Richard Rafer Guy – 222117056

```python
# utils/gemini_client.py

import os
import google.generativeai as genai
from dotenv import load_dotenv

load_dotenv()

API_KEY = os.getenv("GEMINI_API_KEY")

if not API_KEY:
    raise ValueError("GEMINI_API_KEY not found in environment variables.")

genai.configure(api_key=API_KEY)

model = genai.GenerativeModel("gemini-2.0-flash")

def generate_answer(prompt: str) -> str:
    try:
        response = model.generate_content(
            prompt,
            generation_config={"temperature": 0.7}
        )
        return response.text.strip()
    except Exception as e:
        return f"[ERROR] {str(e)}"
```

```python
# utils/io_utils.py

import json

def load_json(path: str):
    with open(path, "r", encoding="utf-8") as f:
        return json.load(f)

def save_json(data, path: str):
    with open(path, "w", encoding="utf-8") as f:
        json.dump(data, f, indent=2, ensure_ascii=False)
```

```python
# main.py - setup code

from utils.io_utils import load_json, save_json
from utils.gemini_client import generate_answer
import re
import os
import argparse
import pandas as pd
import prompts.gameof24_template as gameof24_template
import prompts.writing_template as writing_template
import time
generate_prompt = gameof24_template.generate_prompt

def pars_args():
    parser = argparse.ArgumentParser(description="Run prompting with Gemini API.")
    parser.add_argument(
        "--output",
        type=str,
        default="generate_res/dummy.json",
        help="Output file path (default: generate_res/dummy.json)"
    )
    parser.add_argument(
        "--dataset",
        type=str,
        default="writing",
        help="Dataset type ? game24 or writing (default: writing)"
    )
    parser.add_argument(
        "--n-sample",
        type=str,
        default="5",
        help="Generated samples for each input (default: 5)"
    )

    return parser.parse_args()


# main.py - function handle_game24
def handle_game24(args, n_samples=3):
    dataset_path = os.path.join("dataset", "game24.csv")
    dataset = pd.read_csv(dataset_path)

    current_input_list = [[]]
    new_target_input = []
    generated_samples = []

    output_info = []
    for index, row in dataset.iterrows():
        logging_info = {
            "index": index,
            "input": row["Puzzles"],
            "steps": [],
            "final_score": 0,
        }
        print("\nSolve Puzzle Record : ", index)
        for step_index, _ in enumerate(range(3)):
            generated_samples = []
            if step_index == 0:
                current_input_list = [list(map(int, row["Puzzles"].split()))]
            else:
                current_input_list = new_target_input.copy()

            # STEP 1 GENERATE SAMPLES
            for index, i in enumerate(current_input_list):
                current_input_string = ' '.join(map(str, i))
                prompt = re.sub(r"\{input\}", current_input_string,
                         generate_prompt)
                prompt = re.sub(r"\{n_sample\}", args.n_sample, prompt)
                generated_samples_string = generate_answer(prompt)
                generated_samples_string_split = generated_samples_string
                                                 .split('\n')

                if step_index != 0:
                    generated_samples_string_split = [logging_info["steps"]
                             [step_index - 1]["new_target_path"][index] +
                             item for item in generated_samples_string_split]

                generated_samples.extend(generated_samples_string_split[0:])

            # STEP 2 EVALUATE SAMPLES
            candidate_input_list = []
            for sample in generated_samples:
                match = re.search(r'\(left:\s*([0-9.\s]+(?:\.\.\.)?)\)$', sample)
                if match:
                    raw_values = match.group(1).strip().split()
                    cleaned_values = [val.replace('...', '') for val in raw_values]
                    try:
                        numbers = [float(v) if '.' in v else int(v) for v in
                                  cleaned_values]
                        candidate_input_list.append(numbers)
                    except ValueError:
                        # Lewati jika parsing gagal (misalnya string tidak valid)
                        continue

            print(candidate_input_list)
            print("Evaluating candidates...")

