Program Synthesis Evaluation using LLM

1. Introduction

In this assignment, I evaluate the effectiveness of Gemini-2.0-flash and GPT-3.5-Turbo, on the program synthesis benchmark consisting of 56 problems. I explore prompt engineering strategies, apply postprocessing to improve executability, and calculate Pass@5 metrics to assess model performance.

2. Prompt Engineering

I edited the prompt for GPT-3.5-Turbo to support Markdown JSON formatting.

```
Respond should only with a string in the following JSON format, using Markdown
```

3. Postprocessing

Postprocessing is critical to increase code compilation and evaluation success. I implemented the following:

- **Extraction of code blocks**: Removed natural language prefixes/suffixes like `"Sure, here's the code"`.
- **Filtered reasoning**: Any explanation text included in the LLM response was removed to retain pure code.
- **Normalization**: Ensured all 5 attempts per problem followed the format expected by `evaluator.py`.
- **Fixing JSON Escapes**: I sanitized all `target code` fields by replacing invalid escape sequences (e.g., unescaped newlines, quotes, or backslashes) to ensure that the results conform to valid JSON formatting and can be parsed reliably into `.jsonl`.

4. Evaluation Method

I used the provided `evaluator.py` to execute the synthesized code against hidden test cases. Each problem has 5 generated solutions. If any one of them passes all test cases, it counts as a success (Pass@5).

5. Results

Pass@5 Summary

Model	Easy Passed	Difficult Passed	Total	Total Without HTTP Error	Total Without JSON Error
Gemini-2.0-flash	23	8	56	55	55
GPT-3.5-Turbo	2	3	56	56	24

We can see that half of the target code generated by GPT-3.5-turbo have JSON formatting errors. I found that some of the JSON formatting errors are like wrongly use `\\n` and `\n`. And some of them are like `\\"".join()`, which missing `\\` in front of the secend `"`. And for Gemini-2.0-flash, I found that sometimes Gemini-2.0-flash have http errors, which is

```
ERROR - Failed to generate text: 503 UNAVAILABLE. {'error': {'code': 503, 'mess
```

6. Case Studies

Successful Case: Problem `6cfd3b0a403212ec68bac1667bce9ef1`

Problem: Berland National Library has recently been built in the capital of Berland. In addition, in the library you can take any of the collected works of Berland leaders, the library has a reading room.Today was the pilot launch of an automated reading room visitors' accounting system! The scanner of the system is installed at the entrance to the reading room. It records the events of the form "reader entered room", "reader left room". Every reader is assigned a registration number during the registration procedure at the library — it's a unique integer from 1 to 106. Thus, the system logs events of two forms: "+ ri" — the reader with registration number ri entered the room; "- ri" — the reader with registration number ri left the room. The first launch of the system was a success, it functioned for some period of time, and, at the time of its launch and at the time of its shutdown, the reading room may already have visitors.Significant funds of the budget of Berland have been spent on the design and installation of the system. Therefore, some of the citizens of the capital now demand to explain the need for this system and the benefits that its implementation will bring. Now, the developers of the system need to urgently come up with reasons for its existence.Help the system developers to find the minimum possible capacity of the reading room (in visitors) using the log of the system available to you.

Input Specification: The first line contains a positive integer n (1 ≤ n ≤ 100) — the number of records in the system log. Next follow n events from the system journal in the order in which the were made. Each event was written on a single line and looks as "+ ri" or "- ri", where ri is an integer from 1 to 106, the registration number of the visitor (that is, distinct visitors always have distinct registration numbers). It is guaranteed that the log is not contradictory, that is, for every visitor the

types of any of his two consecutive events are distinct. Before starting the system, and after stopping the room may possibly contain visitors.

Output Specification: Print a single integer — the minimum possible capacity of the reading room.

Notes: In the first sample test, the system log will ensure that at some point in the reading room were visitors with registration numbers 1, 1200 and 12001. More people were not in the room at the same time based on the log. Therefore, the answer to the test is 3.

