```
import torch
from diffusers import DiffusionPipeline, DPMSolverMultistepScheduler
from diffusers.utils import export_to_video


class ZeroscopeTTV:
    """
    ZeroscopeTTV class represents a zero-shot video generation model.

    Args:
        model_name (str): The name of the pre-trained model to use.
        torch_dtype (torch.dtype): The torch data type to use for computations.
        chunk_size (int): The size of chunks for forward chunking.
        dim (int): The dimension along which to split the input for forward chunking.
        num_inference_steps (int): The number of inference steps to perform.
        height (int): The height of the video frames.
        width (int): The width of the video frames.
        num_frames (int): The number of frames in the video.

    Attributes:
        model_name (str): The name of the pre-trained model.
        torch_dtype (torch.dtype): The torch data type used for computations.
        chunk_size (int): The size of chunks for forward chunking.
        dim (int): The dimension along which the input is split for forward chunking.
        num_inference_steps (int): The number of inference steps to perform.
        height (int): The height of the video frames.
```

```
    width (int): The width of the video frames.
    num_frames (int): The number of frames in the video.
    pipe (DiffusionPipeline): The diffusion pipeline for video generation.

Methods:
    forward(task: str = None, *args, **kwargs) -> str:
        Performs forward pass on the input task and returns the path of the generated video.

Examples:
    >>> from swarm_models
    >>> zeroscope = ZeroscopeTTV()
    >>> task = "A person is walking on the street."
    >>> video_path = zeroscope(task)

"""

def __init__(
    self,
    model_name: str = "cerspense/zeroscope_v2_576w",
    torch_dtype=torch.float16,
    chunk_size: int = 1,
    dim: int = 1,
    num_inference_steps: int = 40,
    height: int = 320,
    width: int = 576,
    num_frames: int = 36,
```

```
    *args,
    **kwargs,
):
    self.model_name = model_name
    self.torch_dtype = torch_dtype
    self.chunk_size = chunk_size
    self.dim = dim
    self.num_inference_steps = num_inference_steps
    self.height = height
    self.width = width
    self.num_frames = num_frames

    self.pipe = DiffusionPipeline.from_pretrained(
        model_name, torch_dtype=torch_dtype, *args, **kwargs
    )
    self.pipe.scheduler = DPMSolverMultistepScheduler(
        self.pipe.scheduler.config,
    )
    self.pipe_enable_model_cpu_offload()
    self.pipe.enable_vae_slicing()
    self.pipe.unet.enable_forward_chunking(
        chunk_size=chunk_size, dim=dim
    )

def run(self, task: str = None, *args, **kwargs):
    """
```

```
Performs a forward pass on the input task and returns the path of the generated video.

Args:
    task (str): The input task for video generation.

Returns:
    str: The path of the generated video.
"""
try:
    video_frames = self.pipe(
        task,
        num_inference_steps=self.num_inference_steps,
        height=self.height,
        width=self.width,
        num_frames=self.num_frames,
        *args,
        **kwargs,
    ).frames
    video_path = export_to_video(video_frames)
    return video_path
except Exception as error:
    print(f"Error in [ZeroscopeTTV.forward]: {error}")
    raise error
```