```
import datetime
from datetime import timedelta
from unittest.mock import Mock

import pytest
from dotenv import load_dotenv

from swarm_models.gpt4_vision_api import GPT4VisionAPI
from swarms.prompts.multi_modal_autonomous_instruction_prompt import (
    MULTI_MODAL_AUTO_AGENT_SYSTEM_PROMPT_1,
)
from swarms.structs.agent import Agent
from swarms.structs.task import Task

load_dotenv()


@pytest.fixture
def llm():
    return GPT4VisionAPI()


def test_agent_run_task(llm):
    task = (
        "Analyze this image of an assembly line and identify any"
        " issues such as misaligned parts, defects, or deviations"
        " from the standard assembly process. IF there is anything"
        " unsafe in the image, explain why it is unsafe and how it"
        " could be improved."
    )
    img = "assembly_line.jpg"

    agent = Agent(
        llm=llm,
        max_loops="auto",
        sop=MULTI_MODAL_AUTO_AGENT_SYSTEM_PROMPT_1,
        dashboard=True,
    )

    result = agent.run(task=task, img=img)

    # Add assertions here to verify the expected behavior of the agent's run method
    assert isinstance(result, dict)
    assert "response" in result
    assert "dashboard_data" in result
    # Add more assertions as needed



@pytest.fixture
def task():
    agents = [Agent(llm=llm, id=f"Agent_{i}") for i in range(5)]
    return Task(
        id="Task_1", task="Task_Name", agents=agents, dependencies=[]
    )


# Basic tests


def test_task_init(task):
    assert task.id == "Task_1"
    assert task.task == "Task_Name"
    assert isinstance(task.agents, list)
    assert len(task.agents) == 5
    assert isinstance(task.dependencies, list)


def test_task_execute(task, mocker):
    mocker.patch.object(Agent, "run", side_effect=[1, 2, 3, 4, 5])
    parent_results = {}
    task.execute(parent_results)
    assert isinstance(task.results, list)
    assert len(task.results) == 5
    for result in task.results:
        assert isinstance(result, int)


# Parameterized tests
```

```
@pytest.mark.parametrize("num_agents", [1, 3, 5, 10])

def test_task_num_agents(task, num_agents, mocker):

    task.agents = [Agent(id=f"Agent_{i}") for i in range(num_agents)]

    mocker.patch.object(Agent, "run", return_value=1)

    parent_results = {}

    task.execute(parent_results)

    assert len(task.results) == num_agents



# Exception testing



def test_task_execute_with_dependency_error(task, mocker):

    task.dependencies = ["NonExistentTask"]

    mocker.patch.object(Agent, "run", return_value=1)

    parent_results = {}

    with pytest.raises(KeyError):

        task.execute(parent_results)



# Mocking and monkeypatching tests



def test_task_execute_with_mocked_agents(task, mocker):
    mock_agents = [Mock(spec=Agent) for _ in range(5)]
    mocker.patch.object(task, "agents", mock_agents)
    for mock_agent in mock_agents:
        mock_agent.run.return_value = 1
    parent_results = {}
    task.execute(parent_results)
    assert len(task.results) == 5


def test_task_creation():
    agent = Agent()
    task = Task(id="1", task="Task1", result=None, agents=[agent])
    assert task.id == "1"
    assert task.task == "Task1"
    assert task.result is None
    assert task.agents == [agent]


def test_task_with_dependencies():
    agent = Agent()
    task = Task(
        id="2",
        task="Task2",
        result=None,
        agents=[agent],
        dependencies=["Task1"],
    )
    assert task.dependencies == ["Task1"]


def test_task_with_args():
    agent = Agent()
    task = Task(
        id="3",
        task="Task3",
        result=None,
        agents=[agent],
        args=["arg1", "arg2"],
    )
    assert task.args == ["arg1", "arg2"]


def test_task_with_kwargs():
    agent = Agent()
    task = Task(
        id="4",
        task="Task4",
        result=None,
        agents=[agent],
        kwargs={"kwarg1": "value1"},
    )
    assert task.kwargs == {"kwarg1": "value1"}
```

```
# ... continue creating tests for different scenarios


# Test execute method
def test_execute():
    agent = Agent()
    task = Task(id="5", task="Task5", result=None, agents=[agent])
    # Assuming execute method returns True on successful execution
    assert task.run() is True


def test_task_execute_with_agent(mocker):
    mock_agent = mocker.Mock(spec=Agent)
    mock_agent.run.return_value = "result"
    task = Task(description="Test task", agent=mock_agent)
    task.run()
    assert task.result == "result"
    assert task.history == ["result"]


def test_task_execute_with_callable(mocker):
    mock_callable = mocker.Mock()
    mock_callable.run.return_value = "result"
    task = Task(description="Test task", agent=mock_callable)
    task.run()
    assert task.result == "result"
    assert task.history == ["result"]


def test_task_execute_with_condition(mocker):
    mock_agent = mocker.Mock(spec=Agent)
    mock_agent.run.return_value = "result"
    condition = mocker.Mock(return_value=True)
    task = Task(
        description="Test task", agent=mock_agent, condition=condition
    )
    task.run()
    assert task.result == "result"
    assert task.history == ["result"]


def test_task_execute_with_condition_false(mocker):
    mock_agent = mocker.Mock(spec=Agent)
    mock_agent.run.return_value = "result"
    condition = mocker.Mock(return_value=False)
    task = Task(
        description="Test task", agent=mock_agent, condition=condition
    )
    task.run()
    assert task.result is None
    assert task.history == []


def test_task_execute_with_action(mocker):
    mock_agent = mocker.Mock(spec=Agent)
    mock_agent.run.return_value = "result"
    action = mocker.Mock()
    task = Task(
        description="Test task", agent=mock_agent, action=action
    )
    task.run()
    assert task.result == "result"
    assert task.history == ["result"]
    action.assert_called_once()


def test_task_handle_scheduled_task_now(mocker):
    mock_agent = mocker.Mock(spec=Agent)
    mock_agent.run.return_value = "result"
    task = Task(
        description="Test task",
        agent=mock_agent,
        schedule_time=datetime.now(),
    )
    task.handle_scheduled_task()
    assert task.result == "result"
    assert task.history == ["result"]


def test_task_handle_scheduled_task_future(mocker):
    mock_agent = mocker.Mock(spec=Agent)
    mock_agent.run.return_value = "result"
    task = Task(
        description="Test task",
        agent=mock_agent,
        schedule_time=datetime.now() + timedelta(days=1),
    )
    with mocker.patch.object(
        task.scheduler, "enter"
    ) as mock_enter, mocker.patch.object(
        task.scheduler, "run"
    ) as mock_run:
        task.handle_scheduled_task()
    mock_enter.assert_called_once()
    mock_run.assert_called_once()


def test_task_set_trigger():
    task = Task(description="Test task", agent=Agent())

    def trigger():
        return True
    task.set_trigger(trigger)

    assert task.trigger == trigger



def test_task_set_action():

    task = Task(description="Test task", agent=Agent())


    def action():

        return True


    task.set_action(action)

    assert task.action == action



def test_task_set_condition():

    task = Task(description="Test task", agent=Agent())


    def condition():

        return True


    task.set_condition(condition)

    assert task.condition == condition
```