```
import os
from typing import List

from pydantic import BaseModel, Field
from swarm_models import OpenAIFunctionCaller, OpenAIChat

from swarms.structs.agent import Agent
from swarms.structs.swarm_router import SwarmRouter
from swarms.utils.loguru_logger import initialize_logger
from swarms.structs.agents_available import showcase_available_agents

logger = initialize_logger(log_folder="auto_swarm_builder")


class AgentConfig(BaseModel):
    """Configuration for an individual agent in a swarm"""

    name: str = Field(
        description="The name of the agent", example="Research-Agent"
    )
    description: str = Field(
        description="A description of the agent's purpose and capabilities",
        example="Agent responsible for researching and gathering information",
    )
    system_prompt: str = Field(
        description="The system prompt that defines the agent's behavior",
```

```
        example="You are a research agent. Your role is to gather and analyze information...",
    )
    # max_loops: int = Field(
    #     description="Maximum number of reasoning loops the agent can perform",
    #     example=3,
    # )


class SwarmConfig(BaseModel):
    """Configuration for a swarm of cooperative agents"""

    name: str = Field(
        description="The name of the swarm",
        example="Research-Writing-Swarm",
    )
    description: str = Field(
        description="The description of the swarm's purpose and capabilities",
        example="A swarm of agents that work together to research topics and write articles",
    )
    agents: List[AgentConfig] = Field(
        description="The list of agents that make up the swarm",
        example=[
            AgentConfig(
                name="Research-Agent",
                description="Gathers information",
                system_prompt="You are a research agent...",
```

```
            ),
            AgentConfig(
                name="Writing-Agent",
                description="Writes content",
                system_prompt="You are a writing agent...",
            ),
        ],
    )
    max_loops: int = Field(
        description="The maximum number of loops to run the swarm",
        example=1,
    )



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

# Create an instance of the OpenAIChat class
model = OpenAIChat(
    openai_api_key=api_key, model_name="gpt-4o-mini", temperature=0.1
)



BOSS_SYSTEM_PROMPT = """
Manage a swarm of worker agents to efficiently serve the user by deciding whether to create new agents or delegate tasks. Ensure operations are efficient and effective.
```

Instructions:

1. **Task Assignment**:
   - Analyze available worker agents when a task is presented.
   - Delegate tasks to existing agents with clear, direct, and actionable instructions if an appropriate agent is available.
   - If no suitable agent exists, create a new agent with a fitting system prompt to handle the task.

2. **Agent Creation**:
   - Name agents according to the task they are intended to perform (e.g., "Twitter Marketing Agent").
   - Provide each new agent with a concise and clear system prompt that includes its role, objectives, and any tools it can utilize.

3. **Efficiency**:
   - Minimize redundancy and maximize task completion speed.
   - Avoid unnecessary agent creation if an existing agent can fulfill the task.

4. **Communication**:
   - Be explicit in task delegation instructions to avoid ambiguity and ensure effective task execution.
   - Require agents to report back on task completion or encountered issues.

5. **Reasoning and Decisions**:
   - Offer brief reasoning when selecting or creating agents to maintain transparency.
   - Avoid using an agent if unnecessary, with a clear explanation if no agents are suitable for a task.

Output Format

Present your plan in clear, bullet-point format or short concise paragraphs, outlining task assignment, agent creation, efficiency strategies, and communication protocols.

Notes

- Preserve transparency by always providing reasoning for task-agent assignments and creation.
- Ensure instructions to agents are unambiguous to minimize error.

"""

class AutoSwarmBuilder:
    """A class that automatically builds and manages swarms of AI agents.

    This class handles the creation, coordination and execution of multiple AI agents working together as a swarm to accomplish complex tasks. It uses a boss agent to delegate work and create new specialized agents as needed.

