Final Course Project AI Agent Creation

Objective This capstone project requires student groups to design, implement, and demonstrate an AI agent system that applies key concepts covered throughout the course. The project will provide hands-on experience building and evaluating AI agent systems that incorporate reinforcement learning techniques with language-based planning and decision-making.

Group Members: Objective

- Jemima Egwurube
- Thien Nam Nguyen
- Williane YarroWilliane Yarro

Phase 1: Environment Setup

1. Install Dependencies

```
# Install core AI and utility libraries
!pip install --quiet openai requests google-generativeai

# Install web framework dependencies (Flask, CORS, etc.)
!pip install flask flask-cors shapely requests
```

```
Requirement already satisfied: flask in /usr/local/lib/python3.12/dist-packages (3.1.2)
Collecting flask-cors
  Downloading flask_cors-6.0.1-py3-none-any.whl.metadata (5.3 kB)
Requirement already satisfied: shapely in /usr/local/lib/python3.12/dist-packages (2.1.2)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (2.32.4)
Requirement already satisfied: blinker>=1.9.0 in /usr/local/lib/python3.12/dist-packages (from flask) (1.9.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.12/dist-packages (from flask) (8.3.1)
Requirement already satisfied: itsdangerous>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from flask) (2.
Requirement already satisfied: jinja2>=3.1.2 in /usr/local/lib/python3.12/dist-packages (from flask) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from flask) (3.0.
Requirement already satisfied: werkzeug>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from flask) (3.1.3)
Requirement already satisfied: numpy>=1.21 in /usr/local/lib/python3.12/dist-packages (from shapely) (2.0.2)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reque
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests) (
Downloading flask_cors-6.0.1-py3-none-any.whl (13 kB)
Installing collected packages: flask-cors
Successfully installed flask-cors-6.0.1
```

```
!git clone https://github.com/joelleyarro03/Flood-Agent-App.git

# Move into the project folder
%cd Flood-Agent-App

# Install dependencies
!pip install -r requirements.txt
!pip install openai

```

```
Cloning into 'Flood-Agent-App'...
remote: Enumerating objects: 51, done.
remote: Counting objects: 100% (51/51), done.
remote: Compressing objects: 100% (46/46), done.
remote: Total 51 (delta 3), reused 50 (delta 3), pack-reused 0 (from 0)
Receiving objects: 100% (51/51), 32.31 KiB | 4.04 MiB/s, done.
Resolving deltas: 100% (3/3), done.
/content/Flood-Agent-App
Requirement already satisfied: Flask in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt (lin
```

```
Requirement already satisfied: flask-cors in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt (
Requirement already satisfied: shapely in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt (l
Requirement already satisfied: cachetools in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt
Requirement already satisfied: blinker>=1.9.0 in /usr/local/lib/python3.12/dist-packages (from Flask->-r requi
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.12/dist-packages (from Flask->-r require
Requirement already satisfied: itsdangerous>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from Flask->-r
Requirement already satisfied: jinja2>=3.1.2 in /usr/local/lib/python3.12/dist-packages (from Flask->-r requir
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from Flask->-r re
Requirement already satisfied: werkzeug>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from Flask->-r requ
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reque
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->-r requ
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->-
Requirement already satisfied: numpy>=1.21 in /usr/local/lib/python3.12/dist-packages (from shapely->-r requir
Requirement already satisfied: openai in /usr/local/lib/python3.12/dist-packages (2.8.1)
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from openai) (4.11.
Requirement already satisfied: distro<2,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from openai) (1.9.
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from openai) (0.28
Requirement already satisfied: jiter<1,>=0.10.0 in /usr/local/lib/python3.12/dist-packages (from openai) (0.12
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.12/dist-packages (from openai) (2.
Requirement already satisfied: sniffio in /usr/local/lib/python3.12/dist-packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.12/dist-packages (from openai) (4.67.1)
Requirement already satisfied: typing-extensions<5,>=4.11 in /usr/local/lib/python3.12/dist-packages (from ope
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.12/dist-packages (from anyio<5,>=3.5.0->ope
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->open
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydanti
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/python3.12/dist-packages (from pydantic
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from pydan
```

```
1 !pip install -r requirements.txt openai
```

```
Requirement already satisfied: openai in /usr/local/lib/python3.12/dist-packages (2.8.1)
Requirement already satisfied: Flask in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt (lin
Requirement already satisfied: flask-cors in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt (
Requirement already satisfied: shapely in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt (l
Requirement already satisfied: cachetools in /usr/local/lib/python3.12/dist-packages (from -r requirements.txt
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from openai) (4.11.
Requirement already satisfied: distro<2,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from openai) (1.9.
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from openai) (0.28
Requirement already satisfied: jiter<1,>=0.10.0 in /usr/local/lib/python3.12/dist-packages (from openai) (0.12
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.12/dist-packages (from openai) (2.
Requirement already satisfied: sniffio in /usr/local/lib/python3.12/dist-packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.12/dist-packages (from openai) (4.67.1)
Requirement already satisfied: typing-extensions<5,>=4.11 in /usr/local/lib/python3.12/dist-packages (from ope
Requirement already satisfied: blinker>=1.9.0 in /usr/local/lib/python3.12/dist-packages (from Flask->-r requi
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.12/dist-packages (from Flask->-r require
Requirement already satisfied: itsdangerous>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from Flask->-r
Requirement already satisfied: jinja2>=3.1.2 in /usr/local/lib/python3.12/dist-packages (from Flask->-r requir
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from Flask->-r re
Requirement already satisfied: werkzeug>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from Flask->-r requ
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reque
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->-r requ
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->-
Requirement already satisfied: numpy>=1.21 in /usr/local/lib/python3.12/dist-packages (from shapely->-r requir
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydanti
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/python3.12/dist-packages (from pydantic
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from pydan
```

