Attempter Instructions: HardCodeQA

Please review the following recordings:

- Onboarding Session
- Quality Review Session

Overview

The goal of this project is to **expose model weaknesses** by crafting questions that cannot be solved correctly (get a model failure)..

Key Requirements

- **Questions should be difficult and niche**, requiring deep reasoning, inference, and synthesis.
- **Must not exist online** (cannot be copied or easily found).
- **Answers must be concise** (single value or phrase, max 3 words).
- **Provide two additional confounding incorrect answers**.
- **Use real-world coding scenarios** rather than artificial or toy examples.
- **Follow licensing restrictions** when using external sources.

Six Key Coding Areas

Your questions should fit into one of these six categories. Below is a **detailed explanation of each**, including examples of question types that would challenge the model.

- Some example challenges are included below, but these **should not be repeated!** Please come up with your own ideas for each prompt.
- Questions should resemble stack overview questions like here: https://stackoverflow.com/questions
    - The goal is to make new questions, where the answer or question cannot be found on the internet. Rephrasing, adding complexity, and introducing novel twists are all encouraged to adapt questions from source materials.
    - Direct copying and pasting is not allowed, but you can use parts of an existing question and creatively introduce new elements.

1. Frontend Web & Mobile Development

- **What it Covers**: Client-side programming for web and mobile applications.

- **Languages & Technologies**: JavaScript, TypeScript, HTML, CSS, React, Vue, Angular, Flutter, Swift, Kotlin.
- **Example Challenges**:
    - A tricky edge case in CSS flexbox layout behavior.
    - An obscure bug in JavaScript event bubbling.
    - A security vulnerability in React state management.

2. Backend & DevOps

- **What it Covers**: Server-side logic, API development, infrastructure, deployment, and CI/CD pipelines.
- **Languages & Technologies**: Python (Flask, Django), Node.js, Java (Spring Boot), Docker, Kubernetes, AWS, Terraform.
- **Example Challenges**:
    - A complex race condition in multi-threaded server code.
    - A subtle bug in Kubernetes YAML configurations.
    - An edge case where an API request returns an unexpected 403 error due to IAM role misconfigurations.

3. Data Science & Engineering

- **What it Covers**: Data processing, machine learning, statistical analysis, and large-scale data workflows.
- **Languages & Technologies**: Python (Pandas, NumPy, Scikit-learn), R, SQL, Spark, TensorFlow, PyTorch.
- **Example Challenges**:
    - A misleading statistical assumption in a machine learning model.
    - An edge case where a data pipeline fails under specific data distributions.
    - A computational complexity issue in a NumPy vectorized operation.

4. Database Administration & Development

- **What it Covers**: Database design, query optimization, indexing, transactions, and stored procedures.
- **Languages & Technologies**: SQL (PostgreSQL, MySQL, SQL Server), NoSQL (MongoDB, Firebase).
- **Example Challenges**:
    - A query that performs differently under different indexing strategies.
    - A race condition in a distributed transaction across multiple databases.
    - A logical flaw in a recursive SQL query.

5. Desktop & CLI Application Development

- **What it Covers**: General software development for desktop applications and command-line tools.
- **Languages & Technologies**: Java, C#, C++, Python (PyQt, Tkinter), Rust.
- **Example Challenges**:
    - A GUI framework-specific bug that only occurs on a certain OS version.
    - A memory leak in a long-running CLI application.
    - A misunderstood behavior in Python's subprocess module.

6. Systems Programming & Scripting

- **What it Covers**: Low-level programming, operating system interactions, and automation.
- **Languages & Technologies**: C, Rust, Go, Bash, PowerShell.
- **Example Challenges**:
    - A misunderstood pointer arithmetic issue in C.
    - A hidden inefficiency in a Bash script processing large logs.
    - A security vulnerability in a PowerShell automation script.

