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.")

1. genai.configure(api\_key=API\_KEY)
2. model = genai.GenerativeModel("gemini-2.0-flash")
3. def generate\_answer(prompt: str) -> str:
4. try:
5. response = model.generate\_content(
6. prompt,
7. generation\_config={"temperature": 0.7}
8. )
9. return response.text.strip()
10. except Exception as e:
11. return f"[ERROR] {str(e)}"
12. # utils/io\_utils.py
14. import json
15. def load\_json(path: str):
16. with open(path, "r", encoding="utf-8") as f:
17. return json.load(f)
18. def save\_json(data, path: str):
19. with open(path, "w", encoding="utf-8") as f:
20. json.dump(data, f, indent=2, ensure\_ascii=False)
21. # main.py
22. from utils.io\_utils import load\_json, save\_json
23. from utils.gemini\_client import generate\_answer
24. import re
25. import os
26. import argparse
27. import pandas as pd
28. import time
29. def parse\_args():
30. parser = argparse.ArgumentParser(description="Run prompting with Gemini API.")
31. parser.add\_argument(
32. "--output",
33. type=str,
34. default="generate\_res/dummy.json",
35. help="Output file path (default: generate\_res/dummy.json)"
36. )
37. parser.add\_argument(
38. "--task",
39. type=str,
40. default="musique",
41. help="Dataset type ? (default: musique)"
42. )
43. return parser.parse\_args()
44. def get\_controller(task):
45. if task == "musique":
46. from musique\_controller import MusiqueController
47. return MusiqueController()
48. elif task == "2wiki":
49. from wiki\_controller import WikiController
50. return WikiController()
51. else:
52. raise ValueError(f"Unsupported task: {task}")
53. def main():
54. args = parse\_args()
55. print("Arguments parsed:")
56. for arg, value in vars(args).items():
57. print(f"{arg}: {value}")
58. print("\n")
59. controller = get\_controller(args.task)
60. # get decomposition chains
61. controller.solve()
63. if \_\_name\_\_ == "\_\_main\_\_":
64. main()
65. # wiki\_controller.py
66. from utils.io\_utils import load\_json, save\_json
67. from utils.gemini\_client import generate\_answer
68. import prompts.wiki.template as wiki\_template
69. import re
70. import os
71. import time
72. import json
73. import ast
74. class WikiController:
75. def \_\_init\_\_(self):
76. self.dataset = load\_json('dataset/2wiki.json')
77. def print\_questions(self):
78. """Prints all questions from the dataset."""
79. for entry in self.dataset:
80. print(entry['question'])
81. def replace\_references(self, s, result\_array):
82. def replacer(match):
83. index = int(match.group(1))-1  # Ambil angka dari #3 misalnya
84. if 0 <= index < len(result\_array):
85. return result\_array[index]
86. else:
87. return f"<Invalid index #{index}>"
89. # Ganti semua #angka dengan nilai yang sesuai
90. return re.sub(r"#(\d+)", replacer, s)
91. def get\_generated\_answer(self, ans):
92. if "Answer:" in ans:
93. process\_answer = ans.split("Answer: ", 1)[1].strip()
94. else:
95. process\_answer = ans.strip()
96. return process\_answer
97. def clean\_text(self, text):
98. return re.sub(r'\(.\*?\)|\[.\*?\]', '', text).strip()
99. def get\_references(self, qs):
100. references = re.findall(r'#(\d+)', qs)
101. return [int(ref) for ref in references]
102. def get\_references\_array(self, qs, result\_answers):
103. references = result\_answers[self.get\_references(qs)[0]-1]
104. array\_ver = json.loads(references)
105. return array\_ver
107. def implic\_RAG(self, question, entry):
108. formatted\_context\_list = self.get\_formatted\_context(entry)
109. fulltext\_context = ""
110. for context in formatted\_context\_list:
111. fulltext\_context += context
112. prompt = wiki\_template.impli\_rag\_template.replace('{question}',

question).replace('{context}', fulltext\_context)

1. generated\_ans = generate\_answer(prompt)
3. start = generated\_ans.find("C1:")
4. extracted\_output = generated\_ans[start:]
5. return extracted\_output
7. def chain\_processing(self, qs\_lines, entry):
8. result\_answers = []
9. for index, qs in enumerate(qs\_lines):
10. # Cari yang dalam tanda kurung bulat (command)
11. command\_match = re.search(r"\[(.\*?)\]", qs)
12. command = command\_match.group(1) if command\_match else ''
13. process\_answer = ""
14. if command == "get\_ent\_qa":
15. formatted\_qs = self.replace\_references(qs, result\_answers)
16. prompt = wiki\_template.get\_ent\_qa\_template.replace('{input}',

formatted\_qs).replace('{context}', entry['question'])