            candidate_scores = [0] * len(candidate_input_list)
            for index, candidate in enumerate(candidate_input_list):
                # for _ in range(3):
                candidate_string = ' '.join(map(str, candidate))
                prompt = re.sub(r"\{input\}", candidate_string,
                         gameof24_template.evaluate_prompt)
                result = generate_answer(prompt)

                match = re.search(r'evaluate:\s*(\w+)', result)
                if match:
                    hasil = match.group(1)
                    if hasil == "sure":
                        candidate_scores[index] += 10
                    elif hasil == "likely":
                        candidate_scores[index] += 1
                    else :
                        candidate_scores[index] += 0.01

            time.sleep(60)

            print("Candidate scores:", candidate_scores)

            # STEP 3 FINALIZE ANSWER
            new_target_input = []
            candidate_with_score = [(index, score) for index, score in
                                   enumerate(candidate_scores)]
            sorted_candidate = sorted(candidate_with_score, key=lambda x: x[1],
                               reverse=True)

            top_3_index = [x[0] for x in sorted_candidate[:3]]
            new_target_input = [candidate_input_list[i] for i in top_3_index]

            logging_info["steps"].append({
                "step": step_index + 1,
                "generated_samples": generated_samples,
                "candidate_input_list": [str(x) for x in candidate_input_list],
                "candidate_scores": candidate_scores,
                "new_target_input": [str(x) for x in new_target_input],
                "new_target_path": [generated_samples[i] for i in top_3_index]
            })

        logging_info["final_score"] = "passed" if logging_info["steps"][2]
                                    ["new_target_input"][0] == "[24]" else "failed"
        output_info.append(logging_info)
        time.sleep(60)  # Sleep to avoid rate limiting issues

    # Save output_info to file based on the output argument
    output_path = args.output
    save_json(output_info, output_path)

    return

# main.py - function handle_writing
```

```python
def handle_writing(args, n_samples=5):
    output_path = args.output
    with open(os.path.join("dataset", "writing.txt"), "r") as f:
        writing_data = [line.strip() for line in f.readlines()]

    #  Extract the plan from the generated text
    def extract_plan_only(text):
        lines = text.splitlines()
        plan_lines = []
        in_plan = False

        for line in lines:
            stripped = line.strip()
            if stripped.lower().startswith("plan:"):
                in_plan = True
                continue  # skip the "Plan:" line itself

            if in_plan:
                # jika menemukan baris kosong atau baris tidak dimulai dengan "paragraph", kita berhenti
                if not stripped or not re.match(r'paragraph\s+\d+:', stripped, re.IGNORECASE):
                    Break
                plan_lines.append(stripped)

        return '\n'.join(plan_lines).strip() if plan_lines else "Plan not found."
    def extract_passage_only(text):
        match = re.search(r"Passage:\s*(.*)", text, re.DOTALL)
        if match:
            return match.group(1).strip()
        else:
            return "Passage not found."

    def split_with_dot(text):
        return ". ".join([f"{index + 1}. {s.strip()}" for index, s in enumerate(text.split("."))])

    output_info = []
    for idx, data in enumerate(writing_data):
        logging_info = {
            "id": idx + 1,
            "input": data,
            "steps": []
        }
        print(f"\nWriting Record : {idx + 1}")
        # ToT Prompting
        # 1. Planning Phase ============================
        # 1.1 Generate Plan
        print(f"Generating plans...")
        generated_plans = []
        choices_list = ""
        for index, _ in enumerate(range(n_samples)):
            prompt = re.sub(r"\{input\}", split_with_dot(data),
                    writing_template.generate_sample_plan_prompt)
            generated_plan = generate_answer(prompt)
            generated_plan = extract_plan_only(generated_plan)
            generated_plans.append(generated_plan)
            choices_list += f"Choice {index + 1}: {generated_plan}\n\n"

        # 1.2 Evaluate Plans
        print(f"Evaluating plans...")
        evaluation_score = [0] * n_samples
        for index, _ in enumerate(range(n_samples)):
            vote_prompt = re.sub(r"\{choices\}", choices_list, writing_template
                          .plan_vote_prompt)
            vote_prompt = re.sub(r"\{input\}", split_with_dot(data), vote_prompt)
            voting_result = generate_answer(vote_prompt)

            # Extract the best choice from voting result
            best_choice = None
            best_choice_match = re.search(r"The best choice is (\w+)",
                                voting_result)
            if best_choice_match:
                best_choice = best_choice_match.group(1)
                evaluation_score[int(best_choice) - 1] += 1

        best_plan_index = evaluation_score.index(max(evaluation_score))
        best_plan = generated_plans[best_plan_index] if best_choice else "No Plan"

        logging_info["steps"].append({
            "step": 0,
            "generated_plans": generated_plans,
            "evaluation_score": evaluation_score,
            "best_plan_index": best_plan_index,
            "best_plan": best_plan
        })

        time.sleep(60)  # Sleep to avoid rate limiting issues

        # 2. Writing Passage Phase ============================
        # 2.1 Generate Passage
        print(f"Generating passages based on the best plan...")
        generated_passages = []
        for index, _ in enumerate(range(n_samples)):
            prompt = re.sub(r"\{plan\}", best_plan, writing_template
                    .generate_sample_passage_prompt)
            generated_passage = generate_answer(prompt)
            generated_passage = extract_passage_only(generated_passage)

            generated_passages.append(generated_passage)
            choices_list += f"Choice {index + 1}: {generated_passage}\n\n"

        # 2.2 Evaluate Passage
        print(f"Evaluating passages...")
