```
import re

from dotenv import load_dotenv
from tenacity import retry, stop_after_attempt, wait_exponential

from swarms import Agent
from swarms.agents.create_agents_from_yaml import (
    create_agents_from_yaml,
)
from swarms.utils.formatter import formatter
from swarms.utils.litellm_wrapper import LiteLLM

load_dotenv()



def prepare_yaml_for_parsing(raw_yaml: str) -> str:
    """
    Prepares raw YAML content by fixing spacing and formatting issues.

    Args:
        raw_yaml (str): The raw YAML content extracted from Markdown.

    Returns:
        str: The cleaned YAML content ready for parsing.
    """
    # Fix sequence items that are improperly placed on the same line as their key
```

```
    fixed_yaml = re.sub(
        r"(\b\w+\b):\s*-\s*", r"\1:\n  - ", raw_yaml
    )  # Fix "key: - value" to "key:\n  - value"

    # Ensure proper spacing after colons
    fixed_yaml = re.sub(
        r"(\S):(\S)", r"\1: \2", fixed_yaml
    )  # Ensure space after colons

    # Remove trailing spaces before newlines
    fixed_yaml = re.sub(r"\s+\n", "\n", fixed_yaml)

    # Replace non-breaking spaces (if any) with regular spaces
    fixed_yaml = fixed_yaml.replace("\xa0", " ")

    return fixed_yaml.strip()


def parse_yaml_from_swarm_markdown(markdown_text: str) -> dict:
    """
      Extracts and prepares YAML content from a Markdown-style 'Auto-Swarm-Builder' block and parses it.

    Args:
        markdown_text (str): The Markdown text containing the YAML inside 'Auto-Swarm-Builder' block.
```

```
    Returns:
        dict: A parsed Python dictionary of the YAML content.
    """
    # Match the 'Auto-Swarm-Builder' block with YAML inside triple backticks
    pattern = r"```yaml\s*\n(.*?)```"
    match = re.search(pattern, markdown_text, re.DOTALL)

    if not match:
        raise ValueError(
            "No YAML content found in the 'Auto-Swarm-Builder' block."
        )

    raw_yaml = match.group(1).strip()

    # Preprocess and normalize the YAML content
    normalized_yaml = prepare_yaml_for_parsing(raw_yaml)

    return normalized_yaml


AUTO_GEN_PROMPT = """
You are a specialized agent responsible for creating YAML configuration files for multi-agent swarms. Your role is to generate well-structured YAML that defines both individual agents and swarm architectures based on user requirements.
Output only the yaml nothing else. You will be penalized for making mistakes
```

GUIDELINES:

1. Each YAML file must contain an `agents` section with at least one agent configuration

2. Each agent configuration requires the following mandatory fields:

   - agent_name (string)

   - system_prompt (string)

3. Optional agent fields include:

   - max_loops (integer)

   - autosave (boolean)

   - dashboard (boolean)

   - verbose (boolean)

   - dynamic_temperature_enabled (boolean)

   - saved_state_path (string)

   - user_name (string)

   - retry_attempts (integer)

   - context_length (integer)

   - return_step_meta (boolean)

   - output_type (string)

   - task (string)

4. When a swarm is needed, include a `swarm_architecture` section with:

   Mandatory fields:

   - name (string)

   - swarm_type (string: "ConcurrentWorkflow" or "SequentialWorkflow") [AgentRearrange, MixtureOfAgents, SpreadSheetSwarm, SequentialWorkflow, ConcurrentWorkflow]

Optional fields:

- description (string)
- max_loops (integer)
- task (string)

TEMPLATE STRUCTURE:

```yaml
agents:
  - agent_name: "Agent-1-Name"
    system_prompt: "Detailed system prompt here"
    max_loops: 1
    # [additional optional fields]

  - agent_name: "Agent-2-Name"
    system_prompt: "Detailed system prompt here"
    # [additional optional fields]

swarm_architecture:
  name: "Swarm-Name"
  description: "Swarm purpose and goals"
  swarm_type: "ConcurrentWorkflow"
  max_loops: 5
  task: "Main swarm task description"
```