    Args:
        name (str): The name of the swarm
        description (str): A description of the swarm's purpose
        verbose (bool, optional): Whether to output detailed logs. Defaults to True.
        max_loops (int, optional): Maximum number of execution loops. Defaults to 1.

```
    """


    def __init__(
        self,
        name: str = None,
        description: str = None,
        verbose: bool = True,
        max_loops: int = 1,
    ):
        self.name = name
        self.description = description
        self.verbose = verbose
        self.max_loops = max_loops
        self.agents_pool = []
        logger.info(
            f"Initialized AutoSwarmBuilder: {name} {description}"
        )


    # @retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=4, max=10))
    def run(self, task: str, image_url: str = None, *args, **kwargs):
        """Run the swarm on a given task.


        Args:
            task (str): The task to be accomplished
            image_url (str, optional): URL of an image input if needed. Defaults to None.
            *args: Variable length argument list
```

```
        **kwargs: Arbitrary keyword arguments

    Returns:
        The output from the swarm's execution
    """
    logger.info(f"Running swarm on task: {task}")
    agents = self._create_agents(task, image_url, *args, **kwargs)
    logger.info(f"Agents created {len(agents)}")
    logger.info("Routing task through swarm")
    output = self.swarm_router(agents, task, image_url)
    logger.info(f"Swarm execution complete with output: {output}")
    return output

# @retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=4, max=10))
def _create_agents(self, task: str, *args, **kwargs):
    """Create the necessary agents for a task.

    Args:
        task (str): The task to create agents for
        *args: Variable length argument list
        **kwargs: Arbitrary keyword arguments

    Returns:
        list: List of created agents
    """
    logger.info("Creating agents for task")
```

```
model = OpenAIFunctionCaller(
    system_prompt=BOSS_SYSTEM_PROMPT,
    api_key=os.getenv("OPENAI_API_KEY"),
    temperature=0.1,
    base_model=SwarmConfig,
)

agents_dictionary = model.run(task)
logger.info(f"Agents dictionary: {agents_dictionary}")

# Convert dictionary to SwarmConfig if needed
if isinstance(agents_dictionary, dict):
    agents_dictionary = SwarmConfig(**agents_dictionary)

# Set swarm config
self.name = agents_dictionary.name
self.description = agents_dictionary.description
self.max_loops = getattr(
    agents_dictionary
)  # Default to 1 if not set

logger.info(
    f"Swarm config: {self.name}, {self.description}, {self.max_loops}"
)

# Create agents from config
```

```
    agents = []
    for agent_config in agents_dictionary.agents:
        # Convert dict to AgentConfig if needed
        if isinstance(agent_config, dict):
            agent_config = AgentConfig(**agent_config)

        agent = self.build_agent(
            agent_name=agent_config.name,
            agent_description=agent_config.description,
            agent_system_prompt=agent_config.system_prompt,
        )
        agents.append(agent)

    # Showcasing available agents
    agents_available = showcase_available_agents(
        name=self.name,
        description=self.description,
        agents=agents,
    )

    for agent in agents:
        agent.system_prompt += "\n" + agents_available

    return agents

def build_agent(
```

```
    self,
    agent_name: str,
    agent_description: str,
    agent_system_prompt: str,
    max_loops: int = 1,
):
    """Build a single agent with the given specifications.

    Args:
        agent_name (str): Name of the agent
        agent_description (str): Description of the agent's purpose
        agent_system_prompt (str): The system prompt for the agent

    Returns:
        Agent: The constructed agent instance
    """
    logger.info(f"Building agent: {agent_name}")
    agent = Agent(
        agent_name=agent_name,
        description=agent_description,
        system_prompt=agent_system_prompt,
        llm=model,
        max_loops=max_loops,
        autosave=True,
        dashboard=False,
        verbose=True,
```

```
        dynamic_temperature_enabled=True,
        saved_state_path=f"{agent_name}.json",
        user_name="swarms_corp",
        retry_attempts=1,
        context_length=200000,
        return_step_meta=False,
        output_type="str",  # "json", "dict", "csv" OR "string" soon "yaml" and
        streaming_on=False,
        auto_generate_prompt=True,
    )

    return agent

# @retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=4, max=10))
def swarm_router(
    self,
    agents: List[Agent],
    task: str,
    image_url: str = None,
    *args,
    **kwargs,
):
    """Route tasks between agents in the swarm.

    Args:
        agents (List[Agent]): List of available agents
```

```
            task (str): The task to route
            image_url (str, optional): URL of an image input if needed. Defaults to None.
            *args: Variable length argument list
            **kwargs: Arbitrary keyword arguments

        Returns:
            The output from the routed task execution
        """
        logger.info("Routing task through swarm")
        swarm_router_instance = SwarmRouter(
            name=self.name,
            description=self.description,
            agents=agents,
            swarm_type="auto",
            max_loops=1,
        )

        return swarm_router_instance.run(
            self.name + " " + self.description + " " + task,
        )


example = AutoSwarmBuilder(
    name="ChipDesign-Swarm",
     description="A swarm of specialized AI agents collaborating on chip architecture, logic design, verification, and optimization to create novel semiconductor designs",
```

```
    max_loops=1,
)


print(
    example.run(
        "Design a new AI accelerator chip optimized for transformer model inference. Consider the following aspects: 1) Overall chip architecture and block diagram 2) Memory hierarchy and interconnects 3) Processing elements and data flow 4) Power and thermal considerations 5) Physical layout recommendations -> "
    )
)
```