LLM PROMPTS SAMPLE TASKS:

1] <https://github.com/AK067CSE/chatpdf/blob/main/chatbot_app.py>

2] <https://github.com/AK067CSE/chatpdf/blob/main/query.py>

3] <https://github.com/AK067CSE/chatpdf/blob/main/new.py>

from openai import OpenAI

import requests

# Set up Gemini API with API key

API\_KEY = "your\_gemini\_api\_key\_here" # Replace with your actual Gemini API key

# Define the new Gemini API URL

GEMINI\_API\_URL = f"https://generativelanguage.googleapis.com/v1beta/models/gemini-pro:generateContent?key={API\_KEY}"

# Define prompt templates for different content types

AI\_PROMPTS = {

"analogy": """

Explain [Concept] using an analogy with something familiar. Make it:

- Easy to understand, avoiding technical terms or complexity

- Based on a common experience or idea

- Focused on key similarities and insights to make the concept relatable and clear.

""",

"description": """

Break down [Concept] in a simple, beginner-friendly way. Imagine it's being introduced for the first time. Make it:

- Brief, fun, and easy to follow

- Focused on the main idea and basic functionality.

""",

"codeSnippet": """

Create a concise code snippet to demonstrate [Programming Concept]. Make it:

- Short (under 20 lines) and syntactically correct

- Clear and practical, showcasing a specific use of the concept

- Commented with brief explanations for readability

Use [Programming Language] and provide a hands-on example for easy experimentation.

""",

"funfacts": """

Make learning [Concept] exciting with fun facts. Find:

- Intriguing or surprising details related to the concept

- Simple and memorable explanations

- Vivid descriptions or unexpected connections that spark curiosity

List 2-3 engaging fun facts to enhance enjoyment and retention.

""",

"quiz": """

Reinforce understanding of [Concept] with a short quiz. Include:

- Key questions that cover the concept's essentials

- Clear language and concise questions

- Answers with brief explanations to deepen understanding

Keep it engaging, informative, and balanced in difficulty.

""",

"assignment": """

Design a mini assignment to apply [Concept] practically. Make it:

- Simple and focused, taking under 30 minutes

- Centered on one main aspect of the concept

- Relevant to real-life or relatable scenarios

Help reinforce understanding by creating a hands-on task to apply [Concept].

"""

}

# Define text prompt for content structure

TEXT\_PROMPT = """\

Create a set of maximum {max\_objects} card objects to provide {content\_type} related to the subtopic "{subtopic}" of "{skill\_name}" under the topic "{topic}".

Guidelines:

1. Card ID should follow the format: '{card\_prefix}{{index}}-sub{{index}}' where the index starts from 1.

2. Each card's content should not exceed 6 lines. If needed, add additional content in a new card object.

3. The value of the content property should be of the same data type as provided in the schema/sample.

4. Use </br> after each sentence.

5. Use simple and easy-to-understand vocabulary.

6. Style content with HTML and inline styles for emphasis & key terms (e.g., <span style="color: red;">bold</span>, <span style="color: green;">colors</span>, <i>italic</i>).

7. Use only the element values available in the card render object.

8. If a {content\_type} is irrelevant for the subtopic "{subtopic}" of "{skill\_name}" under the topic "{topic}", provide output as an empty JSON object.

"""

CONTENT\_TYPES\_ARRAY = ["analogy", "description", "codeSnippet", "funfacts", "quiz", "assignment"]

MAX\_OBJECTS = {

"analogy": 1,

"description": 8,

"codeSnippet": 3,

"funfacts": 2,

"quiz": 6,

"assignment": 1

}

# Function to get Gemini response based on a prompt

def get\_gemini\_response(prompt):

headers = {

"Content-Type": "application/json"

}

payload = {

"model": "gemini-pro", # Update with the actual model name if needed

"messages": [{"role": "user", "content": prompt}],

"temperature": 0.5,

"max\_tokens": 1024

}

response = requests.post(GEMINI\_API\_URL, headers=headers, json=payload)

if response.status\_code == 200:

return response.json()["content"] # Adjust based on the actual response structure

else:

print(f"Error: {response.status\_code} - {response.text}")

return ""

# Function to generate array of subtopic objects with content

def generate\_subtopic\_array(subject):

subtopic\_array = []

for subtopic in subject["subtopics"]:

card\_prefix = f"{subject['topic']}-{subtopic['name']}"

for content\_type in AI\_PROMPTS.keys():

if content\_type in subtopic:

# Combine AI\_PROMPTS and TEXT\_PROMPT for a detailed prompt

prompt = (

TEXT\_PROMPT.format(

max\_objects=6, # Define max number of objects if needed

content\_type=content\_type,

subtopic=subtopic["name"],

skill\_name=subject["skill\_name"],

topic=subject["topic"],

card\_prefix=card\_prefix

) + AI\_PROMPTS[content\_type].replace("[Concept]", subtopic["name"])

)

response = get\_gemini\_response(prompt)

# Append content based on content\_type to subtopic array

if content\_type == "analogy":

subtopic\_array.append({

"analogy": {

"cardId": f"{card\_prefix}\_analogy",

"render": [

{"element": "analogy", "about": f"An analogy explaining {subtopic['name']}", "content": response}

]

}

})

elif content\_type == "description":

subtopic\_array.append({

"description": {

"cardId": f"{card\_prefix}\_description",

"render": [

{"element": "description", "about": f"Description of {subtopic['name']}", "content": response}

]

}

})

elif content\_type == "codeSnippet":

subtopic\_array.append({

"codeSnippet": {

"cardId": f"{card\_prefix}\_codeSnippet",

"render": [

{"element": "codeSnippet", "about": f"Code example for {subtopic['name']}", "content": response}

]

}

})

elif content\_type == "funfacts":

subtopic\_array.append({

"funfacts": {

"cardId": f"{card\_prefix}\_funfacts",

"render": [

{"element": "facts", "about": f"Fun facts about {subtopic['name']}", "content": response}

]

}

})

elif content\_type == "quiz":

subtopic\_array.append({

"quiz": {

"cardId": f"{card\_prefix}\_quiz",

"render": [

{"element": "quiz", "about": f"Quiz on {subtopic['name']}", "content": response}

]

}

})

elif content\_type == "assignment":

subtopic\_array.append({

"assignment": {

"cardId": f"{card\_prefix}\_assignment",

"render": [

{"element": "assignment", "about": f"Assignment for {subtopic['name']}", "content": response}

]

}

})

return subtopic\_array

# Sample subject input to test the function

subject = {

"topic": "GPU Computing",

"skill\_name": "Advanced Computing Skills",

"subtopics": [

{"name": "Parallel Processing", "analogy": True, "description": True, "codeSnippet": True, "funfacts": True, "quiz": True, "assignment": True},

{"name": "CUDA Programming", "description": True, "codeSnippet": True, "quiz": True},

]

}

# Generate the subtopic array with responses

subtopic\_cards = generate\_subtopic\_array(subject)

# Make a GET request to the Flask backend

response = requests.get('http://127.0.0.1:5000/response')

if response.status\_code == 200:

backend\_response = response.json()

print(backend\_response['response'])

else:

print('Error fetching response from backend')

# Print the generated cards for inspection

for card in subtopic\_cards:

print(card)