```
import os


from autotemp import AutoTemp

from termcolor import colored


from swarm_models import OpenAIChat

from swarms.structs import SequentialWorkflow



class BlogGen:
    def __init__(
        self,
        api_key,
        blog_topic,
        temperature_range: str = "0.4,0.6,0.8,1.0,1.2",
    ):  # Add blog_topic as an argument
        self.openai_chat = OpenAIChat(
            openai_api_key=api_key, temperature=0.8
        )
        self.auto_temp = AutoTemp(api_key)
        self.temperature_range = temperature_range
        self.workflow = SequentialWorkflow(max_loops=5)

        # Formatting the topic selection prompt with the user's topic
        self.TOPIC_SELECTION_SYSTEM_PROMPT = f"""
         Given the topic '{blog_topic}', generate an engaging and versatile blog topic. This topic should
```

cover areas related to '{blog_topic}' and might include aspects such as current events, lifestyle, technology, health, and culture related to '{blog_topic}'. Identify trending subjects within this realm. The topic must be unique, thought-provoking, and have the potential to draw in readers interested in '{blog_topic}'.

"""

self.DRAFT_WRITER_SYSTEM_PROMPT = """

Create an engaging and comprehensive blog article of at least 1,000 words on '{{CHOSEN_TOPIC}}'. The content should be original, informative, and reflective of a human-like style, with a clear structure including headings and sub-headings. Incorporate a blend of narrative, factual data, expert insights, and anecdotes to enrich the article. Focus on SEO optimization by using relevant keywords, ensuring readability, and including meta descriptions and title tags. The article should provide value, appeal to both knowledgeable and general readers, and maintain a balance between depth and accessibility. Aim to make the article engaging and suitable for online audiences.

"""

self.REVIEW_AGENT_SYSTEM_PROMPT = """

Critically review the drafted blog article on '{{ARTICLE_TOPIC}}' to refine it to high-quality content suitable for online publication. Ensure the article is coherent, factually accurate, engaging, and optimized for search engines (SEO). Check for the effective use of keywords, readability, internal and external links, and the inclusion of meta descriptions and title tags. Edit the content to enhance clarity, impact, and maintain the authors voice. The goal is to polish the article into a professional, error-free piece that resonates with the target audience, adheres to publication standards, and is optimized for both search engines and social media sharing.

"""

```
        self.DISTRIBUTION_AGENT_SYSTEM_PROMPT = """
          Develop an autonomous distribution strategy for the blog article on '{{ARTICLE_TOPIC}}'. Utilize an API to post the article on a popular blog platform (e.g., WordPress, Blogger, Medium) commonly used by our target audience. Ensure the post includes all SEO elements like meta descriptions, title tags, and properly formatted content. Craft unique, engaging social media posts tailored to different platforms to promote the blog article. Schedule these posts to optimize reach and engagement, using data-driven insights. Monitor the performance of the distribution efforts, adjusting strategies based on engagement metrics and audience feedback. Aim to maximize the article's visibility, attract a diverse audience, and foster engagement across digital channels.
        """


    def run_workflow(self):
        try:
            # Topic generation using OpenAIChat
            topic_result = self.openai_chat.generate(
                [self.TOPIC_SELECTION_SYSTEM_PROMPT]
            )
            topic_output = topic_result.generations[0][0].text
            print(
                colored(
                    (
                        "\nTopic Selection Task"
                        f" Output:\n----------------------------\n{topic_output}\n"
                    ),
                    "white",
                )
            )

            chosen_topic = topic_output.split("\n")[0]
            print(
                colored("Selected topic: " + chosen_topic, "yellow")
            )

            # Initial draft generation with AutoTemp
            initial_draft_prompt = (
                self.DRAFT_WRITER_SYSTEM_PROMPT.replace(
                    "{{CHOSEN_TOPIC}}", chosen_topic
                )
            )
            auto_temp_output = self.auto_temp.run(
                initial_draft_prompt, self.temperature_range
            )
              initial_draft_output = auto_temp_output  # Assuming AutoTemp.run returns the best output
    directly
            print(
                colored(
                    (
                        "\nInitial Draft"
                        f" Output:\n----------------------------\n{initial_draft_output}\n"
                    ),
                    "white",
                )
            )

            # Review process using OpenAIChat
            review_prompt = self.REVIEW_AGENT_SYSTEM_PROMPT.replace(
                "{{ARTICLE_TOPIC}}", chosen_topic
            )
            review_result = self.openai_chat.generate([review_prompt])
            review_output = review_result.generations[0][0].text
            print(
                colored(
                    (
                        "\nReview"
                        f" Output:\n----------------------------\n{review_output}\n"
                    ),
                    "white",
                )
            )

            # Distribution preparation using OpenAIChat
            distribution_prompt = (
                self.DISTRIBUTION_AGENT_SYSTEM_PROMPT.replace(
                    "{{ARTICLE_TOPIC}}", chosen_topic
                )
            )
            distribution_result = self.openai_chat.generate(
                [distribution_prompt]
            )
            distribution_output = distribution_result.generations[0][
                0
            ].text
            print(
                colored(
                    (
                        "\nDistribution"
                        f" Output:\n----------------------------\n{distribution_output}\n"
                    ),
                    "white",
                )
            )

            # Final compilation of the blog
            final_blog_content = f"{initial_draft_output}\n\n{review_output}\n\n{distribution_output}"
            print(
                colored(
                    (
                        "\nFinal Blog"
                        f" Content:\n----------------------------\n{final_blog_content}\n"
                    ),
                    "green",
                )
            )
```

```
        except Exception as e:
            print(colored(f"An error occurred: {str(e)}", "red"))



if __name__ == "__main__":
    api_key = os.environ["OPENAI_API_KEY"]
    blog_generator = BlogGen(api_key)
    blog_generator.run_workflow()
```