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:

raise ValueError("GEMINI\_API\_KEY not found in environment variables.")

genai.configure(api\_key=API\_KEY)

1. model = genai.GenerativeModel("gemini-2.0-flash")
2. def generate\_answer(prompt: str) -> str:
3. try:
4. response = model.generate\_content(
5. prompt,
6. generation\_config={"temperature": 0.7}
7. )
8. return response.text.strip()
9. except Exception as e:
10. return f"[ERROR] {str(e)}"
11. # utils/io\_utils.py
12. import json
13. def load\_json(path: str):
14. with open(path, "r", encoding="utf-8") as f:
15. return json.load(f)
16. def save\_json(data, path: str):
17. with open(path, "w", encoding="utf-8") as f:
18. json.dump(data, f, indent=2, ensure\_ascii=False)
19. # main.py - setup
20. from utils.io\_utils import load\_json, save\_json
21. from utils.gemini\_client import generate\_answer
22. import re
23. from prompts.templates import basic\_math\_template, plan\_and\_solve\_template
24. import os
25. import argparse
26. import time
27. INPUT\_FILE = "dataset/gsm8k.json"
28. OUTPUT\_FILE = "generate\_res/gsm8k.json"
29. # Mapping teknik ke fungsi template
30. from prompts.templates import (
31. basic\_math\_template,
32. plan\_and\_solve\_template,
33. plan\_and\_solve\_fewshot\_template,
34. aqua\_plan\_and\_solve\_template,
35. aqua\_plan\_and\_solve\_fewshot\_template,
36. lastletter\_plan\_and\_solve,
37. lastletter\_plan\_and\_solve\_fewshot
38. )
39. TEMPLATE\_MAP = {
40. "basic": basic\_math\_template,
41. "plan-and-solve": plan\_and\_solve\_template,
42. "plan-and-solve-fewshot": plan\_and\_solve\_fewshot\_template,
43. "aqua-plan-and-solve": aqua\_plan\_and\_solve\_template,
44. "aqua-plan-and-solve-fewshot": aqua\_plan\_and\_solve\_fewshot\_template,
45. "lastletter-plan-and-solve": lastletter\_plan\_and\_solve,
46. "lastletter-plan-and-solve-fewshot": lastletter\_plan\_and\_solve\_fewshot,
47. }
48. def parse\_args():
49. parser = argparse.ArgumentParser(description="Run prompting with Gemini API.")
50. parser.add\_argument(
51. "--technique",
52. choices=TEMPLATE\_MAP.keys(),
53. default="basic",
54. help="Prompting technique to use (default: basic)"
55. )
56. parser.add\_argument(
57. "--output",
58. type=str,
59. default="generate\_res/dummy.json",
60. help="Output file path (default: generate\_res/gsm8k.json)"
61. )
62. parser.add\_argument(
63. "--dataset-type",
64. choices=["gsm8k", "aqua", "lastletter"],
65. required=True,
66. help="Dataset type: gsm8k or aqua"
67. )
68. parser.add\_argument(
69. "--dataset",
70. type=str,
71. default="dataset/gsm8k.json",
72. help="Dataset file path (default: dataset/gsm8k.json)"
73. )
74. return parser.parse\_args()
75. # Main.py – function main
76. def main():
77. args = parse\_args()
78. technique = args.technique
79. dataset\_type = args.dataset\_type
80. prompt\_fn = TEMPLATE\_MAP[technique]
81. dataset = load\_json(args.dataset)
82. os.makedirs(os.path.dirname(args.output), exist\_ok=True)
83. if dataset\_type == "gsm8k":
84. results, correct\_count = process\_gsm8k(dataset, prompt\_fn, technique)
85. elif dataset\_type == "aqua":
86. results, correct\_count = process\_aqua(dataset, prompt\_fn, technique)
87. elif dataset\_type == "lastletter":
88. results, correct\_count = process\_lastletter(dataset, prompt\_fn, technique)
89. else:
90. raise ValueError("Unsupported dataset type")
91. accuracy = correct\_count / len(results) \* 100
92. print(f"\n✅ Accuracy: {correct\_count}/{len(results)} correct →

