



Published using Google Docs

[Report abuse](#)

[Learn more](#)


[EXT] Gaiters_Wife Code RTL Instructions

Updated automatically every 5 minutes

Gaiters Wife Instructions

Last Updated: Oct 14, 2024

This document is meant to be used when you first onboard the project to learn everything you need. You should read this document thoroughly.

 Published using Google Docs

[Report abuse](#) [Learn more](#)

[EXT] Gaiters_Wife Code RTL Instructions

Updated automatically every 5 minutes

Reviewer Checklist: [\[EXT\] \[MAKE A COPY\] Gaius Code RTL Reviewer Checklist](#)

Office Hours:



Tasking Workflow

Task Specifications

[Step 1: Review the Instructions](#)

[Step 2: Write a prompt that causes a model 'failure' in the first turn](#)

[Step 3: !\[\]\(7d1d6890825e83a6a4a51febe2dcc7f3_img.jpg\) Evaluate model response](#)

[Step 4: !\[\]\(2bae76de5ebbd5c4d7d47162f1673734_img.jpg\) Generate a perfect response](#)

[Step 5: !\[\]\(b64b40baaee5acddc1eab8538ba84754_img.jpg\) Rate each response by the following](#)

[Final Step - Side by Side Rating and Justification](#)

Welcome to Gaiters Wife! This project is all about writing prompts and evaluating the quality of the responses generated.

What is this project all about?

Welcome to Gaiter's Wife! In this project, you will create data that will teach the model how to identify bad responses and produce good ones. Let's break down the task workflow and things that we consider important and then explain **why** each one is important!

Task specifications:

1. **Write a coding prompt that makes the model fail.**
 - a. You want to write a prompt that makes the model fail in a certain dimension such as instruction following, code executability or code correctness. Ensure your prompt is usable in a practical context.
 - b. **Why:** *by making the model fail, you're targeting an area the model is weak at! This means your data meaningfully improves the model! Think about it, why would we provide data on things the model is good at? For example, if I'm very good at washing dishes, I won't learn much if you show me videos on how to wash dishes. However, if I'm very bad at painting my walls, and you show me a tutorial of how to do it well, then I'll likely get much better at painting and therefore improve my overall skills!*
2. **Rate the poor response on a series of dimensions**
 - a. You will indicate which dimensions the model response has an issue in and explain why.
 - b. **Why:** *by telling the model exactly where it was wrong and explaining why, you're helping it actually identify the issue correctly, which helps it improve. For example, if my golf swing is bad, and you tell me "your golf swing is bad", I don't know what needs fixing. But if you specifically tell me "it's bad because your grip is wrong", then I'm more likely to improve.*

[EXT] Gaiters_Wife Code RTL Instructions

Updated automatically every 5 minutes

- c. **Why:** through doing this you're showing the model what a great response looks like! Now, it sees two responses to the same prompt (your first prompt), the first which shows it what **not to do** and the second shows it what **to do**.

4. **Perform a side by side ranking and explain why one response is better than the other**

- a. At this point, it's clear that the response you wrote is much better (if you performed the task correctly). Now, you rate how much better it is and justify why.
- b. **Why:** a high quality justification helps the model understand what makes the better response 'better' and how to distinguish between different responses to the same prompt!



Tasking Workflow

In all, the general task workflow involves:

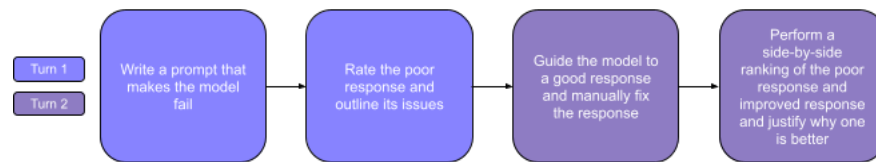
1. Write a prompt that produces a poor-quality response from the model in some way (e.g.: instruction following, code executability or code correctness)
2. Evaluate the Model's Response
 - Once the model generates a response to your prompt, review it carefully. Identify areas where the response falls short or makes mistakes.
 - If the model does not have an issue, edit your prompt and try again until you make the model fail



