```
!pip install git+https://github.com/facebookresearch/segment-anything.git
!pip install -q git+https://github.com/huggingface/transformers.git
!pip install datasets
!pip install -q monai
```

```
Collecting git+https://github.com/facebookresearch/segment-anything.git
  Cloning https://github.com/facebookresearch/segment-anything.git to /tmp/pip-req-build-421wpdwe
  Running command git clone --filter=blob:none --quiet https://github.com/facebookresearch/segment-anything.git /tmp/pip-req-bu
  Resolved https://github.com/facebookresearch/segment-anything.git to commit dca509fe793f601edb92606367a655c15ac00fdf
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: segment_anything
  Building wheel for segment_anything (setup.py) ... done
  Created wheel for segment_anything: filename=segment_anything-1.0-py3-none-any.whl size=36592 sha256=92aea4710f22d118e6786677
  Stored in directory: /tmp/pip-ephem-wheel-cache-1m5yta23/wheels/29/82/ff/04e2be9805a1cb48bec0b85b5a6da6b63f647645750a0e42d4
Successfully built segment_anything
Installing collected packages: segment_anything
Successfully installed segment_anything-1.0
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 528.7/528.7 kB 16.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.8/46.8 kB 4.4 MB/s eta 0:00:00
  Building wheel for transformers (pyproject.toml) ... done
Requirement already satisfied: datasets in /usr/local/lib/python3.12/dist-packages (4.0.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from datasets) (3.19.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from datasets) (2.0.2)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.12/dist-packages (from datasets) (2.32.4)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.12/dist-packages (from datasets) (4.67.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=2025
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (1.0.0rc2)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from datasets) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from datasets) (6.0.3)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=2025.3
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datas
Requirement already satisfied: typer-slim in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.2
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->d
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->data
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2.9.0
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fs
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspe
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->f
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1-
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fs
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->f
Requirement already satisfied: anyio in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub>=0.24.0
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx<1,>=0.23.0->hugg
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas->datase
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.12/dist-packages (from typer-slim->huggingface-hub>=0.24
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.12/dist-packages (from anyio->httpx<1,>=0.23.0->huggingfa
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.7/2.7 MB 36.7 MB/s eta 0:00:00
```

```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.12/dist-packages (4.12.0.88)
Requirement already satisfied: numpy<2.3.0,>=2 in /usr/local/lib/python3.12/dist-packages (from opencv-python) (2.0.2)
```

```
!pip install -q patchify --no-deps
```

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import random
from glob import glob
from PIL import Image
import torch
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader
from datasets import Dataset as HFDataset
from transformers import SamProcessor, SamModel, SamConfig
```

```
from tqdm import tqdm
from statistics import mean
import monai
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!mkdir -p /content/oilspill_dataset
```

```
!wget -q https://zenodo.org/records/10555314/files/dataset.zip -O /content/oilspill_dataset/dataset.zip
```

```
!unzip -q /content/oilspill_dataset/dataset.zip -d /content/oilspill_dataset/
```

```
def load_oil_spill_data(base_path='/content/oilspill_dataset', target_size=(256, 256)):
    """Load oil spill images and masks from the dataset"""

    # Get all image and mask paths
    train_img_paths = sorted(glob(f"{base_path}/train/images/*.jpg"))
    train_mask_paths = sorted(glob(f"{base_path}/train/masks/*.png"))

    print(f"Found {len(train_img_paths)} training images")
    print(f"Found {len(train_mask_paths)} training masks")

    # Load images and masks
    images = []
    masks = []

    for img_path, mask_path in tqdm(zip(train_img_paths, train_mask_paths),
                                    total=len(train_img_paths),
                                    desc="Loading data"):
        # Load image (convert to grayscale or keep RGB based on your need)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, target_size, interpolation=cv2.INTER_LINEAR)

        # Load mask (convert to binary)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        mask = cv2.resize(mask, target_size, interpolation=cv2.INTER_NEAREST)
        mask = (mask > 0).astype(np.uint8)  # Binarize mask

        images.append(img)
        masks.append(mask)

    return np.array(images), np.array(masks)

# Load the data
images, masks = load_oil_spill_data()
print(f"Images shape: {images.shape}")
print(f"Masks shape: {masks.shape}")
def load_oil_spill_data(base_path='/content/oilspill_dataset', target_size=(256, 256)):
    """Load oil spill images and masks from the dataset"""

