

```
1.  # utils/gemini_client.py
2.
3.  import os
4.  import google.generativeai as genai
5.  from dotenv import load_dotenv
6.
7.  load_dotenv()
8.
9.  API_KEY = os.getenv("GEMINI_API_KEY")
10.
11.  if not API_KEY:
12.      raise ValueError("GEMINI_API_KEY not found in environment variables.")
13.
14.  genai.configure(api_key=API_KEY)
15.
16.  model = genai.GenerativeModel("gemini-2.0-flash")
17.
18.  def generate_answer(prompt: str) -> str:
19.      try:
20.          response = model.generate_content(
21.              prompt,
22.              generation_config={"temperature": 0.7}
23.          )
24.          return response.text.strip()
25.      except Exception as e:
26.          return f"[ERROR] {str(e)}"
```

```
1.  # utils/io_utils.py
2.
3.  import json
4.
5.  def load_json(path: str):
6.      with open(path, "r", encoding="utf-8") as f:
7.          return json.load(f)
8.
9.  def save_json(data, path: str):
10.      with open(path, "w", encoding="utf-8") as f:
11.          json.dump(data, f, indent=2, ensure_ascii=False)
```

```
1.  # main.py - setup code
2.
3.  from utils.io_utils import load_json, save_json
4.  from utils.gemini_client import generate_answer
5.  import re
6.  import os
```

```
7.     import argparse
8.     import pandas as pd
9.     import prompts.gameof24_template as gameof24_template
10.    import prompts.writing_template as writing_template
11.    import time
12.    generate_prompt = gameof24_template.generate_prompt
13.
14.    def pars_args():
15.        parser = argparse.ArgumentParser(description="Run prompting with Gemini API.")
16.        parser.add_argument(
17.            "--output",
18.            type=str,
19.            default="generate_res/dummy.json",
20.            help="Output file path (default: generate_res/dummy.json)"
21.        )
22.        parser.add_argument(
23.            "--dataset",
24.            type=str,
25.            default="writing",
26.            help="Dataset type ? game24 or writing (default: writing)"
27.        )
28.        parser.add_argument(
29.            "--n-sample",
30.            type=str,
31.            default="5",
32.            help="Generated samples for each input (default: 5)"
33.        )
34.
35.        return parser.parse_args()
36.
37.
38.    # main.py - function handle_game24
39.    def handle_game24(args, n_samples=3):
40.        dataset_path = os.path.join("dataset", "game24.csv")
41.        dataset = pd.read_csv(dataset_path)
42.
43.        current_input_list = [[]]
44.        new_target_input = []
45.        generated_samples = []
46.
47.        output_info = []
48.        for index, row in dataset.iterrows():
49.            logging_info = {
50.                "index": index,
51.                "input": row["Puzzles"],
52.                "steps": [],
53.                "final_score": 0,
54.            }
```

```
55.         print("\nSolve Puzzle Record : ", index)
56.     for step_index, _ in enumerate(range(3)):
57.         generated_samples = []
58.         if step_index == 0:
59.             current_input_list = [list(map(int, row["Puzzles"].split()))]
60.         else:
61.             current_input_list = new_target_input.copy()
62.
63.         # STEP 1 GENERATE SAMPLES
64.         for index, i in enumerate(current_input_list):
65.             current_input_string = ' '.join(map(str, i))
66.             prompt = re.sub(r"\{input\}", current_input_string,
67.                             generate_prompt)
68.             prompt = re.sub(r"\{n_sample\}", args.n_sample, prompt)
69.             generated_samples_string = generate_answer(prompt)
70.             generated_samples_string_split = generated_samples_string
71.                                     .split('\n')
72.
73.             if step_index != 0:
74.                 generated_samples_string_split = [logging_info["steps"]
75.                                                     [step_index - 1]["new_target_path"][index] +
76.                                                     item for item in generated_samples_string_split]
77.
78.             generated_samples.extend(generated_samples_string_split[0:])
79.
80.         # STEP 2 EVALUATE SAMPLES
81.         candidate_input_list = []
82.         for sample in generated_samples:
83.             match = re.search(r'\(left:\s*([0-9.\s]+(?:\.\.\.\s)?)\)\$', sample)
84.             if match:
85.                 raw_values = match.group(1).strip().split()
86.                 cleaned_values = [val.replace('...', '') for val in raw_values]
87.                 try:
88.                     numbers = [float(v) if '.' in v else int(v) for v in
89.                                cleaned_values]
90.                     candidate_input_list.append(numbers)
91.                 except ValueError:
92.                     # Lewati jika parsing gagal (misalnya string tidak valid)
93.                     continue
94.
