Microsoft API Model Evaluation [Multi Turn]

Introduction

We are focusing on collecting **high-quality** datasets to train and enhance a Large Language Model (LLM) using Reinforcement Learning from Human Feedback (RLHF).

Context

We will interact with the LLM through a Turing in-house developed UI Tool. The interaction will help generate valuable data that will be used to train the model, improve its performance, and address any identified weaknesses.

Objective

To gather a comprehensive and diverse dataset through human-model interactions, which will serve as the foundation for training and refining the LLM in the next phase of development.

Trainer Instructions

As a trainer, you will simulate realistic scenarios by role-playing as a user interacting with a sophisticated AI called "Assistant." Utilize the LLMs similarly to how you would in your day-to-day engineering tasks. Our goal is to collect valuable data that will enhance the model's usefulness in various scenarios while improving its capabilities.

High-level workflow

This provides a high-level overview. The following sections contain detailed information about each aspect.

1. **Receive Task Scope**: You will receive a task scope as metadata.
2. **Design Prompt**: Create a prompt within the provided scope that challenges the model.
3. **Review Responses**: You will be presented with four responses from the model.
4. **Evaluate Responses**: Complete all fields in the evaluation form in the tool for each response.
5. **Provide Details**: Include a preference rating, justification for your choice, and an ideal response.

Constraints & Guidelines

- **Behavioral expectations from personas:**
  - **User Persona**: (1) Resembling real-world usage patterns for different people. (2) Diversity in question structure & formatting. (3) Various backgrounds and skill levels. (4) Can change subject mid-conversation. (5) Can make mistakes.
  - **Assistant Persona**: (1) Well-formatted markdown & code. (2) Correct, Precise, Concise & Engaging Language. (3) Correct, Optimal & Readable Code (4) Responses should feel "Tailored" to the user. Example: A beginner user would need explanations for simple things while an advanced user would not. (5) Every response should be optimal in isolation, providing a comprehensive answer without overflowing the user with information, emphasizing important key points & summarizing less important ones (6) Whenever possible well-known references that support claims should be added as hyperlinks… (Don't use random no-name or opinionated websites/sources)
- **Adhere to the Metadata Constraints:**
  - Adhering to the metadata should strictly manifest in the user prompts. **The assistant is aware only of the user's prompt and not the metadata, so please make sure to include all requirements directly in the prompt.**
  - **Definitions:**
    - **Taxonomy:** A hierarchical classification of topics that will be **strictly** followed, where each level represents a broader or more specific category.
      - For example, "Backend Development > API Development" represents a broader category (Backend) and a specific focus (API Development).
    - **Subtopic:** A focused subject derived from the **taxonomy** that will be **strictly** followed to narrow down the discussion.
      - For example, "Python-based Microservice Architecture in Backend" is a subtopic under API Development.

        - **Use Case:** A practical application showing how the **subtopic** is implemented, which will be **loosely** followed.
            - For example, "Design and Implementation of RESTful APIs using FastAPI with AsyncIO for Enhanced Performance"* illustrates a specific real-world scenario based on the subtopic.
        - **Task Difficulty:** This indicates the difficulty of the prompt for the model, which **loosely** guides the prompt design. Difficulty encompasses not only the complexity of the problem but also the number of requirements included in the prompt. Check out the difficulty rubric to learn more.
        - **Conversation Length:** This refers to the **total number of turns** in a task where the user poses a question or request to the model. This is a **strictly** enforced requirement that applies to the entire task, specifying the minimum and maximum number of turns each task should include.
        - **Conversational Type:** This refers to how you design the prompt, whether it contains a code snippet or not. It encompasses coding problems, general Q&A on any topic, etc. This **strictly** guides the design of the user prompts. Please refer to the different types here.
        - **Programming Language:** This should be the programming language in which the user expects the code output to be, or the language of the code he presents to the assistant. This **strictly** guides the design of the user prompts.

    Important Note:

    - Please remember that the metadata constraints either **'strictly'** or **'loosely'** guide user prompt creation. **'Strictly'** refers to mandatory aspects, like programming language, conversation length etc, that must be followed, while **'loosely'** indicates guidelines that are suggested but can be adjusted or bypassed when necessary.
    - It's **essential and mandatory to consider these constraints when starting the conversation in the first prompt.** After that, allow the conversation to take its natural course, building subsequent turns on the previous responses.

