1. # utils/gemini\_client.py
2. import os
3. import google.generativeai as genai
4. from dotenv import load\_dotenv
5. load\_dotenv()
6. API\_KEY = os.getenv("GEMINI\_API\_KEY")
7. if not API\_KEY:
8. raise ValueError("GEMINI\_API\_KEY not found in environment variables.")
9. genai.configure(api\_key=API\_KEY)
10. model = genai.GenerativeModel("gemini-2.0-flash")
11. def generate\_answer(prompt: str) -> str:
12. try:
13. response = model.generate\_content(
14. prompt,
15. generation\_config={"temperature": 0.7}
16. )
17. return response.text.strip()
18. except Exception as e:
19. return f"[ERROR] {str(e)}"
20. # utils/io\_utils.py
21. import json
22. def load\_json(path: str):
23. with open(path, "r", encoding="utf-8") as f:
24. return json.load(f)
25. def save\_json(data, path: str):
26. with open(path, "w", encoding="utf-8") as f:
27. json.dump(data, f, indent=2, ensure\_ascii=False)
28. # main.py - setup code
29. from utils.io\_utils import load\_json, save\_json
30. from utils.gemini\_client import generate\_answer
31. import re
32. import os
33. import argparse
34. import pandas as pd
35. import prompts.gameof24\_template as gameof24\_template
36. import prompts.writing\_template as writing\_template
37. import time
38. generate\_prompt = gameof24\_template.generate\_prompt
39. def pars\_args():
40. parser = argparse.ArgumentParser(description="Run prompting with Gemini API.")
41. parser.add\_argument(
42. "--output",
43. type=str,
44. default="generate\_res/dummy.json",
45. help="Output file path (default: generate\_res/dummy.json)"
46. )
47. parser.add\_argument(
48. "--dataset",
49. type=str,
50. default="writing",
51. help="Dataset type ? game24 or writing (default: writing)"
52. )
53. parser.add\_argument(
54. "--n-sample",
55. type=str,
56. default="5",
57. help="Generated samples for each input (default: 5)"
58. )
59. return parser.parse\_args()
60. # main.py - function main
61. def main():
62. args = parse\_args()
63. print("Arguments parsed:")
64. for arg, value in vars(args).items():
65. print(f"{arg}: {value}")
66. print("\n")
67. if args.dataset == "game24":
68. handle\_game24(args)
69. elif args.dataset == "writing":
70. handle\_writing(args)
72. if \_\_name\_\_ == "\_\_main\_\_":
73. main()
74. # main.py - function handle\_game24
75. def handle\_game24(args, n\_samples=3):
76. dataset\_path = os.path.join("dataset", "game24.csv")
77. dataset = pd.read\_csv(dataset\_path)
78. current\_input\_list = [[]]
79. new\_target\_input = []
80. generated\_samples = []
81. output\_info = []
82. for index, row in dataset.iterrows():
83. logging\_info = {
84. "index": index,
85. "input": row["Puzzles"],
86. "steps": [],
87. "final\_score": 0,
88. }
89. print("\nSolve Puzzle Record : ", index)
90. for step\_index, \_ in enumerate(range(n\_samples)):
91. generated\_samples = []
92. if step\_index == 0:
93. current\_input\_list = [list(map(int, row["Puzzles"].split()))]
94. else:
95. current\_input\_list = new\_target\_input.copy()
96. # STEP 1 GENERATE SAMPLES
97. for index, i in enumerate(current\_input\_list):
98. current\_input\_string = ' '.join(map(str, i))
99. prompt = re.sub(r"\{input\}", current\_input\_string,

generate\_prompt)

1. prompt = re.sub(r"\{n\_sample\}", args.n\_sample, prompt)
2. generated\_samples\_string = generate\_answer(prompt)
3. generated\_samples\_string\_split = generated\_samples\_string

.split('\n')

1. if step\_index != 0:
2. generated\_samples\_string\_split = [logging\_info["steps"]

[step\_index - 1]["new\_target\_path"][index] +

item for item in generated\_samples\_string\_split]

2. generated\_samples.extend(generated\_samples\_string\_split[0:])
3. # STEP 2 EVALUATE SAMPLES
4. candidate\_input\_list = []
5. for sample in generated\_samples:
6. match = re.search(r'\(left:\s\*([0-9.\s]+(?:\.\.\.)?)\)$', sample)
7. if match:
8. raw\_values = match.group(1).strip().split()
9. cleaned\_values = [val.replace('...', '') for val in raw\_values]
10. try:
11. numbers = [float(v) if '.' in v else int(v) for v in

cleaned\_values]