{accuracy:.2f}%")

1. save\_json(results, args.output)
2. if \_\_name\_\_ == "\_\_main\_\_":
3. main()
4. # main.py - processing lastletter dataset
5. def process\_lastletter(dataset, prompt\_fn, technique):
6. results = []
7. correct\_count = 0
8. for i, entry in enumerate(dataset):
9. question = entry["question"]
10. expected\_answer = entry["answer"]
11. prompt = prompt\_fn(question)
12. print(f"[{i+1}] LastLetter: {question[:60]}...")
13. model\_answer = generate\_answer(prompt)
14. if "Solve:" in model\_answer:
15. plan\_part, solve\_part = model\_answer.split("Solve:", 1)
16. reasoning = plan\_part.strip() + "\nSolve:\n" + solve\_part.strip()
17. full\_text = solve\_part.strip()
18. else:
19. reasoning = model\_answer.strip()
20. full\_text = model\_answer.strip()
21. match = re.search(r"Answer:\s\*([a-zA-Z]+)", full\_text)
22. final\_answer = match.group(1).lower() if match else None
23. is\_correct = final\_answer == expected\_answer if final\_answer else False
24. if is\_correct:
25. correct\_count += 1
26. if technique == "plan-and-solve-fewshot" or technique == "lastletter-plan-

and-solve-fewshot":

1. plan\_match = re.search(r"Plan:\s\*(.\*?)Solve:", model\_answer, re.DOTALL)
2. solve\_match = re.search(r"Solve:\s\*(.\*?)Answer:", model\_answer,

re.DOTALL)

1. results.append({
2. "question": question,
3. "expected\_answer": expected\_answer,
4. "plan": plan\_match.group(1).strip() if plan\_match else "",
5. "solve": solve\_match.group(1).strip() if solve\_match else "",
6. "generated\_answer": final\_answer,
7. "correct": is\_correct
8. })
9. else:
10. results.append({
11. "question": question,
12. "expected\_answer": expected\_answer,
13. "reasoning": reasoning,
14. "generated\_answer": final\_answer,
15. "correct": is\_correct
16. })
17. if (i+1) % 15 == 0:
18. print(f"Delaying for 1 minute after {i+1} iterations...")
19. time.sleep(60)
21. return results, correct\_count
22. # main.py - processing gsm8k dataset
23. def process\_gsm8k(dataset, prompt\_fn, technique):
24. results = []
25. correct\_count = 0
26. for i, entry in enumerate(dataset):
27. question = entry["question"]
28. expected\_answer = entry["answer"]
29. prompt = prompt\_fn(question)
30. print(f"[{i+1}] GSM8K: {question[:60]}...")
31. model\_answer = generate\_answer(prompt)
32. if "Solve:" in model\_answer:
33. plan\_part, solve\_part = model\_answer.split("Solve:", 1)
34. reasoning = plan\_part.strip() + "\nSolve:\n" + solve\_part.strip()
35. full\_text = solve\_part.strip()
36. else:
37. reasoning = model\_answer.strip()
38. full\_text = model\_answer.strip()
39. match = re.search(r"Answer:\s\*([-+]?\d\*\.\d+|\d+)", full\_text)
40. final\_answer = float(match.group(1)) if match else None
41. is\_correct = (round(final\_answer, 2) == round(expected\_answer, 2)) if

final\_answer is not None else False

1. if is\_correct:
2. correct\_count += 1
3. if technique == "plan-and-solve-fewshot":
4. plan\_match = re.search(r"Plan:\s\*(.\*?)Solve:", model\_answer, re.DOTALL)
5. solve\_match = re.search(r"Solve:\s\*(.\*?)Answer:", model\_answer,

re.DOTALL)