- **Prompt Design:**
    - **Solvable with the given context** - Always present the problem to the model clearly and completely, ensuring it's solvable with the given information in under 400 lines of code. *(If you give this prompt to a random freelancer on the internet, he should be able to give you a correct solution while ChaptGPT/llama3 fails)*
    - **The model fails in Solution, Instruction following, or Correctness** - Prompts should aim to challenge the model with hard but very clear problems. (Check response review guidelines in the appendix for more details)

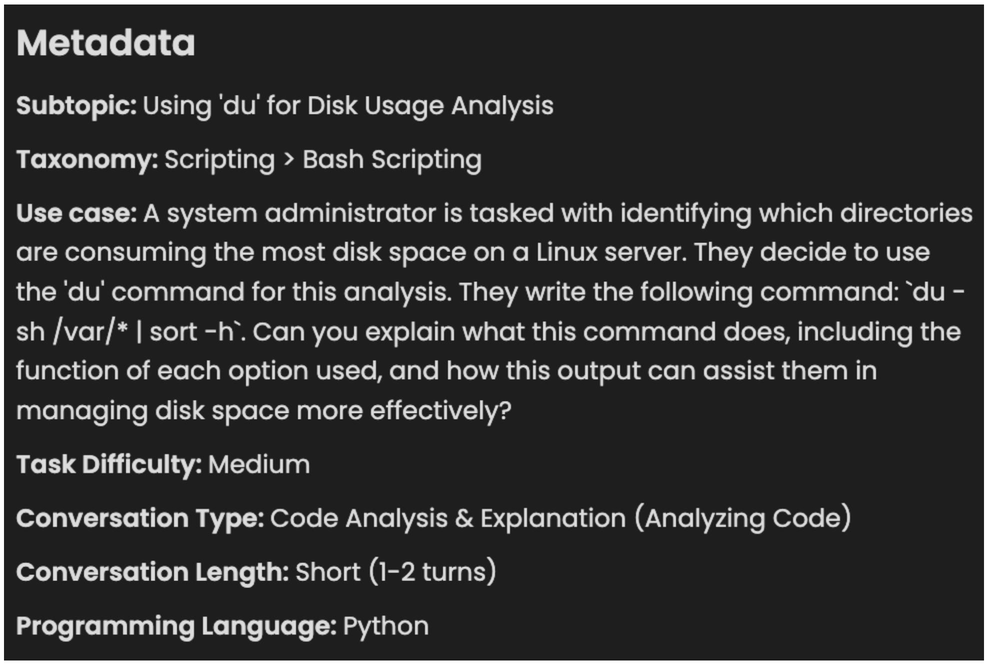
- **Response as Conversation History (only applies to multi-turn tasks):**
    - **Selecting Responses**: At each turn, you'll choose a response from the generated options using the **"Choose as Conversation History"** button. This selected response will be incorporated into the context of your next prompt.

    - **Random Selection**: To ensure diverse data collection, select the response randomly. This approach allows you to make independent choices at each step, without being influenced by previous selections.
    - **Preferred vs. Choose as History:**
        - **Preferred Response**: This is the response you select as the most optimal among the generated ones. It represents your choice of the best response from the available options.
        - **Choose as History**: This action adds a response to the conversation history, making it part of the context for future prompts.

    In summary, the Preferred Response is your optimal choice from the generated options, while "Choose as History" incorporates the response into the ongoing conversation context.

Detailed Workflow

- Access the list of available tasks:
    - Access the labeling tool at https://labeling-ms.turing.com/
    - Click on **Workspace** and select the `[MS Evals - Coding]` from the Project filter.
    - Next to the **Project** filter, there is a **Batch** filter where you can select the batch number you'll be working on. Remember to select the batch with the **'In Progress'** status.
    - Now, you should be able to see a list of tasks under the `Unclaimed` tab. You can select any of the tasks from this section and start working on it.
- Claim a task:
    - Find a task:
        - You can view the details of any task by clicking on the `View Metadata` button corresponding to the task. This will help us to choose the task which is best aligned with our expertise and preference.
        - You can also filter for specific Languages, Domains, or technologies by using the search bar. (Please use the search bar on the top)
    - Once you have selected a task on which you want to work, click the corresponding `View` button to go to the task details page.
    - Click the `Claim` button in the top right corner to allocate this task to yourself. *Additionally, if something is wrong/incorrect with the task, you can collaborate with your Squad Lead to mark it improper by clicking* `Mark Improper`*. For example In the task metadata below, you can see that the taxonomy is related to Bash scripting, while the programming language is Python.*



Metadata

Subtopic: Using 'du' for Disk Usage Analysis

Taxonomy: Scripting > Bash Scripting

Use case: A system administrator is tasked with identifying which directories are consuming the most disk space on a Linux server. They decide to use the 'du' command for this analysis. They write the following command: `du -sh /var/* | sort -h`. Can you explain what this command does, including the function of each option used, and how this output can assist them in managing disk space more effectively?

