   b. Evaluate this using the test.py file that you see in the "Sphere Engine" section and not the text field below that.
7. Evaluate that the reference solution (golden rewrite) follows the skeleton format, answers the prompt, and solves all unit tests
   a. Evaluate this using the solution_name.py file that you see in the "Sphere Engine" section and not the text field below that

General Grading Instructions (How the 1-5 scale is used)

1. **General Grading**
   - Grade to the lowest dimension across all rubrics (e.g. if instruction following is a 2, the task should be rated a 2)
   -
   - If the task meets any criteria under 1-2 Fail, the task is a fail.
   - If the task does not fail and it meets criteria for a 3 on any dimension, then the entire task is a three.
   - All dimensions must be a 4-5 for the task to receive a 4-5.
2. **Choosing 1 vs 2 or 4 vs 5**

   - When deciding between a 1 or 2, select a 1 if the attempter put little to no effort
   - When deciding between a 4 or 5, select a 5 if the task is perfect or impressive
3. **Prompt instructions or task instructions should always take precedence over other dimensions**
   - For example: if the task instructions asks the user to intentionally make spelling mistakes in the prompt, spelling errors in the prompt would not be marked towards a failure.
4.

Rubrics

SFT Prompt

Category	Notes	1-2 (Fail)	3 (Okay)	4-5 (Good/Perfect)
Prompt Clarity and Specificity **UPDATED 05/27**	The prompts can be underspecified but you should be able to fill in the missing details either by inference or making standard assumptions that an expert would make. The customer does not want very structured prompts that lay out all the specifics of the implementation **(05/27)**	- **[Fail - Prompt Unclear]** It's not clear what is being asked, making the prompt unanswerable - **[Fail - Prompt Missing Crucial Details]** Major details are missing that are explicitly required to answer the prompt and are not considered publicly available/common knowledge or you can't make standard assumptions to fill in the gaps		- **[Non-Fail - Prompt Requires a Minor Assumption] (4)** The Prompt has a specific request that doesn't require more than one minor assumption to answer it - **[Non-Fail - Prompt Ambiguous]** It's mostly clear what is being asked, while the request could reasonably be interpreted multiple ways, there is one single most likely interpretation

Category	Notes	1-2 (Fail)	3 (Okay)	4-5 (Good/Perfect)
Feasibility	- [Coding] Ex: "Can you create a new sorting algorithm that runs in constant time?" or "Can you give me all the code to create a social media platform like Instagram? But make it better please"	- **[Fail - Prompt Impractical Request]** Prompt contains an impractical request that can't be answered by an LLM in a single response - **[Fail - Prompt Impossible Request]** Prompt has at least one request that can't be fulfilled at all - **[Fail - Prompt Conflicting Instructions]** Prompt gives conflicting/contradicting instructions that can't be fulfilled simultaneously	N/A	- The prompt is completely actionable by an LLM or chatbot - The prompt contains no conflicting instructions/statements
Prompt Mentions Source	Examples of sources that should not be mentioned are Codewars, Leetcode, Exercism etc. Note: Referencing theory is non-failing e.g. Aider Hangman Benchmark	N/A	**[Non-Fail - Prompt Mentions Source]** All else is good except that the prompt mentions something related to the source of the prompt instructions	- No mention of where the problem was sourced from, no direct references to LeetCode, codewars etc. - E.g., including the link to the problem
Prompt Paraphrasing		- **[Fail - Seed Prompt Not Modified]** The paraphrased prompt is identical to the original prompt (exact verbatim)	- **[Non-Fail - Seed Prompt Minimally Modified]** The paraphrased prompt is not sufficiently different from the seeded prompt. Only the most minimal/small word changes were	• The paraphrased prompt is notably paraphrased. • Pragmatic meaning is the same, and the same unit tests can be applied to test the fulfillment

Category	Notes	1-2 (Fail)	3 (Okay)	4-5 (Good/Perfect)
		- **[Fail - Paraphrased Prompt Different From Seed]** The pragmatic meaning is different, causing an invalidation of the unit tests.	made **-Non-Fail - Paraphrased Prompt Removes Useful Contex**t] The Paraphrase removes additional examples not necessarily required but useful	• Useful examples included
Fluff		- **[Fail - Prompt Has Too Much Fluff]** The prompt has too much fluff or story that's not relevant to solving the coding problem		• **"Too much fluff"** (not allowed) ○ **Notable (more than a few words)** unnecessary storytelling / fictional information • **Actionable context** (allowed) ○ Background information that is actionable /, explaining the underlying theory - Tower exponentiation problems have fascinated mathematicians since ancient times, with early examples appearing in Chinese

Category	Notes	1-2 (Fail)	3 (Okay)	4-5 (Good/Perfect)
				mathematical texts ... --- - Modular exponentiation towers require Euler's theorem to avoid integer overflow: a^k ... --- - Given a list of positive integers \`arr\` of length n and a positive integer m, compute the value of the exponent tower ...