Turn #1 - Prompt

Can you write a python script which would query the database row by row and use go libphonenumber library python-phonenumbers to update a live print out of invalid dani numbers

- Use the retry chat from here button to try again
 - Once you find a response that failed, you will rate it on specific dimensions (e.g. instruction following, code executability, code correctness)
3. "Find" or create a good response for this prompt
 - a. Guide the model to provide a good response by telling it where it made a mistake and giving it hints to produce an error-free response
 - b. Then, you will rewrite and fix the response directly
 - c. It is crucial that the good response not reference the previous/bad response in its code or text. **The good response should be an independent stand-alone response to the original prompt.**
 4. Provide an explanation and side-by-side rating
 - Finally, you'll compare the poor-quality response with the improved, corrected one. You will write a great justification that illustrates why the response is bad



Task Specifications

Sections:

1. [Step 1: Review the Instructions](#)
 - a. [🚲 Turns](#)
2. [Step 2: Write a prompt that causes a model 'failure' in the first turn](#)
3. [Step 3: 🔍 Evaluate model response](#)
4. [Step 4: 🛠️ Generate a perfect response](#)
5. [Step 5: ⭐ Rate each response by the following](#)
6. [Final Step - Side by Side Rating and Justification](#)
7. [For Reviewers - Workflow and Task Rating Rubric](#)

Step 1: Review the Instructions

1. Read the task carefully
2. Ensure your prompt matches the given constraints
3. Aim for diversity - Create prompts that bring in unique variations to avoid repetition.

🚲 Turns

⭐ **Concept:** Turns

On many projects, *turns* refers to the pair of question-and-answer in a back-and-forth conversation you have with a model. So turn #1 is your initial request and the model's response, turn #2 is your follow up, and so on.

On this project, turns are different! Turn #1 is still your initial request and the model response, but you need to think of Turn #2 as a revision of Turn #1. It's advised to think of **each turn by its goal**.

Goal of Turn #1:

1. Write a prompt that makes the model fail
2. Rate the poor response, correctly labeling the dimensions (e.g. instruction following, executability or correctness) where it fails.

Goal of Turn #2:

1. Write a new prompt built on prompt #1 that guides the response to reproduce the same response but fix the issues
2. Manually improve the new response
3. Make sure the new improved response does not mention the first model response.
4. Justify why this response is better than the initial response

Turn #1

Step 2: Write a prompt that causes a model 'failure' in the first turn

⭐ **Purpose**

Your job now will be to write a prompt that:

1. Causes a model failure (a bad response from the model).
 - a. This is an iterative process. You will be required to keep generating a new model response by rewriting

[EXT] Gaiters_Wife Code RTL Instructions

Updated automatically every 5 minutes

1. **Instruction:** The instruction is your main request to the model, it dictates what the model says. **a perfect response will address the instruction completely, safely, and accurately.**
2. **Constraints:** Constraints dictate how the model responds or talks. For example, if you ask for a code that generates a password and you add a constraint to have special characters. **A perfect response will follow these constraints while still addressing your instruction.**

When you're writing prompts, make sure they are clear, challenging, and natural. The goal is to maintain complexity while ensuring that the prompt is something you can assess confidently.

Step 3: 🔍 Evaluate model response

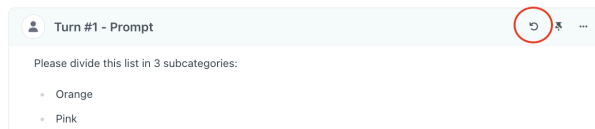
Once you've written this prompt, check that it causes a bad response.

1. **Check that the prompt written has caused the model to fail to answer as intended.**
 - If the model fails to answer correctly → no further action is required on the prompt. ✓
 - If the model does answer the prompt correctly → create another prompt until you are able to cause a failure. ∞

How can I check for this?