    # Get all image and mask paths
    train_img_paths = sorted(glob(f"{base_path}/train/images/*.jpg"))
    train_mask_paths = sorted(glob(f"{base_path}/train/masks/*.png"))

    print(f"Found {len(train_img_paths)} training images")
    print(f"Found {len(train_mask_paths)} training masks")

    # Load images and masks
    images = []
    masks = []

    for img_path, mask_path in tqdm(zip(train_img_paths, train_mask_paths),
                                    total=len(train_img_paths),
                                    desc="Loading data"):
        # Load image (convert to grayscale or keep RGB based on your need)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, target_size, interpolation=cv2.INTER_LINEAR)

        # Load mask (convert to binary)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
```

```
            mask = cv2.resize(mask, target_size, interpolation=cv2.INTER_NEAREST)
            mask = (mask > 0).astype(np.uint8)  # Binarize mask

            images.append(img)
            masks.append(mask)

        return np.array(images), np.array(masks)

    # Load the data
    images, masks = load_oil_spill_data()
    print(f"Images shape: {images.shape}")
    print(f"Masks shape: {masks.shape}")
```

```
Found 811 training images
Found 811 training masks
Loading data: 100%|██████████| 811/811 [00:40<00:00, 19.94it/s]
Images shape: (811, 256, 256, 3)
Masks shape: (811, 256, 256)
Found 811 training images
Found 811 training masks
Loading data: 100%|██████████| 811/811 [00:40<00:00, 19.93it/s]
Images shape: (811, 256, 256, 3)
Masks shape: (811, 256, 256)
```

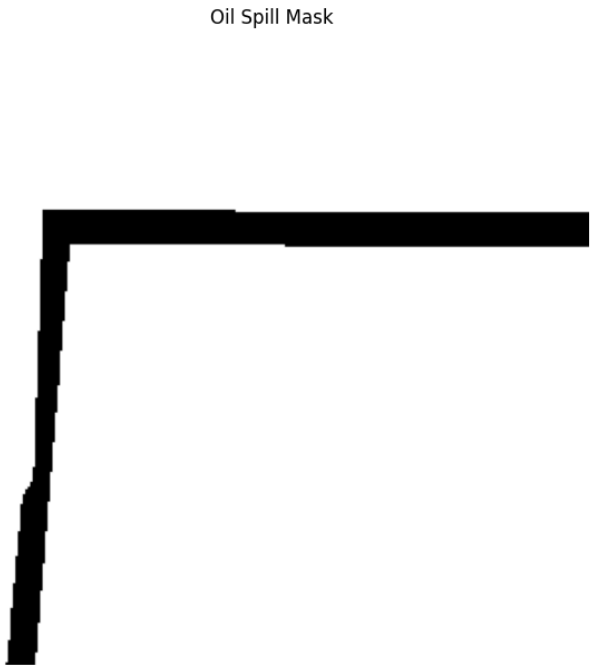
```
valid_indices = [i for i, mask in enumerate(masks) if mask.max() != 0]
filtered_images = images[valid_indices]
filtered_masks = masks[valid_indices]

print(f"Filtered images shape: {filtered_images.shape}")
print(f"Filtered masks shape: {filtered_masks.shape}")
```

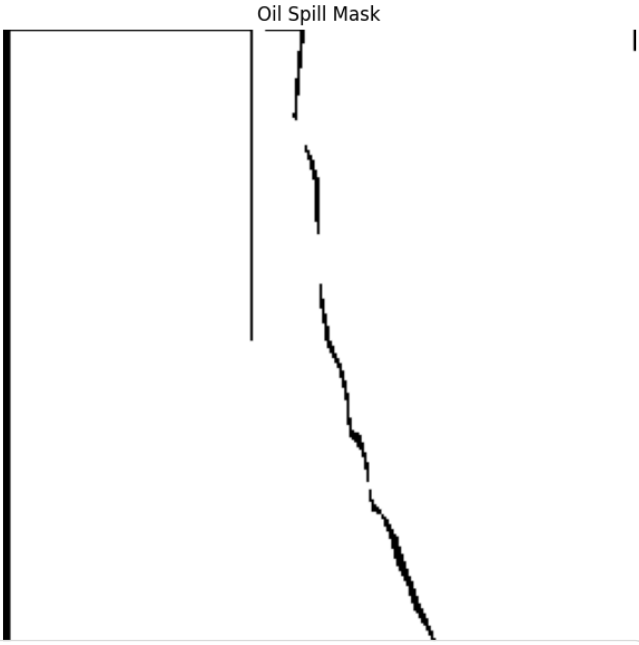
```
Filtered images shape: (811, 256, 256, 3)
Filtered masks shape: (811, 256, 256)
```

