

Ex.No.6 – Development of Python Code Compatible with Multiple AI Tools

Ranen Joseph Solomon
Register Number: 212223060265
Date: 18/10/2025

Abstract—This experiment focuses on designing Python code that interacts with multiple AI systems and external APIs to compare outputs and generate analytical insights. Two powerful AI tools—ChatGPT (GPT-5) and Google Gemini (Gemini 1.5 Pro)—were prompted to generate Python programs for retrieving weather data, comparing multi-API responses, and producing data-driven recommendations. The generated outputs were analyzed for structure, reliability, exception handling, and production readiness. This report documents the complete process, comparative outputs, and observations over a multi-stage evaluation.

I. AIM

To write and implement Python code that integrates with multiple AI tools for automating interactions with external APIs, comparing the generated outputs, and analyzing their practical usability across different models.

II. AI TOOLS REQUIRED

- ChatGPT (OpenAI GPT-5)
- Google Gemini (Gemini 1.5 Pro / Pro Vision)
- Python 3.x
- External APIs: OpenWeatherMap and WeatherAPI

III. EXPLANATION

The experiment follows the persona pattern: “*Act as a Python Programmer.*” The same prompt was provided to both ChatGPT and Gemini to generate code for:

- 1) Fetching weather data from multiple APIs
- 2) Comparing API outputs
- 3) Generating insights from the comparison

This enables an objective evaluation of code style, robustness, architecture, and reliability.

IV. TASK 1: GENERATE PYTHON CODE FOR INTERACTING WITH MULTIPLE APIs

A. Prompt

Act as a Python programmer. Write Python code that fetches current weather data for Chennai from two different APIs—OpenWeatherMap and WeatherAPI. Display the temperature, humidity, and condition from both APIs.

B. ChatGPT (GPT-5) Output

```
import requests

openweathermap_key = "YOUR_OPENWEATHER_KEY"
weatherapi_key = "YOUR_WEATHERAPI_KEY"
city = "Chennai"

url1 = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={openweathermap_key}&units=metric"
data1 = requests.get(url1).json()

url2 = f"http://api.weatherapi.com/v1/current.json?key={weatherapi_key}&q={city}"
data2 = requests.get(url2).json()

ow_temp = data1['main']['temp']
ow_humidity = data1['main']['humidity']
ow_condition = data1['weather'][0]['description']

wa_temp = data2['current']['temp_c']
wa_humidity = data2['current']['humidity']
wa_condition = data2['current']['condition']['text']

print("---- OpenWeatherMap ----")
print(f"Temperature: {ow_temp} C , Humidity: {ow_humidity}%, Condition: {ow_condition}")

print("---- WeatherAPI ----")
print(f"Temperature: {wa_temp} C , Humidity: {wa_humidity}%, Condition: {wa_condition}")
```

1) Analysis of ChatGPT Output:

- Simple and functional
- Suitable for beginner-level demonstration
- No exception handling
- No modular design
- Quick for prototyping

C. Gemini Output (Partial)

```
def get_openweathermap_data(city, api_key):
    params = {"q": city, "appid": api_key, "units": "metric"}
    try:
        response = requests.get(OWM_BASE_URL, params=params, timeout=10)
        response.raise_for_status()
        data = response.json()
        if data.get("cod") != 200:
```

```

        return None
    return {
        "Temperature": f"{data['main']['temp']} C",
        "Humidity": f"{data['main']['humidity']}%",
        "Condition": data["weather"][0]["description"].capitalize()
    }
except Exception:
    return None

```

I) Analysis of Gemini Output:

- Highly modular structure
- Robust exception handling
- Timeout included
- Production-ready formatting
- Clear function definitions

V. TASK 2: PYTHON CODE TO COMPARE TWO API OUTPUTS

A. Prompt

Now write a Python function that compares the results of the two APIs (temperature, humidity, condition) and prints the differences clearly.

B. ChatGPT (GPT-5) Output

```

def compare_weather(ow, wa):
    temp_diff = abs(ow['temp'] - wa['temp'])
    hum_diff = abs(ow['humidity'] - wa['humidity'])

    print("---- Comparison Report ---")
    print(f"Temperature difference: {temp_diff} C")
    print(f"Humidity difference: {hum_diff}%")

    if ow['condition'] != wa['condition']:
        print(f"Condition mismatch: {ow['condition']} vs {wa['condition']}")
    else:
        print("Both APIs report similar conditions.")

```

I) Analysis:

- Easy to understand
- Suitable for conceptual demonstration
- Limited comparison logic

C. Gemini Output (Comparison Module)

```

def compare_weather_data(owm_data, wa_data):
    metrics = ["Temperature", "Humidity", "Condition"]
    print("Metric | OpenWeather | WeatherAPI | Status")
    for metric in metrics:
        owm = owm_data.get(metric)
        wa = wa_data.get(metric)
        status = "MATCH" if owm == wa else "DIFFERENCE"
        print(f"{metric} | {owm} | {wa} | {status}")

```

I) Analysis:

- Tabular comparison format
- Handles missing keys
- Produces clear audit-friendly output

VI. TASK 3: GENERATE INSIGHTS FROM COMPARISON

A. ChatGPT Insight Code

```

def generate_insights(temp_diff, hum_diff):
    if temp_diff < 1 and hum_diff < 5:
        print("Both APIs are consistent.")
    else:
        print("Variation detected. Averaging recommended.")

```

B. Gemini Insight Code (Partial)

```

if temp_difference >= 2.0:
    print("Large temperature difference detected.")
    print(f"Suggested average temperature: {avg_temp:.1f} C")

```

I) Analysis:

- Numeric extraction from strings
- Threshold-based evaluation
- Advisory suggestions

VII. COMPARISON TABLE: CHATGPT VS GEMINI CODE QUALITY

Criteria	ChatGPT	Gemini
Modularity	Basic	Advanced
Error Handling	Minimal	Comprehensive
Readability	High	High
Production Usability	Medium	Very High
Complexity Handling	Moderate	High

VIII. DISCUSSION

The same prompt yields different styles of Python code depending on the AI model. ChatGPT tends to produce concise and beginner-friendly code ideal for quick implementations or educational usage. Gemini, however, leans toward production-grade outputs with robust error handling, structured formatting, and system-level considerations.

This demonstrates the diversity in AI-assisted programming workflows and highlights the importance of selecting the appropriate model based on project requirements.

IX. CONCLUSION

The experiment successfully achieved its objective of generating Python code compatible with multiple AI systems and comparing their outputs. Both ChatGPT and Gemini produced functional programs, but Gemini's version was more aligned with professional development standards. The comparative analysis demonstrates how prompt quality and model architecture influence code generation.

X. RESULT

The corresponding prompt was executed successfully, and the Python code comparison across multiple AI tools was completed.

XI. REFERENCES

- OpenWeatherMap API Documentation
- WeatherAPI Documentation
- OpenAI GPT-5 Developer Guide
- Google Gemini 1.5 Pro API Reference