95.     print(candidate_input_list)
96.     print("Evaluating candidates...")
97.
98.     candidate_scores = [0] * len(candidate_input_list)
99.     for index, candidate in enumerate(candidate_input_list):
100.         # for _ in range(3):
101.             candidate_string = ' '.join(map(str, candidate))
102.             prompt = re.sub(r"\{input\}", candidate_string,
```

```

        gameof24_template.evaluate_prompt)
98.         result = generate_answer(prompt)
99.
100.         match = re.search(r'evaluate:\s*(\w+)', result)
101.         if match:
102.             hasil = match.group(1)
103.             if hasil == "sure":
104.                 candidate_scores[index] += 10
105.             elif hasil == "likely":
106.                 candidate_scores[index] += 1
107.             else :
108.                 candidate_scores[index] += 0.01
109.
110.         time.sleep(60)
111.
112.         print("Candidate scores:", candidate_scores)
113.
114.         # STEP 3 FINALIZE ANSWER
115.         new_target_input = []
116.         candidate_with_score = [(index, score) for index, score in
                                enumerate(candidate_scores)]
117.         sorted_candidate = sorted(candidate_with_score, key=lambda x: x[1],
                                reverse=True)
118.
119.         top_3_index = [x[0] for x in sorted_candidate[:3]]
120.         new_target_input = [candidate_input_list[i] for i in top_3_index]
121.
122.         logging_info["steps"].append({
123.             "step": step_index + 1,
124.             "generated_samples": generated_samples,
125.             "candidate_input_list": [str(x) for x in candidate_input_list],
126.             "candidate_scores": candidate_scores,
127.             "new_target_input": [str(x) for x in new_target_input],
128.             "new_target_path": [generated_samples[i] for i in top_3_index]
129.         })
130.
131.         logging_info["final_score"] = "passed" if logging_info["steps"][2]
                                ["new_target_input"][0] == "[24]" else "failed"
132.         output_info.append(logging_info)
133.         time.sleep(60) # Sleep to avoid rate limiting issues
134.
135.         # Save output_info to file based on the output argument
136.         output_path = args.output
137.         save_json(output_info, output_path)
138.
139.         return
140.
141. # main.py - function handle_writing
```

```
142.     def handle_writing(args, n_samples=5):
143.         output_path = args.output
144.         with open(os.path.join("dataset", "writing.txt"), "r") as f:
145.             writing_data = [line.strip() for line in f.readlines()]
146.
147.         # Extract the plan from the generated text
148.         def extract_plan_only(text):
149.             lines = text.splitlines()
150.             plan_lines = []
151.             in_plan = False
152.
153.             for line in lines:
154.                 stripped = line.strip()
155.                 if stripped.lower().startswith("plan:"):
156.                     in_plan = True
157.                     continue # skip the "Plan:" line itself
158.
159.                 if in_plan:
160.                     # jika menemukan baris kosong atau baris tidak dimulai dengan
161.                     # "paragraph", kita berhenti
162.                     if not stripped or not re.match(r'paragraph\s+\d+:', stripped,
163.                                                     re.IGNORECASE):
164.                         Break
165.                     plan_lines.append(stripped)
166.
167.             return '\n'.join(plan_lines).strip() if plan_lines else "Plan not found."
168.         def extract_passage_only(text):
169.             match = re.search(r"Passage:\s*(.*)", text, re.DOTALL)
170.             if match:
171.                 return match.group(1).strip()
172.             else:
173.                 return "Passage not found."
174.
175.         def split_with_dot(text):
176.             return ". ".join([f"{index + 1}. {s.strip()}" for index, s in
177.                               enumerate(text.split("."))])
178.
179.         output_info = []
180.         for idx, data in enumerate(writing_data):
181.             logging_info = {
182.                 "id": idx + 1,
183.                 "input": data,
184.                 "steps": []
185.             }
186.             print(f"\nWriting Record : {idx + 1}")
187.             # ToT Prompting
188.             # 1. Planning Phase =====
189.             # 1.1 Generate Plan
```

```
187.         print(f"Generating plans...")
188.         generated_plans = []
189.         choices_list = ""
190.         for index, _ in enumerate(range(n_samples)):
191.             prompt = re.sub(r"\{input\}", split_with_dot(data),
                             writing_template.generate_sample_plan_prompt)
192.             generated_plan = generate_answer(prompt)
193.             generated_plan = extract_plan_only(generated_plan)
194.             generated_plans.append(generated_plan)
195.             choices_list += f"Choice {index + 1}: {generated_plan}\n\n"
196.