Task Difficulty: Medium

Conversation Type: Code Analysis & Explanation (Analyzing Code)

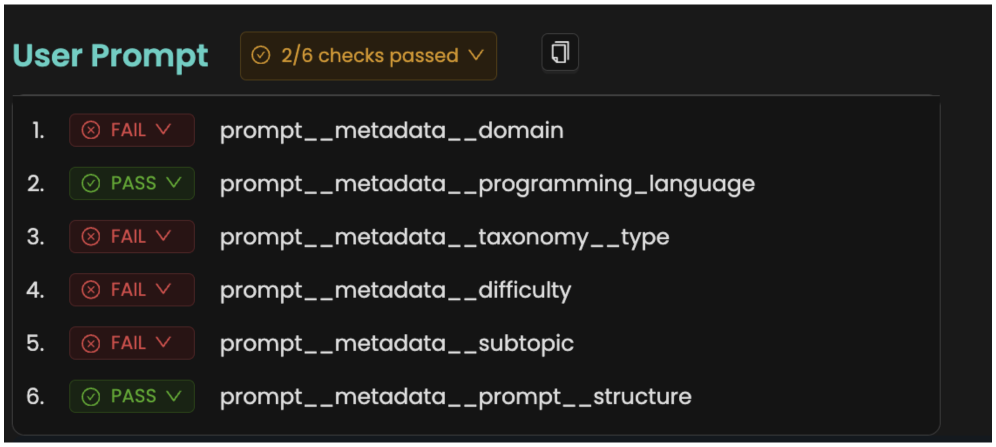
Conversation Length: Short (1-2 turns)

Programming Language: Python

- Once you have claimed a task, you can start working on it immediately and you can also view it under the `In Progress` tab in the **Workspace**.

  Important Note: **Before starting work on the task, make sure to "Claim" it in the tool. You can do this by pressing the `Claim` button on the task details page.**

- Work on the task:
  - To start working on the task, Click the `RLHF Link` button which will take you to the conversation page. Here, you will act as a user by providing prompts to the assistant and evaluating its responses.
  - On the conversation page, clicking `Create New Turn` will add a new field where you can enter the user prompt.
  - Click `Submit Prompt` to generate responses for the prompt you entered.
  - After submitting the prompt, you'll see some automated checks running. **It's okay if some tests fail as long as the prompt you've added adheres to the metadata constraints.** These checks are automated and not always precise, so **they can occasionally fail even for valid prompts.**


User Prompt 2/6 checks passed

1.	FAIL	prompt__metadata__domain
2.	PASS	prompt__metadata__programming_language
3.	FAIL	prompt__metadata__taxonomy__type
4.	FAIL	prompt__metadata__difficulty
5.	FAIL	prompt__metadata__subtopic
6.	PASS	prompt__metadata__prompt__structure

Important Note: These automated checks are not always precise. Your goal is not to pass all these checks in every prompt. Just ensure that your first prompt aligns with the metadata constraints. After that, allow the conversation to take its natural course, building subsequent turns on the previous responses.

- After the model generates responses, each response will have an associated **Evaluation Form**. You will complete this form based on your analysis and submit it using the **`Submit Evaluation`** button. Instructions on how to fill out each form field can be found here. Each field in the form also includes the same description for guidance.
- Please make sure to **run the code inside the sandbox provided in the tool.** Be aware that there are current limitations with the sandbox, including some missing execution environments. Whenever possible, prioritize running the code in the sandbox. **If the sandbox does not support the required environment, you can run the code in your local environment and take screenshots. Upload the screenshot to Google Drive, and paste the link in the "Comments for Reference" field. Make sure that the folder is shared with view access for the organization "turing.com"**

  More details on how to execute the code in the tool's execution environment can be found here. The video contains some references to other documentation that you can ignore: Execution environment.mov

  If you're running the code in the tool, remove any incomplete, erroneous executions, or test runs from the **Code Executions** section. Only keep the relevant ones. Use the red 'Delete' button to remove these entries.