2. Setup Project Structure

```
1 import os
2
```

```
# Create necessary directories for the project structure
os.makedirs('services', exist_ok=True)
os.makedirs('routes', exist_ok=True)

print("✅ Project directories 'services' and 'routes' created.")
```

```
✅ Project directories 'services' and 'routes' created.
```

3. Configure API Key Here we set the Google Gemini API key

```
import os
from google.colab import userdata

# 1. Load Gemini Key
try:
    gemini_key = userdata.get('GEMINI_API_KEY')
    os.environ["GEMINI_API_KEY"] = gemini_key
    print("✅ Gemini key loaded.")
except Exception:
    print("⚠️ GEMINI_API_KEY not found in Secrets.")

# 2. Load OpenAI Key (NEW)
try:
    openai_key = userdata.get('OPENAI_API_KEY')
    os.environ["OPENAI_API_KEY"] = openai_key
    print("✅ OpenAI key loaded.")
except Exception:
    print("⚠️ OPENAI_API_KEY not found in Secrets.")
```

```
🔐 Gemini key loaded securely from Colab Secrets!
```

How to Add the Key to Secrets You need to add it to the Colab interface:

1. Look at the left sidebar of Google Colab and click the Key icon (🔑) called "Secrets".
2. Click "Add new secret".
3. Name: Enter GEMINI_API_KEY || OPENAI_API_KEY (must match the code exactly).
4. Value: Paste your actual API key (starting with AIza...).
5. Important: Toggle the "Notebook access" switch next to the secret to ON (blue).

Now, when you run the cell above, it will safely load your key.

Phase 2: The "Brain" (Agent)

