```
import asyncio
from abc import abstractmethod
from concurrent.futures import ThreadPoolExecutor
from typing import List, Optional

from diffusers.utils import export_to_video

from swarm_models.base_llm import BaseLLM


class BaseTextToVideo(BaseLLM):
    """BaseTextToVideo class represents prebuilt text-to-video models."""

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    @abstractmethod
    def run(self, *args, **kwargs):
        pass

    def __call__(
        self,
        task: Optional[str] = None,
        img: Optional[str] = None,
        *args,
        **kwargs,
):
    """
    Performs forward pass on the input task and returns the path of the generated video.


    Args:
        task (str): The task to perform.


    Returns:
        str: The path of the generated video.
    """
    return self.run(task, img, *args, **kwargs)


def save_video_path(
    self, video_path: Optional[str] = None, *args, **kwargs
):
    """Saves the generated video to the specified path.


    Args:
        video_path (Optional[str], optional): _description_. Defaults to None.


    Returns:
     str: The path of the generated video.
    """
    return export_to_video(video_path, *args, **kwargs)


def run_batched(
    self,
    tasks: List[str] = None,
    imgs: List[str] = None,
    *args,
    **kwargs,
):
    # TODO: Implement batched inference
    tasks = tasks or []
    imgs = imgs or []
    if len(tasks) != len(imgs):
        raise ValueError(
            "The number of tasks and images should be the same."
        )
    return [
        self.run(task, img, *args, **kwargs)
        for task, img in zip(tasks, imgs)
    ]

def run_concurrent_batched(
    self,
    tasks: List[str] = None,
    imgs: List[str] = None,
    *args,
    **kwargs,
):
    tasks = tasks or []
    imgs = imgs or []

    if len(tasks) != len(imgs):

        raise ValueError(

            "The number of tasks and images should be the same."

        )

    with ThreadPoolExecutor(max_workers=4) as executor:

        loop = asyncio.get_event_loop()

        tasks = [

            loop.run_in_executor(

                executor, self.run, task, img, *args, **kwargs

            )

            for task, img in zip(tasks, imgs)

        ]

        return loop.run_until_complete(asyncio.gather(*tasks))


# Run the model in async mode

def arun(

    self,

    task: Optional[str] = None,

    img: Optional[str] = None,

    *args,

    **kwargs,

):

    loop = asyncio.get_event_loop()

    return loop.run_until_complete(

        self.run(task, img, *args, **kwargs)
    )


def arun_batched(
    self,
    tasks: List[str] = None,
    imgs: List[str] = None,
    *args,
    **kwargs,
):
    loop = asyncio.get_event_loop()
    return loop.run_until_complete(
        self.run_batched(tasks, imgs, *args, **kwargs)
    )
```