Objective: Your task is to intake a business problem or activity and create a swarm of specialized LLM agents that can efficiently solve or automate the given problem. You will define the number of agents, specify the tools each agent needs, and describe how they need to work together, including the communication protocols.

Instructions:

1. **Intake Business Problem:**
   - Receive a detailed description of the business problem or activity to automate.
   - Clarify the objectives, constraints, and expected outcomes of the problem.
   - Identify key components and sub-tasks within the problem.

2. **Agent Design:**
   - Based on the problem, determine the number and types of specialized LLM agents required.
   - For each agent, specify:
     - The specific task or role it will perform.
     - The tools and resources it needs to perform its task.
     - Any prerequisite knowledge or data it must have access to.
   - Ensure that the collective capabilities of the agents cover all aspects of the problem.

3. **Coordination and Communication:**
   - Define how the agents will communicate and coordinate with each other.
     - Choose the type of communication (e.g., synchronous, asynchronous, broadcast, direct messaging).
   - Describe the protocol for information sharing, conflict resolution, and task handoff.

4. **Workflow Design:**
   - Outline the workflow or sequence of actions the agents will follow.
   - Define the input and output for each agent.
   - Specify the triggers and conditions for transitions between agents or tasks.
   - Ensure there are feedback loops and monitoring mechanisms to track progress and performance.

5. **Scalability and Flexibility:**
   - Design the system to be scalable, allowing for the addition or removal of agents as needed.
   - Ensure flexibility to handle dynamic changes in the problem or environment.

6. **Output Specification:**
   - Provide a detailed plan including:
     - The number of agents and their specific roles.
     - The tools and resources each agent will use.
     - The communication and coordination strategy.
     - The workflow and sequence of actions.
   - Include a diagram or flowchart if necessary to visualize the system.

## Examples

# Swarm Architectures

Swarms was designed to faciliate the communication between many different and specialized agents from a vast array of other frameworks such as langchain, autogen, crew, and more.

In traditional swarm theory, there are many types of swarms usually for very specialized use-cases and problem sets. Such as Hiearchical and sequential are great for accounting and sales, because there is usually a boss coordinator agent that distributes a workload to other specialized agents.

Name	**Description**	**Code Link**	**Use Cases**
Hierarchical Swarms	A system where agents are organized in a hierarchy, with higher-level agents coordinating lower-level agents to achieve complex tasks.	[Code Link](#)	Manufacturing process optimization, multi-level sales management, healthcare resource coordination
Agent Rearrange	A setup where agents rearrange themselves dynamically based on the task requirements and environmental conditions.	[Code Link](https://docs.swarms.world/en/latest/swarms/structs/agent_rearrange/)	Adaptive manufacturing lines, dynamic sales territory realignment, flexible healthcare staffing
Concurrent Workflows	Agents perform different tasks simultaneously, coordinating to complete a larger goal.	[Code Link](#)	Concurrent production lines, parallel sales operations, simultaneous patient care processes
Sequential Coordination	Agents perform tasks in a specific sequence, where the completion of one task triggers the start of the next.	[Code Link](https://docs.swarms.world/en/latest/swarms/structs/sequential_workflow/)	

Step-by-step assembly lines, sequential sales processes, stepwise patient treatment workflows |
| Parallel Processing | Agents work on different parts of a task simultaneously to speed up the overall process. | [Code Link](#) | Parallel data processing in manufacturing, simultaneous sales analytics, concurrent medical tests |

### Hierarchical Swarm

Overview:
A Hierarchical Swarm architecture organizes the agents in a tree-like structure. Higher-level agents delegate tasks to lower-level agents, which can further divide tasks among themselves. This structure allows for efficient task distribution and scalability.

Use-Cases:

- Complex decision-making processes where tasks can be broken down into subtasks.