- Depending on your prompt and category it may be easy to spot if the model response is wrong, but sometimes it may not be easy.
- Run the code if you are not sure about the answer, but keep in mind **DO NOT OVER DO IT, typically incorrect model responses should be obvious and important.**

1. **If the prompt does not produce an error in the response, keep rewriting the prompt until you encounter a failure (bad response).**
 - Use the 'retry from chat' at the top right to create another prompt and generate a new response.



1. Once you've confirmed that the response produces a failure, proceed to step 4 and Turn #2 of guiding the model to generate a better response.

Turn #2

Step 4: 🍌 Generate a perfect response

- i. **Note that the model failure flagged in Response 1 must be related to prompt 1. Response failures should not be based on additional functionality requests which were not present in prompt 1.**
- b. **Point out the errors the model made, and ask it to provide a new response taking these things into consideration.**
 - o Remember that this is Turn #2, so the model already knows what happened in the prior turn. These modification(s) will help guide the model to write a better response; **this is what we refer to as “guiding”**.
 - o The goal for this step is for you to be able to get the model to provide an ideal response purely through prompt engineering.
 - o **Note: You cannot take out any of the parameters/constraints from the prompt turn 1. The constraints added in prompt2 when compared with prompt1 should NOT be additional reasons to justify a model failure. They should be pointing out more clearly to the model why it failed by adding specifics in prompt 2 so that response 2 is closer to our goal.**
 - o **Our goal is to reach a perfect response 2, writing a more specific prompt 2 which addresses the failures in response 1 makes it easier to reach the perfect response. However, we should just be specifying more clearly, not adding new functionality requests or modifying previous constraints.**

What is “Guiding the Response”?

Guiding the model means ‘steering’ it to give you the kind of answers you want. It’s basically your chance to produce the perfect response **to the original prompt** using an LLM!

You can use this prompt space freely! For example, if I found the first response to write a great response on who the 10 last US presidents are, but the president in spot #9 was incorrect, I can just prompt it to output the list and replace the answer at #9 with the correct president.

Another way I can use it is by giving the model hints! For example, if the model didn’t output a format correctly, I can provide the model with an example on how to write in a certain format.

You do this by being clear about what you ask, like choosing a certain style or focusing on a specific topic. It’s like giving the model directions so it stays on track with your needs.

Strategies for steering:

1. **Use Clear Prompts**
 - a. Clearly state what you need and mention any specific details that are needed or missing.
2. **Use Follow-Up Instructions**
 - a. Provide follow-up requests by specifying what was missing or what needs to change
3. **Guide Through Example**
 - a. Provide examples for the model

1. **Manually adjust the response to ensure it is completely error-free!**

- a. After guiding the model to provide a better response, you should edit the response, if needed, to make it as close to perfect as you can. You should:
 - i. Use the model as a basis for where to start and only apply necessary edits.
 - ii. Research to create the perfect response

Step 5: 🌟 Rate each response by the following

Every prompt you write in this project will generate a model response which you'll rate for these dimensions, (and write a quick justification for each dimension):

Dimension	1 (Major Issues)	2 (Minor Issues)	3 (No Issues)
Instruction Following	Response missed key components of the prompt, rendering it unhelpful to the user.	The response addressed most of the instructions or goal(s) of the prompt, but missed or misinterpreted some small parts. A user would still be reasonably satisfied.	All prompt instructions were followed; response delivered fully on the tasks of the prompt.
Code Executability	<p>Executability issues may stem from poorly written code, missing parameters, incorrect parameter values, missing code blocks, missing imports or other similar issues.</p> <p>Executability issues do not refer to requiring API keys, missing external files or any other situation in which a user's environment would actually be able to run the code, in these cases please try your best to create a mock environment in which to test the code.</p> <p>If the code cannot be executed and this is not due to lack of access to a specific environment, file or API, then select "No".</p>	Some responses contain multiple code blocks. If you can run some, but not all of these, answer "Yes-Partially".	Some responses contain multiple code blocks. If you can run all of these, answer "Yes-Fully".
Code Correctness	<p>Any of the following are true:</p> <p>Text: primary claims contain meaningful inaccuracies (or unfounded claims), such that</p>	<p>Either or both of the following are true:</p> <p>Text: primary claims (central to addressing the prompt) are factual /</p>	All claims in both the explanation and any code comments are factual and accurate; the code (if any) is