1. generated\_ans = generate\_answer(prompt)
2. print(f"Generated answer for get\_ent\_qa: {generated\_ans}")
3. process\_answer = self.get\_generated\_answer(generated\_ans)
4. print(f"Processed answer for get\_ent\_qa: {process\_answer}")
6. elif command == "get\_atr\_qa":
7. references = self.get\_references\_array(qs, result\_answers)
8. ans\_list = []
9. for ref in references:
10. # format and generate prompt template
11. formatted\_qs = re.sub(r"#\d+", ref, qs)
12. cleaned\_text = self.clean\_text(formatted\_qs)
13. print(f"Formatted question: {cleaned\_text}")
14. context = self.implic\_RAG(cleaned\_text, entry)
15. print(f"Context: {context}\n")
16. prompt = wiki\_template.get\_atr\_qa\_template

.replace('{question}', cleaned\_text)

.replace('{context}', context)

2. generated\_ans = generate\_answer(prompt)
3. formatted\_ans = self.get\_generated\_answer(generated\_ans)
4. ans\_list.append(f"{ref} => {formatted\_ans}")
5. print(ans\_list)
6. process\_answer = ans\_list
7. elif command == "comp\_qa":
8. latest\_ans = result\_answers[-1]
9. full\_context = ""
10. for ans in latest\_ans:
11. full\_context += f"{ans}\n"
12. prompt = wiki\_template.comp\_qa\_template.replace('{question}', qs).replace('{context}', full\_context)
13. generated\_ans = generate\_answer(prompt)
14. process\_answer = self.get\_generated\_answer(generated\_ans)
15. print(f"Processed answer for comp\_qa: {process\_answer}")
16. elif command == "EOQ":
17. Break
19. print(f"Processing step {index+1}: {command} -> {process\_answer}")
20. result\_answers.append(process\_answer)
22. return result\_answers
24. def get\_formatted\_context(self, entry):
25. formatted\_data = []
26. formatted\_entry\_context = json.loads(entry['context'])
27. for index, context in enumerate(formatted\_entry\_context):
28. fulltext = ""
29. title = context[0]
30. for text in context[1]:
31. fulltext += text
32. formatted\_context\_text = f"{index+1}. {title} => {fulltext}\n"
33. formatted\_data.append(formatted\_context\_text)
35. return formatted\_data
36. def generate\_chain(self, question):
37. template = wiki\_template.decomp\_template
38. prompt = template.replace('{input}', question)
39. decomp\_chain = generate\_answer(prompt)
41. qs\_lines = [line.replace("QS: ", "").strip() for line in

decomp\_chain.splitlines() if line.startswith("QS:")]

1. print(f"Decomposition chain generated: {qs\_lines}")
2. return qs\_lines
3. def solve(self):
5. output\_log = []
6. score = 0
7. for entry in self.dataset:
8. qs\_lines = self.generate\_chain(entry['question'])
9. output = self.chain\_processing(qs\_lines, entry)
10. def check\_answer(output):
11. return entry['answer'].lower() == output[-1].lower() if output else

False

1. output\_log.append({
2. 'question': entry['question'],
3. 'qs\_lines': qs\_lines,
4. "output": output,
5. "is\_correct": check\_answer(output),
6. })
7. current\_score = 1 if check\_answer(output) else 0
8. score += current\_score
9. save\_json(output\_log, 'generate\_res/dummy.json')
10. time.sleep(60)  # To avoid hitting rate limits
11. print(f"final score: {score}")
12. return True
13. # musique\_controller.py
14. from utils.io\_utils import load\_json, save\_json
15. from utils.gemini\_client import generate\_answer
16. import prompts.musique.template as musique\_template
17. import re
18. import os
19. import time
20. import json
21. import ast
22. class MusiqueController:
23. def \_\_init\_\_(self):
24. self.dataset = load\_json('dataset/musique.json')
25. def print\_questions(self):
26. """Prints all questions from the dataset."""
27. for entry in self.dataset:
28. print(entry['question'])
29. def get\_generated\_answer(self, ans):
30. if "Answer:" in ans:
31. process\_answer = ans.split("Answer: ", 1)[1].strip()
32. else:
33. process\_answer = ans.strip()
34. return process\_answer
36. def get\_fullcontext (self, paragraphs):
37. fulltext\_context = ""
38. for index, paragraph in enumerate(paragraphs):
39. fulltext\_context += f"{paragraph['title']} =>

