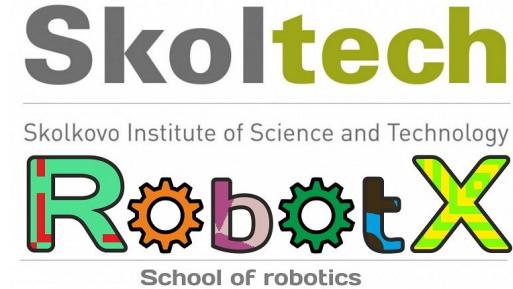


# Летний интенсив 2024

- Yolo V8.
- Трекинг объектов и предсказание движения.
- Расчет кинематики манипулятора.



Преподаватели курса:

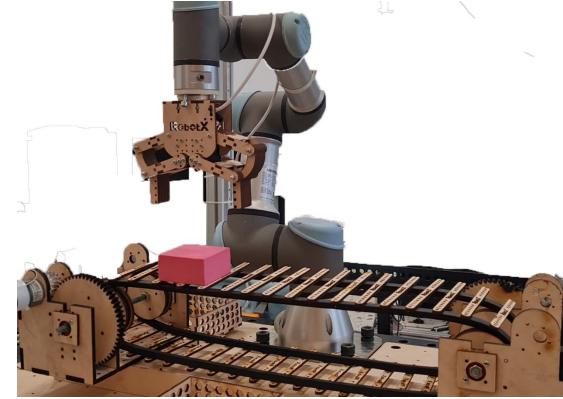
- Бурмистров Степан
- Федосеев Алексей

Руководитель лаборатории:

- Профессор Дмитрий Тетерюков

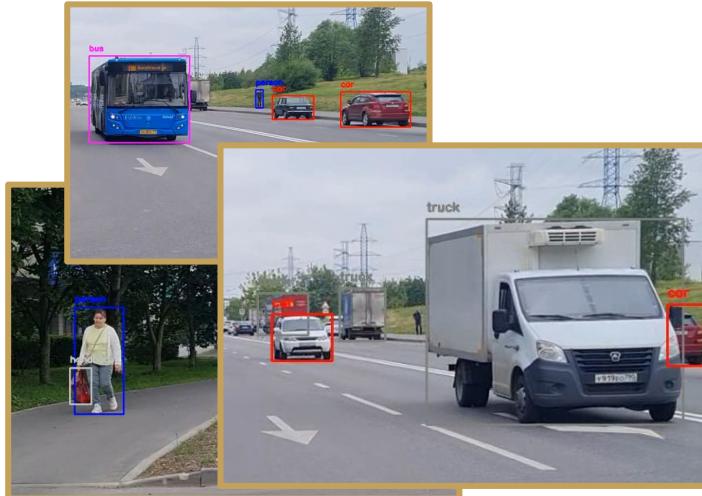
# Выполненные задачи:

- Исследование нейросети YoloV8
- Использование готовых моделей для детекции, сегментации и трекинга объектов
- Подготовка собственного датасета
- Дообучение нейронной сети
- Предсказание траектории движения на основе трекинга



- Разработка конвейера для перемещения объектов в зоне манипулятора
- Тестирование моделей YoloV8 на различных устройствах
- Настройка Jetson Nano для использования YoloV8
- Разработка системы управления манипуляторов с помощью анализа движений и жестов руки с использованием MediaPipe
- Расчет кинематики манипулятора
- Публикация статей по проделанной работе

# Запуск YoloV8 - базовая модель для детекции объектов



Распознавание объектов окружающего мира на основе предобученной модели

```
from ultralytics import YOLO
import cv2

model = YOLO('yolov8n.pt')

def process_image(image_path):
    image = cv2.imread(image_path)
    results = model(image)[0]

    image = results.orig_img
    classes_names = results.names
    classes = results.boxes.cls.cpu().numpy()
    boxes = results.boxes.xyxy.cpu().numpy().astype(np.int32)

    for class_id, box in zip(classes, boxes):
        class_name = classes_names[int(class_id)]
        color = (255,0,255)

        x1, y1, x2, y2 = box
        cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)
        cv2.putText(image, class_name, (x1, y1 - 10),
                   cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

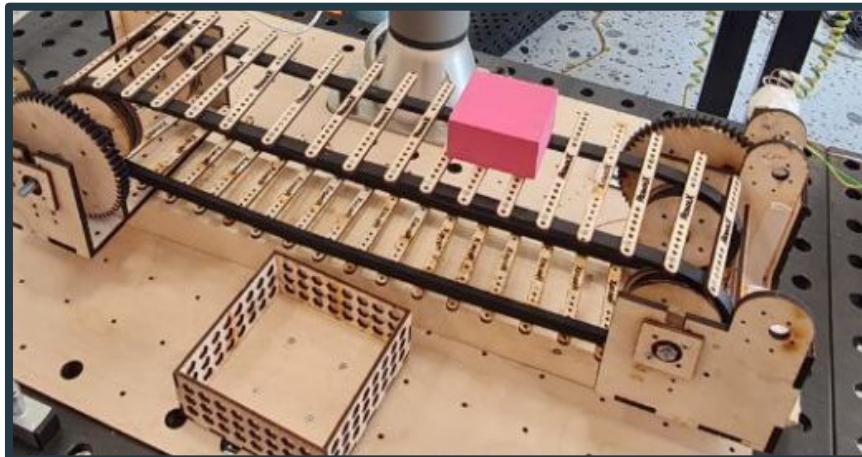
    cv2.imwrite("out.png", image)

process_image('t.png')
```