1. candidate\_input\_list.append(numbers)
2. except ValueError:
3. # Lewati jika parsing gagal (misalnya string tidak valid)
4. continue
6. print(candidate\_input\_list)
7. print("Evaluating candidates...")
9. candidate\_scores = [0] \* len(candidate\_input\_list)
10. for index, candidate in enumerate(candidate\_input\_list):
11. # for \_ in range(3):
12. candidate\_string = ' '.join(map(str, candidate))
13. prompt = re.sub(r"\{input\}", candidate\_string,

gameof24\_template.evaluate\_prompt)

1. result = generate\_answer(prompt)
3. match = re.search(r'evaluate:\s\*(\w+)', result)
4. if match:
5. hasil = match.group(1)
6. if hasil == "sure":
7. candidate\_scores[index] += 10
8. elif hasil == "likely":
9. candidate\_scores[index] += 1
10. else :
11. candidate\_scores[index] += 0.01
13. time.sleep(60)
15. print("Candidate scores:", candidate\_scores)
16. # STEP 3 FINALIZE ANSWER
17. new\_target\_input = []
18. candidate\_with\_score = [(index, score) for index, score in

enumerate(candidate\_scores)]

1. sorted\_candidate = sorted(candidate\_with\_score, key=lambda x: x[1],

reverse=True)

1. top\_3\_index = [x[0] for x in sorted\_candidate[:3]]
2. new\_target\_input = [candidate\_input\_list[i] for i in top\_3\_index]
3. logging\_info["steps"].append({
4. "step": step\_index + 1,
5. "generated\_samples": generated\_samples,
6. "candidate\_input\_list": [str(x) for x in candidate\_input\_list],
7. "candidate\_scores": candidate\_scores,
8. "new\_target\_input": [str(x) for x in new\_target\_input],
9. "new\_target\_path": [generated\_samples[i] for i in top\_3\_index]
10. })
12. logging\_info["final\_score"] = "passed" if logging\_info["steps"][2]

["new\_target\_input"][0] == "[24]" else "failed"

1. output\_info.append(logging\_info)
2. time.sleep(60)  # Sleep to avoid rate limiting issues
3. # Save output\_info to file based on the output argument
4. output\_path = args.output
5. save\_json(output\_info, output\_path)
6. return
7. # main.py - function handle\_writing
8. def handle\_writing(args, n\_samples=5):
9. output\_path = args.output
10. with open(os.path.join("dataset", "writing.txt"), "r") as f:
11. writing\_data = [line.strip() for line in f.readlines()]
12. #  Extract the plan from the generated text
13. def extract\_plan\_only(text):
14. lines = text.splitlines()
15. plan\_lines = []
16. in\_plan = False
17. for line in lines:
18. stripped = line.strip()
19. if stripped.lower().startswith("plan:"):
20. in\_plan = True
21. continue  # skip the "Plan:" line itself
22. if in\_plan:
23. # jika menemukan baris kosong atau baris tidak dimulai dengan

"paragraph", kita berhenti

1. if not stripped or not re.match(r'paragraph\s+\d+:', stripped,

re.IGNORECASE):

1. Break
2. plan\_lines.append(stripped)
3. return '\n'.join(plan\_lines).strip() if plan\_lines else "Plan not found."
4. def extract\_passage\_only(text):
5. match = re.search(r"Passage:\s\*(.\*)", text, re.DOTALL)
6. if match:
7. return match.group(1).strip()
8. else:
9. return "Passage not found."
11. output\_info = []
12. for idx, data in enumerate(writing\_data):
13. logging\_info = {
14. "id": idx + 1,
15. "input": data,
16. "steps": []
17. }
18. print(f"\nWriting Record : {idx + 1}")
19. # ToT Prompting
20. # 1. Planning Phase ============================
21. # 1.1 Generate Plan
22. print(f"Generating plans...")
23. generated\_plans = []
24. choices\_list = ""
25. for index, \_ in enumerate(range(n\_samples)):
26. prompt = re.sub(r"\{input\}", data,

writing\_template.generate\_sample\_plan\_prompt)

