CS598JBR MP Progress Report: Team-25

Chih-Fu Lai, Heyang Li, Jinsong Yuan, Yihang Jiao
University of Illinois Urbana-Champaign
[chihful2,heyangl4,jinsong3,yihangj3]@illinois.edu

1 Team Information

Link to the GitHub Repository:
https://github.com/lycoris106/CS598JBR-Team-25
Link to the Google Colab Workspace:
https://colab.research.google.com/drive/1t-d6AL-alJ5accoeCVUS9A4YbfaPTn0S?usp=sharing listings

2 MP1

2.1

- n = total number of samples generated for a problem
- c = number of correct samples (passes all tests)
- k = number of samples considered

$$pass@k = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}, n \geq k$$

This calculates the probability that at least one of the top-k generated code samples passes all test cases.

2.2

- base-processed pass@1: 0.4
- instruct-processed pass@1: 0.6

2.3

See the attached table

2.4

Issues and resolutions:

- Remove duplicate definition: The models often output multiple versions of the function (in-place version, etc.).
  For example, following is part of the original base model's response of HumanEval/85:

```
def add(lst):
    """Given a non-empty list of integers lst. add
      the even elements that are at odd indices..


    Examples:
        add([4, 2, 6, 7]) ==> 2
    """
    return sum(lst[1::2])


def add(lst):
    """Given a non-empty list of integers lst. add
      the even elements that are at odd indices..


    Examples:
        add([4, 2, 6, 7]) ==> 2
    """
    return sum(lst[1::2])

def add(lst):
    """Given a non-empty list of integers lst. add
      the even elements that are at odd indices..


    Examples:
        add([4, 2, 6, 7]) ==> 2
    """
    return sum(lst[1::2])
```

The response is composed of multiple "add" functions while the last one is incomplete. This is resolved by splitting all the functions and extracting only the first function definition which matches the function header in the prompt when post-processing:

```
def split_multiple_functions(code):
    triple_string_pattern = r'("""[\s\S]*?"""|\'\'\'[\s\S]*?\'\'\')'
    placeholders = {}

    def replacer(match):
        key = f"__PLACEHOLDER_{len(placeholders)}__"
        placeholders[key] = match.group(0)
        return key

    protected_code = re.sub(triple_string_pattern, replacer, code)
    raw_blocks = protected_code.split("\n\n\n")

    restored_blocks = [
        re.sub("|".join(map(re.escape, placeholders.keys())),
               lambda m: placeholders[m.group(0)], block)
        for block in raw_blocks
    ]

    return [b.strip() for b in restored_blocks if b.strip()]

def filter_response(prompt, response):
    """
    Filters out any extraneous text before the first function definition.
    """
```

Table 1: Pairwise comparison of pass/fail results for each of the 20 programs under two different settings, before and after post-processing.

Problem ID	Base Results	Base Results Processed	Instruct Results	Instruct Results Processed
HumanEval/96	✗	✗	✓	✓
HumanEval/117	✗	✗	✗	✓
HumanEval/85	✗	✗	✗	✓
HumanEval/148	✗	✓	✓	✓
HumanEval/64	✗	✗	✗	✗
HumanEval/149	✓	✓	✗	✗
HumanEval/13	✗	✗	✗	✓
HumanEval/95	✗	✗	✗	✗
HumanEval/126	✗	✗	✗	✗
HumanEval/123	✗	✗	✗	✗
HumanEval/162	✗	✗	✗	✗
HumanEval/41	✗	✗	✗	✓
HumanEval/21	✗	✓	✗	✓
HumanEval/61	✗	✗	✗	✓
HumanEval/97	✗	✓	✓	✓
HumanEval/27	✓	✓	✗	✓
HumanEval/1	✗	✗	✓	✓
HumanEval/68	✓	✓	✗	✗
HumanEval/150	✗	✓	✗	✓
HumanEval/112	✗	✓	✗	✗

```
    # find the first function in prompt using regex
    import re
    match = re.search(r'def\s+\w+\s*\(.*?\)\s*(->\s*.+?)?\s*:', prompt)
    if not match:
        return None

    clean_prompt = match.group(0)
    functions = split_multiple_functions(response)
    for func in functions:
        # if the func string contains the clean_prompt, return it
        if clean_prompt in func:
            return func
    return None

```

- Remove redundant statements: The models often generate redundant, sometimes syntatically-wrong testing statements.
  For example, the original instruct model's response of HumanEval/117 contains some test statement at the end, including a call to an unknown "select" function:

```
def select_words(s, n):
    """Given a string s and a natural number n, you have been tasked to implement
    a function that returns a list of all words from string s that contain exactly
    n consonants, in order these words appear in the string s.
    If the string s is empty then the function should return an empty list.
    Note: you may assume the input string contains only letters and spaces.
    Examples:
    select_words("Mary had a little lamb", 4) ==> ["little"]
    select_words("Mary had a little lamb", 3) ==> ["Mary", "lamb"]
    select_words("simple white space", 2) ==> []
    select_words("Hello world", 4) ==> ["world"]
    select_words("Uncle sam", 3) ==> ["Uncle"]
    """
    vowels = "aeiou"
    words = s.split()
    result = []
    for word in words:
        consonants = 0
        for letter in word:
            if letter.lower() not in vowels:
                consonants += 1
        if consonants == n:
            result.append(word)
    return result


print(select_words("Mary had a little lamb", 4))  # ["little"]
print(select_words("Mary had a little lamb", 3))  # ["Mary", "lamb"]
print(select("simple white space", 2))  # []
print(select_words("Hello world", 4))  # ["world"]
print(select_words("Uncle sam", 3))  # ["Uncle"]

```

  We naturally remove these statements with our function extraction shown previously.
- Completion-only: The models always return several full functions instead of completions which are required by HumanEval (like the example HumanEval/117 above). Our

post-process converts the full function output to the completion by deleting function definition and comments.

```
def get_completion(prompt, response):
    response = filter_response(prompt, response)
    if response:
        # remove the definition line from the response
        lines = response.split("\n")
        if len(lines) > 1:
            lines = lines[1:]

        # remove the comments/docstring if any (including the content)
        while lines and (lines[0].strip().startswith('"""') or lines[0].strip().startswith("'''")):
            lines.pop(0)
            while lines and not (lines[0].strip().endswith('"""') or lines[0].strip().endswith("'''")):
                lines.pop(0)
            if lines:
                lines.pop(0)  # remove the ending docstring line
        while lines and lines[0].strip().startswith("#"):
            lines.pop(0)

        return "\n".join(lines)
    return None

```

The base model's output for HumanEval/85 after post-processing:

```
    return sum(lst[1::2])
```

The instruct model's output for HumanEval/117 after post-processing:

```
    vowels = "aeiou"
    words = s.split()
    result = []
    for word in words:
        consonants = 0
        for letter in word:
            if letter.lower() not in vowels:
                consonants += 1
        if consonants == n:
            result.append(word)
    return result

```

- Wrong interpretation: The models sometimes fail to understand more complex function descriptions and generate wrong solutions. This is difficult to resolve with post-processing without human inspection.
- Edge cases: The models sometimes do not consider all edge cases of inputs (even when explicitly stated in prompts). This is difficult to resolve with post-processing without human inspection.

3 MP2

TBA

4 MP3

TBA