## Запуск YoloV8 - демонстрация работы



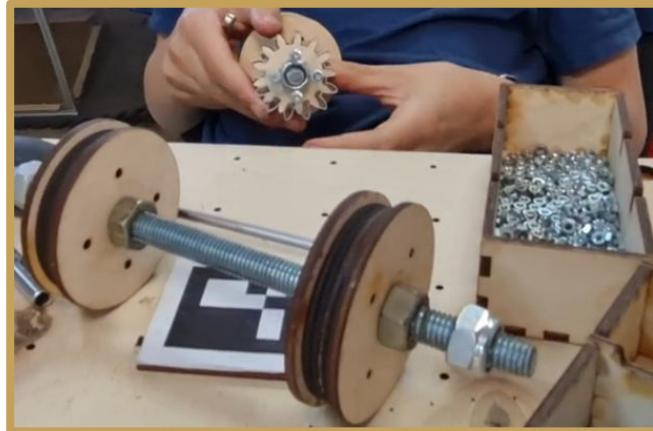
# Разработка конвейера



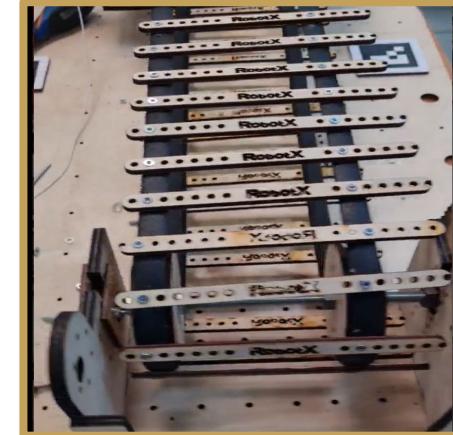
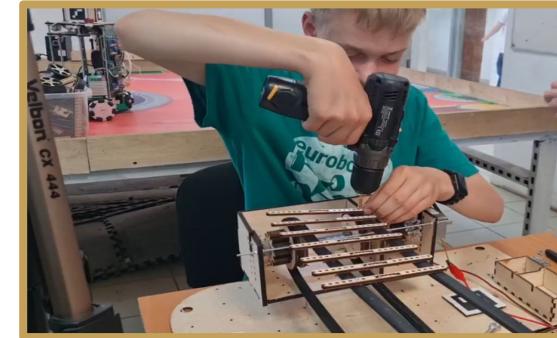
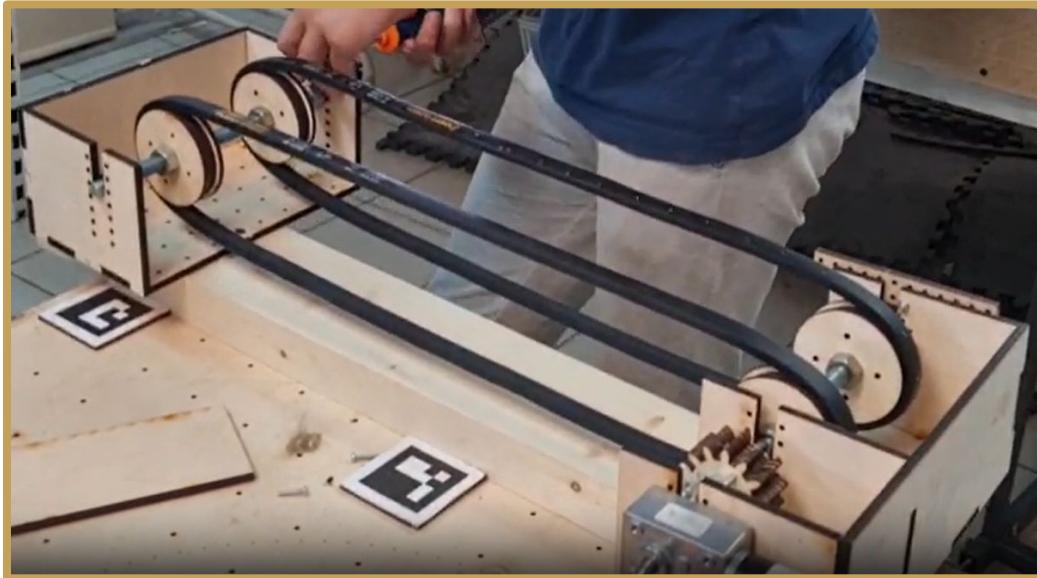
Материал	Фанера
Двигатель	jgb37 (2 шт)
Привод	ремень клиновой 1250мм

## Версия 1.0 - конвейерная лента

- Слишком жесткая лента
- Большая нагрузка на двигатель
- Шпильки без подшипников