1. generated\_plan = generate\_answer(prompt)
2. generated\_plan = extract\_plan\_only(generated\_plan)
3. generated\_plans.append(generated\_plan)
4. choices\_list += f"Choice {index + 1}: {generated\_plan}\n\n"
5. # 1.2 Evaluate Plans
6. print(f"Evaluating plans...")
7. evaluation\_score = [0] \* n\_samples
8. for index, \_ in enumerate(range(n\_samples)):
9. vote\_prompt = re.sub(r"\{choices\}", choices\_list, writing\_template

.plan\_vote\_prompt)

1. voting\_result = generate\_answer(vote\_prompt)
2. # Extract the best choice from voting result
3. best\_choice = None
4. best\_choice\_match = re.search(r"The best choice is (\w+)",

voting\_result)

1. if best\_choice\_match:
2. best\_choice = best\_choice\_match.group(1)
3. evaluation\_score[int(best\_choice) - 1] += 1
5. best\_plan\_index = evaluation\_score.index(max(evaluation\_score))
6. best\_plan = generated\_plans[best\_plan\_index] if best\_choice else "No Plan"
7. logging\_info["steps"].append({
8. "step": 0,
9. "generated\_plans": generated\_plans,
10. "evaluation\_score": evaluation\_score,
11. "best\_plan\_index": best\_plan\_index,
12. "best\_plan": best\_plan
13. })
14. time.sleep(60)  # Sleep to avoid rate limiting issues
15. # 2. Writing Passage Phase ============================
16. # 2.1 Generate Passage
17. print(f"Generating passages based on the best plan...")
18. generated\_passages = []
19. for index, \_ in enumerate(range(n\_samples)):
20. prompt = re.sub(r"\{plan\}", best\_plan, writing\_template

.generate\_sample\_passage\_prompt)

1. generated\_passage = generate\_answer(prompt)
2. generated\_passage = extract\_passage\_only(generated\_passage)
4. generated\_passages.append(generated\_passage)
5. choices\_list += f"Choice {index + 1}: {generated\_passage}\n\n"
7. # 2.2 Evaluate Passage
8. print(f"Evaluating passages...")
9. evaluation\_score = [0] \* n\_samples
10. for index, \_ in enumerate(range(n\_samples)):
11. vote\_prompt = re.sub(r"\{choices\}", choices\_list, writing\_template

.passage\_vote\_prompt)

1. voting\_result = generate\_answer(vote\_prompt)
2. best\_choice = None
3. best\_choice\_match = re.search(r"The best passage is (\w+)",

voting\_result)

1. if best\_choice\_match:
2. best\_choice = best\_choice\_match.group(1)
3. evaluation\_score[int(best\_choice) - 1] += 1
5. best\_passage\_index = evaluation\_score.index(max(evaluation\_score))
6. best\_passage = generated\_passages[best\_passage\_index] if best\_choice else

"No Passage"

