```
import datetime

import os


import streamlit as st

from dotenv import load_dotenv


from swarm_models import OpenAIChat

from swarm_models.gpt4_vision_api import GPT4VisionAPI

from swarm_models.stable_diffusion import StableDiffusion

from swarms.structs import Agent


# Load environment variables

load_dotenv()

openai_api_key = os.getenv("OPENAI_API_KEY")

stability_api_key = os.getenv("STABLE_API_KEY")


# Initialize the models

vision_api = GPT4VisionAPI(api_key=openai_api_key)

sd_api = StableDiffusion(api_key=stability_api_key)

gpt_api = OpenAIChat(openai_api_key=openai_api_key)




class Idea2Image(Agent):

    def __init__(self, llm, vision_api):

        super().__init__(llm=llm)

        self.vision_api = vision_api
```

```
def run(self, initial_prompt, num_iterations, run_folder):
    current_prompt = initial_prompt

    for i in range(num_iterations):
        print(f"Iteration {i}: Image generation and analysis")

        if i == 0:
            current_prompt = self.enrich_prompt(current_prompt)
            print(f"Enriched Prompt: {current_prompt}")

        img = sd_api.generate_and_move_image(
            current_prompt, i, run_folder
        )
        if not img:
            print("Failed to generate image")
            break
        print(f"Generated image at: {img}")

        analysis = (
            self.vision_api.run(img, current_prompt)
            if img
            else None
        )
        if analysis:
            current_prompt += (
                ". " + analysis[:500]
            )  # Ensure the analysis is concise
            print(f"Image Analysis: {analysis}")
        else:
            print(f"Failed to analyze image at: {img}")


def enrich_prompt(self, prompt):
    enrichment_task = (
        "Create a concise and effective image generation prompt"
        " within 400 characters or less, based on Stable"
        " Diffusion and Dalle best practices to help it create"
        " much better images. Starting prompt:"
        f" \n\n'{prompt}'\n\nImprove the prompt with any"
        " applicable details or keywords by considering the"
        " following aspects: \n1. Subject details (like actions,"
        " emotions, environment) \n2. Artistic style (such as"
        " surrealism, hyperrealism) \n3. Medium (digital"
        " painting, oil on canvas) \n4. Color themes and"
        " lighting (like warm colors, cinematic lighting) \n5."
        " Composition and framing (close-up, wide-angle) \n6."
        " Additional elements (like a specific type of"
        " background, weather conditions) \n7. Any other"
        " artistic or thematic details that can make the image"
        " more vivid and compelling. Help the image generator"
        " create better images by enriching the prompt."
    )
```

```
        llm_result = self.llm.generate([enrichment_task])
        return (
            llm_result.generations[0][0].text[:500]
            if llm_result.generations
            else None
        )

    def run_gradio(self, initial_prompt, num_iterations, run_folder):
        results = []
        current_prompt = initial_prompt

        for i in range(num_iterations):
            enriched_prompt = (
                self.enrich_prompt(current_prompt)
                if i == 0
                else current_prompt
            )
            img_path = sd_api.generate_and_move_image(
                enriched_prompt, i, run_folder
            )
            analysis = (
                self.vision_api.run(img_path, enriched_prompt)
                if img_path
                else None
            )
            if analysis:
                current_prompt += (
                    ". " + analysis[:500]
                )  # Ensuring the analysis is concise
            results.append((enriched_prompt, img_path, analysis))

        return results



    # print(
    #           colored("---------------------------------------- MultiModal Tree of Thought agents for Image Generation", "cyan", attrs=["bold"])
    # )
    # # User input and setup
    # user_prompt = input("Prompt for image generation: ")
    # num_iterations = int(
    #     input("Enter the number of iterations for image improvement: ")
    # )
    # run_folder = os.path.join(
    #     "runs", datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
    # )
    # os.makedirs(run_folder, exist_ok=True)

    # print(
    #     colored(
    #         f"---------------------------------- Running Multi-Modal Tree of thoughts agent with {num_iterations}
    iterations", "green"
    #     )
    # )
    # # Initialize and run the agent
    # idea2image_agent = Idea2Image(gpt_api, vision_api)
    # idea2image_agent.run(user_prompt, num_iterations, run_folder)

    # print("Idea space has been traversed.")



    # Load environment variables and initialize the models
    load_dotenv()
    openai_api_key = os.getenv("OPENAI_API_KEY")
    stability_api_key = os.getenv("STABLE_API_KEY")
    vision_api = GPT4VisionAPI(api_key=openai_api_key)
    sd_api = StableDiffusion(api_key=stability_api_key)
    gpt_api = OpenAIChat(openai_api_key=openai_api_key)

    # Define the modified Idea2Image class here

    # Streamlit UI layout
    st.title(
        "Explore the infinite Multi-Modal Idea Space with Idea2Image"
    )
    user_prompt = st.text_input("Prompt for image generation:")
    num_iterations = st.number_input(
        "Enter the number of iterations for image improvement:",
        min_value=1,
        step=1,
    )


    if st.button("Generate Image"):
        run_folder = os.path.join(
            "runs", datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        )
        os.makedirs(run_folder, exist_ok=True)
        idea2image_agent = Idea2Image(gpt_api, vision_api)

        results = idea2image_agent.run_gradio(
            user_prompt, num_iterations, run_folder
        )

        for i, (enriched_prompt, img_path, analysis) in enumerate(
            results
        ):
            st.write(f"Iteration {i+1}:")
            st.write("Enriched Prompt:", enriched_prompt)
            if img_path:
                st.image(img_path, caption="Generated Image")
            else:
                st.error("Failed to generate image")
            if analysis:
                st.write("Image Analysis:", analysis)

        st.success("Idea space has been traversed.")

    # [Add any additional necessary code adjustments]
```