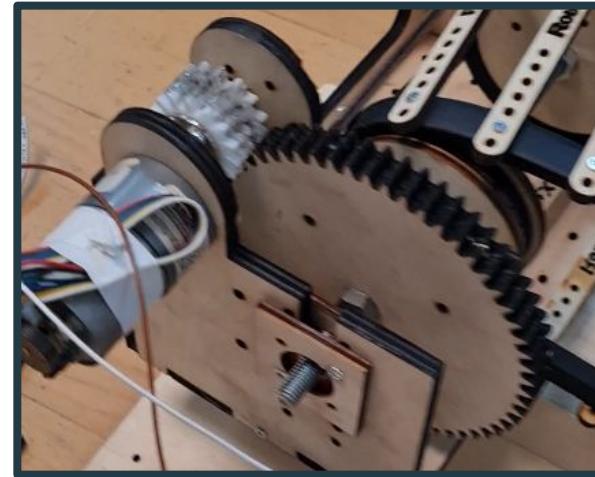
## Версия 2.0 - замена ленты на фанерные элементы



- Снижение нагрузки на двигатели
- Возможность создать ленту нужной длины

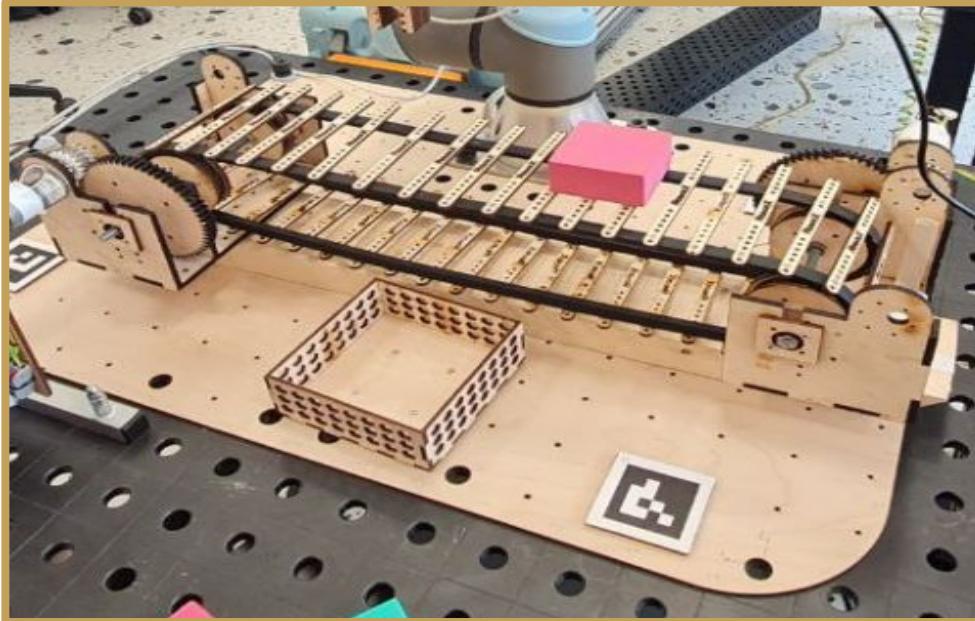
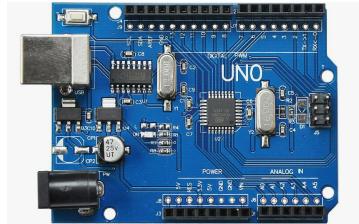
## Версия 3.0 - замена двигателей

- Скорость 27 об/мин
- Червячный редуктор
- Нет возможности провернуть руками
- Отсутствие подшипников



- Скорость двигателя 166 об/мин
- Дополнительный самодельный редуктор 1:4 (15 зуб : 60 зуб)
- Все оси конвейера установлены в подшипники

## Готовое решение и перспективы развития



- Управление через Arduino с контролем скорости и остановки
- Установка красной кнопки и элементов управления
- Автоматический возврат объектов в начало ленты



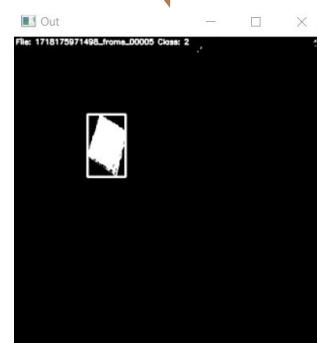
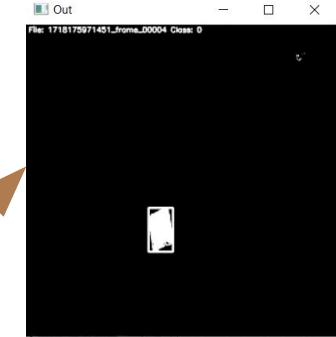
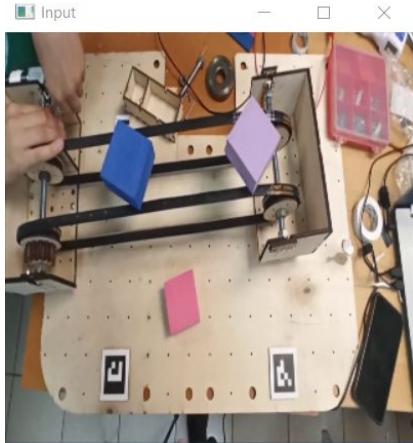
# Подготовка собственного датасета - основные правила

Для датасета важно:

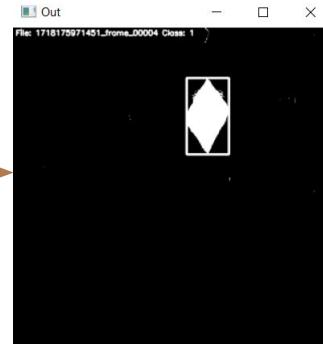
- Большое кол-во фотографий
- Наличие фотографий без объектов
- Точная разметка
- Максимально разные фотографии(на разном фоне, при разном освещении)



## Подготовка собственного датасета: разметка



После подготовки большого кол-ва фотографий необходимо их разметить. Для этого был разработан автоматизированный скрипт.