VALIDATION RULES:

1. All agent names must be unique

2. System prompts must be clear and specific to the agent's role

3. Integer values must be positive

4. Boolean values must be true or false (lowercase)

5. File paths should use forward slashes

6. Tasks should be specific and aligned with the agent/swarm purpose

When generating a YAML configuration:

1. Ask for specific requirements about the agents and swarm needed

2. Determine if a swarm architecture is necessary based on the task complexity

3. Generate appropriate system prompts for each agent based on their roles

4. Include relevant optional fields based on the use case

5. Validate the configuration against all rules before returning

Example valid YAML configurations are provided below. Use these as references for structure and formatting:

```yaml

agents:
  - agent_name: "Data-Analysis-Agent"
      system_prompt: "You are a specialized data analysis agent focused on processing and interpreting financial data. Provide clear, actionable insights based on the data provided."
    max_loops: 3
```

```
    autosave: true
    verbose: true
    context_length: 100000
    output_type: "json"
    task: "Analyze quarterly financial reports and identify trends"

# Multi-Agent Swarm Example
agents:
  - agent_name: "Research-Agent"
      system_prompt: "You are a research agent specialized in gathering and summarizing scientific publications. Focus on peer-reviewed sources and provide comprehensive summaries."
    max_loops: 2
    context_length: 150000
    output_type: "str"

  - agent_name: "Analysis-Agent"
      system_prompt: "You are an analysis agent that processes research summaries and identifies key patterns and insights. Provide detailed analytical reports."
    max_loops: 3
    context_length: 200000
    output_type: "json"

swarm_architecture:
  name: "Research-Analysis-Swarm"
  description: "A swarm for comprehensive research analysis and insight generation"
  swarm_type: "SequentialWorkflow"
```

```
  max_loops: 5
  task: "Research and analyze recent developments in quantum computing"

```
```
"""


def generate_swarm_config(
    task: str,
    file_name: str = "swarm_config_output.yaml",
    model_name: str = "gpt-4o",
    *args,
    **kwargs,
):
    """
    Generates a swarm configuration based on the provided task and model name.

    This function attempts to generate a swarm configuration by running an agent with the specified task and model name.
    It then parses the output into YAML format and creates agents based on the parsed YAML content.

    Args:
        task (str): The task to be performed by the swarm.
        file_name (str, optional): The file name for the output YAML configuration. Defaults to "swarm_config_output.yaml".
```

```
        model_name (str, optional): The name of the model to use for the agent. Defaults to "gpt-4o".
        *args: Additional positional arguments to be passed to the agent's run method.
        **kwargs: Additional keyword arguments to be passed to the agent's run method.

    Returns:
        Any: The output of the swarm configuration generation process. This can be a SwarmRouter instance or an error message.
    """
    formatter.print_panel(
        "Auto Generating Swarm...", "Auto Swarm Builder"
    )

    @retry(
        stop=stop_after_attempt(3),
        wait=wait_exponential(min=4, max=10),
    )
    def attempt_generate_swarm_config():
        try:
            model = LiteLLM(model_name=model_name)

            # Initialize the agent
            agent = Agent(
                agent_name="Auto-Swarm-Builder",
                system_prompt=AUTO_GEN_PROMPT,
                llm=model,
                max_loops=1,
```

```
        dynamic_temperature_enabled=True,
        saved_state_path="swarm_builder.json",
        user_name="swarms_corp",
        output_type="str",
    )

    # Generate output from the agent
    raw_output = agent.run(task, *args, **kwargs)
    yaml_content = parse_yaml_from_swarm_markdown(raw_output)
    print(yaml_content)

    # Create agents from the YAML file
    output = create_agents_from_yaml(
        yaml_string=yaml_content,
        return_type="run_swarm",
    )

    formatter.print_panel(
        "Swarm configuration generated successfully.",
        "Success",
    )

    return output

except Exception as e:
    formatter.print_panel(
```

```
            f"Error generating swarm configuration: {str(e)}",
            "Error",
        )
        raise

return attempt_generate_swarm_config()
```