SWE-bench Annotation Instructions

👋 Hello!

We have a dataset of GitHub issues from various open-source Python repositories. Each issue comes with a PR that successfully solves the issue described. Each PR consists of 2 parts: (1) code that resolves the issue (2) changes to the test files of the repository, which check whether the issue has been resolved.

We intend to use samples in this dataset as a benchmark for coding ability: For each sample, we give an engineer the issue text and ask them to write code to resolve the issue (without revealing the solution from the original PR). Then, we apply the test files from the original PR to their code and run the tests to check whether their solution passes.

Importantly, this setup assumes that:

- The issue description is sufficiently well-specified to understand what the problem is, and what the correct solution should look like.
- The tests are correctly scoped to the issue i.e. they correctly test for a solution exactly as described in the issue description, and do not test any other unrelated functionality.

In this task, you will help to check those assumptions and identify which issue + test samples are suitable for use in our benchmark.

You are now considering an issue from the {repo} repository.

Sample ID: {sample_id}

URL to the PR: {url}

Before continuing, please open the above PR in a separate tab and take a moment to familiarize yourself with the PR and relevant parts of the codebase.

Section 1 - Issue Description

Please take a moment to read the issue description below. If you prefer, you may navigate to the original issue from the PR (you can usually find the original issue linked from the PR): {url}

- **IMPORTANT**: Do not click through on any external links, and do not read the discussion below the GitHub issue. Our setup will only provide the main issue text as shown below, so please answer the questions on the basis of only the text shown here.[1]

{Issue Description}

How well-specified is the issue text?

Imagine that you are an experienced software engineer who has been instructed to create a PR that successfully resolves the above GitHub issue. You have full access to the codebase, and can see the issue description as it is above. But you are not able to ask for clarification and would need to work exclusively from this information.

Is the issue description well-specified enough for a meaningful attempt at a solution?

Question 1.1

- 0: The issue is well-specified and it is clear what is required for a successful solution.
    - Example: pylint #5201
- 1: There are some blanks to fill in about the issue, but there is a sensible interpretation of what is required for a successful solution.
    - Example: sympy #18030
- 2: The issue is vague and there is room for ambiguity. It is unclear what a successful solution would look like.
    - Example: scikit-learn #14520
- 3: It is almost impossible to understand what you are being asked to do without further information.
    - Example: pylint #5743

Please explain your choice above. Include references to specific filenames, function/class names, or lines of code where relevant.

Question 1.2

[Free text, minimum 100 characters]

Section 2 - Tests

You will now consider the tests that will be used to check whether the issue is resolved.

The Gold Patch is the solution for the issue given in the original PR, and the Test Patch contains any new tests that were added in that same PR to verify that the issue was resolved.

[1] We later clarified to annotators that they should assume embedded images in the Issue Description would be visible.

Please carefully study the Test Patch shown below. If you prefer, you may view these in the original PR using the GitHub diff-viewer instead: {url}

{Gold Patch}
{Test Patch}

False Negatives

Given a new candidate solution, we intend to use the Test Patch to check whether the solution correctly resolves the issue.

However, remember that these tests were written with a particular solution in mind (the Gold Patch) so they may be ill-suited for evaluating other valid solutions. We would like to know if the tests are correctly scoped to identify all reasonable solutions to the issue, or if the tests rely on narrow details that would unfairly penalize a new solution that is otherwise correct.

In other words, our setup only works if the tests do not rely on any details that are not present in the issue description. We find that the most common problems occur when there are subtle discrepancies between the tests and the issue text, such as the tests relying on a new function, variable name, or error message that were introduced in the Gold Patch but is not mentioned or differs from the Issue Description. Please check carefully for such discrepancies, and remember that the engineer attempting this issue will not have access to the original PR or the tests.

Note: You may go back to the previous sections to re-familiarize yourself with the issue whenever needed.

Are the tests well-scoped such that all reasonable solutions to the issue should pass the tests?
Question 2.1

- 0: The tests perfectly cover all possible solutions.
    - Example: sympy #13574
- 1: The tests cover the majority of correct solutions, however some unusual solutions may be missed.
    - Example: scikit-learn #12471
- 2: The tests work but some perfectly reasonable solutions may be missed by the tests.
    - Example: sympy #17022
- 3: The tests are too narrow/broad or they look for something different than what the issue is about.
    - Example: scikit-learn #14520

Please explain your choice above.
Question 2.2

[Free text, minimum 100 characters]

Section 3

You will consider the GitHub issue again and answer a final set of questions. You may navigate back to previous sections to see the issue and patches.

Difficulty

We wish to understand how difficult this issue is to solve.

How long would it take (for an experienced software engineer who had a few hours to familiarize themselves with the codebase) to understand the problem described in the GitHub issue, arrive at a solution and write the code for a solution?

Note: If the issue text was previously too ambiguous to solve at all, you may assume that the problem has been clarified sufficiently such that the high-level requirements for the solution are clear ("what"), but the specifics about the solution are left to the engineer to figure out ("how").

Question 3.1

- <15 min fix
    - e.g., a trivial change adding some assertions to a function
- 15 min - 1 hour
    - e.g., a small change that requires a bit of thought
- 1-4 hours
    - e.g., substantially rewriting a function or editing multiple files
- 4 hours
    - e.g., a very esoteric issue that clearly requires a substantial amount of research to fix, changing >100 lines of code

Explain why you chose the difficulty level above.
Question 3.2

Other Issues

Are there any other major issues that haven't been covered? i.e. Any other reasons this sample should not be used in our setup for evaluating coding ability.
Question 3.3

- 0: No
- 1: Yes

(Optional) Do you have any other notes you wish to add? If you answered 'Yes' to the previous question, please explain here.
Question 3.4
[Free text]

Confidence

Overall, how confident are you in the answers you have given for this sample?
Question 3.5
[1,2,3,4,5]