Программа сохраняет исходные фото, и текстовые файлы с координатами

# Дообучение нейронной сети

После этого размеченные фотографии проверяются, и разделяются на 3 части:

- **train** 70% всего датасета; используется при обучении
- **test** 20% всего сета; используется после каждой эпохи для определения ошибки
- **valid** 10% датасета; используется в конце обучения для определения результата всего обучения

Также для обучения нейросети нужно сделать файл конфигурации - `data.yaml`

Пример данного файла:

```
1 train: dataset/train/images
2 val: dataset/valid/images
3 test: dataset/test/images
4
5 nc: 4
6 names: ['blueCube', 'purpleCube', 'redCube', 'errorCube']
```

# Дообучение нейронной сети

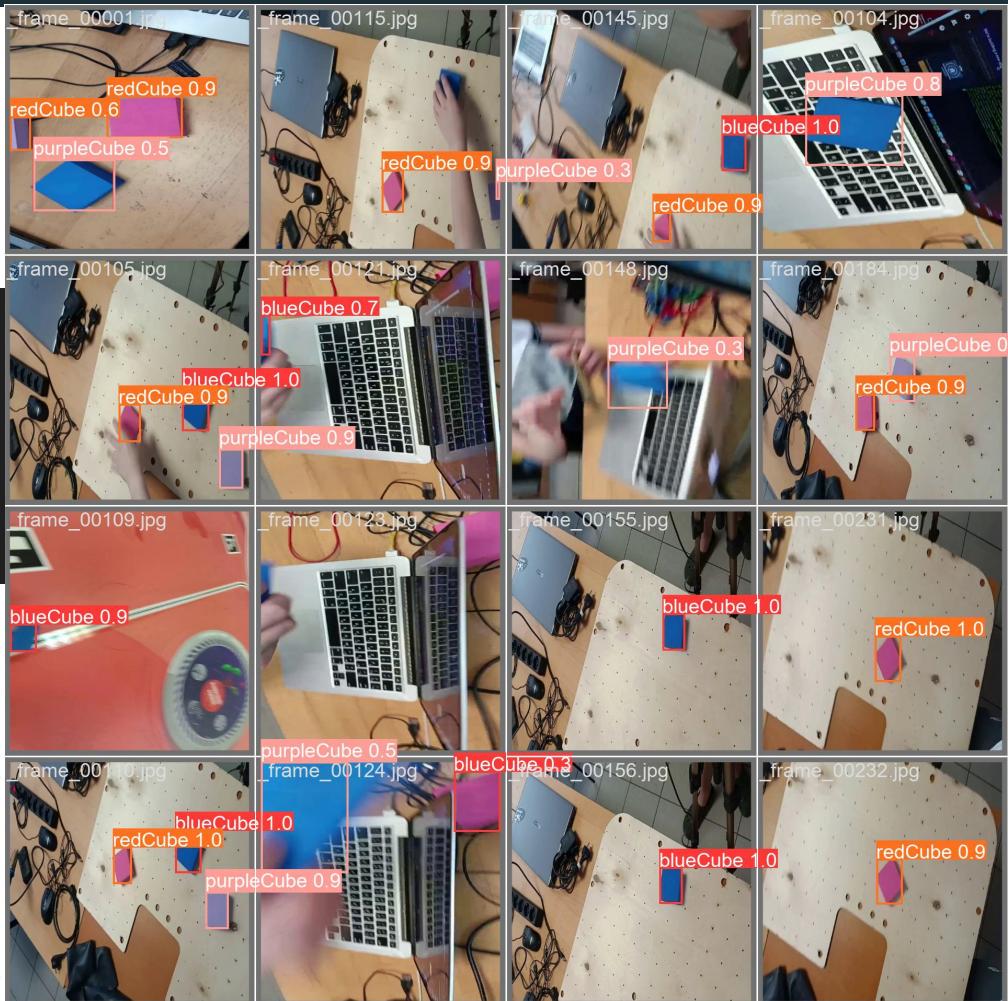
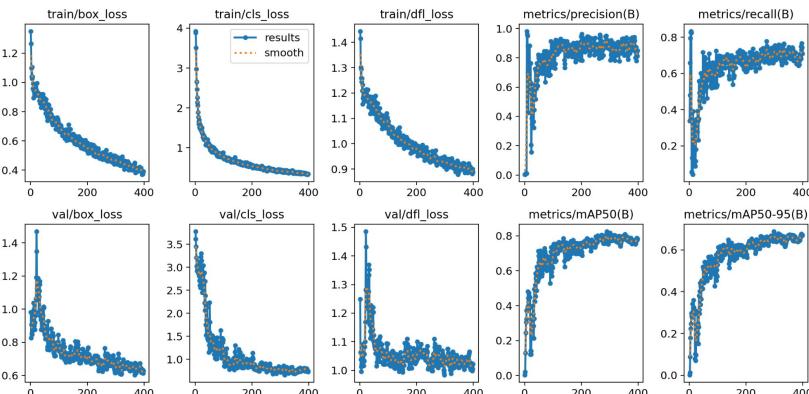
Вот пример обучения  
нейросети на 500 эпох

```
import os
from ultralytics import YOLO

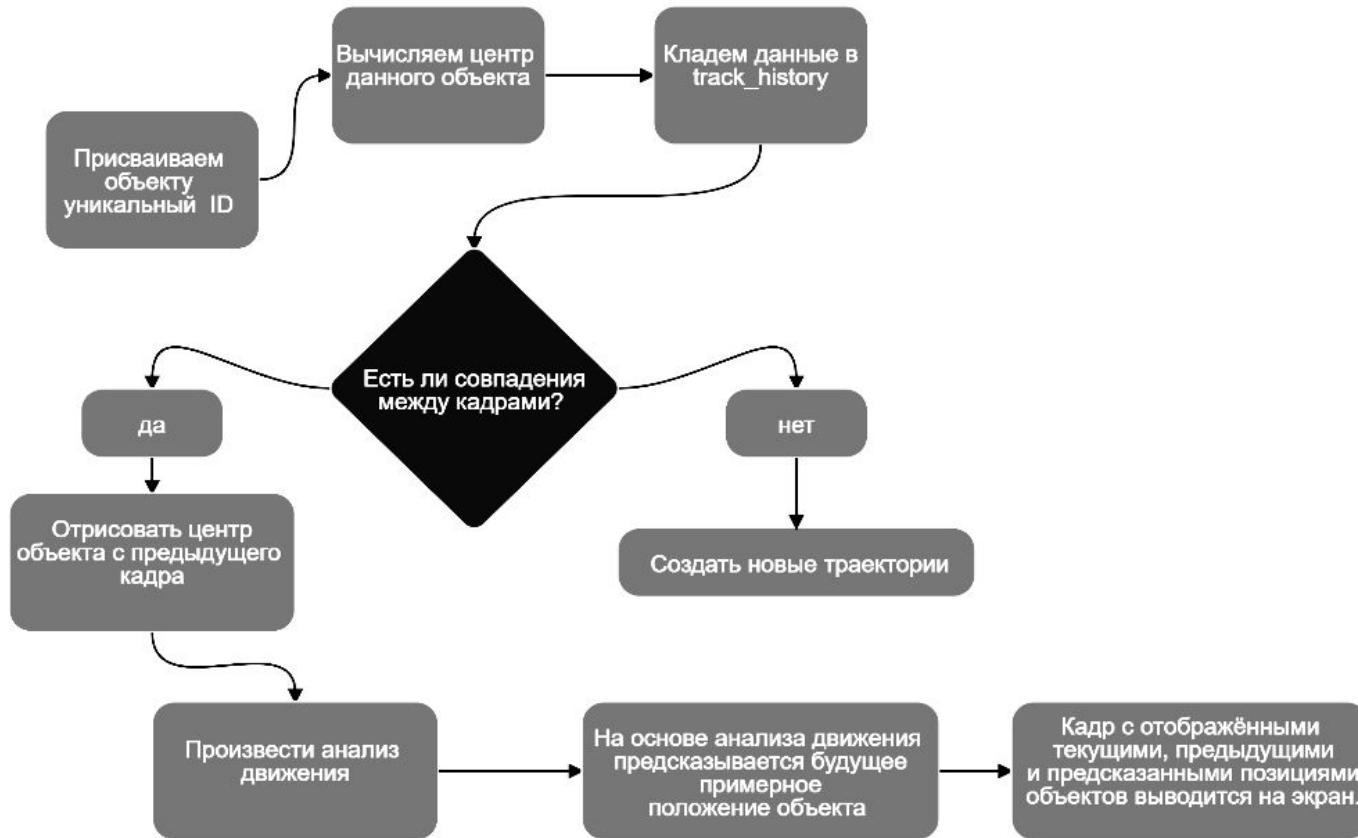
current_dir = os.path.dirname(os.path.abspath(__file__))

data_path = os.path.join(current_dir, 'data.yaml')
model = YOLO(os.path.join(current_dir, 'yolov8n.pt'))

epochs = 500
batch = 64
imgsz = 640
if __name__ == '__main__':
    model.train(data=data_path, epochs=epochs, batch=batch, imgsz=imgsz, name='red', device='cuda')
```