Model Failure

Category	Notes	1-2 (Fail)	3 (Okay)	4-5 (Good/Perfect)

Model Failure **UPDATED 05/27**		**[Fail - Model Passes All Unit Tests]** The model provides a script to the original prompt that passes all unit tests (make sure that the tests are accurate), meaning that the prompt is not difficult enough **[Fail - Faulty Unit Tests]** The model fails only because of incorrect unit tests for the problem or a unit test that doesn't make the most likely interpretation of ambiguity in the prompt **(05/27)** ~~**[Fail - Ambiguous Model Failure]**~~ ~~The model response failure is based on an unlikely interpretation of the prompt (i.e., the request can be reasonably interpreted in multiple ways and the model picked one of the least likely valid interpretations)~~ **(05/27)** ~~Note: If the prompt is underspecified the model must default to the most likely interpretation~~	N/A	• The model fails at least one valid unit test. **OR** • The model fails to compile due to a syntax error

Golden Unit Tests

Evaluate this using the test.py file that you see in the "Sphere Engine" section and not the text field below that. (05/27)

Category	Notes	1-2 (Fail)	3 (Okay)	4-5 (Good/Perfect)
Unit Tests Outputs	• Run the unit tests locally and validate the output • Note that the in-task output format/structure might be different from the output locally. • We should assert the correct values in our unit tests, else we mark it as incorrect!	- **[Fail - Faulty Unit Test Output]** The unit test outputs are incorrect/mismatch - **[Fail - Mismatch Skeleton Code]** The unit test mismatch the skeleton code provided	N/A	All unit tests outputs are correct and match the skeleton code
Unit Test Quality UPDATED 05/27	The unit test suite is expected to include at least some edge cases. Given that underspecified prompts are allowed, we can run into disagreements on how to handle a specific edge case. To handle this, as long as you think an edge case unit	- **[Fail - Coverage Core Functionality]** The core functionality is not being covered - **[Fail - Meaningful Assertions]** Values asserted in unit tests are not meaningful Note: We explicitly test for edge	- **[Non-Fail - Edge Cases]** At least some edge cases are covered. - **[Non-Fail - Duplicate Tests/Assertions]** The same assertion is tested more than once. - **[Non-Fail - Repetitiveness]** The unit	ALL SHOULD BE TRUE FOR A 4 or 5: - All core functionality is tested. - Exhaustive edge case coverage - Tests are independent and don't rely on shared state - Tests are deterministic. Tests will always output the same result on the

	test makes a reasonable interpretation which also aligns with the golden solution (i.e., golden solution passes the unit test), that's fine **(05/27)**	cases. - **[Fail - Edge Cases]** The unit tests fail to include any main edge cases - **[Fail - Test Independence]** The unit tests rely on modified shared state or similar. Running the tests in different orders results in different outputs. - **[Fail - Test Flakiness]** The unit tests are not deterministic. Running the same test on the same code leads to different outputs.	tests are not identical, but are very repetitive or test similar scenarios. Removing these redundant scenarios, you would not lose out on any meaningful test cases as they test the same logic essentially. e.g. first testing the model results with an input of 6 and then only incrementing the value asserted slightly in the next call e.g. 7 or 8 **(05/27)** However, some parameters are greatly affected by a small increment (e.g. K where K defines the dimensions of an array e.g. 1d vs 3d)	same code. - No duplicate tests or assertions

Model Guidance

Category	1-2 (Fail)	3 (Okay)	4-5 (Good/Perfect)	Notes

Model Guidance	- **[Fail - Objectively Incorrect Model Guidance]** The model guidance has incorrect suggestions that will lead to an incorrect reference solution - **[Fail - Incoherent Model Failure]** The model guidance is incoherent, considering the initial model failure - **[Fail - Incoherent Golden Response]** The model guidance is incoherent, considering the golden response Note: The guidance is appended above the golden response.	N/A	The model guidance is correct and covers the main failure	

SFT Response Rubric

Evaluate this using the solution_name.py file that you see in the “Sphere Engine” section and not the text field below that. **(05/27)**

Category	Notes	1-2 (Fail)	3 (Okay)	4-5 (Good/Perfect)

Reference Solution **UPDATED 05/27**	Do not audit the comments. Those will be stripped upon delivery to comply with the Aider benchmark. Aider solution is code-only without comments.	**[Fail - Reference Solution Incorrect]** - The reference solution does not pass all the unit tests (assuming the unit tests are accurate and comprehensive) - The unit tests are incorrect or incomplete and the reference solution doesn't solve the coding problem in the prompt **(05/27)** - **[Fail - Mismatch Skeleton Code]** The reference solution does not follow the correct skeleton/scaffolding	- **[Non-Fail - Reference Solution Mentions Source]** Passes Unit Tests but mentions source material either in the natural language guidance or golden solution itself	Passes all unit tests No Mention of Source Material Correct skeleton/scaffolding is used