  Important Note:

        - Please remember that it's very important to run and test the code generated by the model, identify any issues, and highlight them appropriately in the Evaluation Form.
        - If you're running the code on your local machine and it is taking a lot of time to set up the environment, you can skip executing the code. Instead, please add a comment in the '**Comments for Reference**' field explaining that it was taking a significant amount of time, and due to time constraints, you were unable to execute it.
    - There is also a **Model Preference Evaluation** form where you can select the response you prefer, provide details on why you prefer it, and include an **Ideal response**. Instructions on how to fill out each form field can be found here.
    - For tasks with a **Conversation Length** greater than 1, you must select a response as conversation history. You can do this by clicking the `Choose as Conversation History` button. For more details, refer to the **Constraints and Guidelines** section under **Response as Conversation History**.

      Important Note:

        - The `Create New Turn` button will remain disabled until a response is selected as conversation history.
        - There's also a `Regenerate` button next to every response. You should only press the regenerate button if the model fails to generate any response at all. If the response is incorrect due to hallucination or other issues, **do not regenerate it.** Instead, focus on evaluating the incorrect response thoroughly, as this is crucial for identifying and addressing model behavior.
    - Once you have completed your conversation and its analysis, you can click the `Done` button in the bottom-right corner to save the whole conversation.

      Important Note: Please avoid using the `Reset Prompt` button, as it will delete all your progress. **It's recommended not to use it.**

- Submit the task:
    - To submit the task, open the task detail page in the labeling tool and click the `Complete` button. You will also find a completion form where you should enter the time it took to complete the task and any notes you wish to share with the reviewer who will assess the task.
    - Once submitted, you can see the task under the `Completed` tab.
- Expect the review:
    - Some of your tasks will be reviewed by your Squad Lead and other Squad Leads.
    - When any of your tasks are reviewed, you will get a notification of it through email.
- Fix the task:

    - If your task gets a score of less than 4, then you have to edit that task to fix it based on the comments of the reviewer.
    - To fix a task, go to that task in the labeling tool using the link in the email or by browsing in the labeling tool.
    - Click the **`RLHF Link`** button to go to the conversation page. Here, we will use the **`Review Again`** button to put the conversation in the edit model. Now, we will follow a similar workflow as steps 3 and 4 to make changes in the task and submit it again.
    - **Note:** If a task requires a re-work, please try to complete that task asap, maximum within 24 hours.

      Please watch the **General Overview** video, which demonstrates all the instructions provided earlier.

Expectations

- Time Limits:
    - New task credits:
        - Research overhead: **10 mins**
        - Per turn duration: **40 mins**
    - Redo task credits: **20 mins**
- Quality:
    - You should maintain an average quality score which is >= 4.3.
    - Copy/Pasting prompts from any online website (also includes other LLM-generated prompts/responses like GPT) is considered **plagiarism**. Learn more about the responsible use of LLMs in this document: Responsible Use of LLMs

Please reach out to your lead for assistance in case you're struggling to adhere to this.

QA Instructions

These instructions are **mandatory** for squad leads and reviewers to read. If you are working on the task, you can skip them.

Grading Rubrics

Grade	Prompt Quality	Evaluation Quality
5 (Very High)	The prompt adheres to all constraints and resembles real-world usage patterns for different people.	The trainer provides a comprehensive evaluation of the assistant response and adheres to all response review guidelines.

4 (High)	The prompt adheres to most of the constraints but can be optimized to align better with the instructions.	The trainer covers most of the evaluation criteria but can be optimized to be more comprehensive and include more failure categories.
3 (Borderline)	The prompt does not follow some constraints or guidelines.	The trainer fails to notice some important criteria and the evaluation can be significantly improved.
2 (Poor)	The prompt does not follow most of the constraints and fails to depict real-world scenarios.	The trainer misses most of the response review guidelines and fails to provide constructive criticism.
1 (Very Poor)	The prompt fails to follow any constraints and real-world scenarios.	The evaluation adds no value and fails to highlight all major issues.