```
dataset_dict = {
    "image": [Image.fromarray(img) for img in filtered_images],
    "label": [Image.fromarray(mask) for mask in filtered_masks],
}

dataset = HFDataset.from_dict(dataset_dict)
print(f"Dataset created with {len(dataset)} samples")
```

```
Dataset created with 811 samples
```

```
def visualize_sample(dataset, idx=None):
    """Visualize a random or specific sample from dataset"""
    if idx is None:
        idx = random.randint(0, len(dataset)-1)

    example_image = dataset[idx]["image"]
    example_mask = dataset[idx]["label"]

    fig, axes = plt.subplots(1, 2, figsize=(12, 6))

    axes[0].imshow(np.array(example_image))
    axes[0].set_title("Oil Spill Image")
    axes[0].axis('off')

    axes[1].imshow(example_mask, cmap='gray')
    axes[1].set_title("Oil Spill Mask")
    axes[1].axis('off')

    plt.tight_layout()
    plt.show()

# Visualize a few samples
for _ in range(3):
    visualize_sample(dataset)
```






```
def get_bounding_box(ground_truth_map):
    """Extract bounding box from binary mask with random perturbation"""
    y_indices, x_indices = np.where(ground_truth_map > 0)

    if len(x_indices) == 0 or len(y_indices) == 0:
        # Return a default box if mask is empty
        return [0, 0, ground_truth_map.shape[1], ground_truth_map.shape[0]]

    x_min, x_max = np.min(x_indices), np.max(x_indices)
    y_min, y_max = np.min(y_indices), np.max(y_indices)

    # Add perturbation to bounding box coordinates
    H, W = ground_truth_map.shape
    x_min = max(0, x_min - np.random.randint(0, 20))
    x_max = min(W, x_max + np.random.randint(0, 20))
    y_min = max(0, y_min - np.random.randint(0, 20))
    y_max = min(H, y_max + np.random.randint(0, 20))

    bbox = [x_min, y_min, x_max, y_max]
    return bbox
```

```
class SAMDataset(Dataset):
    """Custom dataset for SAM training"""

    def __init__(self, dataset, processor):
        self.dataset = dataset
        self.processor = processor
```

```
    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        item = self.dataset[idx]
        image = item["image"]
        ground_truth_mask = np.array(item["label"])

        # Get bounding box prompt
        prompt = get_bounding_box(ground_truth_mask)

        # Prepare image and prompt for the model
        inputs = self.processor(image, input_boxes=[[prompt]], return_tensors="pt")

        # Remove batch dimension
        inputs = {k: v.squeeze(0) for k, v in inputs.items()}

        # Add ground truth segmentation
        inputs["ground_truth_mask"] = ground_truth_mask

        return inputs
```

```
processor = SamProcessor.from_pretrained("facebook/sam-vit-base")
train_dataset = SAMDataset(dataset=dataset, processor=processor)

# Create DataLoader
train_dataloader = DataLoader(train_dataset, batch_size=4, shuffle=True, drop_last=False)

print("DataLoader created successfully!")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
preprocessor_config.json: 100%  466/466 [00:00<00:00, 48.6kB/s]
DataLoader created successfully!
```

```
model = SamModel.from_pretrained("facebook/sam-vit-base")

# Freeze vision encoder and prompt encoder
for name, param in model.named_parameters():
    if name.startswith("vision_encoder") or name.startswith("prompt_encoder"):
        param.requires_grad_(False)

print("Model loaded and configured!")
```

```
model.safetensors: 100%  375M/375M [00:18<00:00, 17.1MB/s]
Model loaded and configured!
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")

model.to(device)

