```
import os
import subprocess as sp
from pathlib import Path

from dotenv import load_dotenv
from PIL import Image

from swarm_models.base_multimodal_model import BaseMultiModalModel

try:
    import google.generativeai as genai
    from google.generativeai.types import GenerationConfig
except ImportError as error:
    print(f"Error importing google.generativeai: {error}")
    print("Please install the google.generativeai package")
    print("pip install google-generativeai")
    sp.run(["pip", "install", "--upgrade", "google-generativeai"])


load_dotenv()


# Helpers
def get_gemini_api_key_env():
    """Get the Gemini API key from the environment
    Raises:
        ValueError: _description_

    Returns:
        _type_: _description_
    """
    key = os.getenv("GEMINI_API_KEY")
    if key is None:
        raise ValueError("Please provide a Gemini API key")
    return str(key)


# Main class
class Gemini(BaseMultiModalModel):
    """Gemini model

    Args:
        model_name (str, optional): _description_. Defaults to "gemini-pro".
        gemini_api_key (str, optional): _description_. Defaults to get_gemini_api_key_env.
        return_safety (bool, optional): _description_. Defaults to False.
        candidates (bool, optional): _description_. Defaults to False.
        stream (bool, optional): _description_. Defaults to False.
        candidate_count (int, optional): _description_. Defaults to 1.
        stop_sequence ([type], optional): _description_. Defaults to ['x'].
        max_tokens (int, optional): _description_. Defaults to 100.
        temperature (float, optional): _description_. Defaults to 0.9.
```

```
Methods:
    run: Run the Gemini model
    process_img: Process the image
    chat: Chat with the Gemini model
    list_models: List the Gemini models
    stream_tokens: Stream the tokens
    process_img_pil: Process img



Examples:
    >>> from swarm_models import Gemini
    >>> gemini = Gemini()
    >>> gemini.run(
            task="A dog",
            img="dog.png",
        )
"""

def __init__(
    self,
    model_name: str = "gemini-pro-vision",
    gemini_api_key: str = get_gemini_api_key_env,
    return_safety: bool = False,
    candidates: bool = False,
    stream: bool = False,
    candidate_count: int = 1,
    transport: str = "rest",
    stop_sequence=["x"],
    max_tokens: int = 100,
    temperature: float = 0.9,
    system_prompt: str = None,
    *args,
    **kwargs,
):
    super().__init__(model_name, *args, **kwargs)
    self.model_name = model_name
    self.gemini_api_key = gemini_api_key
    self.safety = return_safety
    self.candidates = candidates
    self.stream = stream
    self.candidate_count = candidate_count
    self.stop_sequence = stop_sequence
    self.max_tokens = max_tokens
    self.temperature = temperature
    self.system_prompt = system_prompt

    # Configure the API key
    genai.configure(
        api_key=gemini_api_key,
        transport=transport,
            *args,
            **kwargs,
        )

        # Prepare the generation config
        self.generation_config = GenerationConfig(
            candidate_count=candidate_count,
            # stop_sequence=stop_sequence,
            max_output_tokens=max_tokens,
            temperature=temperature,
            *args,
            **kwargs,
        )

        # Initialize the model
        self.model = genai.GenerativeModel(
            model_name, *args, **kwargs
        )

        # Check for the key
        if self.gemini_api_key is None:
            raise ValueError("Please provide a Gemini API key")

    def system_prompt_prep(
        self,
        task: str = None,
    *args,

    **kwargs,

):

    """System prompt



    Args:

        system_prompt (str, optional): _description_. Defaults to None.

    """

    PROMPT = f"""



    {self.system_prompt}



    ######



    {task}



    """

    return PROMPT



def run(

    self,

    task: str = None,

    img: str = None,

    *args,

    **kwargs,

) -> str:
```

```
"""Run the Gemini model


Args:
    task (str, optional): textual task. Defaults to None.
    img (str, optional): img. Defaults to None.


Returns:
    str: output from the model
"""
try:
    prepare_prompt = self.system_prompt_prep(task)
    if img:
        # process_img = self.process_img(img, *args, **kwargs)
        process_img = self.process_img_pil(img)
        response = self.model.generate_content(
            contents=[prepare_prompt, process_img],
            generation_config=self.generation_config,
            stream=self.stream,
            *args,
            **kwargs,
        )
        return response.text
    else:
        response = self.model.generate_content(
            prepare_prompt,
            stream=self.stream,
                *args,
                **kwargs,
            )
            return response.text
    except Exception as error:
        print(f"Error running Gemini model: {error}")
        print(f"Please check the task and image: {task}, {img}")
        raise error

def process_img(
    self,
    img: str = None,
    type: str = "image/png",
    *args,
    **kwargs,
):
    """Process the image

    Args:
        img (str, optional): _description_. Defaults to None.
        type (str, optional): _description_. Defaults to "image/png".

    Raises:
        ValueError: _description_
        ValueError: _description_
        ValueError: _description_
    """
    try:
        if img is None:
            raise ValueError("Please provide an image to process")
        if type is None:
            raise ValueError("Please provide the image type")
        if self.gemini_api_key is None:
            raise ValueError("Please provide a Gemini API key")

        # Load the image
        img = [
            {"mime_type": type, "data": Path(img).read_bytes()}
        ]
    except Exception as error:
        print(f"Error processing image: {error}")

def chat(
    self,
    task: str = None,
    img: str = None,
    *args,
    **kwargs,
) -> str:
    """Chat with the Gemini model

    Args:
        task (str, optional): _description_. Defaults to None.
        img (str, optional): _description_. Defaults to None.

    Returns:
        str: _description_
    """
    chat = self.model.start_chat()
    response = chat.send_message(task, *args, **kwargs)
    response1 = response.text
    print(response1)
    response = chat.send_message(img, *args, **kwargs)

def list_models(self) -> str:
    """List the Gemini models

    Returns:
        str: _description_
    """
    for m in genai.list_models():
        if "generateContent" in m.supported_generation_methods:
            print(m.name)

def stream_tokens(self, content: str = None):
    """Stream the tokens

    Args:
        content (t, optional): _description_. Defaults to None.
    """
    for chunk in content:
        print(chunk.text)
        print("_" * 80)


def process_img_pil(self, img: str = None):
    """Process img


    Args:
        img (str, optional): _description_. Defaults to None.


    Returns:
        _type_: _description_
    """
    img = Image.open(img)
    return img
```