```
from typing import List

import requests
import torch
from PIL import Image
from transformers import SamModel, SamProcessor

device = "cuda" if torch.cuda.is_available() else "cpu"


class SAM:
    """
    Class representing the SAM (Segmentation and Masking) model.

    Args:
                    model_name (str): The name of the pre-trained SAM model. Default is "facebook/sam-vit-huge".
        device (torch.device): The device to run the model on. Default is the current device.
          input_points (List[List[int]]): The 2D location of a window in the image to segment. Default is [[450, 600]].
        *args: Additional positional arguments.
        **kwargs: Additional keyword arguments.

    Attributes:
        model_name (str): The name of the pre-trained SAM model.
        device (torch.device): The device to run the model on.
```

```
        input_points (List[List[int]]): The 2D location of a window in the image to segment.
        model (SamModel): The pre-trained SAM model.
        processor (SamProcessor): The processor for the SAM model.

    Methods:
        run(task=None, img=None, *args, **kwargs): Runs the SAM model on the given image and returns the segmentation scores and masks.
        process_img(img: str = None, *args, **kwargs): Processes the input image and returns the processed image.

    """

    def __init__(
        self,
        model_name: str = "facebook/sam-vit-huge",
        device=device,
        input_points: List[List[int]] = [[450, 600]],
        *args,
        **kwargs,
    ):
        self.model_name = model_name
        self.device = device
        self.input_points = input_points

        self.model = SamModel.from_pretrained(
            model_name, *args, **kwargs
```

```
    ).to(device)

    self.processor = SamProcessor.from_pretrained(model_name)

def run(self, task: str = None, img: str = None, *args, **kwargs):
    """
    Runs the SAM model on the given image and returns the segmentation scores and masks.

    Args:
        task: The task to perform. Not used in this method.
        img: The input image to segment.
        *args: Additional positional arguments.
        **kwargs: Additional keyword arguments.

    Returns:
        Tuple: A tuple containing the segmentation scores and masks.

    """
    img = self.process_img(img)

    # Specify the points of the mask to segment
    input_points = [
        self.input_points
    ]  # 2D location of a window in the image

    # Preprocess the image
```

```
    inputs = self.processor(
        img, input_points=input_points, return_tensors="pt"
    ).to(device)

    with torch.no_grad():
        outputs = self.model(**inputs)  # noqa: E999

    masks = self.processor.image_processor.post_process_masks(
        outputs.pred_masks.cpu(),
        inputs["original_sizes"].cpu(),
        inputs["reshaped_input_sizes"].cpu(),
    )
    scores = outputs.iou_scores

    return scores, masks

def process_img(self, img: str = None, *args, **kwargs):
    """
    Processes the input image and returns the processed image.

    Args:
        img (str): The URL or file path of the input image.
        *args: Additional positional arguments.
        **kwargs: Additional keyword arguments.

    Returns:
```

```
    Image: The processed image.

"""
raw_image = Image.open(
    requests.get(img, stream=True, *args, **kwargs).raw
).convert("RGB")

return raw_image
```