4. Create services/agent.py This is the core logic of your application. It contains three main components:

1. Tools: The get_nws_hourly_forecast function fetches real weather data.
2. Memory: A simple in-memory storage (AGENT_MEMORY) to track interaction history and adjust "risk sensitivity" based on feedback.
3. The Agent: The run_flood_agent_two_calls function orchestrates the logic—fetching data, asking Gemini for a risk assessment, and then asking Gemini to explain it to a resident.

```
%%writefile services/agent.py
import os
import json
import time
import random
import re
from typing import Dict, Any, List, Optional
```

```python

import requests
import google.generativeai as genai
import openai  # <--- NEW IMPORT

# ---------------------------------------------------------------------
# CONFIGURATION
# ---------------------------------------------------------------------
GEMINI_API_KEY = os.getenv("GEMINI_API_KEY")
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") # <--- NEW KEY

if GEMINI_API_KEY:
    genai.configure(api_key=GEMINI_API_KEY)

# ---------------------------------------------------------------------
# HELPER: LLM SWITCHER (NEW FUNCTION)
# ---------------------------------------------------------------------
def _call_llm(provider: str, system_msg: str, user_msg: str) -> str:
    """Surgical abstraction to switch between providers."""

    # OPTION A: OPENAI
    if provider.lower() == "openai":
        if not OPENAI_API_KEY:
            return "Error: OPENAI_API_KEY not set."

        client = openai.OpenAI(api_key=OPENAI_API_KEY)
        try:
            response = client.chat.completions.create(
                model="gpt-4o",  # or "gpt-3.5-turbo"
                messages=[
                    {"role": "system", "content": system_msg},
                    {"role": "user", "content": user_msg}
                ],
                temperature=0.7
            )
            return response.choices[0].message.content
        except Exception as e:
            return f"OpenAI Error: {e}"

    # OPTION B: GEMINI (Default)
    else:
        if not GEMINI_API_KEY:
            return "Error: GEMINI_API_KEY not set."

        try:
            model = genai.GenerativeModel("gemini-2.5-flash", system_instruction=system_msg)
            response = model.generate_content(user_msg)
            return response.text
        except Exception as e:
            return f"Gemini Error: {e}"

# ---------------------------------------------------------------------
# TOOL: NWS hourly forecast API (Unchanged)
# ---------------------------------------------------------------------
NWS_HEADERS = {
    "User-Agent": "Panacea Flood Agent (student-project@example.com)",
    "Accept": "application/geo+json",
}

def get_nws_hourly_forecast(lat: float, lon: float, hours: int = 6) -> Dict[str, Any]:
    try:
        points_url = f"https://api.weather.gov/points/{lat},{lon}"
        resp_points = requests.get(points_url, headers=NWS_HEADERS, timeout=10)
        resp_points.raise_for_status()
        points_data = resp_points.json()

        hourly_url = points_data["properties"]["forecastHourly"]
        resp_hourly = requests.get(hourly_url, headers=NWS_HEADERS, timeout=10)
        resp_hourly.raise_for_status()
        hourly_data = resp_hourly.json()

        periods = hourly_data.get("properties", {}).get("periods", [])
        if not periods:
            return {"error": "No hourly forecast periods returned from NWS"}