# Optimizer and loss function
optimizer = Adam(model.mask_decoder.parameters(), lr=1e-5, weight_decay=0)
seg_loss = monai.losses.DiceCELoss(sigmoid=True, squared_pred=True, reduction='mean')
```

```
Using device: cuda
```

```
Start coding or generate with AI.
```

```
def train_model(model, dataloader, optimizer, seg_loss, num_epochs=10, save_path='/content/drive/MyDrive/models/oil_spill_sam.p
    """Train the SAM model"""

    model.train()

    for epoch in range(num_epochs):
        epoch_losses = []

        for batch in tqdm(dataloader, desc=f"Epoch {epoch+1}/{num_epochs}"):
            # Forward pass
            outputs = model(
```

```
                pixel_values=batch["pixel_values"].to(device),
                input_boxes=batch["input_boxes"].to(device),
                multimask_output=False
            )

            # Compute loss
            predicted_masks = outputs.pred_masks.squeeze(1)
            ground_truth_masks = batch["ground_truth_mask"].float().to(device)
            loss = seg_loss(predicted_masks, ground_truth_masks.unsqueeze(1))

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            epoch_losses.append(loss.item())

        mean_loss = mean(epoch_losses)
        print(f'Epoch {epoch+1}/{num_epochs} - Mean loss: {mean_loss:.4f}')

        # Save checkpoint every epoch
        os.makedirs(os.path.dirname(save_path), exist_ok=True)
        torch.save(model.state_dict(), save_path)
        print(f"Model saved to {save_path}")

