```
import unittest
from unittest.mock import patch
from swarms import create_agents_from_yaml
import os


class TestCreateAgentsFromYaml(unittest.TestCase):

    def setUp(self):
        # Mock the environment variable for API key
        os.environ["OPENAI_API_KEY"] = "fake-api-key"

        # Mock agent configuration YAML content
        self.valid_yaml_content = """
        agents:
          - agent_name: "Financial-Analysis-Agent"
            model:
              openai_api_key: "fake-api-key"
              model_name: "gpt-4o-mini"
              temperature: 0.1
              max_tokens: 2000
            system_prompt: "financial_agent_sys_prompt"
            max_loops: 1
            autosave: true
            dashboard: false
            verbose: true
```

```
  dynamic_temperature_enabled: true
  saved_state_path: "finance_agent.json"
  user_name: "swarms_corp"
  retry_attempts: 1
  context_length: 200000
  return_step_meta: false
  output_type: "str"
  task: "How can I establish a ROTH IRA to buy stocks and get a tax break?"

- agent_name: "Stock-Analysis-Agent"
  model:
    openai_api_key: "fake-api-key"
    model_name: "gpt-4o-mini"
    temperature: 0.2
    max_tokens: 1500
  system_prompt: "stock_agent_sys_prompt"
  max_loops: 2
  autosave: true
  dashboard: false
  verbose: true
  dynamic_temperature_enabled: false
  saved_state_path: "stock_agent.json"
  user_name: "stock_user"
  retry_attempts: 3
  context_length: 150000
  return_step_meta: true
```

```
        output_type: "json"
        task: "What is the best strategy for long-term stock investment?"
    """

@patch(
    "builtins.open",
    new_callable=unittest.mock.mock_open,
    read_data="",
)
@patch("yaml.safe_load")
def test_create_agents_return_agents(
    self, mock_safe_load, mock_open
):
    # Mock YAML content parsing
    mock_safe_load.return_value = {
        "agents": [
            {
                "agent_name": "Financial-Analysis-Agent",
                "model": {
                    "openai_api_key": "fake-api-key",
                    "model_name": "gpt-4o-mini",
                    "temperature": 0.1,
                    "max_tokens": 2000,
                },
                "system_prompt": "financial_agent_sys_prompt",
                "max_loops": 1,
```

```
            "autosave": True,
            "dashboard": False,
            "verbose": True,
            "dynamic_temperature_enabled": True,
            "saved_state_path": "finance_agent.json",
            "user_name": "swarms_corp",
            "retry_attempts": 1,
            "context_length": 200000,
            "return_step_meta": False,
            "output_type": "str",
            "task": "How can I establish a ROTH IRA to buy stocks and get a tax break?",
        }
    ]
}

# Test if agents are returned correctly
agents = create_agents_from_yaml(
    "fake_yaml_path.yaml", return_type="agents"
)
self.assertEqual(len(agents), 1)
self.assertEqual(
    agents[0].agent_name, "Financial-Analysis-Agent"
)

@patch(
    "builtins.open",
```

```
    new_callable=unittest.mock.mock_open,
    read_data="",
)
@patch("yaml.safe_load")
@patch(
    "swarms.Agent.run", return_value="Task completed successfully"
)
def test_create_agents_return_tasks(
    self, mock_agent_run, mock_safe_load, mock_open
):
    # Mock YAML content parsing
    mock_safe_load.return_value = {
        "agents": [
            {
                "agent_name": "Financial-Analysis-Agent",
                "model": {
                    "openai_api_key": "fake-api-key",
                    "model_name": "gpt-4o-mini",
                    "temperature": 0.1,
                    "max_tokens": 2000,
                },
                "system_prompt": "financial_agent_sys_prompt",
                "max_loops": 1,
                "autosave": True,
                "dashboard": False,
                "verbose": True,
```

```
            "dynamic_temperature_enabled": True,
            "saved_state_path": "finance_agent.json",
            "user_name": "swarms_corp",
            "retry_attempts": 1,
            "context_length": 200000,