{paragraph['paragraph\_text']}\n"

1. return fulltext\_context
2. def implic\_RAG(self, question, paragraphs):
4. fulltext\_context = self.get\_fullcontext(paragraphs)
5. prompt = musique\_template.impli\_rag\_template.replace('{question}',

question).replace('{context}', fulltext\_context)

1. generated\_ans = generate\_answer(prompt)
2. start = generated\_ans.find("C1:")
3. extracted\_output = generated\_ans[start:]
4. output\_lines = extracted\_output.splitlines()
5. output\_cleaned = [line.split(":", 1)[-1].strip() for line in output\_lines]
6. # Gabungkan kembali konteks menjadi string yang bersih
7. cleaned\_output = "\n".join(output\_cleaned)
8. return cleaned\_output
9. def chain\_processing(self, qs\_lines, paragraphs):
10. result\_answers = []
11. for index, qs in enumerate(qs\_lines):
12. process\_answer = ""
14. if index == 0:
15. context = self.get\_fullcontext(paragraphs)
17. prompt = musique\_template.starter\_qa.replace('{question}',

qs['question']).replace('{context}', context)

1. generated\_ans = generate\_answer(prompt)
2. process\_answer = self.get\_generated\_answer(generated\_ans)
3. else:
4. prev\_ans = result\_answers[-1]
5. context = self.get\_fullcontext(paragraphs)
6. formatted\_qs = re.sub(r"#\d+", prev\_ans, qs['question'])
7. prompt = musique\_template.finisher\_template.replace('{question}',

formatted\_qs).replace('{context}', context)