Goals

What We Want

- **Difficult, niche coding questions** requiring deep understanding.
- **Challenging problems that expose LLM limitations** in reasoning and fact retrieval.
- **Grounded in real-world scenarios**, not theoretical or academic-only questions.

Restrictions

- **Question must not be found online** (question and answer must not exist verbatim or near-verbatim).
- **No direct copying** (questions should be unique, even if inspired by existing materials).
- **Use only commercially licensed sources** for supporting materials (e.g., code snippets, datasets).

Task Overview

1. **Question (prompt) Creation**

    - Find **inspiration** from books, academic papers, technical documentation, forums, etc.
        - **Or come up with your own challenges from scratch!**

    - Adapt existing material by making it harder, more nuanced, or changing the context.
    - Ensure the **question requires reasoning**, not just fact recall.
2. **Model Testing**
    - Run the question through the model
    - The model **should fail** (answer incorrectly or be unable to answer).
3. **Answer & Justification**

    - Provide **a single correct answer** (max 3 words).
    - Add **two confounding incorrect answers**.
    - Justify why the correct answer is correct.

Guidelines

1. Adapt and Build, Don't Copy

- **You can use**: Publicly available information, books, technical docs.
- **You cannot**: Copy exact wording from any source.
- **How to adapt**:
    - Make questions **harder** by adding complexity.
    - Change the **context** while keeping the core difficulty.
    - Introduce **misleading but plausible answer choices**.

2. Question Design

Good questions require:

- **Inference**: Using indirect clues.
- **Synthesis**: Combining multiple sources.
- **Complex reasoning**: Going beyond basic knowledge.
- **Real-world applicability**: Avoiding toy problems.

3. Answer Guidelines

- **Single short answer** (max 3 words).
- **Two confounding incorrect answers**.
- **Clear and objective truth** (agreed upon by experts).
- **Not yes/no or multiple-choice** (avoid limiting possible answers).

4. Using External Resources

- **Allowed**: Images, code snippets, data.

- **License must allow commercial use** (check before using).
- **Cite the source URL** but do **not** include it in the question itself.

Example Prompt 1:

Note that this is an extremely simplified answer to illustrate the process:
I recently wrote the following Python script, how many times will the while loop execute?

```python
max_iterations = 20
curr_iteration = 1

while curr_iteration <= max_iterations:
      curr_iteration += 1
      if curr_iteration % 10 == 9:
            break
```

Model Response:

Let's trace the execution of your `while` loop:

1. Initialization:
    - `max_iterations` is set to 20.
    - `curr_iteration` is set to 1.
2. First Iteration:
    - The condition `curr_iteration <= max_iterations` (1 <= 20) is True.
    - `curr_iteration` becomes `curr_iteration + 1`, so `curr_iteration` is now 2.
    - The condition `curr_iteration % 10 == 9` (2 % 10 == 9) is False. The `break` statement is not executed.
3. Second Iteration:
    - The condition `curr_iteration <= max_iterations` (2 <= 20) is True.
    - `curr_iteration` becomes `curr_iteration + 1`, so `curr_iteration` is now 3.
    - The condition `curr_iteration % 10 == 9` (3 % 10 == 9) is False. The `break` statement is not executed.

4. ... (The loop continues in this manner)

5. Twenty First Iteration:

    - The condition `curr_iteration <= max_iterations` (20 <= 20) is True (just before the 21st iteration).
    - `curr_iteration` becomes `curr_iteration + 1`, so `curr_iteration` is now 21.

Therefore, the `while` loop will execute 21 times.

Correct Answer:

9

Confounding Answers:

20

21

Note that the correct answer must be three words or less, but the model's total response can be longer as it provides justification.

Justification:

The correct answer is 9 as the if statement in the while loop will trigger once the curr_iteration = 9 as the condition 9 % 10 = 9. This will then trigger the break statement which will exit the loop.

