Problem Statement 5: AI-Powered Appointment Scheduler Assistant

Problem Statement:

Build a backend service that parses natural language or document-based appointment requests and converts them into structured scheduling data. The system should handle both typed text and noisy image inputs (e.g., scanned notes, emails). Interns must design a pipeline with OCR, entity extraction, normalization, and final structured JSON output with guardrails for ambiguity.

Solution:

By reading the problem statement, what I understand is that I have:

- **Input:** text, image, or text + image.
- **Processing:** convert any image to base64 (or use as-is) and send it via a POST request to the server. The server then extracts the relevant information — mapping the appointment topic, description, location, date, and time.
- **Output:** a structured JSON containing the reminder/appointment information. This output can then be retrieved by the client using a GET request.

So it's clear that **the input and output format remains consistent across all approaches**; the only difference lies in **how the processing is executed**.

Approach 1: Train Your Own Models

I could collect datasets — or generate synthetic datasets manually using human intelligence — and then train **separate models** for NLP and computer vision:

- NLP could be handled using **LSTMs** or other sequence models.
- Images could be processed using **CNNs** or Keras-based pipelines.
- I could even try a multimodal pipeline combining both.

Problem: I don't have a large or reliable dataset, so the accuracy would likely be very poor. The model might overfit on small synthetic data and generalize poorly in real scenarios.

Approach 2: Use Open-Source Hugging Face Models

I could use large transformer-based models like **Qwen or LLaMA**:

- These give much better results due to pretraining and strong generalization.
- Running them locally, however, requires **huge compute resources**, including GPUs with high memory.
- The cost of continuous operation is high, as inference would be running constantly.

Problem: While accuracy is good, the infrastructure cost is prohibitive for local setups.

Approach 3: Use Gemini-2.5-Flash via API (Current Approach)

A better alternative is to use a smaller **State-of-the-Art Language Model (SLM)** like **Gemini-2.5-Flash** through an **API**:

- Smaller than the giant 27B parameter models, so faster and lighter.
- Can run asynchronously via API, enabling **fast, scalable processing**.
- Provides **SOTA results** with good accuracy and generalization.
- Cost-efficient: I pay only when the API is called, rather than maintaining heavy local infrastructure.

Tech Stack:

- **Language:** Python (JavaScript can also be used)
- **Server Framework:** FastAPI, Pydantic
- **Modeling Framework:** LangChain
- **Model:** Gemini-2.5-Flash

Implementation Notes:

- Followed LangChain documentation for clean, maintainable code.
- Supports **multimodal input** (text + image).
- The system converts images to base64, sends them to the model, and receives structured JSON.

Approach 4 (Future Work): MCP Agent

In the future, I could build a **Microservice or Chat Agent** using this setup:

- Deploy it as a **serverless agent** accessible via WhatsApp, PushAI, or similar platforms.
- This would remove the need for a frontend.
- Could also integrate other functionalities, similar to the **YouTube video summarization agent** I built earlier.
- The agent would work like a personal assistant, handling scheduling and reminders seamlessly.

Summary

- Input/output remains constant; approaches differ in **execution and infrastructure**.
- Approach 1: DIY models → data limitations → low accuracy.
- Approach 2: Large Hugging Face models → high compute → costly.
- Approach 3 (current): Gemini-2.5 Flash API → SOTA, fast, cost-efficient.
- Approach 4 (future): Agent-based deployment → fully automated, frontend-less, handy for multiple channels.

DISCLAIMER (by Pranav Mittal) : AI is used for clean and maintainable code and doc. Code Logic and the words and ideas in doc are entirely mine.