1. generated\_ans = generate\_answer(prompt)
2. process\_answer = self.get\_generated\_answer(generated\_ans)
3. result\_answers.append(process\_answer)
4. return result\_answers
6. def solve(self):
8. output\_log = []
9. score = 0
10. for entry in self.dataset:
12. output = self.chain\_processing(entry['question\_decomposition'], entry['paragraphs'])
13. print("processing index: ", entry['id'])
14. output\_log.append({
15. 'question': entry['question'],
16. 'qs\_lines': entry['question\_decomposition'],
17. 'result': output,
18. 'expected\_answer': entry['answer'],
19. "is\_correct": entry['answer'].lower() == output[-1].lower()
20. })
21. # break
22. save\_json(output\_log, 'generate\_res/dummy.json')
23. time.sleep(20)  # To avoid hitting rate limits
24. print(f"final score: {score}")
25. return True
26. # prompts/wiki/template.py
27. decomp\_template = '''
28. QC: Which film came out first, The Love Route or Engal Aasan?
29. QS: [get\_ent\_qa] Which films are being compared in the question?
30. QS: (foreach)[get\_atr\_qa] What is the release date of #1?
31. QS: [comp\_qa] Which film came out first based on the release date?
32. QC: Are Matraville Sports High School and Wabash High School both located in the same country?
33. QS: [get\_ent\_qa] Which schools are being compared in the question?
34. QS: (foreach)[get\_atr\_qa] What is the location of #1?
35. QS: [comp\_qa] Are the schools located in the same country?
36. QC: Are Alison Skipper and Diane Gilliam Fisher from the same country?
37. QS: [get\_ent\_qa] Who are the people being compared in the question?
38. QS: (foreach)[get\_atr\_qa] What is the nationality of #1?
39. QS: [comp\_qa] Do they have the same nationality?
40. QC: Do the movies Bloody Birthday and The Beckoning Silence, originate from the same country?
41. QS: [get\_ent\_qa] Which movies are being compared in the question?
42. QS: (foreach)[get\_atr\_qa] What is the country of origin of #1?
43. QS: [comp\_qa] Do the movies originate from the same country?
44. QC: Are both businesses, Vakıfbank and Infopro Sdn Bhd, located in the same country?
45. QS: [get\_ent\_qa] Which businesses are being compared in the question?
46. QS: (foreach)[get\_atr\_qa] What is the location of #1?
47. QS: [comp\_qa] Are the businesses located in the same country?
48. QC: Does Mukasa Mbidde have the same nationality as Erich Maas?
49. QS: [get\_ent\_qa] Who are the people being compared in the question?
50. QS: (foreach)[get\_atr\_qa] What is the nationality of #1?
51. QS: [comp\_qa] Do they have the same nationality?
52. QC: {input}
53. Output:
54. QS: <QS-1>
55. QS: <QS-2>
56. .....
57. QS: <QS-N>
58. '''
59. get\_ent\_qa\_template = '''
60. QC: Are Matraville Sports High School and Wabash High School both located in the same country?
61. Q: Which schools are being compared in the question?
62. Answer: ["Matraville Sports High School", "Wabash High School"]
63. QC: Are Alison Skipper and Diane Gilliam Fisher from the same country?
64. Q: Which people are being compared in the question?
65. Answer: ["Alison Skipper", "Diane Gilliam Fisher"]
66. QC: Do the movies Bloody Birthday and The Beckoning Silence, originate from the same country?
67. Q: Which movies are being compared in the question?
68. Answer: ["Bloody Birthday", "The Beckoning Silence"]
69. QC: Are both businesses, Vakıfbank and Infopro Sdn Bhd, located in the same country?
70. Q: Which businesses are being compared in the question?
71. Answer: ["Vakıfbank", "Infopro Sdn Bhd"]
72. QC: Did the movies Pony Express (Film) and The Da Vinci Code (Film), originate from the same country?
73. Q: Which people are being compared in the question?
74. Answer: ["Sam Earle", "Felix Luckeneder"]
75. QC: Are Sam Earle and Felix Luckeneder from the same country?
76. Q: Which locations are being compared in the question?
77. Answer: ["Lesser Slave Lake", "Medeweger See"]
78. QC: Are both Lesser Slave Lake and Medeweger See located in the same country?
79. Q: Which films are being compared in the question?
80. Answer: ["Alsino And The Condor", "1922 (2017 Film)"]
81. QC: Are Alsino And The Condor and 1922 (2017 Film) both from the same country?
82. Q: Which bands are being compared in the question?
83. Answer: ["Cinematic Sunrise", "Kingston Falls"]
84. QC: Are the movies Carnival Of Souls and Uvanga, from the same country?
85. Q: Which schools are being compared in the question?
86. Answer: ["St. Mary High School (Rutherford, New Jersey)", "Mother Teresa High School"]
87. Input:
88. QC: {context}
89. Q: {input}
90. Output (your answer MUST be in the same format "Answer: <your answer here>", you MUST add "Answer: " before your answer):
91. Answer: <your answer here>
92. '''
93. get\_atr\_qa\_template = '''
94. Q: {question}
95. Context:
96. {context}
97. Output:
98. Answer: <Your answer>
99. '''
100. comp\_qa\_template = '''
101. QS: {question}
102. Context:
103. {context}
104. Output (your answer MUST be in the same format "Answer: <your answer here>", you MUST add "Answer: " before your answer):
105. Answer: <your answer here>
106. '''
107. impli\_rag\_template = '''
108. Given the following context data and a specific question, please provide the top 3 most relevant contexts that help answer the question. The output should be formatted as follows:
109. Context:
110. C1: "..."
111. C2: "..."
112. C3: "..."
113. Context Data:
114. {context}
115. Question: {question}
116. Output:
117. C1: <Best Context>
118. C2: <Second Best Context>
119. C3: <Third Best Context>
120. '''
121. # prompts/musique/template.py
122. starter\_qa = '''
123. Input:
124. Q: {question}
125. Context:
126. {context}
127. Output (your answer MUST be in the same format "Answer: <your answer here>", you MUST add "Answer: " before your answer):
128. Answer: <your answer here>
129. '''
130. finisher\_template = '''
131. Input:
132. Q: {question}
133. Context:
134. {context}
135. Output (your answer MUST be in the same format "Answer: <your answer here>", you MUST add "Answer: " before your answer):
136. Answer: <your answer here>
137. '''
138. impli\_rag\_template = '''
139. Given the following context data and a specific question, please provide the top 3 most relevant contexts that help answer the question. The output should be formatted as follows:
140. Context:
141. C1: "..."
142. C2: "..."
143. C3: "..."
144. Context Data:
145. {context}
146. Question: {question}
147. Output:
148. C1: <Best Context>
149. C2: <Second Best Context>
150. C3: <Third Best Context>
151. '''