**PRECOG TASK**

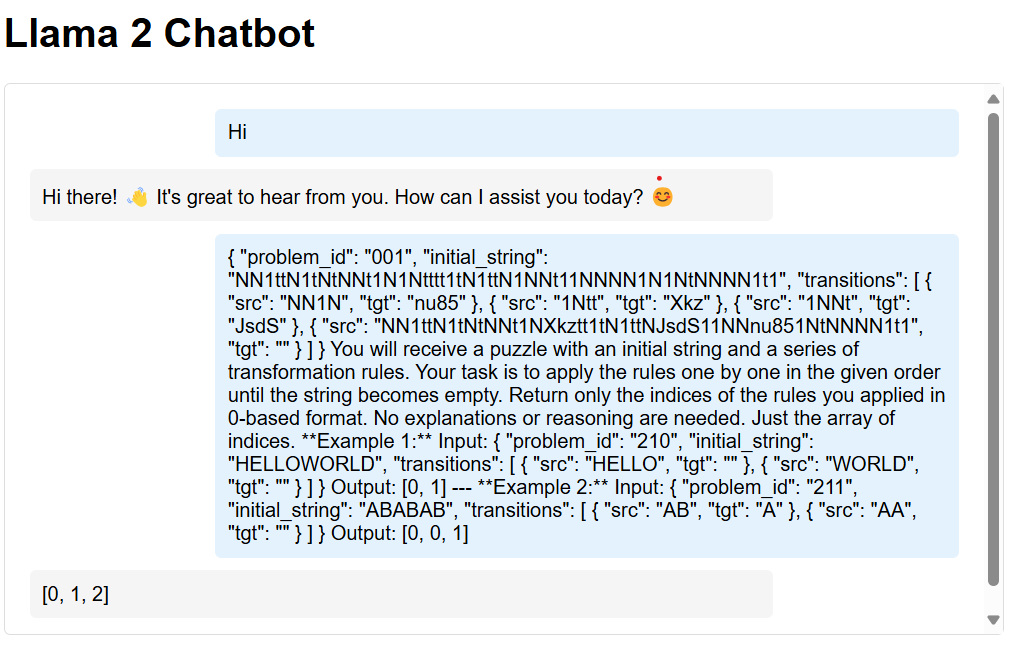
**MATH AND AI**

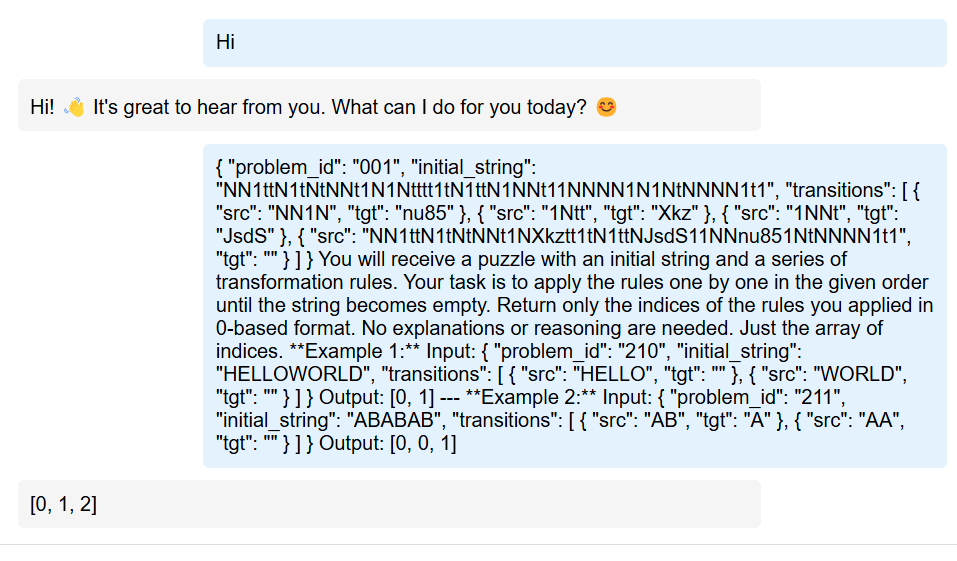
Task-1

Let us consider an empty string. We will keep applying a set of rules on it. Let S be the resultant string. Then we reverse the rules and start applying them on S. We will then reach the empty string. Therefore , we have found a valid way to generate puzzles . I used 3 different techniques. There are 3 different types of puzzles. One, where we start with an empty string, one where we start with a non-empty string . The third kind involves repeatedly generating substrings which overlap with each other which makes it tough to solve. Puzzle\_001 to Puzzle\_079 are medium, 106 to 145 are easy , 202 to 212 are hard.

Task-2

I have used 2 LLM’s ChatGPT and Gemini Flash 1.5. I also tried to run local LLMs like Llama 2 and Mistral. First image is Llama 3.2 and second is Mistral





The local LLMs were giving 0 accuracy outputs, therefore I switched to ChatGPT and Gemini. I ran zero-shot, few shot and chain of thought prompts on both these models for all my puzzles and recorded their output.

ChatGPT links:- <https://chatgpt.com/share/67aa42c8-6d88-800f-84ce-17a7ca83a7c7>

https://chatgpt.com/share/67aa4304-0f44-800f-a982-bb898cb24f94

Zero-shot prompt:- You will receive a JSON object containing an initial string and a list of transformation rules. Apply each rule in the given order until the string becomes empty. Return only an array of the 0-based indices of the applied rules in the order they were applied.

Rules can be applied multiple times. Return nothing else but the array of indices.

Few-shot prompt:- You will receive a puzzle with an initial string and a series of transformation rules. Your task is to apply the rules one by one in the given order until the string becomes empty. Return only the indices of the rules you applied in 0-based format. No explanations or reasoning are needed. Just the array of indices.

\*Example 1:\*

Input:

{

"problem\_id": "210",

"initial\_string": "HELLOWORLD",

"transitions": [

