# Multi-Agent Examples

### `SequentialWorkflow`

Sequential Workflow enables you to sequentially execute tasks with `Agent` and then pass the output into the next agent and onwards until you have specified your max loops.

```python
from swarms import Agent, SequentialWorkflow

from swarm_models import Anthropic

# Initialize the language model agent (e.g., GPT-3)
llm = Anthropic()

# Initialize agents for individual tasks
agent1 = Agent(
    agent_name="Blog generator",
    system_prompt="Generate a blog post like stephen king",
    llm=llm,
    max_loops=1,
    dashboard=False,
    tools=[],
)
agent2 = Agent(
    agent_name="summarizer",
    system_prompt="Sumamrize the blog post",
    llm=llm,
    max_loops=1,
    dashboard=False,
    tools=[],
)

# Create the Sequential workflow
workflow = SequentialWorkflow(
    agents=[agent1, agent2], max_loops=1, verbose=False
)

# Run the workflow
workflow.run(
    "Generate a blog post on how swarms of agents can help businesses grow."
)

```

## `AgentRearrange`

Inspired by Einops and einsum, this orchestration techniques enables you to map out the relationships between various agents. For example you specify linear and sequential relationships like `a -> a1 -> a2 -> a3` or concurrent relationships where the first agent will send a message to 3 agents all at once: `a -> a1, a2, a3`. You can customize your workflow to mix sequential and concurrent relationships. [Docs Available:](https://docs.swarms.world/en/latest/swarms/structs/agent_rearrange/)

```python
from swarms import Agent, AgentRearrange

from swarm_models import Anthropic

# Initialize the director agent

director = Agent(
    agent_name="Director",
    system_prompt="Directs the tasks for the workers",
    llm=Anthropic(),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    state_save_file_type="json",
    saved_state_path="director.json",
)
```

```
# Initialize worker 1

worker1 = Agent(
    agent_name="Worker1",
    system_prompt="Generates a transcript for a youtube video on what swarms are",
    llm=Anthropic(),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    state_save_file_type="json",
    saved_state_path="worker1.json",
)


# Initialize worker 2
worker2 = Agent(
    agent_name="Worker2",
    system_prompt="Summarizes the transcript generated by Worker1",
    llm=Anthropic(),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    state_save_file_type="json",

    saved_state_path="worker2.json",

)




# Create a list of agents

agents = [director, worker1, worker2]


# Define the flow pattern

flow = "Director -> Worker1 -> Worker2"


# Using AgentRearrange class

agent_system = AgentRearrange(agents=agents, flow=flow)

output = agent_system.run(

    "Create a format to express and communicate swarms of llms in a structured manner for youtube"

)

print(output)

```

## `HierarhicalSwarm`

Coming soon...

## `GraphSwarm`

```python
import os


from dotenv import load_dotenv



from swarms import Agent, Edge, GraphWorkflow, Node, NodeType


from swarm_models import OpenAIChat


load_dotenv()


api_key = os.environ.get("OPENAI_API_KEY")


llm = OpenAIChat(
    temperature=0.5, openai_api_key=api_key, max_tokens=4000
)
agent1 = Agent(llm=llm, max_loops=1, autosave=True, dashboard=True)
agent2 = Agent(llm=llm, max_loops=1, autosave=True, dashboard=True)


def sample_task():
    print("Running sample task")
    return "Task completed"


wf_graph = GraphWorkflow()
wf_graph.add_node(Node(id="agent1", type=NodeType.AGENT, agent=agent1))
wf_graph.add_node(Node(id="agent2", type=NodeType.AGENT, agent=agent2))
wf_graph.add_node(
    Node(id="task1", type=NodeType.TASK, callable=sample_task)
)
wf_graph.add_edge(Edge(source="agent1", target="task1"))
wf_graph.add_edge(Edge(source="agent2", target="task1"))

wf_graph.set_entry_points(["agent1", "agent2"])
wf_graph.set_end_points(["task1"])

print(wf_graph.visualize())

# Run the workflow
results = wf_graph.run()
print("Execution results:", results)

```

## `MixtureOfAgents`

This is an implementation from the paper: "Mixture-of-Agents Enhances Large Language Model Capabilities" by together.ai, it achieves SOTA on AlpacaEval 2.0, MT-Bench and FLASK, surpassing GPT-4 Omni. Great for tasks that need to be parallelized and then sequentially fed into another loop

```python
from swarms import Agent, OpenAIChat, MixtureOfAgents
```

```
# Initialize the director agent
director = Agent(
    agent_name="Director",
    system_prompt="Directs the tasks for the accountants",
    llm=OpenAIChat(),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    state_save_file_type="json",
    saved_state_path="director.json",
)

# Initialize accountant 1
accountant1 = Agent(
    agent_name="Accountant1",
    system_prompt="Prepares financial statements",
    llm=OpenAIChat(),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    state_save_file_type="json",
    saved_state_path="accountant1.json",
)

# Initialize accountant 2
accountant2 = Agent(
    agent_name="Accountant2",
    system_prompt="Audits financial records",
    llm=OpenAIChat(),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
    stopping_token="<DONE>",
    state_save_file_type="json",
    saved_state_path="accountant2.json",
)

# Create a list of agents
agents = [director, accountant1, accountant2]


# Swarm
swarm = MixtureOfAgents(
    name="Mixture of Accountants",
    agents=agents,
    layers=3,
    final_agent=director,
)


# Run the swarm
out = swarm.run("Prepare financial statements and audit financial records")
print(out)
```