```
import os
import torch
from diffusers import StableDiffusion3Pipeline
from PIL import Image

from typing import List

from diffusers import (
    AutoencoderKLCogVideoX,
    CogVideoXPipeline,
    CogVideoXTransformer3DModel,
    FluxPipeline,
)
from diffusers.utils import export_to_video
from loguru import logger
from pydantic import BaseModel, Field
from transformers import T5EncoderModel

from swarms import Agent
from swarm_models.openai_function_caller import OpenAIFunctionCaller

hf_token = os.getenv("HF_TOKEN")


IMG_GEN_SYS_PROMPT = """
```

System Prompt for Image Generation Agent

Objective: Generate visually stunning and highly detailed images by crafting precise, extensive prompts tailored for models like MidJourney, Stable Diffusion, and others.

Instructions for the Agent:

1. **Understand the Request**:
   - **Identify Key Elements**: Extract the core themes, styles, emotions, and subjects from the user's request. Break down the request into specific elements such as the environment, characters, actions, color schemes, and artistic styles.
   - **Users Intent**: Clarify the user's intent, focusing on the desired visual impact, mood, and details they want to emphasize.

2. **Crafting the Prompt**:
   - **Visual Details**: Describe the scene with vivid and detailed language. Include specific elements like lighting (e.g., "soft diffused sunlight"), textures (e.g., "cracked, ancient stone"), and colors (e.g., "deep sapphire blue sky").
   - **Style and Artistic Influence**: Reference particular art styles, eras, or techniques if relevant (e.g., "Renaissance painting," "cyberpunk aesthetic," "surrealism with a dream-like quality").
   - **Complex Compositions**: For more intricate requests, break down the composition into layers. Describe the foreground, midground, and background in detail.

- **Dynamic Elements**: Incorporate motion or action where applicable (e.g., "wind blowing through the trees," "waves crashing against the rocks").

- **Emotion and Atmosphere**: Convey the desired mood or atmosphere (e.g., "eerie and unsettling," "tranquil and serene").

3. **Enhance with Advanced Techniques**:

- **Perspective and Depth**: Specify camera angles, perspectives, and focal points (e.g., "wide-angle shot from a low perspective," "close-up of the characters face with a shallow depth of field").

- **Post-Processing Effects**: Suggest potential post-processing effects like "soft glow around the edges," "high contrast and saturation," or "vintage film grain effect."

- **Use of Lighting**: Describe the type of lighting in detail, considering sources, intensity, and direction (e.g., "dramatic backlighting casting long shadows," "warm golden hour light").

4. **Iterative Refinement**:

- **Ask Clarifying Questions**: If any details are ambiguous or if multiple interpretations are possible, ask the user for clarification.

- **Provide Multiple Variations**: Offer different versions of the prompt to explore various interpretations and artistic directions.

- **Feedback Loop**: Incorporate user feedback to refine and enhance the prompt, ensuring it aligns with their vision.

Example Prompts:

1. **Fantasy Landscape**:

- "A majestic, towering castle perched on a cliff overlooking a vast, misty forest. The scene is bathed in the golden light of dawn, with rays breaking through the clouds. The castle features Gothic architecture with tall spires and intricate stone carvings. In the foreground, wildflowers sway gently in the breeze, while distant mountains rise in the background, shrouded in a mysterious fog."

2. **Cyberpunk Cityscape**:

- "A bustling, neon-lit city at night, drenched in rain. Skyscrapers with holographic billboards tower above narrow streets crowded with people wearing futuristic attire. The scene is awash in vibrant hues of electric blue, pink, and purple. Steam rises from manholes, and the reflections of lights ripple on wet pavement. The atmosphere is tense, with a sense of underlying danger."

3. **Surreal Dreamscape**:

- "An endless desert of smooth, shifting sands under a sky filled with swirling, colorful auroras. Floating above the sands are massive, translucent orbs, each containing a different fantastical landscape within. The largest orb, in the center, reveals a tranquil forest with towering, bioluminescent trees. The scene is otherworldly, with a calm, dream-like quality."

Final Output:

- The agent should generate prompts that align with the users vision, maximizing the potential of image-generation models to produce highly detailed, visually captivating images.

