SwarmRouter Documentation

The `SwarmRouter` class is a flexible routing system designed to manage different types of swarms for task execution. It provides a unified interface to interact with various swarm types, including `AgentRearrange`, `MixtureOfAgents`, `SpreadSheetSwarm`, `SequentialWorkflow`, `ConcurrentWorkflow`, and finally `auto` which will dynamically select the most appropriate swarm for you by analyzing your name, description, and input task. We will be continuously adding more swarm architectures as we progress with new developments.

Classes

SwarmLog

A Pydantic model for capturing log entries.

Attribute	Type	Description
`id`	str	Unique identifier for the log entry.
`timestamp`	datetime	Time of log creation.
`level`	str	Log level (e.g., "info", "error").
`message`	str	Log message content.
`swarm_type`	SwarmType	Type of swarm associated with the log.
`task`	str	Task being performed (optional).
`metadata`	Dict[str, Any]	Additional metadata (optional).

SwarmRouter

Main class for routing tasks to different swarm types.

Attribute	Type	Description
`name`	str	Name of the SwarmRouter instance.
`description`	str	Description of the SwarmRouter instance.
`max_loops`	int	Maximum number of loops to perform.
`agents`	List[Union[Agent, Callable]]	List of Agent objects or callable functions to be used in the swarm.
`swarm_type`	SwarmType	Type of swarm to be used.
`autosave`	bool	Flag to enable/disable autosave.
`flow`	str	The flow of the swarm.
`return_json`	bool	Flag to enable/disable returning the result in JSON format.
`auto_generate_prompts`	bool	Flag to enable/disable auto generation of prompts.
`swarm`	Union[AgentRearrange, MixtureOfAgents, SpreadSheetSwarm, SequentialWorkflow, ConcurrentWorkflow]	Instantiated swarm object.
`logs`	List[SwarmLog]	List of log entries captured during operations.

Methods:

Method	Parameters	Description
`__init__`	`self, name: str, description: str, max_loops: int, agents: List[Union[Agent, Callable]], swarm_type: SwarmType, autosave: bool, flow: str, return_json: bool, auto_generate_prompts: bool, *args, **kwargs`	Initialize the SwarmRouter.

| `reliability_check` | `self` | Perform reliability checks on the SwarmRouter configuration. |

| `_create_swarm` | `self, task: str = None, *args, **kwargs` | Create and return the specified swarm type or automatically match the best swarm type for a given task. |

| `_log` | `self, level: str, message: str, task: str = "", metadata: Dict[str, Any] = None` | Create a log entry and add it to the logs list. |

| `run` | `self, task: str, *args, **kwargs` | Run the specified task on the selected or matched swarm. |

| `batch_run` | `self, tasks: List[str], *args, **kwargs` | Execute a batch of tasks on the selected or matched swarm type. |

| `threaded_run` | `self, task: str, *args, **kwargs` | Execute a task on the selected or matched swarm type using threading. |

| `async_run` | `self, task: str, *args, **kwargs` | Execute a task on the selected or matched swarm type asynchronously. |

| `get_logs` | `self` | Retrieve all logged entries. |

| `concurrent_run` | `self, task: str, *args, **kwargs` | Execute a task on the selected or matched swarm type concurrently. |

| `concurrent_batch_run` | `self, tasks: List[str], *args, **kwargs` | Execute a batch of tasks on the selected or matched swarm type concurrently. |

Installation

To use the SwarmRouter, first install the required dependencies:

```bash
pip install swarms swarm_models
```

Basic Usage

```python
import os
from dotenv import load_dotenv
from swarms import Agent, SwarmRouter, SwarmType
from swarm_models import OpenAIChat

load_dotenv()

# Get the OpenAI API key from the environment variable
api_key = os.getenv("GROQ_API_KEY")

# Model
model = OpenAIChat(
    openai_api_base="https://api.groq.com/openai/v1",
    openai_api_key=api_key,
    model_name="llama-3.1-70b-versatile",
    temperature=0.1,
)

# Define specialized system prompts for each agent
DATA_EXTRACTOR_PROMPT = """You are a highly specialized private equity agent focused on data extraction from various documents. Your expertise includes:
1. Extracting key financial metrics (revenue, EBITDA, growth rates, etc.) from financial statements and reports
```

