Ex.No:06
Date:16.10.24

Development of Python Code Compatible with Multiple AI Tools.

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.

To create a Python-based solution that integrates multiple AI tools, interacts with APIs, compares outputs, and generates actionable insights, we need to design a system with the following components:

1. **API Interaction**: Access and fetch data from external APIs.
2. **AI Tools Integration**: Use AI models or APIs to process or analyze the data.
3. **Comparison and Analysis**: Compare results from multiple sources.
4. **Actionable Insights Generation**: Derive and present insights.

Python Code

```
import requests import

json

from transformers import pipeline


# Example AI tools and APIs integration

# Using Hugging Face Transformers for text analysis and OpenAI API for summarization.


def fetch_api_data(api_url, headers=None, params=None):

  """Fetch data from an API."""

try:

    response = requests.get(api_url, headers=headers, params=params)

response.raise_for_status()      return response.json()    except

requests.exceptions.RequestException as e:

    print(f"Error fetching data from API: {e}")

return None
```

```
def analyze_with_transformers(data, task="sentiment-analysis"):
"""Analyze data using Hugging Face Transformers pipeline."""
try:
    analyzer = pipeline(task)
results = analyzer(data)
return results     except
Exception as e:
    print(f"Error analyzing with Transformers: {e}")
return None


def summarize_with_openai(api_key, text, model="text-davinci-003"):
   """Summarize text using OpenAI GPT API."""
url = "https://api.openai.com/v1/completions"
headers = {
    "Authorization": f"Bearer {api_key}",
    "Content-Type": "application/json"
   }    data
= {
    "model": model,
    "prompt": f"Summarize the following text:\n{text}",
    "temperature": 0.7,
    "max_tokens": 150
   }
try:
    response = requests.post(url, headers=headers, json=data)
response.raise_for_status()
    return response.json().get("choices")[0].get("text").strip()
except requests.exceptions.RequestException as e:    print(f"Error
with OpenAI API: {e}")    return None
```

```
def compare_outputs(output1, output2):
"""Compare outputs from different tools."""     if
output1 == output2:
        return "The outputs are consistent."
    return f"Discrepancies found:\nTool 1 Output: {output1}\nTool 2 Output: {output2}"


def generate_insights(outputs):
    """Generate actionable insights based on processed outputs."""
insights = []     for output in outputs:          if "positive" in
output.lower():
            insights.append("Positive sentiment detected. Consider emphasizing this in messaging.")
elif "negative" in output.lower():
            insights.append("Negative sentiment detected. Mitigation strategies may be required.")
else:
            insights.append("Neutral sentiment. No immediate action required.")
return insights


#      Example      usage      if
__name__ == "__main__":
    # Step 1: Fetch data from a public API (e.g., a news API)
api_url = "https://jsonplaceholder.typicode.com/posts"     data
= fetch_api_data(api_url)


    # Step 2: Process data with AI tools
    if data:
        text_data = [post['body'] for post in data[:3]]  # Analyzing the first 3 posts
        transformers_output = analyze_with_transformers(text_data, task="sentiment-analysis")


        #    OpenAI    API    Key    (replace    with    your    own    key)
openai_api_key = "your_openai_api_key_here"
```

```
    openai_summaries = [summarize_with_openai(openai_api_key, text) for text in text_data]

    # Step 3: Compare outputs
    for i, (transformers_result, summary) in enumerate(zip(transformers_output, openai_summaries)):
        print(f"\nComparison for Text {i + 1}:")
        print(compare_outputs(transformers_result, summary))

    # Step 4: Generate actionable insights
    insights = generate_insights([result['label'] for result in transformers_output])
print("\nGenerated Insights:") for insight in insights: print(f"-
{insight}")
```

Explanation

1. **API Data Fetching**: ○ fetch_api_data: Makes GET requests to APIs to fetch data.
2. **AI Tools Integration**:
   - ○ analyze_with_transformers: Uses Hugging Face's pipeline for sentiment analysis.
   - ○ summarize_with_openai: Uses OpenAI's GPT API to summarize text.
3. **Comparison of Outputs**: ○ compare_outputs: Checks for consistency or discrepancies between outputs.
4. **Generating Insights**:
   - ○ generate_insights: Analyzes outputs to provide actionable insights.

Execution

- Replace your_openai_api_key_here with a valid OpenAI API key.
- Ensure the necessary libraries (requests, transformers) are installed.
- Run the script to see the interaction of API data with AI tools, comparison of outputs, and insights generation.

This modular approach allows you to integrate additional APIs or tools seamlessly for more advanced workflows.

Conclusion

The Python-based solution demonstrates how to seamlessly integrate multiple AI tools and APIs to automate tasks such as data fetching, analysis, comparison, and actionable insight generation. By combining public APIs, Hugging Face's Transformers library, and OpenAI's GPT, the system offers a robust framework for handling diverse data processing needs.

The key achievements include:

1. **Efficiency**: Automated interaction with APIs and AI models reduces manual effort and speeds up analysis.
2. **Versatility**: Modular design allows for easy integration of additional AI tools and APIs for diverse applications.
3. **Insights Generation**: Combines outputs from different AI models to deliver meaningful and actionable insights.

This approach highlights the potential of leveraging AI in tandem with external APIs to create intelligent systems for various domains, including business analytics, content creation, and datadriven decision-making.