EXP-06 DATE	Development of Python Code Compatible with Multiple AI Tools

KRISHNA PRAKAASH D M - 212221230052

Aim

To write and implement Python code that integrates with multiple AI tools to automate the task of interacting with APIs, comparing outputs, and generating actionable insights.

Procedure:

Python code to create a system that interacts with multiple AI tools, compares outputs, and generates actionable insights. The example assumes you're working with two hypothetical AI tools (AI_Tool_1 and AI_Tool_2) through APIs, comparing their outputs on text analysis tasks and generating insights based on these comparisons.

Code Overview

- **Step 1:** Interact with APIs of multiple AI tools.
- **Step 2:** Process and compare the results.
- **Step 3:** Generate insights based on the comparison.

Code Implementation

```
import requests

class AIAutomation:
    def __init__(self, api_keys):
        Initialize with API keys for each AI tool.
        self.api_keys = api_keys
        self.api_urls = {
            'AI_Tool_1': 'https://api.aitool1.com/analyze',
            'AI_Tool_2': 'https://api.aitool2.com/analyze'
        }
```

```
def call_ai_tool(self, tool_name, text):
    Calls the specified AI tool API with the given text.
    url = self.api_urls.get(tool_name)
    headers = {'Authorization': f'Bearer {self.api_keys[tool_name]}'}
    payload = {'text': text}
    try:
        response = requests.post(url, headers=headers, json=payload)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print (f"Error calling {tool_name}: {e}")
        return None
def analyze_text(self, text):
    Analyzes the text using multiple AI tools and generates actionable insights.
    results = {}
            for tool in self.api_urls.keys():
        results[tool] = self.call_ai_tool(tool, text)
    insights = self.generate_insights(results)
    return insights


def generate_insights(self, results):
    Generates insights based on comparison of AI tools' outputs.
    insights = {}
    if results['AI_Tool_1'] and results['AI_Tool_2']:
        sentiment1 = results['AI_Tool_1']. get ('sentiment')
        sentiment2 = results['AI_Tool_2']. get ('sentiment')
```

```
        if sentiment1 == sentiment2:
            insights ['Sentiment Analysis'] = f"Both tools agree on sentiment: {sentiment1}"
        else:
            insights ['Sentiment Analysis'] = f"Tools disagree on sentiment. AI_Tool_1: {sentiment1}, AI_Tool_2: {sentiment2}"
        topics1 = set(results['AI_Tool_1'].get('key_topics', []))
        topics2 = set(results['AI_Tool_2'].get('key_topics', []))
        common_topics = topics1.intersection(topics2)
        unique_to_tool_1 = topics1 - topics2
        unique_to_tool_2 = topics2 - topics1
        insights ['Key Topics'] = {
            'Common Topics': list(common_topics),
            'Unique to AI_Tool_1': list(unique_to_tool_1),
            'Unique to AI_Tool_2': list(unique_to_tool_2)
        }
    return insights
api_keys = {
  'AI_Tool_1': 'YOUR_API_KEY_FOR_TOOL_1',
  'AI_Tool_2': 'YOUR_API_KEY_FOR_TOOL_2'
}

ai_automation = AIAutomation(api_keys)
text_to_analyze = "OpenAI has been at the forefront of AI innovation and research."
insights = ai_automation.analyze_text(text_to_analyze)
print(insights)
```

Explanation of Each Component

- **call_ai_tool method:** Interacts with each AI tool using API calls, passing in text data for analysis.
- **analyze_text method:** Executes the analysis by calling multiple AI tools and gathering their outputs.
- **generate_insights method:** Compares the outputs from each AI tool. For example, it checks if the sentiment analysis results align and compares key topics extracted from each tool, categorizing them as common or unique.

Extending the Functionality

- **Automated Scheduling:** Implement cron jobs or use Python's schedule library to automate text analysis at specified intervals.
- **Error Handling and Logging:** Add robust error handling, especially for network errors, and log results for future reference.
- **Data Visualization:** Use libraries like matplotlib or plotly to visualize insights if you need graphical representations of results.

Purpose of the Code

The code is designed to:

1. **Automate Interactions with AI Tools:** Integrate with APIs of different AI services, removing the need for manual input.
2. **Compare Outputs:** Analyze and evaluate the results from multiple AI tools, providing a balanced perspective.
3. **Generate Actionable Insights:** Process and extract meaningful conclusions to aid decision-making or deeper analysis.

Details About Each Component

1. API Integration

- The call_ai_tool function is a generic wrapper to interact with AI APIs. It:
    - Uses requests for HTTP POST requests.
    - Passes the input text to the AI API.
    - Processes and returns the JSON response.

Enhancements:

- Add support for GET requests if required by specific APIs.
- Include retry logic using requests.adapters.Retry for better resilience.

2. Data Analysis and Comparison

- **Sentiment Analysis Example:**
  - Tools may classify text into positive, neutral, or negative sentiments.
  - If two tools agree, you get a more confident result; if not, you may need to dig deeper into why.
- **Key Topics Example:**
  - Extracts topics or keywords from responses.
  - Compares overlaps (common topics) and differences (unique topics to each tool).

Enhancements:

- Compare additional fields like entity recognition, summarization, or topic classification.
- Introduce weights for tool reliability (e.g., prioritize one tool for sentiment but another for keyword extraction).

3. Insights Generation

- The generate_insights function aggregates data into actionable conclusions:
  - Agreement or disagreement between tools is highlighted.
  - Unique contributions from each tool are listed.

4.Possible Use Cases

1. **Market Research:**
   - Analyze customer feedback from multiple tools to understand sentiment and trends.
2. **Content Moderation:**

    - Use different tools to validate flagged content for bias-free moderation.
3. **Text Classification:**
    - Enhance NLP-based classification by combining strengths of various tools.
4. **Educational Insights:**
    - Compare AI-based summaries of academic articles for better understanding.

5.Enhancements and Future Additions

1. Multi-Tool Integration

- Add support for more AI tools like OpenAI's GPT API, AWS Comprehend, or Google Cloud NLP.
- Use environment variables (os.environ) to manage sensitive API keys securely.

2. Parallel Processing

- For large-scale tasks, send requests to APIs asynchronously using libraries like asyncio or aiohttp.

3. Result Storage and Visualization

- Store results in databases (e.g., PostgreSQL, MongoDB) for future analysis.
- Visualize insights using matplotlib, seaborn, or dash for graphical dashboards.

4. Modular Design

- Split functionality into different classes or files for scalability:
    - **APIToolConnector**: Handles API communication.
    - **AIResultComparator**: Compares and processes outputs.
    - **InsightsGenerator**: Aggregates and formats insights.

6. Error Handling and Logging

- Use Python’s logging module for detailed logs (e.g., errors, response times, and API health).

- Handle rate limits (e.g., add delays when APIs reject requests due to excessive calls).

Example Workflow for a Practical Scenario

Analyzing Product Reviews:

1. Feed customer reviews into multiple AI tools for:
    - Sentiment Analysis.
    - Key Phrase Extraction.
    - Language Detection.
2. Compare:
    - Sentiment consistency across tools.
    - Overlaps in identified key phrases.
    - Discrepancies for manual review.
3. Generate:
    - Insights like "Positive reviews are consistent across tools but differ on key highlights."