2. Identifying and extracting important contract terms from legal documents

3. Pulling out relevant market data from industry reports and analyses

4. Extracting operational KPIs from management presentations and internal reports

5. Identifying and extracting key personnel information from organizational charts and bios

Provide accurate, structured data extracted from various document types to support investment analysis."""

SUMMARIZER_PROMPT = """You are an expert private equity agent specializing in summarizing complex documents. Your core competencies include:

1. Distilling lengthy financial reports into concise executive summaries

2. Summarizing legal documents, highlighting key terms and potential risks

3. Condensing industry reports to capture essential market trends and competitive dynamics

4. Summarizing management presentations to highlight key strategic initiatives and projections

5. Creating brief overviews of technical documents, emphasizing critical points for non-technical stakeholders

Deliver clear, concise summaries that capture the essence of various documents while highlighting information crucial for investment decisions."""

```
# Initialize specialized agents
data_extractor_agent = Agent(
    agent_name="Data-Extractor",
    system_prompt=DATA_EXTRACTOR_PROMPT,
    llm=model,
    max_loops=1,
    autosave=True,
    verbose=True,
```

```
    dynamic_temperature_enabled=True,
    saved_state_path="data_extractor_agent.json",
    user_name="pe_firm",
    retry_attempts=1,
    context_length=200000,
    output_type="string",
)

summarizer_agent = Agent(
    agent_name="Document-Summarizer",
    system_prompt=SUMMARIZER_PROMPT,
    llm=model,
    max_loops=1,
    autosave=True,
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="summarizer_agent.json",
    user_name="pe_firm",
    retry_attempts=1,
    context_length=200000,
    output_type="string",
)

# Initialize the SwarmRouter
router = SwarmRouter(
    name="pe-document-analysis-swarm",
```

```
    description="Analyze documents for private equity due diligence and investment decision-making",
    max_loops=1,
    agents=[data_extractor_agent, summarizer_agent],
    swarm_type="ConcurrentWorkflow",
    autosave=True,
    return_json=True,
)

# Example usage
if __name__ == "__main__":
    # Run a comprehensive private equity document analysis task
    result = router.run(
        "Where is the best place to find template term sheets for series A startups? Provide links and references"
    )
    print(result)

    # Retrieve and print logs
    for log in router.get_logs():
        print(f"{log.timestamp} - {log.level}: {log.message}")
```

Advanced Usage

Changing Swarm Types

You can create multiple SwarmRouter instances with different swarm types:

```python
sequential_router = SwarmRouter(
    name="SequentialRouter",
    agents=[agent1, agent2],
    swarm_type="SequentialWorkflow"
)

concurrent_router = SwarmRouter(
    name="ConcurrentRouter",
    agents=[agent1, agent2],
    swarm_type="ConcurrentWorkflow"
)
```

Automatic Swarm Type Selection

You can let the SwarmRouter automatically select the best swarm type for a given task:

```python
auto_router = SwarmRouter(
    name="AutoRouter",
    agents=[agent1, agent2],
    swarm_type="auto"
```

)

result = auto_router.run("Analyze and summarize the quarterly financial report")
```

## Use Cases

### AgentRearrange

Use Case: Optimizing agent order for complex multi-step tasks.

