# MixtureOfAgents Class Documentation

## Overview

The `MixtureOfAgents` class represents a mixture of agents operating within a swarm. The workflow of the swarm follows a parallel sequential parallel final output agent process. This implementation is inspired by concepts discussed in the paper: https://arxiv.org/pdf/2406.04692.

The class is designed to manage a collection of agents, orchestrate their execution in layers, and handle the final aggregation of their outputs through a designated final agent. This architecture facilitates complex, multi-step processing where intermediate results are refined through successive layers of agent interactions.

## Class Definition

### MixtureOfAgents

```python
class MixtureOfAgents(BaseSwarm):
```

### Attributes

| Attribute | Type | Description | Default |

`agents`	`List[Agent]`	The list of agents in the swarm.	`None`
`flow`	`str`	The flow of the swarm.	`parallel -> sequential -> parallel -> final output agent`
`max_loops`	`int`	The maximum number of loops to run.	`1`
`verbose`	`bool`	Flag indicating whether to print verbose output.	`True`
`layers`	`int`	The number of layers in the swarm.	`3`
`rules`	`str`	The rules for the swarm.	`None`
`final_agent`	`Agent`	The agent to handle the final output processing.	`None`
`auto_save`	`bool`	Flag indicating whether to auto-save the metadata to a file.	`False`
`saved_file_name`	`str`	The name of the file where the metadata will be saved.	`"moe_swarm.json"`

## Methods

`__init__`

#### Parameters

Parameter	Type	Description	Default
`name`	`str`	The name of the swarm.	`"MixtureOfAgents"`
`description`	`str`	A brief description of the swarm.	`"A swarm of agents that run in parallel and sequentially."`
`agents`	`List[Agent]`	The list of agents in the swarm.	`None`
`max_loops`	`int`	The maximum number of loops to run.	`1`
`verbose`	`bool`	Flag indicating whether to print verbose output.	`True`
`layers`	`int`	The number of layers in the swarm.	`3`
`rules`	`str`	The rules for the swarm.	`None`
`final_agent`	`Agent`	The agent to handle the final output processing.	`None`
`auto_save`	`bool`	Flag indicating whether to auto-save the metadata to a file.	`False`
`saved_file_name`	`str`	The name of the file where the metadata will be saved.	`"moe_swarm.json"`

### `agent_check`

```python
def agent_check(self):
```

#### Description

Checks if the provided `agents` attribute is a list of `Agent` instances. Raises a `TypeError` if the validation fails.

#### Example Usage

```python
moe_swarm = MixtureOfAgents(agents=[agent1, agent2])
moe_swarm.agent_check()  # Validates the agents
```

### `final_agent_check`

```python
def final_agent_check(self):
```

#### Description

Checks if the provided `final_agent` attribute is an instance of `Agent`. Raises a `TypeError` if the validation fails.

#### Example Usage

```python
moe_swarm = MixtureOfAgents(final_agent=final_agent)
moe_swarm.final_agent_check()  # Validates the final agent
```

### `swarm_initialization`

```python
def swarm_initialization(self):
```

#### Description

Initializes the swarm by logging the swarm name, description, and the number of agents.

#### Example Usage

```python
moe_swarm = MixtureOfAgents(agents=[agent1, agent2])
moe_swarm.swarm_initialization()  # Initializes the swarm
```

### `run`

```python
def run(self, task: str = None, *args, **kwargs):
```

#### Parameters

Parameter	Type	Description	Default
`task`	`str`	The task to be performed by the swarm.	`None`
`*args`	`tuple`	Additional arguments.	`None`
`**kwargs`	`dict`	Additional keyword arguments.	`None`

#### Returns

Type	Description
`str`	The conversation history as a string.

#### Description

Runs the swarm with the given task, orchestrates the execution of agents through the specified layers, and returns the conversation history.

#### Example Usage

```python
moe_swarm = MixtureOfAgents(agents=[agent1, agent2], final_agent=final_agent)
history = moe_swarm.run(task="Solve this problem.")
print(history)
```

## Detailed Explanation

### Initialization

The `__init__` method initializes the swarm with the provided parameters, sets up the conversation rules, and invokes the initialization of the swarm. It also ensures the validity of the `agents` and `final_agent` attributes by calling the `agent_check` and `final_agent_check` methods respectively.

### Agent Validation

The `agent_check` method validates whether the `agents` attribute is a list of `Agent` instances, while the `final_agent_check` method validates whether the `final_agent` is an instance of `Agent`. These checks are crucial to ensure that the swarm operates correctly with the appropriate agent types.

### Swarm Initialization

The `swarm_initialization` method logs essential information about the swarm, including its name, description, and the number of agents. This provides a clear starting point for the swarm's operations and facilitates debugging and monitoring.

### Running the Swarm

The `run` method is the core of the `MixtureOfAgents` class. It orchestrates the execution of agents through multiple layers, collects their outputs, and processes the final output using the `final_agent`. The conversation history is maintained and updated throughout this process, allowing for a seamless flow of information and responses.

During each layer, the method iterates over the agents, invokes their `run` method with the current conversation history, and logs the outputs. These outputs are then added to the conversation, and the history is updated for the next layer.

