TASK 2

Evaluation of Prompting Techniques for Solving Sed Puzzles Using LLMs

Overview

This document presents the evaluation of various prompting techniques used to solve sed puzzle tasks with traditional Large Language Models (LLMs). We systematically tested different strategies, including zero-shot prompting, few-shot prompting, Chain of Thought (CoT) prompting, and lookahead prompting across multiple models, including Gemini, ChatGPT, Llama 2, and Claude AI. The results are documented across different difficulty levels, ranging from Level 0 to "Impossible."

Experimental Setup

Each model was prompted using a standard JSON output format:

```
{
  "problem_id": "000",
  "solution": [0, 1]
}
```

The goal was to find any valid sequence of transitions that converts the given start string into an empty string.

Results

Gemini API

Zero-shot Prompting

- **Level 0: 4/10 problems solved**
- **Level 1: 0/10 problems solved**

- **Level 2: 0/10 problems solved**
- **Level 3: 0/10 problems solved**
- **Level 4: 0/4 problems solved (response parsing errors)**
- **Impossible: 0/2 problems solved (response parsing errors)**

Few-shot Prompting

- **Level 0: 3/10 problems solved**
- **Level 1: 2/10 problems solved**
- **Level 2: 0/10 problems solved**
- **Level 3: 0/10 problems solved (response parsing errors)**
- **Level 4: 0/2 problems solved (1 invalid solution, 1 response parsing error)**
- **Impossible: 0/2 problems solved**

Chain of Thought Prompting

- **Level 0: 5/10 problems solved**
- **Level 1: 0/10 problems solved**
- **Level 2: 0/5 problems solved (reduced string size significantly)**
- **Level 3: 0/5 problems solved (2 parsing errors)**
- **Level 4: 0/3 problems solved**
- **Impossible: 0/2 problems solved**

Lookahead Prompting

- **Level 0: 4/10 problems solved**
- **Level 1: 3/10 problems solved**
- **Level 2: Reduced 32-character string to 3 characters (0/5 fully solved)**
- **Level 3: 0/5 problems solved**
- **Level 4: Reduced one problem's length by half (0/4 fully solved)**
- **Impossible: 0/1 problems solved**

ChatGPT

Zero-shot Prompting

- **Level 0: Solved**
- **Level 1: Reduced string to 1 character**
- **Level 2: Not solved**
- **Level 3: Not solved**
- **Level 4 & Impossible: Not solved**

Few-shot Prompting

- **Level 0: Solved**
- **Level 1: Reduced string to 1/3rd length**
- **Level 2 & 3: Failed, returned another string of same length**
- **Level 4: Halved original string**
- **Impossible: Not solved**

Chain of Thought Prompting

- **Level 0: Solved**
- **Level 1: Solved**
- **Level 2: Reduced string from 5 to 3 characters**
- **Level 3 & Impossible: Not solved**

Lookahead Prompting

- **Level 0: Solved**
- **Level 1: Failed**
- **Level 2: 40% solved**
- **Level 3 & 4 & Impossible: Not solved**

Llama 2

Chain of Thought Prompting

- **Level 0 & Level 1: Not solved**
- **Llama 2 performed poorly and was slow; further testing was discontinued.**

Claude AI

Chain of Thought Prompting

- **Level 0: Solved**
- **Level 1: Reduced string from 11 to 2 characters**
- **Level 2: Reduced string from 9 to 4 characters**
- **Level 3: Reduced string from 17 to 9 characters**
- **Level 4 & Impossible: Not solved**

Zero-shot Prompting

- **Level 0: Solved**
- **Level 1: Solved**
- **Level 2: Reduced string from 10 to 2 characters**
- **Impossible: Reduced 6-line string to 5 lines**

Few-shot Prompting

- **Level 0 & Level 1: Solved**
- **Level 2: Reduced string from 10 to 2 characters**

Lookahead Prompting

- **Level 0 & Level 1: Solved**
- **Level 2: Reduced 6-character string to 1 character**
- **Level 3: Reduced 25-character string to 13 characters**

Analysis & Observations

1. **Zero-shot prompting generally worked best for the simplest problems (Level 0) but failed for complex cases.**
2. **Few-shot prompting improved results slightly but struggled with higher difficulty levels.**
3. **Chain of Thought prompting showed improved reasoning but often resulted in long responses that failed to parse correctly.**
4. **Lookahead prompting helped with intermediate difficulty problems but still failed for the hardest levels.**
5. **Claude AI outperformed other models, showing better reductions in string lengths even when not fully solving puzzles.**
6. **Llama 2 was too slow and ineffective, making it unsuitable for this task.**

Conclusion

Traditional LLMs struggle with complex sed puzzles, especially at higher difficulty levels. Among the tested models:

- **Claude AI performed the best in terms of reducing string lengths.**
- **ChatGPT provided more structured solutions but still failed in harder cases.**
- **Gemini API performed well for Level 0 but struggled with parsing issues.**
- **Llama 2 was the least effective.**