{ "src": "HELLO", "tgt": "" },

{ "src": "WORLD", "tgt": "" }

]

}

Output:

[0, 1]

---

\*Example 2:\*

Input:

{

"problem\_id": "211",

"initial\_string": "ABABAB",

"transitions": [

{ "src": "AB", "tgt": "A" },

{ "src": "AA", "tgt": "" }

]

}

Output:

[0, 0, 1]  
  
Chain of thought prompt:- You will receive a puzzle with an initial string and a series of transformation rules. Your task is to apply the rules one by one in the given order until the string becomes empty. Return only the indices of the rules you applied in 0-based format.

Think step by step about how to transform the string, but only output the indices in the end. Do not include intermediate steps in the output, only the final array of indices.

---

\*Example:\*

Input:

{

"problem\_id": "210",

"initial\_string": "HELLOWORLD",

"transitions": [

{ "src": "HELLO", "tgt": "" },

{ "src": "WORLD", "tgt": "" }

]

}

1. Start with "HELLOWORLD".

2. Apply rule 0 ("HELLO" → ""), resulting in "WORLD".

3. Apply rule 1 ("WORLD" → ""), resulting in "".

Output:

[0, 1]

I had to stick to these prompts since the LLM kept giving code to solve the puzzle instead of solving the puzzle.

Task-3

I have 2 metrics but I have implemented only one. I have implemented the accuracy. The results are available in the puzzles folder of the task. ChatGPT performed better than Gemini in zero-shot prompts, few-shot prompts and chain of thought prompts.

The other metric is as follows:- Sort the solution array and LLM array. If the length’s are different , pad the shorter array with zeroes. Now add up all the absolute differences between respective elements in the array. We will call this distance. An LLM is said to perform better than another LLM if the sum of distances across all answers is smaller than the other one.

Task-4(BONUS)

Humans can solve puzzles better which are mathematically deducible. One such example is puzzle\_212.json which none of the LLMs could solve. Machines solve longer puzzles better. Examples of such puzzles are those between puzzle\_106.json and puzzle\_114.json.