

limitations of current human priors. Specifically, we first engage human annotators to convert the reasoning trace into a more natural, human conversational style. The modified data pairs are then used as examples to prompt an LLM to rewrite additional data in a similar style. All LLM-generated outputs subsequently undergo a second round of human verification to ensure quality and consistency.

Listing 1 | Prompt for producing a human-readable solution.

```
## Question
{question}

## Thought process
{thought_process}

---
Based on the above thought process, provide a clear, easy-to-follow, and well-formatted
solution to the question. Use the same language as the question.

The solution must strictly follow these requirements:
- Stay faithful and consistent with the given thought process. Do not add new reasoning
  steps or conclusions not shown in the original.
- Show key steps leading to final answer(s) in clear, well-formatted LaTeX.
- Use \boxed{} for final answer(s).
- Be clean and concise. Avoid colloquial language. Do not use phrases like "thought
  process" in the solution.

Your response should start with the solution right away, and do not include anything
else. Your task is solely to write the solution based on the provided thought
process. Do not try to solve the question yourself.
```

Specifically, we begin by gathering thousands of high-quality, diverse reasoning prompts. For each prompt, we generate multiple reasoning trajectories using DeepSeek-R1-Zero with a relatively high temperature of 1.0. Next, we filter these generations to retain only those with correct final answers and a readable format. For mathematical outputs, we use `sympy` (<https://www.sympy.org/>) for parsing and expression comparison; and for formatting, we apply rules such as repetition detection and language-mixing filtering. Finally, we prompt DeepSeek-V3 to refine both the reasoning and the summaries to ensure proper formatting and a human-friendly expression. In particular, to resolve language mixing, we instruct DeepSeek-V3 to “Translate the thinking process to the same language as the question.” Since DeepSeek-R1-Zero’s summary only provided the final answer, we use the summary prompt in Listing 1 to produce a concise, human-readable solution that outlines both the reasoning steps and the final result.

For code data, we collect a large set of competitive programming problems. In detail, We have compiled an extensive collection of competitive programming problems from multiple online judge (OJ) platforms, specifically 5151 problems from Codeforces and 2504 problems from AtCoder. Since the original test cases are not publicly available from these platforms, we developed a methodology to create reliable test cases for each problem.

Our approach involves using DeepSeek-V2.5 to generate candidate test cases, followed by a rigorous validation process. Specifically, we prompted DeepSeek-V2.5 to write Python programs that generate test cases tailored to each problem’s requirements as shown in Listing 2.

After obtaining numerous candidate test cases, we implemented a two-phase filtering procedure. First, we used correct submissions to eliminate invalid test cases that produced incorrect outputs. Then, we strategically selected subsets of test cases that successfully identified

flaws in incorrect submissions. This process ensured our final test cases properly differentiated between correct and incorrect solutions for each problem.

Listing 2 | Prompt for generating test cases for code problem.

```
<USER>
Question:
You are given a string word and an integer k.

A substring s of word is complete if:

Each character in s occurs exactly k times.
The difference between two adjacent characters is at most 2. That is, for any two
adjacent characters c1 and c2 in s, the absolute difference in their positions in
the alphabet is at most 2.
Return the number of complete substrings of word.

A substring is a non-empty contiguous sequence of characters in a string.

**Input**
The first line contains a single integer t the number of test cases.
The first line of each test case contain one string word.
The second line of each test case contain one integer k.
It is guaranteed that word consists only of lowercase English letters, and the sum of |
word| over all test cases does not exceed 10^5.

**Output**
For each test case, print one integer the number of complete substrings of word.

**Example**
Input
```
2
igigee
2
aaabbbccc
3
```

Output
```
3
6
```

Construct some random input generators to generate diverse and adversarial inputs,
which are large enough to make incorrect codes exceed the time limit.
</USER>

<ASSISTANT>
First construct a random input generator for large data with 10 testcases of the same
length.