Workflow

- Access the list of completed tasks
    - Access the labeling tool at https://labeling-ms.turing.com/
    - Click workspace and select the project **`MS Evals - Coding`** from the dropdown.
    - Now, you should be able to see a list of tasks under the **`Completed`** tab. You can select any of the tasks from this section and start working on it.
- Click the **`Review`** button for the task you want to review.
- You'll find four main fields in the form. Make use of the fields as follows:
    - **General Quality Score:** Use this field to add general comments regarding the whole task. Highlight any positive aspects the trainer consistently followed or note any issues that exist across turns.
    - **Prompt Quality:** Evaluate how well the prompt adheres to the metadata constraints, its comprehensiveness, whether it contains code, and whether the prompts for later turns follow a natural flow, among other aspects.
    - **Evaluation Quality:** Use this field to highlight issues with the evaluation form submissions. Include comments regarding whether the fields are appropriately filled, and ensure there are no inconsistencies, etc.
    - **Ideal Response:** In this section, validate the ideal response based on how well it adheres to our guidelines, etc.
- In the **Type of Review**, you'll select the kind of review you will be adding for the task. Please refer to this document to learn more about it.
- Provide **Feedback** for the task.
- Add the time it took to complete the review
- Click on **Submit**

  - Select Do Nothing for a rating >= 4
  - Select Request Changes for a rating < 4 and ensure they are **reworked**.

Expectations

- Time Limits:
  - New task credits:
    - Research overhead: 10 mins
    - Per turn duration: 25 mins
  - Redo task credits: 20 mins
- Reviews need to be very accurate & high quality. Low-quality / partial reviews are not acceptable and may result in consequences for the reviewer.
  - We allow for a longer duration for conducting a review to be able to go deep and produce a high-quality & deep review.

Appendix

Difficulty Rubric (Relative to the model)

We can judge whether a response is easy, medium, or hard by whether it contains at least one of the following properties… We iteratively assess difficulties from hard to easy and stop once we have a match. **(Dimensions are OR'ed together)**

Name	**Dimensionality** **Number of Constraints in the prompt**	**Code Output** **Lines of Code elicited in response**	**Complexity** **A minimum grade of FANG engineer that can solve within 1 hour**
Hard - Any of >>	8+	150+	Principal Engineer
Medium - Any of >>	4-7	50-150	Senior Engineer
Easy - Any of >>	2-3	5-49	Entry/Mid Engineer
Trivial - **All** of >>	1-2	1-10	Student

Some more details on different levels of task difficulty, detailing the experience required, knowledge needed, and the complexity of the solutions involved:

- **Easy:**
  - **Experience level:** Entry/Mid Level Engineer

    - **Knowledge required:** limited domain/algorithmics knowledge or implementation context (architecture, libraries, pre-existing code)
    - **Ambiguity of prompt:** little ambiguity in the question (in case of underspecification, good default behaviors are easy to come up with or not important), limited complexity of specifications (in #instructions)
    - **Complexity of solution:** solution is easy to explain (e.g., code doesn’t need comments to be understood) and to test for/debug (limited corner cases)
- **Medium:**
    - **Experience level:** Senior Engineer
    - **Knowledge required:** may require knowledge of standard algorithms and data structures to get an optimal solution, knowledge of common libraries and concepts or additional code context.
    - **Ambiguity of prompt:** medium ambiguity in the prompt (e.g., needs to come up with reasonable ad-hoc data representation or class structure without explicit guidance), multiple requirements should be satisfied or multiple bugs should be found
    - **Complexity of solution:** involves corner cases that should be dealt with separately; explanation of the solution requires some abstraction or decomposition of the problem into a few subproblems
- **Hard:**
    - **Experience level:** Principal Engineer
    - **Knowledge required:** require expert domain knowledge, or information on the specific application or deployment scenario, including substantial specific API/code context
    - **Ambiguity of prompt:** finding good solutions need non-trivial design decisions regarding data structures, algorithms or code architecture/design patterns
    - **Complexity of solution:** finding a solution requires solving several non-trivial subproblems or finding non-trivial bugs; problem involves tricky corner cases, explaining the solution to a non-expert requires adding context

Conversational Type:

- **Code Authoring (Code Generation):** This involves generating code from a text description. By providing details in the prompt, it **elicits a response from the model that contains the generated code.**
- **Code Analysis & Explanation (Analyzing Code):** This includes activities such as refactoring, compression, optimization, translation, and precise edits based on a text description. **It is required to include the code in the prompt.** For example, in refactoring, providing the code to be refactored is mandatory; without it, the purpose of the task is defeated.
- **Coding Q&A & Assistance (Talk about Code):** This involves general Q&A that doesn't necessarily include code but rather focuses on programming concepts, problem-solving, and solutions.

