```
import os

from swarm_models import OpenAIChat
from swarms import Agent
from fluid_api_agent.main import fluid_api_request
from dotenv import load_dotenv


load_dotenv()

# Get the OpenAI API key from the environment variable
api_key = os.getenv("GROQ_API_KEY")

# Model
model = OpenAIChat(
    openai_api_base="https://api.groq.com/openai/v1",
    openai_api_key=api_key,
    model_name="llama-3.1-70b-versatile",
    temperature=0.1,
)


def omni_api(task: str) -> str:
    """
    Omni API Function: Calls any API dynamically based on the task description.
```

This function leverages the `fluid_api_request` method to process a given task and make the necessary API call dynamically. It is designed to be highly flexible, allowing users to interact with a wide variety of APIs without needing predefined configurations.

Parameters:

---

task : str

A descriptive string outlining the API call or task to be performed.

The description should include enough detail for `fluid_api_request`

to determine the appropriate API endpoint, request type, and payload.

Returns:

---

dict

A dictionary containing the response data from the API call.

The structure of the response will vary based on the API being accessed.

Raises:

ValueError

If the task string is insufficiently descriptive or cannot be mapped

to a valid API request.

HTTPError

If the API call results in an HTTP error (e.g., 404 Not Found, 500 Server Error).

Examples:

---

1. Call a weather API to fetch the current weather for a city:

```
task = "Fetch the current weather for New York City"
response = omni_api(task)
print(response)
```

2. Retrieve stock prices for a specific company:

```
task = "Get the latest stock price for Apple Inc."
response = omni_api(task)
print(response)
```

3. Post a message to a Slack channel:

```
task = "Post 'Hello, Team!' to the #general channel in Slack"
response = omni_api(task)
print(response)
```

Notes:

- The `fluid_api_request` function must be implemented to interpret the `task` string and handle API calls accordingly.
- Security and authentication for APIs should be managed within `fluid_api_request`.

```
    """
    return str(fluid_api_request(task))


# Define the system prompt tailored for the API expert
API_AGENT_SYS_PROMPT = """
You are a highly specialized financial API expert.
Your expertise lies in analyzing financial data, making investment recommendations, and
interacting with APIs to retrieve, process, and present data effectively.
You use tools like 'omni_api' to fetch data dynamically, ensuring accuracy and up-to-date results.

Instructions:
1. Always query relevant APIs to gather insights for tasks.
2. When suggesting investments, ensure a diversified portfolio based on the user's budget, risk
appetite, and growth potential.
3. Verify API responses and retry calls if necessary to ensure data accuracy.
"""

# Customize the agent for financial API tasks
agent = Agent(
    agent_name="API-Finance-Expert",
        agent_description="An API expert agent specialized in financial analysis and investment
planning.",
    system_prompt=API_AGENT_SYS_PROMPT,
    max_loops=1,  # Allow a few iterations for refining outputs
    llm=model,
```

```
    dynamic_temperature_enabled=True,  # Enable temperature adjustments for optimal creativity
    user_name="swarms_corp",
    retry_attempts=5,  # Retry API calls to ensure reliability
    context_length=8192,  # Context length for comprehensive analysis
    return_step_meta=False,
    output_type="str",  # Output tables or results in markdown format
    auto_generate_prompt=False,  # Use the custom system prompt for guidance
    max_tokens=4000,
    saved_state_path="api_finance_expert.json",
    tools=[omni_api],  # Integrate the omni_api tool
)

# Run the agent with a financial task
agent.run(
    "Fetch the current price for eth",
    all_cores=True,  # Utilize all processing cores for efficiency
)
```