In [ ]:
```python
# Импорт необходимых библиотек
import cv2
import numpy as np
import os
from sahi import AutoDetectionModel
from sahi.predict import get_sliced_prediction
from pathlib import Path
from boxmot import DeepOCSORT
from IPython.display import display, Image
```

In [ ]:
```python
# Создание директорий


# Создаем директории для сохранения изображений и меток
os.makedirs("images", exist_ok=True)
os.makedirs("labels", exist_ok=True)


#Инициализация трекера и модели детектирования

tracker = DeepOCSORT(
    model_weights=Path('osnet_x0_25_msmt17.pt'),
    device='cpu',
    fp16=False,
)

detection_model = AutoDetectionModel.from_pretrained(
    model_type='yolov8',
    model_path='/Users/stepan/Desktop/Dataset_do/best (9).pt',
    confidence_threshold=0.5,
    device="cpu",
)
```

```
2024-03-30 22:20:12.736 | SUCCESS  | boxmot.appearance.reid_model_factory:load_pretrained_weights:207 - Successf
ully loaded pretrained weights from "osnet_x0_25_msmt17.pt"
2024-03-30 22:20:12.736 | WARNING  | boxmot.appearance.reid_model_factory:load_pretrained_weights:211 - The foll
owing layers are discarded due to unmatched keys or layer size: ('classifier.weight', 'classifier.bias')
2024-03-30 22:20:12.736 | WARNING  | boxmot.appearance.reid_model_factory:load_pretrained_weights:211 - The foll
owing layers are discarded due to unmatched keys or layer size: ('classifier.weight', 'classifier.bias')
```

In [ ]:
```python
# Функция для создания многоугольной маски

def create_polygon_mask(image_shape, points):
    mask = np.zeros(image_shape[:2], dtype=np.uint8)
    points = np.array(points, dtype=np.int32)
    cv2.fillPoly(mask, [points], 255)
    return mask
```

In [ ]:
```python
# Задание параметров камеры и видеофайла для сохранения вывода

vid = cv2.VideoCapture("/Users/stepan/Desktop/Dataset_do/video/file10.mp4")
vid.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
vid.set(cv2.CAP_PROP_FRAME_HEIGHT, 360)

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output4.mp4', fourcc, 30, (640, 360))

# Определение точек области
roi_points = [(10, 50), (200, 50), (1048, 100), (1048, 580), (350, 300), (300, 250), (10, 250)]
```

In [ ]:
```python
# Основной цикл обработки кадров видео

people_in_roi = 0
people_outside_roi = 0

for i in range(1):
    ret, im = vid.read()
    if not ret:
        break

    # Создание многоугольной маски
    mask = create_polygon_mask(im.shape, roi_points)
    outside_roi_mask = cv2.bitwise_not(mask)
    outside_roi_area = cv2.bitwise_xor(im, im, mask=outside_roi_mask)

    # Получение результатов детекции
    result = get_sliced_prediction(
        im,
        detection_model,
        slice_height=256,
```

```
        slice_width=256,
        verbose=1,
        overlap_height_ratio=0.3,
        overlap_width_ratio=0.3
    )

    # Преобразование результатов детекции в треки
    dets = np.zeros([len(result.object_prediction_list), 6], dtype=np.float32)
    for ind, object_prediction in enumerate(result.object_prediction_list):
        bbox = object_prediction.bbox.to_voc_bbox()
        dets[ind, :4] = bbox
        dets[ind, 4] = object_prediction.score.value
        dets[ind, 5] = object_prediction.category.id

        cv2.rectangle(im, (int(dets[ind, 0]), int(dets[ind, 1])), (int(dets[ind, 2]), int(dets[ind, 3])), (0, 25

    people_detected_in_roi = 0
    people_detected_outside_roi = 0


    tracks = tracker.update(dets, im)

    # Обработка каждого трека
    for track in tracks:
        xyxys = tracks[:, 0:4].astype('int')
        ids = tracks[:, 4].astype('int')
        confs = tracks[:, 5].round(decimals=2)
        clss = tracks[:, 6].astype('int')

        for xyxy, id, conf, cls in zip(xyxys, ids, confs, clss):

            r = []
            x1, y1, x2, y2 = xyxy

            center_x, center_y = int((x1 + x2) / 2), int((y1 + y2) / 2)

            # Проверяем принадлежность точки маске ROI

            is_inside_roi = mask[center_y, center_x] > 0

            if is_inside_roi:
                people_detected_in_roi += 1


            else:
                people_detected_outside_roi += 1

    # Рисуем контур области интереса
    cv2.polylines(im, [np.array(roi_points)], True, (255, 0, 0), 2)

    # Отображение количества людей
    cv2.putText(im, f'People in ROI: {people_in_roi}', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
    cv2.putText(im, f'People outside ROI: {people_outside_roi}', (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,

    # Объединяем область вне интереса с изображением для отображения
    im_with_roi = cv2.addWeighted(im, 1, outside_roi_area, 1, 0)
    out.write(im_with_roi)

    # Отображаем результат в Jupyter Notebook

    _, jpg_image = cv2.imencode('.jpg', im_with_roi)
    display(Image(data=jpg_image))

    # Проверяем на нажатие клавиши Q для выхода
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Освобождаем камеру и закрываем все окна
vid.release()
out.release()
cv2.destroyAllWindows()
```

```
Performing prediction on 18 number of slices.
```