After all layers are completed, the final output agent processes the entire conversation history, and the metadata is created and optionally saved to a file. This metadata includes details about the layers, agent runs, and final output, providing a comprehensive record of the swarm's execution.

## Additional Information and Tips

### Common Issues and Solutions

- **Type Errors**: Ensure that all agents in the `agents` list and the `final_agent` are instances of the `Agent` class. The `agent_check` and `final_agent_check` methods help validate this.
- **Verbose Logging**: Use the `verbose` flag to control the verbosity of the output. This can help with debugging or reduce clutter in the logs.

- **Auto-Save Feature**: Utilize the `auto_save` flag to automatically save the metadata to a file. This can be useful for keeping records of the swarm's operations without manual intervention.

### References and Resources

For further reading and background information on the concepts used in the `MixtureOfAgents` class, refer to the paper: https://arxiv.org/pdf/2406.04692.

### Usage Examples

#### Example 1: Basic Initialization and Run

```python
from swarms import MixtureOfAgents, Agent

from swarm_models import OpenAIChat

# Define agents
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",

)

# Initialize the MixtureOfAgents

moe_swarm = MixtureOfAgents(agents=[director, accountant1, accountant2], final_agent=director)

# Run the swarm

history = moe_swarm.run(task="Perform task X.")

print(history)

~~~

Example 2: Verbose Output and Auto-Save

```python

from swarms import MixtureOfAgents, Agent

from swarm_models import OpenAIChat

# Define Agents

# Define agents

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",

)
```

```python
# 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",
)

# Initialize the MixtureOfAgents with verbose output and auto-save enabled
moe_swarm = MixtureOfAgents(
    agents=[director, accountant1, accountant2],
    final_agent=director,
    verbose=True,
    auto_save=True
)

# Run the swarm
history = moe_swarm.run(task="Analyze data set Y.")
print(history)
```

#### Example 3: Custom Rules and Multiple Layers

```python
from swarms import MixtureOfAgents, Agent

from swarm_models import OpenAIChat

# Define agents
# 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",
)

# Initialize the MixtureOfAgents with custom rules and multiple layers
```

```
moe_swarm = MixtureOfAgents(
    agents=[director, accountant1, accountant2],
    final_agent=director,
    layers=5,
    rules="Custom rules for the swarm"
)

# Run the swarm
history = moe_swarm.run(task="Optimize process Z.")
print(history)
```

This comprehensive documentation provides a detailed understanding of the `MixtureOfAgents` class, its attributes, methods, and usage. The examples illustrate how to initialize and run the swarm, demonstrating its flexibility and capability to handle various tasks and configurations.

## Conclusion

The `MixtureOfAgents` class is a powerful and flexible framework for managing and orchestrating a swarm of agents. By following a structured approach of parallel and sequential processing, it enables the implementation of complex multi-step workflows where intermediate results are refined through multiple layers of agent interactions. This architecture is particularly suitable for tasks that require iterative processing, collaboration among diverse agents, and sophisticated aggregation of outputs.

### Key Takeaways

1. **Flexible Initialization**: The class allows for customizable initialization with various parameters, enabling users to tailor the swarm's configuration to their specific needs.
2. **Robust Agent Management**: With built-in validation methods, the class ensures that all agents and the final agent are correctly instantiated, preventing runtime errors and facilitating smooth execution.
3. **Layered Processing**: The layered approach to processing allows for intermediate results to be iteratively refined, enhancing the overall output quality.
4. **Verbose Logging and Auto-Save**: These features aid in debugging, monitoring, and record-keeping, providing transparency and ease of management.
5. **Comprehensive Documentation**: The detailed class and method documentation, along with numerous usage examples, provide a clear and thorough understanding of how to leverage the `MixtureOfAgents` class effectively.

### Practical Applications

The `MixtureOfAgents` class can be applied in various domains, including but not limited to:

- **Natural Language Processing (NLP)**: Managing a swarm of NLP models to process, analyze, and synthesize text.
- **Data Analysis**: Coordinating multiple data analysis agents to process and interpret complex data sets.
- **Optimization Problems**: Running a swarm of optimization algorithms to solve complex problems in fields such as logistics, finance, and engineering.
- **AI Research**: Implementing experimental setups that require the collaboration of multiple AI

models or agents to explore new methodologies and approaches.

### Future Extensions

The `MixtureOfAgents` framework provides a solid foundation for further extensions and customizations, including:

- **Dynamic Layer Configuration**: Allowing layers to be added or removed dynamically based on the task requirements or intermediate results.
- **Advanced Agent Communication**: Enhancing the communication protocols between agents to allow for more sophisticated information exchange.
- **Integration with Other Frameworks**: Seamlessly integrating with other machine learning or data processing frameworks to leverage their capabilities within the swarm architecture.

In conclusion, the `MixtureOfAgents` class represents a versatile and efficient solution for orchestrating multi-agent systems, facilitating complex task execution through its structured and layered approach. By harnessing the power of parallel and sequential processing, it opens up new possibilities for tackling intricate problems across various domains.