- Multi-stage workflows such as data processing pipelines or hierarchical reinforcement learning.

```mermaid
graph TD
    A[Root Agent] --> B1[Sub-Agent 1]
    A --> B2[Sub-Agent 2]
    B1 --> C1[Sub-Agent 1.1]
    B1 --> C2[Sub-Agent 1.2]
    B2 --> C3[Sub-Agent 2.1]
    B2 --> C4[Sub-Agent 2.2]
```

### Parallel Swarm

Overview:
In a Parallel Swarm architecture, multiple agents operate independently and simultaneously on different tasks. Each agent works on its own task without dependencies on the others. [Learn more here in the docs:](https://docs.swarms.world/en/latest/swarms/structs/agent_rearrange/)

Use-Cases:
- Tasks that can be processed independently, such as parallel data analysis.
- Large-scale simulations where multiple scenarios are run in parallel.

```mermaid
graph LR
    A[Task] --> B1[Sub-Agent 1]
    A --> B2[Sub-Agent 2]
    A --> B3[Sub-Agent 3]
    A --> B4[Sub-Agent 4]
```

### Sequential Swarm

Overview:
A Sequential Swarm architecture processes tasks in a linear sequence. Each agent completes its task before passing the result to the next agent in the chain. This architecture ensures orderly processing and is useful when tasks have dependencies. [Learn more here in the docs:](https://docs.swarms.world/en/latest/swarms/structs/agent_rearrange/)

Use-Cases:
- Workflows where each step depends on the previous one, such as assembly lines or sequential data processing.

- Scenarios requiring strict order of operations.

```mermaid
graph TD
    A[First Agent] --> B[Second Agent]
    B --> C[Third Agent]
    C --> D[Fourth Agent]
```

### Round Robin Swarm

Overview:

In a Round Robin Swarm architecture, tasks are distributed cyclically among a set of agents. Each agent takes turns handling tasks in a rotating order, ensuring even distribution of workload.

Use-Cases:

- Load balancing in distributed systems.

- Scenarios requiring fair distribution of tasks to avoid overloading any single agent.

```mermaid
graph TD
    A[Coordinator Agent] --> B1[Sub-Agent 1]
    A --> B2[Sub-Agent 2]
    A --> B3[Sub-Agent 3]
    A --> B4[Sub-Agent 4]
    B1 --> A
    B2 --> A
    B3 --> A
    B4 --> A
```

### SpreadSheet Swarm

Overview:

The SpreadSheet Swarm makes it easy to manage thousands of agents all in one place: a csv file. You can initialize any number of agents and then there is a loop parameter to run the loop of agents on the task. Learn more in the [docs here](https://docs.swarms.world/en/latest/swarms/structs/spreadsheet_swarm/)

Use-Cases:

- Multi-threaded execution: Execution agents on multiple threads

- Save agent outputs into CSV file

- One place to analyze agent outputs

```mermaid

graph TD
    A[Initialize SpreadSheetSwarm] --> B[Initialize Agents]
    B --> C[Load Task Queue]
    C --> D[Run Task]

    subgraph Agents
    D --> E1[Agent 1]

    D --> E2[Agent 2]

    D --> E3[Agent 3]

end


E1 --> F1[Process Task]

E2 --> F2[Process Task]

E3 --> F3[Process Task]


F1 --> G1[Track Output]

F2 --> G2[Track Output]

F3 --> G3[Track Output]


subgraph Save Outputs

    G1 --> H[Save to CSV]

    G2 --> H[Save to CSV]

    G3 --> H[Save to CSV]

end


H --> I{Autosave Enabled?}

I --> |Yes| J[Export Metadata to JSON]

I --> |No| K[End Swarm Run]


%% Style adjustments

classDef blackBox fill:#000,stroke:#f00,color:#fff;

class A,B,C,D,E1,E2,E3,F1,F2,F3,G1,G2,G3,H,I,J,K blackBox;
```

### Mixture of Agents Architecture

```mermaid
graph TD
    A[Task Input] --> B[Layer 1: Reference Agents]
    B --> C[Agent 1]
    B --> D[Agent 2]
    B --> E[Agent N]

    C --> F[Agent 1 Response]
    D --> G[Agent 2 Response]
    E --> H[Agent N Response]

    F & G & H --> I[Layer 2: Aggregator Agent]
    I --> J[Aggregate All Responses]
    J --> K[Final Output]

```