    return model

# Train the model
trained_model = train_model(model, train_dataloader, optimizer, seg_loss, num_epochs=10)
```

```
Epoch 1/10:   0%|          | 0/203 [00:00<?, ?it/s]
Epoch 1/10:   0%|          | 1/203 [00:04<14:47,  4.39s/it]
Epoch 1/10:   1%|          | 2/203 [00:06<09:23,  2.80s/it]
Epoch 1/10:   1%|▏         | 3/203 [00:07<07:39,  2.30s/it]
Epoch 1/10:   2%|▏         | 4/203 [00:09<06:48,  2.05s/it]
Epoch 1/10:   2%|▏         | 5/203 [00:11<06:28,  1.96s/it]
Epoch 1/10:   3%|▎         | 6/203 [00:12<06:08,  1.87s/it]
Epoch 1/10:   3%|▎         | 7/203 [00:14<05:54,  1.81s/it]
Epoch 1/10:   4%|▍         | 8/203 [00:16<05:45,  1.77s/it]
Epoch 1/10:   4%|▍         | 9/203 [00:18<05:39,  1.75s/it]
Epoch 1/10:   5%|▍         | 10/203 [00:19<05:34,  1.73s/it]
Epoch 1/10:   5%|▌         | 11/203 [00:21<05:31,  1.73s/it]
Epoch 1/10:   6%|▌         | 12/203 [00:23<05:33,  1.75s/it]
Epoch 1/10:   6%|▋         | 13/203 [00:25<05:33,  1.76s/it]
Epoch 1/10:   7%|▋         | 14/203 [00:26<05:29,  1.75s/it]
Epoch 1/10:   7%|▋         | 15/203 [00:28<05:25,  1.73s/it]
Epoch 1/10:   8%|▊         | 16/203 [00:30<05:22,  1.72s/it]
Epoch 1/10:   8%|▊         | 17/203 [00:31<05:20,  1.72s/it]
Epoch 1/10:   9%|▉         | 18/203 [00:33<05:18,  1.72s/it]
Epoch 1/10:   9%|▉         | 19/203 [00:35<05:16,  1.72s/it]
Epoch 1/10:  10%|▉         | 20/203 [00:37<05:21,  1.76s/it]
Epoch 1/10:  10%|▉         | 21/203 [00:38<05:22,  1.77s/it]
Epoch 1/10:  11%|▉         | 22/203 [00:40<05:17,  1.76s/it]
Epoch 1/10:  11%|█         | 23/203 [00:42<05:14,  1.75s/it]
Epoch 1/10:  12%|█         | 24/203 [00:44<05:12,  1.74s/it]
Epoch 1/10:  12%|█▏        | 25/203 [00:45<05:09,  1.74s/it]
Epoch 1/10:  13%|█▏        | 26/203 [00:47<05:09,  1.75s/it]
Epoch 1/10:  13%|█▎        | 27/203 [00:49<05:11,  1.77s/it]
Epoch 1/10:  14%|█▍        | 28/203 [00:51<05:12,  1.79s/it]
Epoch 1/10:  14%|█▍        | 29/203 [00:53<05:08,  1.77s/it]
Epoch 1/10:  15%|█▍        | 30/203 [00:54<05:05,  1.77s/it]
Epoch 1/10:  15%|█▌        | 31/203 [00:56<05:03,  1.77s/it]
Epoch 1/10:  16%|█▌        | 32/203 [00:58<05:01,  1.76s/it]
Epoch 1/10:  16%|█▋        | 33/203 [01:00<05:00,  1.77s/it]
Epoch 1/10:  17%|█▋        | 34/203 [01:01<04:58,  1.77s/it]
Epoch 1/10:  17%|█▋        | 35/203 [01:03<05:02,  1.80s/it]
Epoch 1/10:  18%|█▊        | 36/203 [01:05<05:00,  1.80s/it]
Epoch 1/10:  18%|█▊        | 37/203 [01:07<04:57,  1.79s/it]
Epoch 1/10:  19%|█▊        | 38/203 [01:09<04:55,  1.79s/it]
Epoch 1/10:  19%|█▉        | 39/203 [01:10<04:53,  1.79s/it]
Epoch 1/10:  20%|█▉        | 40/203 [01:12<04:51,  1.79s/it]
Epoch 1/10:  20%|██        | 41/203 [01:14<04:50,  1.79s/it]
Epoch 1/10:  21%|██        | 42/203 [01:16<04:52,  1.82s/it]
Epoch 1/10:  21%|██        | 43/203 [01:18<04:54,  1.84s/it]
Epoch 1/10:  22%|██▏       | 44/203 [01:20<04:50,  1.83s/it]
Epoch 1/10:  22%|██▏       | 45/203 [01:21<04:48,  1.82s/it]
Epoch 1/10:  23%|██▎       | 46/203 [01:23<04:45,  1.82s/it]
Epoch 1/10:  23%|██▎       | 47/203 [01:25<04:43,  1.82s/it]
Epoch 1/10:  24%|██▎       | 48/203 [01:27<04:40,  1.81s/it]
Epoch 1/10:  24%|██▍       | 49/203 [01:29<04:43,  1.84s/it]
Epoch 1/10:  25%|██▍       | 50/203 [01:31<04:45,  1.87s/it]
Epoch 1/10:  25%|██▌       | 51/203 [01:32<04:41,  1.85s/it]
Epoch 1/10:  26%|██▌       | 52/203 [01:34<04:38,  1.85s/it]
Epoch 1/10:  26%|██▌       | 53/203 [01:36<04:35,  1.84s/it]
Epoch 1/10:  27%|██▋       | 54/203 [01:38<04:33,  1.84s/it]
Epoch 1/10:  27%|██▋       | 55/203 [01:40<04:31,  1.84s/it]
```

Epoch 1/10: 28%|[redacted] | 56/203 [01:42<04:35, 1.87s/it]

```
# Save the trained model to Google Drive
save_path = '/content/drive/MyDrive/ColabNotebooks/models/SAM/oil_spill_sam_final.pth'

# Create directory if it doesn't exist
os.makedirs(os.path.dirname(save_path), exist_ok=True)

# Save the model
torch.save(trained_model.state_dict(), save_path)
print(f"Model saved successfully to {save_path}")
```

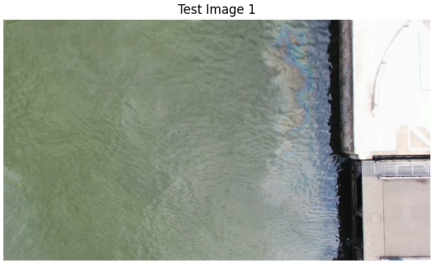
```
Model saved successfully to /content/drive/MyDrive/ColabNotebooks/models/SAM/oil_spill_sam_final.pth
```

