# OpenAI Assistant

The OpenAI Assistant class provides a wrapper around OpenAI's Assistants API, integrating it with the swarms framework.

## Overview

The `OpenAIAssistant` class allows you to create and interact with OpenAI Assistants, providing a simple interface for:

- Creating assistants with specific roles and capabilities
- Adding custom functions that the assistant can call
- Managing conversation threads
- Handling tool calls and function execution
- Getting responses from the assistant

## Insstallation

```bash
pip install swarms
```

## Basic Usage

```python

from swarms import OpenAIAssistant
```

#Create an assistant
assistant = OpenAIAssistant(
    name="Math Tutor",
    instructions="You are a helpful math tutor.",
    model="gpt-4o",
    tools=[{"type": "code_interpreter"}]
)

#Run a Task
response = assistant.run("Solve the equation: 3x + 11 = 14")
print(response)

# Continue the conversation in the same thread
follow_up = assistant.run("Now explain how you solved it")
print(follow_up)
~~~

Function Calling

The assistant supports custom function integration:

```python

def get_weather(location: str, unit: str = "celsius") -> str:
    # Mock weather function
~~~

```
    return f"The weather in {location} is 22 degrees {unit}"

Add function to assistant
assistant.add_function(
    description="Get the current weather in a location",
    parameters={
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "City name"
            },
            "unit": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"],
                "default": "celsius"
            }
        },
        "required": ["location"]
    }
)
```

## API Reference

### Constructor

```python
OpenAIAssistant(
    name: str,
    instructions: Optional[str] = None,
    model: str = "gpt-4o",
    tools: Optional[List[Dict[str, Any]]] = None,
    file_ids: Optional[List[str]] = None,
    metadata: Optional[Dict[str, Any]] = None,
    functions: Optional[List[Dict[str, Any]]] = None,
)
```

### Methods

#### run(task: str) -> str

Sends a task to the assistant and returns its response. The conversation thread is maintained between calls.

#### add_function(func: Callable, description: str, parameters: Dict[str, Any]) -> None

Adds a callable function that the assistant can use during conversations.

#### add_message(content: str, file_ids: Optional[List[str]] = None) -> None

Adds a message to the current conversation thread.

## Error Handling

The assistant implements robust error handling:

- Retries on rate limits
- Graceful handling of API errors
- Clear error messages for debugging
- Status monitoring for runs and completions

## Best Practices

1. Thread Management
   - Use the same assistant instance for related conversations
   - Create new instances for unrelated tasks
   - Monitor thread status during long-running operations

2. Function Integration
   - Keep functions simple and focused
   - Provide clear descriptions and parameter schemas
   - Handle errors gracefully in custom functions
   - Test functions independently before integration

3. Performance
   - Reuse assistant instances when possible
   - Monitor and handle rate limits appropriately
   - Use appropriate polling intervals for status checks
   - Consider implementing timeouts for long-running operations

## References

- [OpenAI Assistants API Documentation](https://platform.openai.com/docs/assistants/overview)

- [OpenAI Function Calling Guide](https://platform.openai.com/docs/guides/function-calling)

- [OpenAI Rate Limits](https://platform.openai.com/docs/guides/rate-limits)