# Трекинг и предсказания



```
# Словарь для хранения истории треков объектов (используем deque для эффективного ограничения размера)
track_history = defaultdict(
    lambda: deque(maxlen=30)) # Максимальная длина истории - 30 кадров (примерно 1 секунда при 30 FPS)
# Обнаружение и трекинг объектов
results = model.track(frame, persist=True)

# Получение информации о частоте кадров
fps = cap.get(cv2.CAP_PROP_FPS) or 30 # По умолчанию 30 FPS

# Отрисовка результатов
if results[0].boxes is not None and results[0].boxes.id is not None:
    boxes = results[0].boxes.xywh.cpu()
    track_ids = results[0].boxes.id.int().cpu().tolist()
    classes = results[0].boxes.cls.cpu().numpy()
    class_names = results[0].names

    for box, track_id, class_id in zip(boxes, track_ids, classes):
        x, y, w, h = box
        class_name = class_names[int(class_id)]

        # Добавление текущих координат в историю трека
        track_history[track_id].append((int(x), int(y)))

        # Отрисовка линии трекинга
        for i in range(1, len(track_history[track_id])):
            cv2.line(frame, track_history[track_id][i - 1], track_history[track_id][i], (0, 255, 0), 2)
            predicted_x, predicted_y = predict_position(track_history[track_id], 0.3, fps)
```

```
def predict_position(track, time_ahead, fps, smoothing_factor=0.75):
    """
        Функция линейной экстраполяции с усреднением для более плавного предсказания позиции.
    """

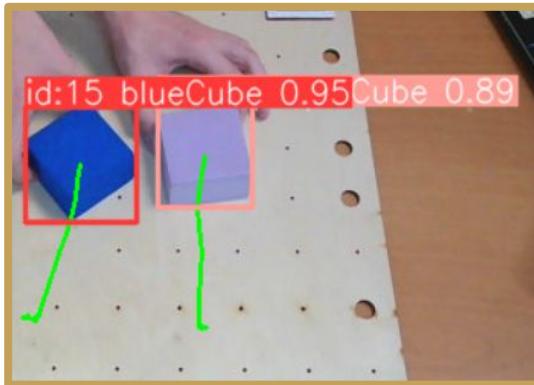
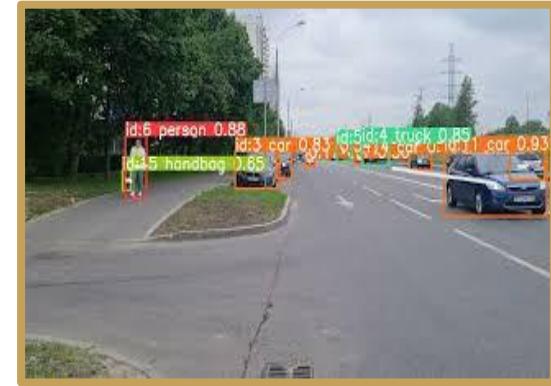
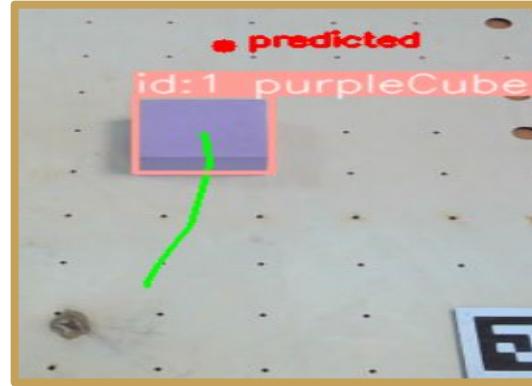
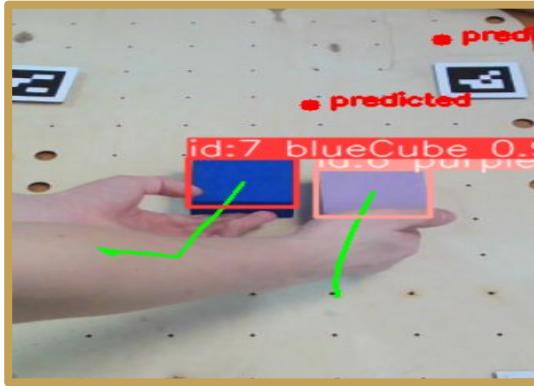
    if len(track) < 2:
        return None, None

    # Вычисляем скорость с использованием последних N кадров (сглаживание)
    N = min(len(track), 10) # Берем не более 10 последних кадров
    x_speed = (track[-1][0] - track[-N][0]) * fps / N
    y_speed = (track[-1][1] - track[-N][1]) * fps / N

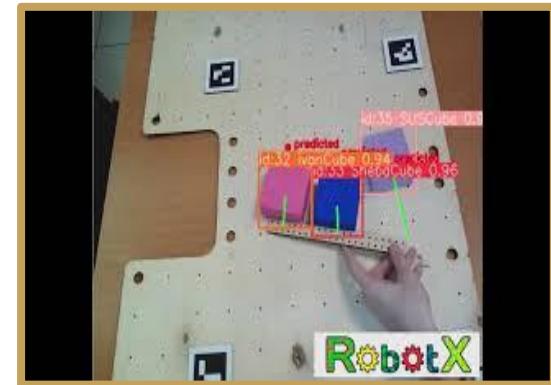
    # Экспоненциальное сглаживание скорости
    if len(track) > 2:
        x_speed = smoothing_factor * x_speed + (1 - smoothing_factor) * (track[-1][0] - track[-2][0]) * fps
        y_speed = smoothing_factor * y_speed + (1 - smoothing_factor) * (track[-1][1] - track[-2][1]) * fps

    # Предсказываем позицию через time_ahead секунд
    x_pred = track[-1][0] + x_speed * time_ahead
    y_pred = track[-1][1] + y_speed * time_ahead

    return x_pred, y_pred
```