```python
import random
import string

def generate_random_string(l_min, l_max):
 length = random.randint(l_min, l_max)
 seq = ''.join(random.choices(string.ascii_lowercase, k=length))
 k = random.randint(1, length)

```

```

 return f"{{seq}}\n{{k}}"

t = 10 # the number of test cases
testcase = [generate_random_string(10000, 10000) for _ in range(t)]

print(t)
print("\n".join(testcase))
"""

First construct a random input generator for large data with 10 testcases of different
length.

'''python
import random
import string

def generate_random_string(l_min, l_max):
 length = random.randint(l_min, l_max)
 seq = ''.join(random.choices(string.ascii_lowercase, k=length))
 k = random.randint(1, length)
 return f"{{seq}}\n{{k}}", len(seq)

t = 10 # the number of test cases
testcase = []
cum_length = 0
for i in range(t):
 # each test case should have length >= 1
 cur testcase, cur_length = generate_random_string(1, 100000 - cum_length - (t -
 i))
 testcase.append(cur testcase)
 cum_length += cur_length

print(t)
print("\n".join(testcase))
"""

Construct a random input generator for large data with only one test case.

'''python
import random
import string

def generate_random_string(l_min, l_max):
 ...

t = 1 # the number of test cases
testcase = [generate_random_string(90000, 100000) for _ in range(t)]

print(t)
print("\n".join(testcase))
"""

Construct an input generator for large adversarial inputs of a string of a repeated
single character.

'''python
import random
import string

def generate_single_repeated_char(l_min, l_max):
 length = random.randint(l_min, l_max)

```

```

char = random.choice(string.ascii_lowercase)
seq = char * length
k = random.randint(1, length)
return f"{{seq}}\n{{k}}"

t = 1 # the number of test cases
testcase = [generate_single_repeated_char(90000, 100000) for _ in range(t)]

print(t)
print("\n".join(testcase))
"""

Construct an input generator for large adversarial inputs of a string of two
alternating characters.

'''python
import random
import string

def generate_alternating_chars(l_min, l_max):
 length = random.randint(l_min, l_max)
 char1, char2 = random.sample(string.ascii_lowercase, 2)
 seq = ''.join(char1 if i % 2 == 0 else char2 for i in range(length))
 k = random.randint(1, length)
 return f"{{seq}}\n{{k}}"

t = 1 # the number of test cases
testcase = [generate_alternating_chars(90000, 100000) for _ in range(t)]

print(t)
print("\n".join(testcase))
"""

Construct an input generator for large adversarial inputs of a string of sequential
characters from alphabet.

'''python
...
'''

</ASSISTANT>

Question:
Question Description

Construct some random input generators to generate large, diverse and adversarial
inputs, which are large enough to testing time complexity and to make incorrect
codes exceed the time limit.

Use the format used in the above example by returning several input generators in
different code blocks. Each of these generators prints EXACTLY ONE input directly
into stdout.

```

In addition, we employ few-shot prompting for DeepSeek-V3 to generate responses to simple math problems, such as "1 + 1 = ?", ensuring that the outputs remain concise and appropriately structured. We provide the prompt for a simple math problem in Listing 3.

Listing 3 | Prompt for generating chain-of-thought for simple math problem.

```

Question
How much is 5+4?

```

```

Response
<think>

I need to add the numbers 5 and 4. Starting with 5, if I add 4 to it, the total will be
9. Therefore, the sum of 5 and 4 is 9.

</think>

Solution:

We are asked to calculate the sum of 5 and 4.

Step 1: Start with the number 5.

Step 2: Add 4 to it.

\[
5 + 4 = 9
\]

Final Answer: \(\boxed{9}\)

Question
what is 1 plus 2

Response
<think>

I need to determine the sum of 1 and 2.

Adding these two numbers together, 1 plus 2 equals 3.

Therefore, the answer is 3.

</think>

Sure! Let's solve the problem step by step.

Problem: What is $(1 + 2)$?

Solution:

To find the sum of 1 and 2, simply add the two numbers together:

\[
1 + 2 = 3
\]

Answer:

\(\boxed{3}\)

Question
{question}

Response

```

---