```
def load_test_data(base_path='/content/oilspill_dataset'):
    """Load test images"""
    test_img_paths = sorted(glob(f"{base_path}/test/images/*.jpg"))

    test_images = []
    for img_path in test_img_paths:
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        test_images.append(img)

    return test_images, test_img_paths

def create_grid_points(array_size, grid_size=10):
    """Create a grid of points for prompting"""
    x = np.linspace(0, array_size-1, grid_size)
    y = np.linspace(0, array_size-1, grid_size)
    xv, yv = np.meshgrid(x, y)

    xv_list = xv.tolist()
    yv_list = yv.tolist()

    input_points = [[[int(x), int(y)] for x, y in zip(x_row, y_row)]
                    for x_row, y_row in zip(xv_list, yv_list)]

    input_points = torch.tensor(input_points).view(1, 1, grid_size*grid_size, 2)
    return input_points

def predict_oil_spill(model, processor, image, device, use_points=True):
    """Predict oil spill segmentation on a single image"""

    pil_image = Image.fromarray(image)

    if use_points:
        # Use grid of points as prompt
        h, w = image.shape[:2]
        grid_size = min(h, w) // 25  # Adaptive grid size
        input_points = create_grid_points(min(h, w), grid_size)
        inputs = processor(pil_image, input_points=input_points, return_tensors="pt")
    else:
        # No prompt
        inputs = processor(pil_image, return_tensors="pt")

    inputs = {k: v.to(device) for k, v in inputs.items()}

    model.eval()
    with torch.no_grad():
        outputs = model(**inputs, multimask_output=False)

    # Apply sigmoid and threshold
    seg_prob = torch.sigmoid(outputs.pred_masks.squeeze(1))
    seg_prob = seg_prob.cpu().numpy().squeeze()
    seg_mask = (seg_prob > 0.5).astype(np.uint8)

    return seg_mask, seg_prob

# Load test data
test_images, test_paths = load_test_data()
print(f"Loaded {len(test_images)} test images")

# Predict on a few test images
for i in range(min(5, len(test_images))):
    test_image = test_images[i]
    seg_mask, seg_prob = predict_oil_spill(trained_model, processor, test_image, device)

    fig, axes = plt.subplots(1, 3, figsize=(18, 6))

    axes[0].imshow(test_image)
```

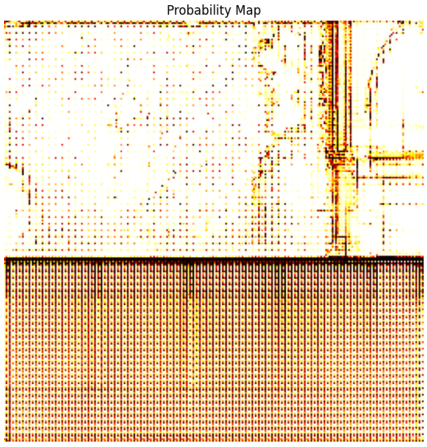
```
        axes[0].set_title(f"Test Image {i+1}")
        axes[0].axis('off')

        axes[1].imshow(seg_prob, cmap='hot')
        axes[1].set_title("Probability Map")
        axes[1].axis('off')

        axes[2].imshow(seg_mask, cmap='gray')
        axes[2].set_title("Predicted Mask")
        axes[2].axis('off')

        plt.tight_layout()
        plt.show()