- **Conversational Coding (Multi-turn):** This is a mixture of the previous three types, allowing for multi-turn interactions that may involve code generation, analysis, and general coding assistance.

Response Review Guidelines

These instructions will help you fill out the **Evaluation Form** attached to every response.

Mistake Severity Definitions

- **Major Issues** - Mistakes that negatively affect the user experience in significant/critical ways (The user gets little or no value, perhaps negative value).
- **Moderate Issues** - Mistakes that partially affect the user experience. (The user gets some value, but significant improvement could be made).
- **Minor Issues** - Mistakes that may not affect the user experience or affect it in trivial ways. (The user still gets most of the value, perhaps there's room for improvement).
- **No Issues** - No mistakes are made. (User gets full & optimal value)

Instruction Following

Is the provided response **on-point** & **respects all constraints** in the user prompt? Is it tailored to the User's skill level?

Core areas we should look for in this case:

- Comprehends and adheres to all constraints and requests of user
- Addresses all the requests of the user (Exceptions will be requests that are outside the capability of the LLM. For example: Give me a sorting algorithm with O(log(n)) time OR Give me the production-ready React app to track student data.)
- Focus remains on the user's request
- Not too short to skip the important and helpful information
- Not too verbose to include unnecessary details
- Well-tailored for the skill level of the user

Accuracy

Does the AI's response **correctly** and **completely** address the information and code requirements?

Core areas we should look for in this case:

- Factual correctness
- Comprehensive Answer (No missing key points)
- Code syntax errors
- Code functional errors

Efficiency

Is the AI's response **optimal** in terms of the approach, code complexity, case coverage, and the method suggested in response to the user's prompt?

Core areas we should look for in this case:

- Optimality in terms of **Time and Memory complexity** (It is fine if assistant gives an algorithm which is efficient and used in mainstream rather than a complex algorithm which optimizes on time/memory a little bit more.)
- Handles all the edge cases
- Takes care of security aspect of code
- During Q&A, suggest optimal answer to the user

Presentation

Is the **presentation** of the AI's response clear, well-organized, and appropriate for the user's skill level?

Core areas we should look for in this case:

- Is well structured
- Uses Markdown and LaTeX capabilities to present response in best way possible
- Code is well formatted, commented, and readable
- Does not use too formal language
- Does not make use of informal slang or a language which is not appropriate.
- Does not make typos or grammatical errors

Up-to-date

Ensures that code uses the latest libraries, functions, and modern practices while avoiding outdated or deprecated methods.

Core areas we should look for in this case:

- Use modern, non-deprecated libraries and up-to-date methods (e.g. **f-strings** in Python, **let/const** in Javascript)
- Recommend efficient and current best practices.

Executable Code

Classify the completeness and usability of the code:

- **No Issues:** Code should run "out of the box" (ignoring whether it contains bugs)
- **Minor Issues:** The response updates a single function, meant to be copied into the user's broader code.
- **Moderate Issues:** The response updates part of the code, with comments like **"// rest of your code here**"
- **Major Issues:** The response provides a skeleton code block with partially implemented functions, using like **"// your code here"**
- **N/A:** If the code is not included in the response.

Important Note: Right out of the box means that the code will run in the appropriate context, provided the necessary environment (libraries, runtimes, etc.) is set up. If the code snippet is part of a larger codebase, it should work seamlessly when inserted into that context.

Expected Output

This field assesses whether the output provided in the response aligns with the actual output of the code when executed.

- **Example:** You asked about a Fibonacci function, and the model generated code with example usage, stating that running the function with an input of 5 would return 10. **In this case, the expected output is part of the response—meaning the model is indicating what the output should be**. You need to verify if the function actually returns 10 after execution and select Yes/No based on that. **If the output is not specified and the model only provided the code without stating the expected result, select N/A.**

Other Issues

Are there any **other significant issues** in the AI's response that affect its overall performance or user experience but were not covered in the predefined categories? Please refer to Microsoft - Failure Categories LookUp for failure categories.

Rating Justification

We will use this field to put:

- Reasoning for the marked Major/Minor Issues.
- Details of the 'Other Issues' if it is marked Minor/Major.