            "return_step_meta": False,
            "output_type": "str",
            "task": "How can I establish a ROTH IRA to buy stocks and get a tax break?",
        }
    ]
}

# Test if tasks are executed and results are returned
task_results = create_agents_from_yaml(
    "fake_yaml_path.yaml", return_type="tasks"
)
self.assertEqual(len(task_results), 1)
self.assertEqual(
    task_results[0]["agent_name"], "Financial-Analysis-Agent"
)
self.assertIsNotNone(task_results[0]["output"])

@patch(
    "builtins.open",
    new_callable=unittest.mock.mock_open,
    read_data="",
```

```
)
@patch("yaml.safe_load")
def test_create_agents_return_both(
    self, mock_safe_load, mock_open
):
    # Mock YAML content parsing
    mock_safe_load.return_value = {
        "agents": [
            {
                "agent_name": "Financial-Analysis-Agent",
                "model": {
                    "openai_api_key": "fake-api-key",
                    "model_name": "gpt-4o-mini",
                    "temperature": 0.1,
                    "max_tokens": 2000,
                },
                "system_prompt": "financial_agent_sys_prompt",
                "max_loops": 1,
                "autosave": True,
                "dashboard": False,
                "verbose": True,
                "dynamic_temperature_enabled": True,
                "saved_state_path": "finance_agent.json",
                "user_name": "swarms_corp",
                "retry_attempts": 1,
                "context_length": 200000,
```

```
            "return_step_meta": False,
            "output_type": "str",
            "task": "How can I establish a ROTH IRA to buy stocks and get a tax break?",
        }
      ]
    }

    # Test if both agents and tasks are returned
    agents, task_results = create_agents_from_yaml(
        "fake_yaml_path.yaml", return_type="both"
    )
    self.assertEqual(len(agents), 1)
    self.assertEqual(len(task_results), 1)
    self.assertEqual(
        agents[0].agent_name, "Financial-Analysis-Agent"
    )
    self.assertIsNotNone(task_results[0]["output"])

@patch(
    "builtins.open",
    new_callable=unittest.mock.mock_open,
    read_data="",
)
@patch("yaml.safe_load")
def test_missing_agents_in_yaml(self, mock_safe_load, mock_open):
    # Mock YAML content with missing "agents" key
```

```
    mock_safe_load.return_value = {}

    # Test if the function raises an error for missing "agents" key
    with self.assertRaises(ValueError) as context:
        create_agents_from_yaml(
            "fake_yaml_path.yaml", return_type="agents"
        )
    self.assertTrue(
        "The YAML configuration does not contain 'agents'."
        in str(context.exception)
    )

@patch(
    "builtins.open",
    new_callable=unittest.mock.mock_open,
    read_data="",
)
@patch("yaml.safe_load")
def test_invalid_return_type(self, mock_safe_load, mock_open):
    # Mock YAML content parsing
    mock_safe_load.return_value = {
        "agents": [
            {
                "agent_name": "Financial-Analysis-Agent",
                "model": {
                    "openai_api_key": "fake-api-key",
```

```
            "model_name": "gpt-4o-mini",
            "temperature": 0.1,
            "max_tokens": 2000,
        },
        "system_prompt": "financial_agent_sys_prompt",
        "max_loops": 1,
        "autosave": True,
        "dashboard": False,
        "verbose": True,
        "dynamic_temperature_enabled": True,
        "saved_state_path": "finance_agent.json",
        "user_name": "swarms_corp",
        "retry_attempts": 1,
        "context_length": 200000,
        "return_step_meta": False,
        "output_type": "str",
        "task": "How can I establish a ROTH IRA to buy stocks and get a tax break?",
    }
  ]
}

# Test if an error is raised for invalid return_type
with self.assertRaises(ValueError) as context:
    create_agents_from_yaml(
        "fake_yaml_path.yaml", return_type="invalid_type"
    )
```

```
        self.assertTrue(
            "Invalid return_type" in str(context.exception)
        )



if __name__ == "__main__":
    unittest.main()
```