```python
rearrange_router = SwarmRouter(
    name="TaskOptimizer",
    description="Optimize agent order for multi-step tasks",
    max_loops=3,
    agents=[data_extractor, analyzer, summarizer],
    swarm_type="AgentRearrange",
    flow=f"{data_extractor.name} -> {analyzer.name} -> {summarizer.name}"
)

result = rearrange_router.run("Analyze and summarize the quarterly financial report")
```

MixtureOfAgents
```

Use Case: Combining diverse expert agents for comprehensive analysis.

```python
mixture_router = SwarmRouter(
    name="ExpertPanel",
    description="Combine insights from various expert agents",
    max_loops=1,
    agents=[financial_expert, market_analyst, tech_specialist],
    swarm_type="MixtureOfAgents"
)

result = mixture_router.run("Evaluate the potential acquisition of TechStartup Inc.")
```

SpreadSheetSwarm

Use Case: Collaborative data processing and analysis.

```python
spreadsheet_router = SwarmRouter(
    name="DataProcessor",
    description="Collaborative data processing and analysis",
    max_loops=1,
    agents=[data_cleaner, statistical_analyzer, visualizer],
    swarm_type="SpreadSheetSwarm"
)
```

result = spreadsheet_router.run("Process and visualize customer churn data")
```

### SequentialWorkflow

Use Case: Step-by-step document analysis and report generation.

```python
sequential_router = SwarmRouter(
    name="ReportGenerator",
    description="Generate comprehensive reports sequentially",
    max_loops=1,
    agents=[data_extractor, analyzer, writer, reviewer],
    swarm_type="SequentialWorkflow"
)

result = sequential_router.run("Create a due diligence report for Project Alpha")
```

ConcurrentWorkflow

Use Case: Parallel processing of multiple data sources.

```python
concurrent_router = SwarmRouter(

```
    name="MultiSourceAnalyzer",
    description="Analyze multiple data sources concurrently",
    max_loops=1,
    agents=[financial_analyst, market_researcher, competitor_analyst],
    swarm_type="ConcurrentWorkflow"
)

result = concurrent_router.run("Conduct a comprehensive market analysis for Product X")
```

Auto Select (Experimental)

Autonomously selects the right swarm by conducting vector search on your input task or name or description or all 3.

```python
concurrent_router = SwarmRouter(
    name="MultiSourceAnalyzer",
    description="Analyze multiple data sources concurrently",
    max_loops=1,
    agents=[financial_analyst, market_researcher, competitor_analyst],
      swarm_type="auto" # Set this to 'auto' for it to auto select your swarm. It's match words like concurrently multiple -> "ConcurrentWorkflow"
)

result = concurrent_router.run("Conduct a comprehensive market analysis for Product X")
```

```

## Advanced Features

### Batch Processing

To process multiple tasks in a batch:

```python
tasks = ["Analyze Q1 report", "Summarize competitor landscape", "Evaluate market trends"]
results = router.batch_run(tasks)
```

Threaded Execution

For non-blocking execution of a task:

```python
result = router.threaded_run("Perform complex analysis")
```

Asynchronous Execution

For asynchronous task execution:

```python

result = await router.async_run("Generate financial projections")

```

Concurrent Execution

To run a single task concurrently:

```python
result = router.concurrent_run("Analyze multiple data streams")
```

Concurrent Batch Processing

To process multiple tasks concurrently:

```python
tasks = ["Task 1", "Task 2", "Task 3"]
results = router.concurrent_batch_run(tasks)
```

Best Practices

1. Choose the appropriate swarm type based on your task requirements.
2. Provide clear and specific tasks to the swarm for optimal results.
3. Regularly review logs to monitor performance and identify potential issues.
4. Use descriptive names and descriptions for your SwarmRouter and agents.
```

5. Implement proper error handling in your application code.

6. Consider the nature of your tasks when choosing a swarm type (e.g., use ConcurrentWorkflow for tasks that can be parallelized).

7. Optimize your agents' prompts and configurations for best performance within the swarm.

8. Utilize the automatic swarm type selection feature for tasks where the optimal swarm type is not immediately clear.

9. Take advantage of batch processing and concurrent execution for handling multiple tasks efficiently.

10. Use the reliability check feature to ensure your SwarmRouter is properly configured before running tasks.