Gemini API Evaluation

We ran a large number of problems using the Gemini API with the following Python script:

```
import json

import requests

import time

import os



# Define paths

PUZZLE_FOLDER = os.path.join(os.path.dirname(__file__), "Puzzles")

SOLUTION_FOLDER = os.path.join(os.path.dirname(__file__), "Solutions")



# Ensure the solutions directory exists

os.makedirs(SOLUTION_FOLDER, exist_ok=True)



# Load a puzzle from JSON file

def load_puzzles(file_path):

   with open(file_path, 'r') as f:

       data = json.load(f)

   return [data] if isinstance(data, dict) else data



# Generate a Lookahead Prompt

def generate_prompt(problem):

   # Example to guide the model
```

```
   example_problem = {
       "problem_id": "001",
       "initial_string": "abcabc",
       "transitions": [
           {"src": "abc", "tgt": ""},
           {"src": "bc", "tgt": "c"}
       ]
   }

   example_solution = """
Example:
Problem ID: 001
Initial string: "abcabc"
Transitions:
0. "abc" → ""
1. "bc" → "c"

### Lookahead Strategy:
1. Identify which transitions can be applied at each step.
2. Predict the resulting string **before applying each transformation**.
3. Choose the transition that brings the string **closest to empty** in the fewest steps.

### Step-by-step reasoning:
```

```
1. The initial string is "abcabc".

2. Applying transition 0 ("abc" → ""), we predict the string will become "abc".

3. Applying transition 0 again, we predict the string will become "" (empty).

4. The optimal solution sequence is **[0, 0]**.



Final Answer (JSON format):

{

   "problem_id": "001",

   "solution": [0, 0]

}

""".strip()



   # Convert the current problem into text format

   transitions_text = "\n".join(

       f"{i}. \"{t['src']}\" → \"{t['tgt']}\""

       for i, t in enumerate(problem["transitions"])

   )



   lookahead_prompt = f"""

Solve the following problem using a **Lookahead Strategy** before providing the JSON
output.



{example_solution}
```

```
Now solve this new problem:



Problem ID: {problem["problem_id"]}

Initial string: "{problem["initial_string"]}"

Transitions:

{transitions_text}



### Lookahead Strategy:

1. Identify possible transitions at each step.

2. **Predict** the resulting string **before applying any transformation**.

3. Select the transition that minimizes remaining transformations.

4. Repeat until the string becomes empty.



### Step-by-step reasoning:

1. Start with the initial string: "{problem["initial_string"]}".

2. Apply the lookahead approach to determine the best sequence.

3. Stop when the string is empty.



Final Answer (JSON format):

{{

   "problem_id": "{problem["problem_id"]}",

   "solution": [index1, index2, ...]  # Must be within 25 steps.

}}

""".strip()
```

```
    print("\n===== GENERATED LOOKAHEAD PROMPT =====\n")
    print(lookahead_prompt)
    print("\n=======================================\n")

    return lookahead_prompt

# Call Gemini API
def call_gemini_api(prompt, api_key):
    url = "https://generativelanguage.googleapis.com/v1/models/gemini-pro:generateContent"
    headers = {"Content-Type": "application/json"}
    data = {"contents": [{"parts": [{"text": prompt}]}], "generationConfig": {"temperature": 0.3}}

    response = requests.post(f"{url}?key={api_key}", headers=headers, json=data)
    return response.json()

# Process all puzzles in the "Puzzles" folder
def run_experiments(api_key):
    puzzle_files = [f for f in os.listdir(PUZZLE_FOLDER) if f.endswith(".json")]

    for puzzle_file in puzzle_files:
        puzzle_path = os.path.join(PUZZLE_FOLDER, puzzle_file)
        puzzles = load_puzzles(puzzle_path)
```

```
        for puzzle in puzzles:
            prompt = generate_prompt(puzzle)  # Generate Lookahead prompt
            response = call_gemini_api(prompt, api_key)  # Get API response
            print(f"Raw response for problem {puzzle['problem_id']}: {response}")

            try:
                response_text =
response["candidates"][0]["content"]["parts"][0]["text"]
                response_text = response_text.strip("```json").strip("``` ").strip()
                solution_data = json.loads(response_text)
            except (KeyError, json.JSONDecodeError, TypeError) as e:
                print(f"Failed to parse response for problem {puzzle['problem_id']} -
{e}")
                solution_data = {"problem_id": puzzle["problem_id"], "solution": []}

            solution_file = os.path.join(SOLUTION_FOLDER,
f"{puzzle['problem_id']}.json")
            with open(solution_file, "w") as f:
                json.dump(solution_data, f, indent=4)

            print(f"Solution saved: {solution_file}")
            time.sleep(1)  # Avoid hitting rate limits

# Usage Example
```

```
if __name__ == "__main__":

   API_KEY =”AIzaSyA0qTSCFDjmtz9V6dYpnXcl3COrJuJQkIg”  # Use environment variable for
security

   run_experiments(API_KEY)
```

By using this script, we were able to generate and validate a large dataset for Gemini.

ChatGPT & Claude Evaluation

For ChatGPT and Claude, we had to manually generate prompts, paste them into the web interface, and verify solutions. Claude frequently hit usage limits, so only easier problems were tested. Due to this, Claude showed artificially higher accuracy.

LLaMA 2 Evaluation

We downloaded LLaMA 2 and ran it locally using `ollama`. However, it was slow and consistently provided incorrect answers even for simple test cases. Due to poor performance, its use was discontinued.

Conclusion & Observations

1. **Gemini API: Allowed us to automate a large number of problem-solving attempts. Despite failures in higher levels, it was effective in generating and testing solutions at scale.**
2. **ChatGPT & Claude: Required manual interaction, making them less scalable. Claude performed better on easier problems but was limited by API restrictions.**
3. **LLaMA 2: Too slow and unreliable to be useful.**

Our study highlights that prompting strategies significantly impact the success of LLMs on sed puzzles. Lookahead prompting improved performance, but models

still struggled with complex cases. Automating the process via APIs yielded the best large-scale evaluation results.