Sample Inputs: ['6\n+ 12001\n- 12001\n- 1\n- 1200\n+ 1\n+ 7', '2\n- 1\n- 2', '2\n+ 1\n- 1']

Sample Outputs: ['3', '2', '1']

Generated Code (Pass):

```
def solve():
    n = int(input())
    events = []
    for _ in range(n):
    events.append(input().split())

    present = set()
    max_capacity = 0
    current_capacity = 0

    for event in events:op = event[0]
        reader_id = int(event[1])

        if op == '+':
            if reader_id not in present:
                present.add(reader_id)
                current_capacity += 1
                max_capacity = max(max_capacity, current_capacity)
            else:
                if reader_id in present:
                    present.remove(reader_id)
                    current_capacity -= 1
        else:
            max_capacity += 1

    print(max_capacity)

solve()
```

Failed Case: Problem `0937a7e2f912fc094cc4275fd47cd457`

Problem: Petya loves football very much, especially when his parents aren't home. Each morning he comes to the yard, gathers his friends and they play all day. From time to time they have a break to

have some food or do some chores (for example, water the flowers).The key in football is to divide into teams fairly before the game begins. There are n boys playing football in the yard (including Petya), each boy's football playing skill is expressed with a non-negative characteristic ai (the larger it is, the better the boy plays). Let's denote the number of players in the first team as x, the number of players in the second team as y, the individual numbers of boys who play for the first team as pi and the individual numbers of boys who play for the second team as qi. Division n boys into two teams is considered fair if three conditions are fulfilled: Each boy plays for exactly one team (x\u2009+\u2009y\u2009=\u2009n). The sizes of teams differ in no more than one (|x\u2009-\u2009y|\u2009\u2264\u20091). The total football playing skills for two teams differ in no more than by the value of skill the best player in the yard has. More formally: Your task is to help guys divide into two teams fairly. It is guaranteed that a fair division into two teams always exists.

Input Specification: The first line contains the only integer n (2\u2009\u2264\u2009n\u2009\u2264\u2009105) which represents the number of guys in the yard. The next line contains n positive space-separated integers, ai (1\u2009\u2264\u2009ai\u2009\u2264\u2009104), the i-th number represents the i-th boy's playing skills.

Output Specification: On the first line print an integer x — the number of boys playing for the first team. On the second line print x integers — the individual numbers of boys playing for the first team. On the third line print an integer y — the number of boys playing for the second team, on the fourth line print y integers — the individual numbers of boys playing for the second team. Don't forget that you should fulfil all three conditions: x + y = n, |x − y| ≤ 1, and the condition that limits the total skills. If there are multiple ways to solve the problem, print any of them. The boys are numbered starting from one in the order in which their skills are given in the input data. You are allowed to print individual numbers of boys who belong to the same team in any order.

Note: Let's consider the first sample test. There we send the first and the second boy to the first team and the third boy to the second team. Let's check all three conditions of a fair division. The first limitation is fulfilled (all boys play), the second limitation on the sizes of groups (|2 − 1| = 1 ≤ 1) is fulfilled, the third limitation on the difference in skills ((2 + 1) − (1) = 2 ≤ 2) is fulfilled.

Sample Inputs: ["3\n1 2 1","5\n2 3 3 1 1"]

Sample Outputs: ["2\n1 2 \n1\n3","3\n4 1 3 \n2\n5 2"]

Generated Code (Fail):

```
def solve():
    n = int(input())
    a = list(map(int, input().split()))

    players = []
    for i in range(n):
        players.append((a[i], i + 1))

    team1 = []
    team2 = []

    if n % 2 == 0:
        x = n // 2
        y = n // 2
    else:
        x = (n + 1) // 2
        y = n // 2

    for i in range(x):
        team1.append(players[i][1])

    for i in range(x, n):
        team2.append(players[i][1])

    print(x)
    print(*team1)
    print(y)
    print(*team2)

solve()
```