```

```
        simplified = []
        for p in periods[:hours]:
            simplified.append({
                "startTime": p.get("startTime"),
                "temperature": p.get("temperature"),
                "precipChance": (p.get("probabilityOfPrecipitation", {}) or {}).get("value"),
                "shortForecast": p.get("shortForecast"),
                "windSpeed": p.get("windSpeed"),
            })

        return {"data": simplified}

    except Exception as e:
        return {"error": f"NWS tool failed: {e}"}

# ---------------------------------------------------------------------
# MEMORY & FALLBACKS (Unchanged)
# ---------------------------------------------------------------------
AGENT_MEMORY: Dict[str, Any] = {
    "interactions": [],
    "feedback_history": [],
    "risk_sensitivity": 0.5,
}

def _store_interaction(record: Dict[str, Any]) -> None:
    AGENT_MEMORY["interactions"].append(record)
    if len(AGENT_MEMORY["interactions"]) > 100:
        AGENT_MEMORY["interactions"].pop(0)

def register_feedback(payload: Dict[str, Any]) -> Dict[str, Any]:
    score = int(payload.get("score", 3))
    score = max(1, min(5, score))
    reward = score - 3

    old_sensitivity = AGENT_MEMORY["risk_sensitivity"]
    new_sensitivity = max(0.2, min(0.8, old_sensitivity + (0.05 * reward)))
    AGENT_MEMORY["risk_sensitivity"] = new_sensitivity

    return {"ok": True, "policy": {"risk_sensitivity": new_sensitivity}}

def _fallback_predict_flood(lat: float, lon: float, hours: int) -> Dict[str, Any]:
    avg_temp = round(random.uniform(20.0, 30.0), 1)
    max_precip_prob = round(random.uniform(0.1, 0.9), 2)
    sensitivity = AGENT_MEMORY["risk_sensitivity"]

    if max_precip_prob > sensitivity + 0.25:
        trend = "Conditions suggest a higher chance of heavy rain."
        advice = "Avoid low-lying roads."
    else:
        trend = "No strong indication of heavy rainfall."
        advice = "Normal travel is likely fine."

    return {
        "summary": {"hours": hours, "avg_temp": avg_temp, "max_precip_prob": max_precip_prob, "will_precip": max_precip_pr
        "trend": trend,
        "advice": advice,
        "source": "fallback_simulation",
    }

def _predict_flood_from_nws(lat: float, lon: float, hours: int) -> Dict[str, Any]:
    nws_result = get_nws_hourly_forecast(lat, lon, hours)
    if "error" in nws_result:
        base = _fallback_predict_flood(lat, lon, hours)
        base["tool_error"] = nws_result["error"]
        return base

    periods = nws_result["data"]
    precip_values = [p.get("precipChance") or 0 for p in periods]
    max_precip_prob = (max(precip_values) / 100.0) if precip_values else 0.0
    avg_temp = sum((p.get("temperature") or 0) for p in periods) / max(len(periods), 1)

    sensitivity = AGENT_MEMORY["risk_sensitivity"]
    if max_precip_prob > sensitivity:
        trend = "Forecast suggests periods of rain."
        advice = "Stay weather-aware near bayous."
```

```
    else:
        trend = "Forecast does not show a strong signal for heavy rain."
        advice = "Normal activity is likely fine."

    return {
        "summary": {"hours": hours, "avg_temp": round(avg_temp, 1), "max_precip_prob": round(max_precip_prob, 2), "will_pr
        "trend": trend,
        "advice": advice,
        "source": "nws_hourly_forecast",
    }

# ---------------------------------------------------------------------
# MAIN AGENT FUNCTION (UPDATED)
# ---------------------------------------------------------------------
def run_flood_agent_two_calls(
    lat: float,
    lon: float,
    hours: int,
    location_name: str = "Unknown",
    mode: Optional[str] = None,
    llm_provider: str = "gemini"  # <--- NEW ARGUMENT
) -> Dict[str, Any]:

    if mode == "sim_high":
        base = _fallback_predict_flood(lat, lon, hours)
        base["summary"]["max_precip_prob"] = 0.9
        base["trend"] = "Simulated scenario: heavy, persistent rain."
    else:
        base = _predict_flood_from_nws(lat, lon, hours)

    summary = base["summary"]
    risk_sensitivity = AGENT_MEMORY["risk_sensitivity"]

    # Call 1: Assessment (JSON)
    system_msg_1 = "You are a flood risk analyst. Respond with STRICT JSON only."
    user_msg_1 = f"Forecast Summary: {summary}. Risk sensitivity: {risk_sensitivity}. Generate JSON with keys: risk_level,

    # USE HELPER INSTEAD OF DIRECT CALL
    assessment_text = _call_llm(llm_provider, system_msg_1, user_msg_1)

    try:
        match = re.search(r"```json\n([\s\S]*?)\n```", assessment_text)
        model_assessment = json.loads(match.group(1) if match else assessment_text)
    except Exception as e:
        model_assessment = {"risk_level": "unknown", "justification": f"Failed parsing: {e} | Raw: {assessment_text}"}

    # Call 2: Explanation (Natural Language)
    system_msg_2 = "You are a calm flood-risk explainer. This is a simulation."
    user_msg_2 = f"Summary: {summary}. Assessment: {model_assessment}. Write a clear explanation for a resident."

    # USE HELPER INSTEAD OF DIRECT CALL
    explanation = _call_llm(llm_provider, system_msg_2, user_msg_2)

    result = {
        "base": base,
        "model_assessment": model_assessment,
        "explanation": explanation,
        "meta": {
            "tools_used": {llm_provider: True}, # <--- REFLECTS CHOICE
            "risk_sensitivity": risk_sensitivity
        }
    }
    _store_interaction({"lat": lat, "result": result})
    return result

def run_panacea_flood_insight(lat, lon, hours=6, location_name="Houston", neighborhood=None, mode=None, llm_provider="gemi
    # Updated wrapper to pass the provider
    core = run_flood_agent_two_calls(lat, lon, hours, location_name, mode, llm_provider=llm_provider)
    core["explanation"] = "This summary is provided by Panacea's Passage...\n\n" + core["explanation"]
    return core
```

```
Overwriting services/agent.py
```

Phase 3: The API (Route Layer)