The model's answer of 21 is incorrect as it does not consider the if clause and its subsequent break. 21 is the correct answer if the while loop runs without breaking.

The answer 20 is also incorrect and represents a misunderstanding of the behavior of the while loop. Once curr_iteration is set to 20, the while loop will trigger once more and curr_iteration will be set to 21. 21 >= 20 which will cause the loop to exit before beginning another iteration.

Example Prompt 2: (do not copy!)

I recently wrote a script which extracts CSV files from a ZIP file, and processes the data using the pandas module in Python. Parts of the code are shown below:

```python
import pandas as pd
from zipfile import ZipFile
```

```
file_list = [
   '/tmp/archived.zip',
   '/tmp/archived_2.zip',
   '/tmp/log.txt'
]

zip_files  = [files for file in file_list if
file.endswith(".zip")]

for file in zip_files:
    with open(f"{file}", "rb") as obj:
        with ZipFile(obj, "r") as zip_ref:
            for csv_files in zip_ref.namelist():
                if csv_files.endswith(".csv"):
                    with zip_ref.open(csv_files, "r") as
file_csv:
                        header = pd.read_csv(file_csv,
encoding='ISO-8859-1').columns.tolist()
                        file_csv.seek(0)
```

The code is currently incomplete. Since the CSV files are so large, I can't load an entire CSV file into memory at once. That's why we use compressed ZIP files. We would like to save the header information, and save the total row count of each CSV file to a variable called "counter".. What is the minimum number of file pointer resets needed to extract the header, and get a total row count of a CSV file within a ZIP file assuming we don't change the current code logic?

Review Criteria

1. **Prompt matches the assigned category** ✅
   The question fits within the designated coding category (Frontend, Backend, Data Science, etc.).
   - No category mismatches (e.g., a database question shouldn't be in the Systems Programming category).
   - The prompt must have only 1 correct answer, and the answer must be 3 words or less in length!
2. **Model comes to an incorrect final answer** ✅
   - The LLM **fails to provide the correct answer** or **gives an incorrect response**.

3. **Two confounding incorrect answers are included** ✅
    - The **incorrect answers are plausible** but **wrong**.
    - They **could realistically be chosen by the model** but lead to failure.
4. **Correct answer is accurate and well explained** ✅
    - The **correct answer is precise and objective** (max 3 words).

Additional Criteria:

5. **Prompt is original and non-trivial**
   The question **is not copied** from online sources.
    - It **isn't too basic or widely known** (should require reasoning, not just fact recall).
6. **Question is challenging but solvable**
    - The question **is difficult enough to trip up LLMs** but **can still be answered correctly by a human expert**.
7. **No ambiguity in phrasing**
   The question is **clearly worded**, avoiding multiple valid interpretations.
    - No unnecessary complexity in language or structure. Model just needs to fail
8. **Code (if included) is properly formatted**
    - Any code snippet **is syntactically correct and properly formatted** for readability.
    - Code aligns with the question's intent and **doesn’t contain accidental hints** that make it too easy.
9. **No unnecessary extra information**
    - The question **stays focused on testing model weaknesses** and does not include **irrelevant details**.
    - **Avoids filler text** or excessive context unless necessary for difficulty.
10. **Prompt does not require external references**

- The model should be able to **answer based on included knowledge alone** (not require an external document, dataset, or obscure research paper that’s not provided).

Frequently Asked Questions

Question

- Can I use any other forms of response limiting such as:
    - “I am in a rush, please answer in a concise fashion”
    - “Please just give me the answer, my team is working under a deadline”

Answer

- No, any form of response limitation is not permitted. The response can have extensive justification as long as the answer itself is expressed in three words or less within the response.

Question

- Does the entire model response need to be three words or less?

Answer

- No, only the portion of the response that corresponds to the answer must be three words or less.

Question

- If the model gives the correct response, with incorrect reasoning, is that a valid submission?

Answer

- No, the model must give an incorrect answer