"""

```
class ImageGenSpec(BaseModel):
    negative_prompt: str = Field(
        ...,
          description="The negative prompt for the image or video generation modelto generate the image",
    )
    prompt: str = Field(
        ...,
        description="The prompt for the image or video generation model to generate the image",
    )
    task_type: str = Field(
        ...,
         description="The type of task for the image or video generation model to generate the image like image or video",
        # examples=["image", "video"],
    )



class MultipleImgGenSpec(BaseModel):
    plan: str = Field(
        ...,
        description="The plan for the prompt to be injected into the image or video generation model to generate a really good creation that aligns with the user's vision",
    )
    creations: List[ImageGenSpec] = Field(
```

```
        ...,
        description="The list of image or video generation model prompts for the swarm",
    )



class FluxDev:
    def __init__(self, height: int = 1024, width: int = 1024):
        self.height = height
        self.width = width
        self.pipe = FluxPipeline.from_pretrained(
            "black-forest-labs/FLUX.1-dev",
            torch_dtype=torch.bfloat16,
            hf_token=hf_token,
        )

    def run(self, task: str, *args, **kwargs):
        import torch

        image = self.pipe(
            task,
            height=self.height,
            width=self.width,
            guidance_scale=3.5,
            num_inference_steps=50,
            max_sequence_length=512,
            generator=torch.Generator("gpu").manual_seed(0),
```

```
            *args,
            **kwargs,
        ).images[0]

        task_name_under_score = task.replace(" ", "_")

        name = f"{task_name_under_score}.png"

        image.save(name)

        return name


class ImageGenerator:
    def __init__(
        self,
        model_name="stabilityai/stable-diffusion-3-medium-diffusers",
        device="cpu",
        negative_prompt="",
        num_inference_steps=28,
        guidance_scale=7.0,
        output_folder=".",
    ):
        self.pipe = StableDiffusion3Pipeline.from_pretrained(
            model_name, torch_dtype=torch.float16
        )
```

```
    self.pipe = self.pipe.to(device)
    self.negative_prompt = negative_prompt
    self.num_inference_steps = num_inference_steps
    self.guidance_scale = guidance_scale
    self.output_folder = output_folder

def run(self, prompt):
    image = self.pipe(
        prompt,
        negative_prompt=self.negative_prompt,
        num_inference_steps=self.num_inference_steps,
        guidance_scale=self.guidance_scale,
    ).images[0]

    # Convert tensor to PIL Image
    image = Image.fromarray(image.cpu().numpy().astype("uint8"))

    # Save image
    image_path = os.path.join(
        self.output_folder, f"{prompt.replace(' ', '_')}.png"
    )
    image.save(image_path)

    return image_path
```

```
class VideGenerator:
    def __init__(
        self,
        height: int = 1024,
        width: int = 1024,
        save_video_path: str = "output.mp4",
        folder_path: str = "video_generator_frames",
        name: str = "THUDM/CogVideoX-5b",
        num_videos_per_prompt: int = 1,
        num_inference_steps: int = 50,
        device: str = "cpu",
        *args,
        **kwargs,
    ):
        # To get started, PytorchAO needs to be installed from the GitHub source and PyTorch Nightly.
        # Source and nightly installation is only required until next release.
        self.height = height
        self.width = width
        self.save_video_path = save_video_path
        self.folder_path = folder_path
        self.name = name
        self.num_videos_per_prompt = num_videos_per_prompt
        self.num_inference_steps = num_inference_steps
        self.device = device

        # quantization = int8_weight_only
```

```
text_encoder = T5EncoderModel.from_pretrained(
    name,
    subfolder="text_encoder",
    torch_dtype=torch.bfloat16,
)
# quantize_(text_encoder, quantization())

transformer = CogVideoXTransformer3DModel.from_pretrained(
    name,
    subfolder="transformer",
    torch_dtype=torch.bfloat16,
)
# quantize_(transformer, quantization())

vae = AutoencoderKLCogVideoX.from_pretrained(
    name,
    subfolder="vae",
    torch_dtype=torch.bfloat16,
)

# quantize_(vae, quantization())

# Create pipeline and run inference
self.pipe = CogVideoXPipeline.from_pretrained(
    name,
```

```