[EXT] Gaiters_Wife Code RTL Instructions

Updated automatically every 5 minutes

<p>the design or usage of a library, or a response that mischaracterizes what the code does.</p> <p>Code: has one or more of the following problems:</p> <p>Functionality: The code does not, or will not, produce the proper intended output or is broken in a logical/functional fashion.</p> <p>Safety: the code would create safety or security risks if used, such as relying on libraries with known vulnerabilities or failing to sanitize user inputs.</p> <p>Do not use this to flag responses that make simplifying assumptions that a user would reasonably be expected to notice and improve, such as using a hard-coded password in a clearly visible location.</p> <p>Performance: the code is unnecessarily slow, for instance, due to using a quadratic algorithm where a (log-)linear option exists, or repeatedly concatenating long strings instead of using a stringbuilder.</p> <p>Documentation: the comments contain meaningful inaccuracies that make the code very hard to understand.</p> <p>Keep in mind that the code may be functional for the prompter, even if it does not compile or run on your setup. For instance, a response that points to a file only accessible to the prompter, or</p>	<p>/Examples include: an otherwise correct explanation of a library that uses an incorrect link, or a description of a system that misconstrues a small detail of its design.</p> <p>Code: has minor problems where the main functionality of the code is correct; e.g., it fails to handle an edge case, or is correct but has misleading comments.</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	unless it contains errors that would (likely) manifest in the prompter's programming context.		
--	-----------------------------------------------------------------------------------------------	--	--

Final Step - Side by Side Rating and Justification

★ Concept: Rating

The rating aspect is simply marking which items the response “checks off”. This is the simplest part of the task, but one that needs the closest attention to detail.

Any task that doesn't perform correct ratings risks a low-quality score, so please do not rush through this section!

1. **Evaluate the two responses**
2. **Provide a side-by-side rating for the two model responses**
 - **Ratings of 1-3:**
 - 1: Response 1 is clearly better than Response 2, with almost no room for improvement.
 - 2: Response 1 is better than Response 2, but there's some room for minor improvements.
 - 3: Response 1 is somewhat better than Response 2, but not significantly.
 - **Ratings of 4:**
 - Response 1 and Response 2 are of similar quality, and neither stands out as better than the other.
 - **Ratings of 5-7:**
 - 5: Response 2 is slightly better than Response 1, but the difference is small.
 - 6: Response 2 is better than Response 1 and provides a noticeably better solution.
 - 7: Response 2 is clearly superior to Response 1, with no contest.
3. **Write the justification: After scoring which of the responses are better, you should provide a justification for why that score was chosen.**
 - Keep your justifications concise, yet also as informative as possible for which response is better.
 - The justification must adhere to the following guidelines:
 - The justification should be between 50-75 words long
 - The justification must provide a clear, objective, relevant explanation for why response 2 is better than response 1.
 - The justification must be **specific** and detailed.
 - The justification must not use first-person language (“I”) -or- include phrases like “The AI,” “the chatbot,” “the model,” “the LLM,” etc. (regardless of capitalization) with the exception of mentioning @response 1 and @response 2.
 - The justification should be without any grammar or spelling errors. Use the Grammarly extension to help with this.

👉 Writing Justifications



[EXT] Gaiters_Wife Code RTL Instructions

Updated automatically every 5 minutes

- **Start with the Verdict:** Begin with a clear statement about which response is better.
 - Example: "Response B is much better than Response A."
 - **Address Key Issues:** Mention if there are any problems with Instruction Following, code execution, or code correctness in the responses. Provide **evidence**.
 - Example: "Response B follows the instructions, while Response A contains factual errors."
 - **Consider Other Factors:** If there are issues with safety, harmfulness, Content Quality, or Writing Style, mention them.
 - Example: "Response B is concise and well-formatted, while Response A is too wordy."
-