Important Note: Please use this field to provide descriptions regarding the issues you identified in other fields. **It's important to fill in this field appropriately, as it is crucial for helping the task reviewer understand your thought process.**

The convention we use is to add the field name before the description. For example, if you find issues with 'Presentation,' you can describe it like this: 'Presentation: Lack of comments ... etc.'

Comments for Reference

We will use this field to put:

Any notes that you want to convey to the reviewer? You can also include links to screenshots or a sandbox where you have tested something related to the response.

Final Score

Evaluate the models based on their ranking or preference compared to each other. **Remember, this score is relative to the other responses generated in the same turn.**The rating scale is as follows:

1. Terrible
2. Bad
3. Fair
4. Good
5. Exemplary

Important Note: The **Final Score** for different responses can be the same. **Relative to each other means comparing the responses against one another, but it doesn't mean that two scores cannot be identical**. In some cases, multiple responses may be of equal quality, meet the criteria equally well, or have comparable strengths and weaknesses. **In these situations, it is acceptable to assign the same score.** The key is to ensure that the scoring accurately reflects your preference or ranking of the responses relative to each other. **So, while distinct scores are encouraged when clear differences exist, identical scores are valid if two or more responses are truly on par with each other.**

Preference Evaluation

These instructions will help you fill out the **Model Preference Evaluation** form, which should be completed for each turn.

Preference

Select the response you prefer from among the generated options. Choose the one that aligns best with the requirements in the prompt, is the most optimal in terms of clarity and completeness, and effectively addresses the given problem.

Preference Explanation

Provide your reasons for choosing one response over the others. Elaborate on why you selected that specific response.

Ideal Response

This field should include what you intend the model to generate for the given prompt. It should follow markdown syntax and must not contain the issues or limitations you highlighted in the evaluation form for the responses.You can read more about how to structure the ideal response in the **Presentation Guidelines**.

Important Note: Please remember that **we cannot use the response generated by the models as the Ideal Response**; doing so defeats the purpose of having a separate field for it. You may use parts of the generated response but not the entire response as is. For example, if the model's generated response lacks comments, you can add them or improve the explanations when incorporating it into the Ideal Response. **It is prohibited to use the response generated by the models in its original form**.

Presentation Guidelines

Structure

1. Briefly introduce the **purpose** of the response and the **context** it addresses.
2. Organize the response into **clear, logical sections**.
3. Summarize the **key points** made in the response and any **conclusions** derived.

Code

1. Include relevant code snippets when discussing programming concepts or demonstrating solutions.
2. Code should be **well-formatted** and highlighted using backticks or triple backticks in the markdown. Ensure the code is **readable and well-commented** to explain its functionality.
3. Please use the appropriate programming language tag before adding a code snippet. For Python, use the **python** tag and enclose your code with triple backticks (```python).
   Example:
   ```python
   # Here is your code
   ```

Ideal Response Code

- Follow PEP8 Guidelines.
- Precede logical code chunks with an empty line and inline comments explaining what the chunk does (Optimized for beginners)
- Use docstrings if required; you can skip it if not required.
- Correct, Optimal, Rich & Readable Code
- Optimize for ease of usability
  - Parametrized functions
  - Call the function with a couple of critical path test cases

Markdown

Considering we want to (1) provide correct information with the right level of detail **(2) that is formatted well for the user,** please do the following to ensure **(2)** is achieved:

1. Use appropriate spacing between sections and paragraphs for better readability.
2. Use headings and subheadings to organize content into sections, as needed.
3. Use **bold** for emphasis, especially for key terms or important points, as needed.
4. **Use bulleted and numbered lists** to break down information into manageable chunks (see below).
5. Integrate **images** into the conversation by linking image URLs (see below).
6. **Use LaTeX to write equations** (see below).

a. User’s turns may or may not use LaTeX.
b. AI turns **must** use LaTeX — formatting is key for AI responses.

Remember: The ultimate goal is to provide accurate, consumable, engaging information. Follow these guidelines, but let us know your feedback on how to improve your AI responses.