1. results.append({
2. "question": question,
3. "expected\_answer": expected\_answer,
4. "plan": plan\_match.group(1).strip() if plan\_match else "",
5. "solve": solve\_match.group(1).strip() if solve\_match else "",
6. "generated\_answer": final\_answer,
7. "correct": is\_correct
8. })
9. else:
10. results.append({
11. "question": question,
12. "expected\_answer": expected\_answer,
13. "reasoning": reasoning,
14. "generated\_answer": final\_answer,
15. "correct": is\_correct
16. })
17. if (i+1) % 15 == 0:
18. print(f"Delaying for 1 minute after {i+1} iterations...")
19. time.sleep(60)
20. return results, correct\_count
21. # main.py - processing aqua dataset
22. def process\_aqua(dataset, prompt\_fn, technique):
23. results = []
24. correct\_count = 0
25. for i, entry in enumerate(dataset):
26. question = entry["question"]
27. options = entry["options"]
28. expected\_choice = entry["correct"]
29. prompt = prompt\_fn(question, options)
30. print(f"[{i+1}] AQUA: {question[:60]}...")
31. model\_answer = generate\_answer(prompt)
32. # Ekstrak pilihan jawaban akhir
33. choice\_match = re.search(r"Answer:\s\*([A-E])", model\_answer)
34. predicted\_choice = choice\_match.group(1) if choice\_match else ""
35. is\_correct = predicted\_choice == expected\_choice
36. if is\_correct:
37. correct\_count += 1
38. # Handle output format tergantung teknik
39. if technique == "aqua-plan-and-solve":
40. # Ambil seluruh reasoning sebelum "Answer:"
41. reasoning\_match = re.search(r"(.\*)Answer:\s\*[A-E]", model\_answer,

re.DOTALL)

1. reasoning\_text = reasoning\_match.group(1).strip() if reasoning\_match

else model\_answer.strip()

1. results.append({
2. "question": question,
3. "options": options,
4. "plan-and-solve": reasoning\_text,
5. "generated\_answer": predicted\_choice,
6. "expected\_answer": expected\_choice,
7. "correct": is\_correct
8. })
9. else:
10. # Default untuk few-shot dan lainnya
11. plan\_match = re.search(r"Plan:\s\*(.\*?)(Solve:|Answer:)", model\_answer,

re.DOTALL | re.IGNORECASE)

1. solve\_match = re.search(r"Solve:\s\*(.\*?)(Answer:)", model\_answer,

re.DOTALL | re.IGNORECASE)

1. plan = plan\_match.group(1).strip() if plan\_match else ""
2. solve = solve\_match.group(1).strip() if solve\_match else ""
3. results.append({
4. "question": question,
5. "options": options,
6. "plan": plan,
7. "solve": solve,
8. "generated\_answer": predicted\_choice,
9. "expected\_answer": expected\_choice,
10. "correct": is\_correct
11. })
12. if (i+1) % 15 == 0:
13. print(f"Delaying for 1 minute after {i+1} iterations...")
14. time.sleep(60)
15. return results, correct\_count
16. # prompts/templates.py
17. def basic\_math\_template(question: str) -> str:
18. return f"""
19. You are a helpful and accurate math problem solver.
20. Question:
21. {question}
22. Provide only the final numerical answer.
23. """
24. def plan\_and\_solve\_template(question: str) -> str:
25. return f"""
26. You are a helpful and logical math tutor. When solving a math problem, follow these steps:
27. 1. First create a plan with bullet points.
28. 2. Then solve the problem step-by-step.
29. 3. Finally, output the final answer in the format: Answer: <FINAL ANSWER MUST BE NUMBER ONLY>.
30. Question:
31. {question}
32. Plan:
33. –
34. Solve:
35. """
36. def plan\_and\_solve\_fewshot\_template(question: str) -> str:
37. few\_shot\_examples = """
38. You are a logical and helpful math tutor. For each question, follow this format:
39. Plan:
40. - step 1
41. - step 2
42. ...
43. Solve:
44. <detailed solution steps>
45. Answer: <final numeric answer>
46. ---
47. Question: Jamie buys 3 pens for $2 each and a notebook for $4. What is the total cost?
48. Plan:
49. - Calculate cost of pens.
50. - Add cost of notebook.
51. Solve:
52. 3 pens × $2 = $6
53. $6 + $4 = $10
54. Answer: 10
55. ---
56. Question: A box contains 5 red balls, 3 green balls, and 2 blue balls. How many total balls are there?
57. Plan:
58. - Add all the balls.
59. Solve:
60. 5 + 3 + 2 = 10
61. Answer: 10
62. ---
63. Question: {question}
64. Plan:
65. -
66. """
67. return few\_shot\_examples.strip().replace("{question}", question)
68. def aqua\_plan\_and\_solve\_template(question: str, options: list[str]) -> str:
69. options\_text = "\n".join(options)
70. return f"""
71. You are a helpful and logical math tutor. For the following multiple-choice math question, follow these steps:
72. 1. \*\*Analyze\*\* the question carefully and determine the key values.
73. 2. \*\*Plan\*\* how to solve the question using logical steps.
74. 3. \*\*Solve\*\* the problem step-by-step with clarity.
75. 4. At the end, clearly write your final answer in the format: Answer: <MUST BE FINAL OPTION LETTER ONLY>
76. Question:
77. {question}
78. Options:
79. {options\_text}
80. Begin your solution:
81. Plan:
82. –
83. """.strip()
84. def aqua\_plan\_and\_solve\_fewshot\_template(question: str, options: list[str]) -> str:
85. options\_text = "\n".join(options)
86. few\_shot\_examples = """
87. You are a logical and accurate math tutor. For each multiple-choice math problem, follow this format:
88. - First write a step-by-step plan
89. - Then solve the question
90. - Then select the correct answer using: Answer: <option letter>
91. ---
92. Question: Sarah has 5 pencils. She gives 2 to her friend. How many does she have left?
93. Options:
94. A) 1
95. B) 2
96. C) 3
97. D) 4
98. E) 5
99. Plan:
100. - Start with total pencils
101. - Subtract the number she gave away
102. Solve:
103. 5 - 2 = 3
104. Answer: C
105. ---
106. Question: {question}
107. Options:
108. {options\_text}
109. Plan:
110. –
111. """.strip()
112. return few\_shot\_examples.replace("{question}",

