```
from typing import Any, Optional


import torch
from diffusers import AutoPipelineForText2Image


from swarm_models.base_multimodal_model import BaseMultiModalModel



class OpenDalle(BaseMultiModalModel):
    """OpenDalle model class


    Attributes:
            model_name (str): The name or path of the model to be used. Defaults to
"dataautogpt3/OpenDalleV1.1".
        torch_dtype (torch.dtype): The torch data type to be used. Defaults to torch.float16.
        device (str): The device to be used for computation. Defaults to "cuda".


    Examples:
        >>> from swarm_models.open_dalle import OpenDalle
        >>> od = OpenDalle()
        >>> od.run("A picture of a cat")


    """


    def __init__(
        self,
        model_name: str = "dataautogpt3/OpenDalleV1.1",
        torch_dtype: Any = torch.float16,
        device: str = "cuda",
        *args,
        **kwargs,
    ):
        """
        Initializes the OpenDalle model.

        Args:
                model_name (str, optional): The name or path of the model to be used. Defaults to
"dataautogpt3/OpenDalleV1.1".
            torch_dtype (torch.dtype, optional): The torch data type to be used. Defaults to torch.float16.
            device (str, optional): The device to be used for computation. Defaults to "cuda".
            *args: Variable length argument list.
            **kwargs: Arbitrary keyword arguments.
        """
        self.pipeline = AutoPipelineForText2Image.from_pretrained(
            model_name, torch_dtype=torch_dtype, *args, **kwargs
        ).to(device)

    def run(self, task: Optional[str] = None, *args, **kwargs):
        """Run the OpenDalle model

        Args:
            task (str, optional): The task to be performed. Defaults to None.
            *args: Variable length argument list.
            **kwargs: Arbitrary keyword arguments.


        Returns:
            [type]: [description]
        """
        try:
            if task is None:
                raise ValueError("Task cannot be None")
            if not isinstance(task, str):
                raise TypeError("Task must be a string")
            if len(task) < 1:
                raise ValueError("Task cannot be empty")
            return self.pipeline(task, *args, **kwargs).images[0]
        except Exception as error:
            print(f"[ERROR][OpenDalle] {error}")
            raise error
```