1. print("Best passage:", best\_passage)
2. logging\_info["steps"].append({
3. "step": 1,
4. "generated\_passages": generated\_passages,
5. "evaluation\_score": evaluation\_score,
6. "best\_passage\_index": best\_passage\_index,
7. "best\_plan": best\_passage
8. })
9. time.sleep(60)  # Sleep to avoid rate limiting issues
10. print(f"\n")
11. output\_info.append(logging\_info)
12. save\_json(output\_info, output\_path)
14. return
15. # prompts/gameof24\_template.py
16. generate\_prompt = """
17. Select exactly two of the input numbers, combine them using one of the operators (+, -, , /), and replace them with the result—reducing the list by one number.
18. IMPORTANT:
19. You MUST provide 5 possible next steps that are STRATEGICALLY HELPFUL toward reaching 24.
20. Do NOT combine numbers randomly — choose pairs that can help reduce the input set toward 24.
21. if the number 24 already exists in the input, you are NOT done. You still MUST combine two numbers into one. Do NOT stop early just because 24 appears — all input numbers must be used exactly once, in a valid operation chain.
22. Input: 2 8 8 14
23. Possible next steps:
24. 2 + 8 = 10 (left: 8 10 14)
25. 8 / 2 = 4 (left: 4 8 14)
26. 14 + 2 = 16 (left: 8 8 16)
27. 2 \* 8 = 16 (left: 8 14 16)
28. 8 - 2 = 6 (left: 6 8 14)
29. Input: 4 5
30. Possible next steps:
31. 4 + 5 = 9 (left: 9)
32. 5 - 4 = 1 (left: 1)
33. 4 \* 5 = 20 (left: 20)
34. Input: 3 6 9
35. Possible next steps:
36. 3 + 6 = 9 (left: 9 9)
37. 9 - 6 = 3 (left: 3 3)
38. 6 \* 3 = 18 (left: 9 18)
39. Input: {input}
40. Possible next steps:
41. """
42. evaluate\_prompt = """Evaluate if the given numbers can reach exactly 24 by combining each number exactly once using +, -, \*, or /.
43. Each input number must be used exactly one time — no numbers can be left unused or reused.
44. input: 10 14
45. 10 + 14 = 24
46. evaluate: sure
47. input: 11 12
48. 11 + 12 = 23
49. 12 - 11 = 1
50. 11 \* 12 = 132
51. 11 / 12 = 0.91
52. evaluate: impossible
53. input: 4 4 10
54. 4 + 4 + 10 = 8 + 10 = 18
55. 4 \* 10 - 4 = 40 - 4 = 36
56. (10 - 4) \* 4 = 6 \* 4 = 24
57. evaluate: sure
58. input: 5 7 8
59. (8 - 5) \* 7 = 3 \* 7 = 21
60. evaluate: likely
61. input: 10 10 11
62. too big
63. evaluate: impossible
64. input: 1 3 3
65. 1 \* 3 \* 3 = 9
66. (1 + 3) \* 3 = 12
67. evaluate: impossible
68. input: {input}
69. ...
70. evaluate: {answer with sure/likely/impossible}
71. """
72. final\_prompt = """Use numbers and basic arithmetic operations (+ - \* /) to obtain 24. Given an input and an answer, give a judgement (sure/impossible) if the answer is correct, i.e. it uses each input exactly once and no other numbers, and reach 24.
73. Input: 4 4 6 8
74. Answer: (4 + 8) \* (6 - 4) = 24
75. Judge:
76. sure
77. Input: 2 9 10 12
78. Answer: 2 \* 12 \* (10 - 9) = 24
79. Judge:
80. sure
81. Input: 4 4 6 8
82. Answer: (4 + 8) \* (6 - 4) + 1 = 25
83. Judge:
84. impossible
85. Input: {input}
86. Answer: {answer}
87. Judge:"""
88. # prompts/writing\_template.py
89. generate\_sample\_plan\_prompt = """Write a coherent passage of 4 short paragraphs. The end sentence of each paragraph must be: It isn't difficult to do a handstand if you just stand on your hands. It caught him off guard that space smelled of seared steak. When she didn’t like a guy who was trying to pick her up, she started using sign language. Each person who knows you has a different perception of who you are.
90. Make a plan only for the passage. Your output MUST be of the following format:
91. Plan:
92. paragraph 1: [write the first paragraph plan here]
93. paragraph 2: [write the second paragraph plan here]
94. paragraph 3: [write the third paragraph plan here]
95. paragraph 4: [write the fourth paragraph plan here]
96. """
97. generate\_sample\_passage\_prompt = """Write a coherent passage of 4 short paragraphs. The end sentence of each paragraph must be: It isn't difficult to do a handstand if you just stand on your hands. It caught him off guard that space smelled of seared steak. When she didn’t like a guy who was trying to pick her up, she started using sign language. Each person who knows you has a different perception of who you are.
98. Write the passage based on that plan.
99. Plan:
100. {plan}
101. Your output should be of the following format:
102. Passage:
103. <Your output passage here>
104. """
105. plan\_vote\_prompt = '''Given a creative writing plan and several proposed paragraph ideas (choices), analyze which idea aligns best with the plan in terms of coherence, creativity, relevance, and thematic fit.
106. Conclude with a final decision in this format:
107. "The best choice is {s}", where s is the integer ID of the best choice.
108. Choices:
109. {choices}
110. '''
111. passage\_vote\_prompt = '''Given several complete passages written based on the same creative writing prompt, analyze which passage is the most effective in terms of coherence, creativity, emotional impact, narrative consistency, and how well it fulfills the intended prompt or structure.
112. Evaluate each passage critically and justify your reasoning. At the end, conclude with your final decision in this format:
113. "The best passage is {s}", where s is the integer ID of the best passage.
114. Passages:
115. {choices}
116. '''
117. score\_prompt = '''Analyze the following passage, then at the last line conclude "Thus the coherency score is {s}", where s is an integer from 1 to 10.
118. '''