Assignment: Building a Prompt-Driven LLM Evaluation Framework with Gemini API and MLflow

🎯 **Level: Advanced |** ⏱️ **Duration: 4–6 hours**

🗂️ **Technologies: Python, Gemini 1.5 Flash API (Free Tier), MLflow, JSON, Object-Oriented Programming, Streamlit (optional), Prompt Engineering**

🔍 Problem Statement

You are working as a PromptOps Engineer for a Generative AI startup. Your task is to **design, test, evaluate, and version prompt templates** for various use cases (text summarization, code debugging, math reasoning, and multimodal captioning) using the **Gemini API**.

You must **modularize the solution using Python classes and functions**, and track **prompt latency, token usage, output quality (basic heuristics), and versioning** with **MLflow**.

☑️ Assignment Objectives

1. Design zero-shot and few-shot prompt templates for summarization, code completion, math reasoning, and captioning.
2. Build a **modular prompt execution engine** using OOP (classes, functions).
3. Track prompt performance (latency, token usage, response length) using MLflow.
4. Allow prompt reuse using templating with input variables.
5. Evaluate and compare prompt effectiveness using Gemini 1.5 Flash.
6. BONUS: Build a simple Streamlit UI to test live prompts.

🧱 Assignment Structure

🔨 Part 1: Setup & API Integration

- ✅ Create a config file to securely load your Gemini API Key.
- ✅ Create a utility function `call_gemini(prompt, mode='text')` to handle prompt execution and token counting.

🔧 Part 2: Prompt Manager Class

Define a class `PromptManager` with the following methods:

```
class PromptManager:
    def __init__(self, task_type):
        ...
    def load_fewshot_examples(self):
        ...
    def build_prompt(self, user_input):
        ...
    def evaluate_prompt(self, response):
        ...
```

Use subclasses to implement task-specific logic:

class SummarizationPrompt(PromptManager):

class MathQAPrompt(PromptManager):

class CodeDebugPrompt(PromptManager):

class MultimodalCaptionPrompt(PromptManager):

📊 Part 3: MLflow Logging

Use MLflow to log:

- Prompt text
- Task type
- Tokens used
- Response length
- Latency (start to end)
- Prompt version
- Output (as artifact)

```
with mlflow.start_run():
    mlflow.log_param("prompt_type", ...)
    mlflow.log_param("mode", ...)
    mlflow.log_metric("latency", ...)
    mlflow.log_artifact("response.txt")
```

Part 4: Test 5 Use Cases

Test the following prompt styles:

1. **Zero-shot summarization**
2. **Few-shot summarization**
3. **Chain-of-thought for math**
4. **Bug-fix prompt for Python code**
5. **Multimodal caption generation (mock image description)**

Add responses and observations in a CSV file: `prompt_eval_results.csv`.

🚀 Part 5: Bonus (Optional)

- Build a small Streamlit UI to test any of the above prompts live.
- Add dropdown to choose prompt style and task type.

📋 Deliverables

- ✅ Python code: prompt_engine.py, gemini_utils.py, mlflow_tracker.py
- ✅ Sample prompt template files
- ✅ `prompt_eval_results.csv`
- ✅ MLflow tracking screenshot
- ✅ Streamlit UI (optional)
- ✅ README with setup instructions and prompt examples

🧠 Skills Assessed

- Prompt design and reasoning
- Object-oriented programming
- API integration (Gemini)
- Evaluation & tracking using MLflow
- Modular software engineering practices
- (Optional) UI development and prompt testing in real-time