197.     # 1.2 Evaluate Plans
198.     print(f"Evaluating plans...")
199.     evaluation_score = [0] * n_samples
200.     for index, _ in enumerate(range(n_samples)):
201.         vote_prompt = re.sub(r"\{choices\}", choices_list, writing_template
                                .plan_vote_prompt)
202.         voting_result = generate_answer(vote_prompt)
203.
204.         # Extract the best choice from voting result
205.         best_choice = None
206.         best_choice_match = re.search(r"The best choice is (\w+)",
                                         voting_result)
207.         if best_choice_match:
208.             best_choice = best_choice_match.group(1)
209.             evaluation_score[int(best_choice) - 1] += 1
210.
211.     best_plan_index = evaluation_score.index(max(evaluation_score))
212.     best_plan = generated_plans[best_plan_index] if best_choice else "No Plan"
213.
214.     logging_info["steps"].append({
215.         "step": 0,
216.         "generated_plans": generated_plans,
217.         "evaluation_score": evaluation_score,
218.         "best_plan_index": best_plan_index,
219.         "best_plan": best_plan
220.     })
221.
222.     time.sleep(60) # Sleep to avoid rate limiting issues
223.
224. # 2. Writing Passage Phase =====
225. # 2.1 Generate Passage
226.     print(f"Generating passages based on the best plan...")
227.     generated_passages = []
228.     for index, _ in enumerate(range(n_samples)):
229.         prompt = re.sub(r"\{plan\}", best_plan, writing_template
                                .generate_sample_passage_prompt)
230.         prompt = re.sub(r"\{input\}", split_with_dot(data), prompt)
231.         generated_passage = generate_answer(prompt)
```

```
232.         generated_passage = extract_passage_only(generated_passage)
233.
234.         generated_passages.append(generated_passage)
235.         choices_list += f"Choice {index + 1}: {generated_passage}\n\n"
236.
237.     # 2.2 Evaluate Passage
238.     print(f"Evaluating passages...")
239.     evaluation_score = [0] * n_samples
240.     for index, _ in enumerate(range(n_samples)):
241.         vote_prompt = re.sub(r"\{choices\}", choices_list, writing_template
                               .passage_vote_prompt)
242.         voting_result = generate_answer(vote_prompt)
243.
244.         best_choice = None
245.         best_choice_match = re.search(r"The best passage is (\w+)",
                                         voting_result)
246.         if best_choice_match:
247.             best_choice = best_choice_match.group(1)
248.             evaluation_score[int(best_choice) - 1] += 1
249.
250.     best_passage_index = evaluation_score.index(max(evaluation_score))
251.     best_passage = generated_passages[best_passage_index] if best_choice else
        "No Passage"
252.     print("Best passage:", best_passage)
253.
254.     logging_info["steps"].append({
255.         "step": 1,
256.         "generated_passages": generated_passages,
257.         "evaluation_score": evaluation_score,
258.         "best_passage_index": best_passage_index,
259.         "best_plan": best_passage
260.     })
261.
262.     time.sleep(60) # Sleep to avoid rate limiting issues
263.     print(f"\n")
264.
265.     output_info.append(logging_info)
266.     save_json(output_info, output_path)
267.
268.     Return
269.
270. # main.py - function main
271. def main():
272.     args = parse_args()
273.     print("Arguments parsed:")
274.     for arg, value in vars(args).items():
275.         print(f"{arg}: {value}")
276.     print("\n")
```

```
277.  
278.     if args.dataset == "game24":  
279.         handle_game24(args, args.n_sample)  
280.     elif args.dataset == "writing":  
281.         handle_writing(args, args.n_sample)  
282.  
283.     if __name__ == "__main__":  
284.         main()
```

```
1.     # prompts/gameof24_template.py  
2.  
3.     generate_prompt = """  
4.     Select exactly two of the input numbers, combine them using one of the operators  
5.     (+, -, , /), and replace them with the result—reducing the list by one number.  
6.  
7.     IMPORTANT:  
8.     1. You MUST provide 5 possible next steps that are STRATEGICALLY HELPFUL toward  
9.     reaching 24.  
10.    2. Do NOT combine numbers randomly – choose pairs that can help reduce the input set  
11.    toward 24.  