question).replace("{options\_text}", options\_text)

1. def lastletter\_plan\_and\_solve(question: str) -> str:
2. return f"""
3. You are a helpful and logical assistant. When solving a word problem, follow these steps:
4. 1. First create a plan with bullet points.
5. 2. Then solve the problem step-by-step.
6. 3. Finally, output the final answer in the format: Answer: <FINAL ANSWER MUST BE A LOWERCASE STRING>.
7. Question:
8. {question}
9. Plan:
10. –
11. Solve:
12. """.strip()
13. def lastletter\_plan\_and\_solve\_fewshot(question: str) -> str:
14. few\_shot\_examples = """
15. You are a logical and helpful assistant. For each string manipulation task, follow this format:
16. Plan:
17. - Break the sentence into individual words.
18. - Take the last letter of each word.
19. - Concatenate the letters.
20. - Convert to lowercase if needed.
21. Solve:
22. Whitney Erika Tj Benito → ["Whitney", "Erika", "Tj", "Benito"]
23. Last letters → ["y", "a", "j", "o"]
24. Concatenate → "yajo"
25. Answer: yajo
26. ---
27. Question: Lucky Mireya Jj Kc
28. Plan:
29. - Break the sentence into individual words.
30. - Take the last letter of each word.
31. - Concatenate the letters.
32. - Convert to lowercase if needed.
33. Solve:
34. Lucky Mireya Jj Kc → ["Lucky", "Mireya", "Jj", "Kc"]
35. Last letters → ["y", "a", "j", "c"]
36. Concatenate → "yajc"
37. Answer: yajc
38. ---
39. Question: Caleb Chase Eleazar Chanel
40. Plan:
41. - Break the sentence into individual words.
42. - Take the last letter of each word.
43. - Concatenate the letters.
44. - Convert to lowercase if needed.
45. Solve:
46. Caleb Chase Eleazar Chanel → ["Caleb", "Chase", "Eleazar", "Chanel"]
47. Last letters → ["b", "e", "r", "l"]
48. Concatenate → "berl"
49. Answer: berl
50. ---
51. Question: {question}
52. Plan:
53. –
54. """.strip()
55. return few\_shot\_examples.replace("{question}", question)