Tables (use Markdown)	Use Markdown for formatting tables. You can use an online generator to convert tables into Markdown. | Syntax | Description | | ----------- | ----------- | | Header | Title | | Paragraph | Text |
Links (use Markdown)	Use the Markdown syntax to format links. My favorite search engine is [Duck Duck Go](https://duckduckgo.com).
Equations (use LaTeX)	User’s turns may or may not use LaTeX. AI turns **must** use LaTeX — formatting is key for AI responses. The question below is about cells. `The EMF of an alkaline hydrogen-oxygen fuel cell is +1.23 V box The standard electrode potential for one of the electrodes in the alkaline hydrogen-oxygen fuel cell is $2H_20(l) + 2e^- \rightarrow 2OH^-(aq) + H_2(g)\quad E^\degree = -0.83 V$`

Lists	Use the Markdown syntax for bullet formatting.
	
	`- First item of the list`
	`- Second item of the list`
	`    - First sub-item of the list`
	`    - Second sub-item of the list`
	`- Third item of the list`

Images (use Markdown)	Integrate images as URLs into the conversation. Use any free image hosting sites to get an image URL. Use the Markdown syntax to add links for any images. `See the [![image of a molecule](/assets/images/shiprock.jpg "Caption for the image")](https://www.flickr.com) here.`

Behavioral

Core Objectives

- **Assist Users:** The assistant response should always focus on helping users achieve their goals effectively and efficiently.
- **Ethical Responsibilities:** Responses should prioritize the well-being of individuals and society, avoiding harmful content or advice.
- **Professionalism:** Interactions should reflect positivity, maintaining social norms, legality, and professionalism.

Response Guidelines

- **Accuracy & Honesty:** Provide precise information and openly acknowledge any uncertainties or limitations.
- **Respect & Neutrality:** Maintain neutrality, avoid biases, and respect user privacy and intellectual property.

Default Behaviors

- **Safety First:** Never provide harmful, illegal, or unethical content.
- **Politeness and Respect:** Ensure the tone is polite, understanding, and non-confrontational.
- **Transparency:** Be transparent about capabilities and limitations, ensuring users have clear expectations.

For more detailed information & instructions, please refer to the OpenAI Model Spec Document to understand desired behaviors from an ideal model response.

Good Example Tasks

- **Example 1 (4 Model Responses):**
  - **Task:** Link
  - **Comments:**
    - **Prompt Design:** As you can see, the prompt adheres to the metadata constraints, taking into account which constraints should be strictly followed and which can be loosely followed.
    - **Response Evaluation:** Evaluation fields are appropriately filled. The **Rating Justification** includes a clear explanation of the issues in a neatly formatted manner, while **Comments for Reference** highlight where the reviewer of the task can find the output.
    - **Ideal Response:** The ideal response is comprehensive, covering all the requirements of the prompt, and is neatly formatted.

Terms

Number of Turns

This term refers to the total number of exchanges in a conversation where the user poses a question or request. For example:

- 1 Turn = User, Assistant
- 2 Turns = User, Assistant, User, Assistant
- 3 Turns = User, Assistant, User, Assistant, User, Assistant… etc

Note that every user prompt must have an assistant response replying to it to be considered complete.

UI Guide

https://labeling-ms.turing.com/ should always be your home. It will be playing the role of the sheets for task management, review submission, and overall tracking for our work… as well as a gateway that takes you to other tools to do the actual job when needed.

Workspace Tabs

- `Unclaimed`: Shows available tasks you can claim.
- `In Progress`: Displays tasks you are currently working on (up to 3 allowed at a time).
- `Completed`: Lists tasks you've completed.
- `Reviewed`: Indicates your tasks that have been reviewed (For now, you will receive a notification for your reviewed tasks).
- `Improper`: Shows improper tasks.

Important Guidelines

Microsoft - Failure Categories LookUp
Please refer to this guide as a reference to understand the various categories of potential failure types we should be mindful of while evaluating responses. However, please be cautious and avoid limiting yourself solely to the examples provided in the guide.

Responsible Use of LLMs
The purpose of this guide is to offer trainers a comprehensive understanding of when it is appropriate to utilize Language Model Models (LLMs) at different stages of your workflow. Furthermore, it will also highlight instances when it is not recommended to rely on LLMs.

Microsoft - Time Management Evaluation tasks
Optimizing Time for RLHF Tasks and Planning Your Day

Suggestions for Multi-Turn Prompt Creation

Here are some suggestions for prompt creation to help you understand how to use the metadata and work on multi-turn tasks. If you have any tips that have helped you and could benefit others, please feel free to share them, and we'll include them in the document.

FAQs

Microsoft Coding FAQs