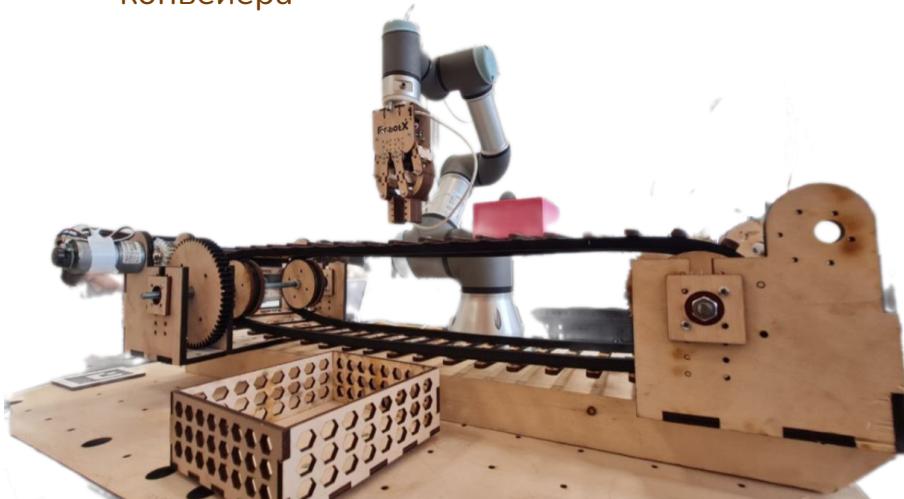
<https://habr.com/ru/articles/821971>





<https://github.com/robotx-school/June-Yolo-2024>

Код для подбора с  
конвейера



# Работа с манипулятором UR3

```
key_thread = threading.Thread(target=check_for_stop_key)
cam_thread = threading.Thread(target=control_camera)

cam_thread.start()
key_thread.start()

cam_thread.join()
key_thread.join()
```

Разделяем код на 2 потока: 1 из них постоянно ожидает нажатие кнопки shift для экстренной остановки. Второй выполняет основную работу - ожидает кубик, и отправляет манипулятор.

# Работа с манипулятором UR3

```
global cap
OK, frame = cap.read()
print("FPS:",Fps())
if not OK:
    print("Error image hasn't been read")
    os._exit(0)
    time.sleep(0.01)
print("image read OK:")
cv2.imshow("source image", frame)
frame = cv2.resize(frame, (640,480))
frame = warp_field(frame,dictionary)
predict_list, track, coords_list, counter_moving,vis = check_cube(frame,fps)
#out_list = check_cube(frame,fps)

print(predict_list["redCube"])
if predict_list["redCube"] != [] and coords_list["redCube"] != [] and predict_list["redCube"] is not None:
    if(170 <= predict_list["redCube"][1] <= 270 and 180 <= predict_list["redCube"][0] <= 230 and 170 <= coords_list["redCube"][1] <= 270):
        print(f"MOVE MANIPULATOR TO X: {predict_list['redCube'][0]}; Y: {predict_list['redCube'][1]} in 1 second")
        move_coord = (predict_list['redCube'][0],200)
        bot_thread = threading.Thread(target=robot_task)
        bot_thread.start()
        bot_thread.join()
        os._exit(0)
cv2.imshow("Visualisation", vis)

fps, fps_start_time, fps_counter = Fps(fps, fps_start_time, fps_counter)
```



# Arm2Arm - система управления манипулятором с помощью жестов рук.

## ЖЕСТЫ

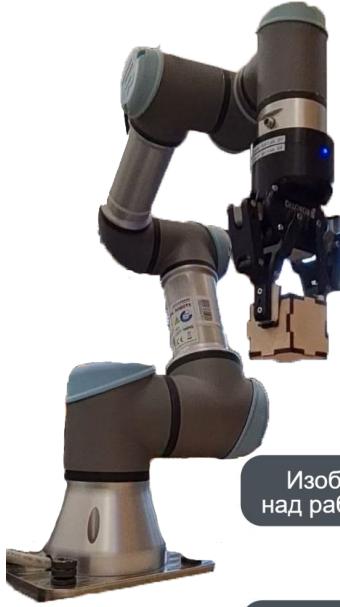
СЖАТЬ/РАЗЖАТЬ  
ЗАХВАТ

ВКЛ/ВЫКЛ  
УПРАВЛЕНИЕ

ПОВОРОТ +

ПОВОРОТ -





Выполнение команд манипулятором UR-3  
(перемещение, вращение, захват)