5. Create routes/agent.py This file handles the web requests. It uses Flask Blueprints to organize routes. It receives data from the web (latitude, longitude) and passes it to the Service layer we defined above.

```
%%writefile routes/agent.py
from flask import Blueprint, request, jsonify
from services.agent import run_flood_agent_two_calls, register_feedback, run_panacea_flood_insight

bp = Blueprint("agent", __name__)

@bp.get("/agent/predict")
def agent_predict_route():
    try:
        lat = float(request.args.get("lat"))
        lon = float(request.args.get("lon"))
        hours = int(request.args.get("hours", 6))
        location = request.args.get("location", "Unknown")

        # Capture the provider from URL (default to gemini)
        provider = request.args.get("provider", "gemini")

    except:
        return jsonify({"error": "Invalid inputs"}), 400

    # Pass the provider to the core function
    result = run_flood_agent_two_calls(lat, lon, hours, location_name=location, llm_provider=provider)
    return jsonify(result)

@bp.post("/agent/feedback")
def agent_feedback_route():
    data = request.get_json(silent=True) or {}
    return jsonify(register_feedback(data))

@bp.get("/panacea/risk")
def panacea_risk_route():
    try:
        lat = float(request.args.get("lat"))
        lon = float(request.args.get("lon"))
        provider = request.args.get("provider", "gemini") # Capture here too
    except:
        return jsonify({"error": "Invalid lat/lon"}), 400

    result = run_panacea_flood_insight(
        lat=lat,
        lon=lon,
        hours=int(request.args.get("hours", 6)),
        location_name=request.args.get("location", "Houston"),
        mode=request.args.get("mode"),
        llm_provider=provider # Pass it down
    )
    return jsonify(result)
```

```
Overwriting routes/agent.py
```

Phase 4: The Application Entry

6. Create app.py This is the main entry point. It creates the Flask application and registers the routes (Blueprints).

```
%%writefile app.py
from flask import Flask
from flask_cors import CORS

# Import all existing blueprints
from routes.health import bp as health_bp
from routes.flood import bp as flood_bp
from routes.hospital import bp as hospital_bp
from routes.geocode import bp as geocode_bp
```

```
from routes.geocode import bp as geocode_bp

# NEW agent route
from routes.agent import bp as agent_bp

def create_app():
    app = Flask(__name__)
    CORS(app)

    # Register blueprints
    app.register_blueprint(health_bp, url_prefix="/api")
    app.register_blueprint(flood_bp, url_prefix="/api")
    app.register_blueprint(hospital_bp, url_prefix="/api")
    app.register_blueprint(agent_bp, url_prefix="/api")   # NEW
    app.register_blueprint(geocode_bp, url_prefix="/api")

    @app.get("/")
    def root():
        return {"ok": True, "app": "Panacea backend"}

    return app

if __name__ == "__main__":
    app = create_app()
    app.run(port=3000, debug=True)
```

```
Overwriting app.py
```

Phase 5: Execution & Testing

7. Run a Test Prediction Now that we have written the files, we can import the module and run a test without starting the whole web server. This confirms that the Agent logic, NWS API connection, and Gemini integration are working.

```
import sys
import os

# Add the current working directory to sys.path
# This is crucial for Python to find modules like 'services.agent'
# when the 'services' directory is a subdirectory of the current working directory.
current_dir = os.getcwd()
if current_dir not in sys.path:
    sys.path.insert(0, current_dir)

import importlib
import services.agent
importlib.reload(services.agent)

# Now get the function from the reloaded module
run_flood_agent_two_calls = services.agent.run_flood_agent_two_calls

# Example: Houston coordinates
lat = 29.76
lon = -95.37
hours = 6
location_name = "Houston"

result = run_flood_agent_two_calls(lat, lon, hours, location_name)

print("Keys in result:", result.keys())
print("\nMeta info:")
print(result["meta"])

print("\nModel assessment:")
print(result["model_assessment"])

print("\nExplanation (first 1200 chars):")
print(result["explanation"][:1200])
```

```
WARNING:tornado.access:429 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Benum-encoding%3D
Keys in result: dict_keys(['base', 'model_assessment', 'explanation', 'meta'])

Meta info:
{'tools_used': {'gemini': True}, 'risk_sensitivity': 0.5}

Model assessment:
{'risk_level': 'unknown', 'justification': 'Failed parsing: Expecting value: line 1 column 1 (char 0) | Raw: G

Explanation (first 1200 chars):
Hello there. Here's a calm update on the situation over the next few hours.

For the next 6 hours, we're expecting mild weather with an average temperature around 58 degrees Fahrenheit. C

Now, regarding our official flood risk assessment, our system encountered a temporary technical issue. This me

Please be assured, our technical team is actively addressing this to get the system back online swiftly. We wi

Based on the clear weather forecast, the immediate outlook remains calm. We will provide the full, system-gene
```