■ TASK SUMMARY:
Update the backend (Flask) to support multiple free fallback AI options.
If OpenAI is unavailable or quota exceeded, automatically use Gemini, Hugging Face, or local mock logic.

■ Implementation requirements:

1■■ ENV Setup
- Use environment variables in `.env`:
```
MODEL=auto
GEMINI_API_KEY=your_google_api_key_here (optional)
OPENROUTER_API_KEY=optional
DATABASE_URL=postgresql://postgres:YOUR_PASSWORD@localhost:5432/minddecode
SESSION_SECRET=random-long-string
```
- Do NOT depend on any paid OpenAI key.
- If no valid AI key is found, fallback gracefully to local mock output.

2■■ Backend Logic (Flask)
- Create a helper module `ai_utils.py` that automatically decides which AI to use:
  Priority:
    a. Gemini (if GEMINI_API_KEY available)
    b. OpenRouter (if OPENROUTER_API_KEY available)
    c. Hugging Face free inference (no key)
    d. Local mock generator (fallback)

- Each model should accept a “prompt” and return a “response” string.

Example flow:
```
from ai_utils import generate_ai_response

@app.route("/predict_exam", methods=["POST"])
def predict_exam():
    data = request.json
    prompt = data.get("prompt", "")
    response = generate_ai_response(prompt)
    return jsonify({"response": response})
```

3■■ ai_utils.py logic example:

If Gemini API key exists → use requests.post("https://generativelanguage.googleapis.com/v1beta/models/gemini-pro:generate

If Hugging Face → use pipeline("text-generation", model="distilgpt2")

If nothing → simulate:

```
import random
topics = ["Data Structures", "Thermodynamics", "Control Systems", "Networks"]
return f"Predicted Topics: {random.sample(topics, 3)}"
```