    print("Oil spill detection pipeline complete!")
```

```
Loaded 254 test images
```

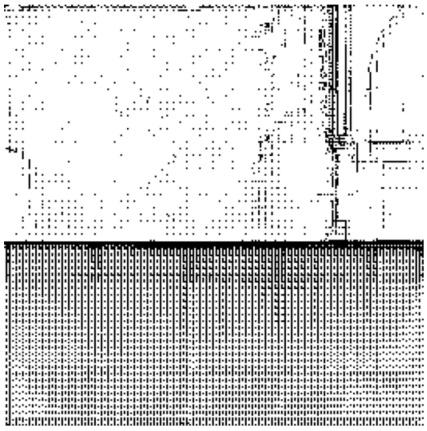
Test Image 1



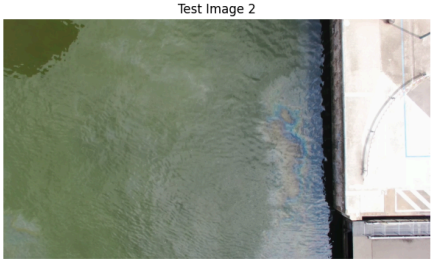
Probability Map



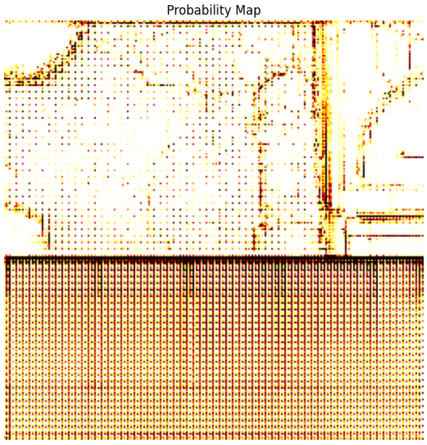
Predicted Mask



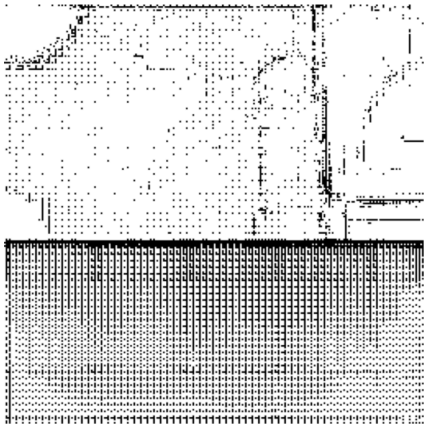
Test Image 2



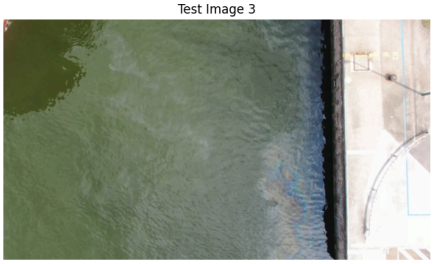
Probability Map



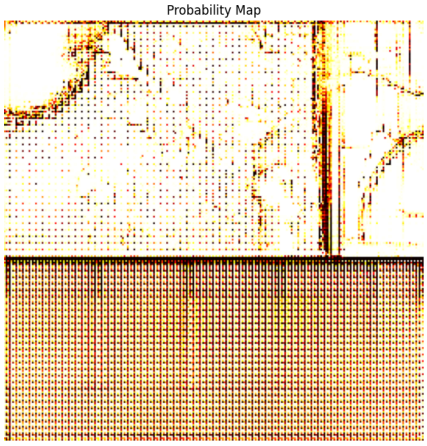
Predicted Mask



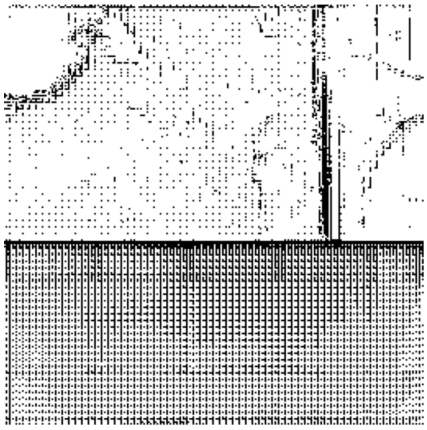
Test Image 3



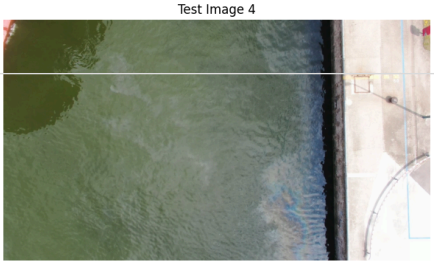
Probability Map



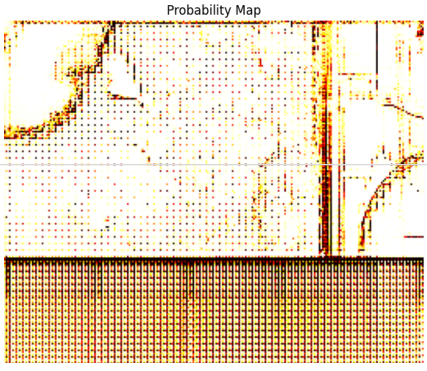
Predicted Mask



Test Image 4



Probability Map



Predicted Mask