        text_encoder=text_encoder,
        transformer=transformer,
        vae=vae,
        torch_dtype=torch.bfloat16,
        *args,
        **kwargs,
    )
    self.pipe.enable_model_cpu_offload()
    self.pipe.vae.enable_tiling()

def run(self, task: str, *args, **kwargs):
    video = self.pipe(
        prompt=task,
        num_videos_per_prompt=self.num_videos_per_prompt,
        num_inference_steps=self.num_inference_steps,
        num_frames=49,
        guidance_scale=6,
        generator=torch.Generator(device=self.device).manual_seed(
            42
        ),
        *args,
        **kwargs,
    ).frames[0]

    export_to_video(video, self.save_video_path, fps=8)
```

```
        return logger.info(f"Video saved at {self.save_video_path}")


class MediaCreatorAgent:
    """
    A class representing an image generator agent.

    Attributes:
        agent_loops (int): The number of loops the agent will run.
        img_height (int): The height of the generated image.
        img_width (int): The width of the generated image.
        max_tokens (int): The maximum number of tokens used for generating the image.

    Methods:
         run(task: str, *args, **kwargs) -> str: Runs the agent to generate an image based on the given
task.

    Example usage:
                    agent = MediaCreatorAgent:(agent_loops=1, img_height=1024, img_width=1024,
max_tokens=3000)
        img_path = agent.run("generate_image", task_args, task_kwargs)
    """

    def __init__(
        self,
        agent_loops: int = 1,
```

```
    img_height: int = 1024,
    img_width: int = 1024,
    max_tokens: int = 3000,
    num_videos_per_prompt: int = 1,
):
    self.agent_loops = agent_loops
    self.img_height = img_height
    self.img_width = img_width
    self.max_tokens = max_tokens
    self.num_videos_per_prompt = num_videos_per_prompt

    # Example usage:
    # Initialize the function caller
    model = OpenAIFunctionCaller(
        system_prompt=IMG_GEN_SYS_PROMPT,
        temperature=0.7,
        base_model=MultipleImgGenSpec,
        parallel_tool_calls=False,
        openai_api_key=os.getenv("OPENAI_API_KEY"),
    )

    # Initialize the agent
    self.agent = Agent(
        agent_name="Image-Generation-Agent",
        system_prompt=IMG_GEN_SYS_PROMPT,
        llm=model,
```

```
            max_loops=1,
            autosave=True,
            dashboard=False,
            verbose=True,
            dynamic_temperature_enabled=True,
            user_name="swarms_corp",
            retry_attempts=1,
            context_length=200000,
            # return_step_meta=True,
            disable_print_every_step=False,
            # output_type="json",
            max_tokens=self.max_tokens,
        )

        # Img Generator Model
        # self.img_gen_model = FluxDev()
        self.img_gen_model = ImageGenerator()

        # Video Generator Model
        self.video_gen_model = VideGenerator(
            num_videos_per_prompt=self.num_videos_per_prompt,
            device="cpu",
        )

    def parse_task_type(
        self, task_type: str = None, prompt: str = None
```

```
):
    if task_type == "image":
        logger.info("Generating Image")
        return self.img_gen_model.run(prompt)
    elif task_type == "video":
        logger.info("Generating Video")
        return self.video_gen_model.run(prompt)
    else:
        raise ValueError(
            "Invalid Task Type. Please provide a valid task type."
        )

def run(self, task: str, *args, **kwargs):
    logger.info(
        f"Running Image Generation Agent for Task: {task}"
    )
    out = self.agent.run(task, *args, **kwargs)
    out = eval(out)
    print(out)

    plan = out["plan"]
    logger.info(f"Plan: {plan}")

    # Now we need to parse multiple schemas
    for img_gen_spec in out["creations"]:
        # plan = img_gen_spec["plan"]
```

```
        negative_prompt = img_gen_spec["negative_prompt"]
        prompt = img_gen_spec["prompt"]
        task_type = img_gen_spec["task_type"]

        logger.info(
            f"Negative Prompt: {negative_prompt}: Prompt for Image: {prompt}"
        )

        return self.parse_task_type(task_type, prompt)


# Example Usage
agent = MediaCreatorAgent(
    agent_loops=1, img_height=1024, img_width=1024, max_tokens=3000
)

# Example Task
task = "Generate an image and a video of a beautiful image of a serene lake at sunset with vibrant
colors and reflections of the sky on the water."
out = agent.run(task)
print(out)
```