12.    3. if the number 24 already exists in the input, you are NOT done. You still MUST  
13.    combine two numbers into one. Do NOT stop early just because 24 appears – all input  
14.    numbers must be used exactly once, in a valid operation chain.  
15.  
16.    Input: 2 8 8 14  
17.    Possible next steps:  
18.    2 + 8 = 10 (left: 8 10 14)  
19.    8 / 2 = 4 (left: 4 8 14)  
20.    14 + 2 = 16 (left: 8 8 16)  
21.    2 * 8 = 16 (left: 8 14 16)  
22.    8 - 2 = 6 (left: 6 8 14)  
23.  
24.    Input: 4 5  
25.    Possible next steps:  
26.    4 + 5 = 9 (left: 9)  
27.    5 - 4 = 1 (left: 1)  
28.    4 * 5 = 20 (left: 20)  
29.  
30.    Input: 3 6 9  
31.    Possible next steps:  
32.    3 + 6 = 9 (left: 9 9)  
33.    9 - 6 = 3 (left: 3 3)  
34.    6 * 3 = 18 (left: 9 18)  
35.  
36.    Input: {input}  
37.    Possible next steps:  
38.    """
```



```
31.
32.     evaluate_prompt = """Evaluate if the given numbers can reach exactly 24 by
    combining each number exactly once using +, -, *, or /.
33.     Each input number must be used exactly one time – no numbers can be left unused or
    reused.
34.
35.     input: 10 14
36.     10 + 14 = 24
37.     evaluate: sure
38.
39.     input: 11 12
40.     11 + 12 = 23
41.     12 - 11 = 1
42.     11 * 12 = 132
43.     11 / 12 = 0.91
44.     evaluate: impossible
45.
46.     input: 4 4 10
47.     4 + 4 + 10 = 8 + 10 = 18
48.     4 * 10 - 4 = 40 - 4 = 36
49.     (10 - 4) * 4 = 6 * 4 = 24
50.     evaluate: sure
51.
52.     input: 5 7 8
53.     (8 - 5) * 7 = 3 * 7 = 21
54.     evaluate: likely
55.
56.     input: 10 10 11
57.     too big
58.     evaluate: impossible
59.
60.     input: 1 3 3
61.     1 * 3 * 3 = 9
62.     (1 + 3) * 3 = 12
63.     evaluate: impossible
64.
65.     input: {input}
66.     ...
67.     evaluate: {answer with sure/likely/impossible}
68.     """
69.
70.     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.
71.     Input: 4 4 6 8
72.     Answer: (4 + 8) * (6 - 4) = 24
73.     Judge:
```

```
74.     sure
75.     Input: 2 9 10 12
76.     Answer: 2 * 12 * (10 - 9) = 24
77.     Judge:
78.     sure
79.     Input: 4 4 6 8
80.     Answer: (4 + 8) * (6 - 4) + 1 = 25
81.     Judge:
82.     impossible
83.     Input: {input}
84.     Answer: {answer}
85.     Judge: ""
```

```
1.     # prompts/writing_template.py
2.
3.     generate_sample_plan_prompt = """Write a coherent passage of 4 short paragraphs.
4.     The end sentence of each paragraph must be: {input}
5.
6.     Make a plan only for the passage. Your output MUST be of the following format:
7.
8.     Plan:
9.     paragraph 1: [write the first paragraph plan here]
10.    paragraph 2: [write the second paragraph plan here]
11.    paragraph 3: [write the third paragraph plan here]
12.    paragraph 4: [write the fourth paragraph plan here]
13.    """
14.
15.    generate_sample_passage_prompt = """Write a coherent passage of 4 short paragraphs.
16.    The end sentence of each paragraph must be: {input}
17.
18.    Write the passage based on that plan.
19.
20.    Plan:
21.    {plan}
22.
23.    Your output should be of the following format:
24.
25.    Passage:
26.    <Your output passage here>
27.    """
28.
29.    plan_vote_prompt = '''Given a creative writing plan and several proposed paragraph
30.    ideas (choices), analyze which idea aligns best with the plan in terms of
    coherence, creativity, relevance, and thematic fit.
    Conclude with a final decision in this format:
    "The best choice is {s}", where s is the integer ID of the best choice.
```

```
31.
32.     Choices:
33.     {choices}
34.     '''
35.
36.     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.
37.
38.     Evaluate each passage critically and justify your reasoning. At the end, conclude
    with your final decision in this format:
39.     "The best passage is {s}", where s is the integer ID of the best passage.
40.
41.     Passages:
42.     {choices}
43.     '''
44.
45.     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.
46.     '''
```