        evaluation_score = [0] * n_samples
        for index, _ in enumerate(range(n_samples)):
            vote_prompt = re.sub(r"\{choices\}", choices_list, writing_template.passage_vote_prompt)
            voting_result = generate_answer(vote_prompt)

            best_choice = None
            best_choice_match = re.search(r"The best passage is (\w+)", voting_result)
            if best_choice_match:
                best_choice = best_choice_match.group(1)
                evaluation_score[int(best_choice) - 1] += 1

        best_passage_index = evaluation_score.index(max(evaluation_score))
        best_passage = generated_passages[best_passage_index] if best_choice else "No Passage"
        print("Best passage:", best_passage)

        logging_info["steps"].append({
            "step": 1,
            "generated_passages": generated_passages,
            "evaluation_score": evaluation_score,
            "best_passage_index": best_passage_index,
            "best_plan": best_passage
        })

        time.sleep(60)  # Sleep to avoid rate limiting issues
        print(f"\n")

        output_info.append(logging_info)
        save_json(output_info, output_path)

    Return

# main.py - function main
def main():
    args = parse_args()
    print("Arguments parsed:")
    for arg, value in vars(args).items():
        print(f"{arg}: {value}")
    print("\n")
```

```python

    if args.dataset == "game24":
        handle_game24(args, args.n_sample)
    elif args.dataset == "writing":
        handle_writing(args, args.n_sample)

if __name__ == "__main__":
    main()
```

```python
# prompts/gameof24_template.py

generate_prompt = """
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.

IMPORTANT:
1. You MUST provide 5 possible next steps that are STRATEGICALLY HELPFUL toward reaching 24.
2. Do NOT combine numbers randomly — choose pairs that can help reduce the input set toward 24.
3. 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.

Input: 2 8 8 14
Possible next steps:
2 + 8 = 10 (left: 8 10 14)
8 / 2 = 4 (left: 4 8 14)
14 + 2 = 16 (left: 8 8 16)
2 * 8 = 16 (left: 8 14 16)
8 - 2 = 6 (left: 6 8 14)

Input: 4 5
Possible next steps:
4 + 5 = 9 (left: 9)
5 - 4 = 1 (left: 1)
4 * 5 = 20 (left: 20)

Input: 3 6 9
Possible next steps:
3 + 6 = 9 (left: 9 9)
9 - 6 = 3 (left: 3 3)
6 * 3 = 18 (left: 9 18)

Input: {input}
Possible next steps:
"""
```

```

evaluate_prompt = """Evaluate if the given numbers can reach exactly 24 by combining each number exactly once using +, -, *, or /.
Each input number must be used exactly one time – no numbers can be left unused or reused.

input: 10 14
10 + 14 = 24
evaluate: sure

input: 11 12
11 + 12 = 23
12 - 11 = 1
11 * 12 = 132
11 / 12 = 0.91
evaluate: impossible

input: 4 4 10
4 + 4 + 10 = 8 + 10 = 18
4 * 10 - 4 = 40 - 4 = 36
(10 - 4) * 4 = 6 * 4 = 24
evaluate: sure

input: 5 7 8
(8 - 5) * 7 = 3 * 7 = 21
evaluate: likely

input: 10 10 11
too big
evaluate: impossible

input: 1 3 3
1 * 3 * 3 = 9
(1 + 3) * 3 = 12
evaluate: impossible

input: {input}
...
evaluate: {answer with sure/likely/impossible}
"""

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.
Input: 4 4 6 8
Answer: (4 + 8) * (6 - 4) = 24
Judge:
```

```
sure
Input: 2 9 10 12
Answer: 2 * 12 * (10 - 9) = 24
Judge:
sure
Input: 4 4 6 8
Answer: (4 + 8) * (6 - 4) + 1 = 25
Judge:
impossible
Input: {input}
Answer: {answer}
Judge:"""
```

```
# prompts/writing_template.py

generate_sample_plan_prompt = """Write a coherent passage of 4 short paragraphs. The end sentence of each paragraph must be: {input}

Make a plan only for the passage. Your output MUST be of the following format:

Plan:
paragraph 1: [write the first paragraph plan here]
paragraph 2: [write the second paragraph plan here]
paragraph 3: [write the third paragraph plan here]
paragraph 4: [write the fourth paragraph plan here]
"""

generate_sample_passage_prompt = """Write a coherent passage of 4 short paragraphs. The end sentence of each paragraph must be: {input}

Write the passage based on that plan.

Plan:
{plan}

Your output should be of the following format:

Passage:
<Your output passage here>
"""

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.

Conclude with a final decision in this format:
"The best choice is {s}", where s is the integer ID of the best choice.
```

```

Choices:
{choices}
'''

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.

Evaluate each passage critically and justify your reasoning. At the end, conclude with your final decision in this format:
"The best passage is {s}", where s is the integer ID of the best passage.

Passages:
{choices}
'''

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.
'''
```