Изображение с камеры  
над рабочим пространством

Развертка изображения по  
АгУсо маркерам

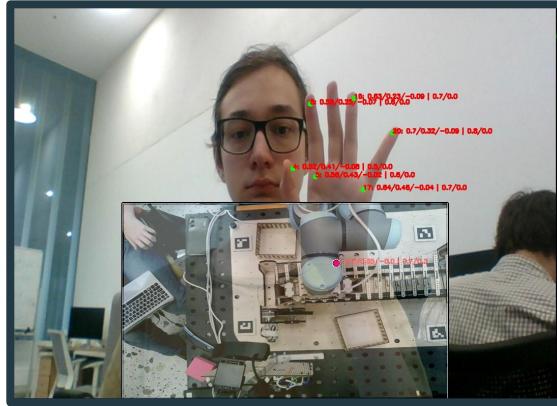
Рабочее место  
оператора

Визуализация

Изображение с камеры рабочего места

MediaPipe  
(Получение и обработка координат  
фаланг пальцев рук)

Распознавание жестов,  
расчёт координат искомой точки  
в координатах манипулятора  
по положению рук



In use: True  
Grip/Finger | Angle no  
**ARM2ARM**  
Hand gesture  
capture system for  
manipulator control

**GESTURES:**

- GRAB/RELEASE
- ON/OFF CONTROL
- ROTATE +
- ROTATE -

**RobotX**  
Skoltech  
Moscow Institute of Science  
and Technology June 2024





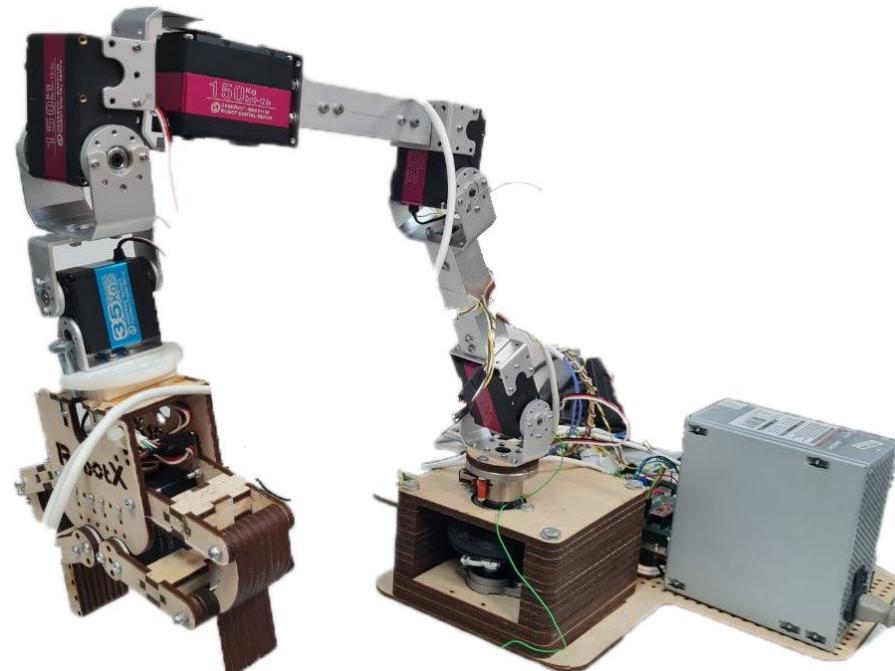
<https://github.com/robotx-school/arm2arm>



# DIY-манипулятор (Кинематика)

Разработка библиотеки Python для управления четырехзвенного манипулятора собственного производства.

Программе на вход подаются координаты точки захвата манипулятора в миллиметрах. Для расчета искомой точки используется простая модель обратной кинематики, рассчитывающая координаты в каждой плоскости, с использованием обратных тригонометрических функций и теоремы косинусов.



# DIY-манипулятор (Кинематика)

## Алгоритм расчета обратной кинематики манипулятора

1. Расчет угла поворота манипулятора и положения всех звеньев

```
def calc4pos(self, x, y, z, sizes = [180, 190, 240, 130]):  
    dist = self.calcdist(x, y)  
    rot = self.calcrot(x, dist)  
    h = sizes[0]  
    l1, l2, l3 = sizes[1:]  
    p0 = (dist, z+h)  
    p1 = (dist, p0[1]+l3)  
    p3 = (0, h)  
    p4 = (0, 0)  
    c = self.calcdist(p1[0]-p3[0], p1[1]-p3[1])  
    cosa = p1[0]/c  
    sina = p1[1]/c  
    cx, cy = self.calcPPosByCos(l3, l2, c) ##  
    p2x = cx * cosa - cy * sina  
    p2y = cx * sina + cy * cosa  
    p2 = (p2x, p2y)  
    return (rot, p0, p1, p2, p3, p4)
```

2. Расчет углов каждого звена

```
def calc3angle(self, A, B, C):  
    BA = (A[0] - B[0], A[1] - B[1])  
    BC = (C[0] - B[0], C[1] - B[1])  
    BABC = BA[0]*BC[0] + BA[1]*BC[1]  
    lBA = math.sqrt(BA[0]**2 + BA[1]**2)  
    lBC = math.sqrt(BC[0]**2 + BC[1]**2)  
    cosa = BA[0]*BC[0]/(lBA*lBC)
```

return

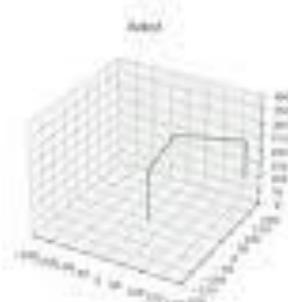
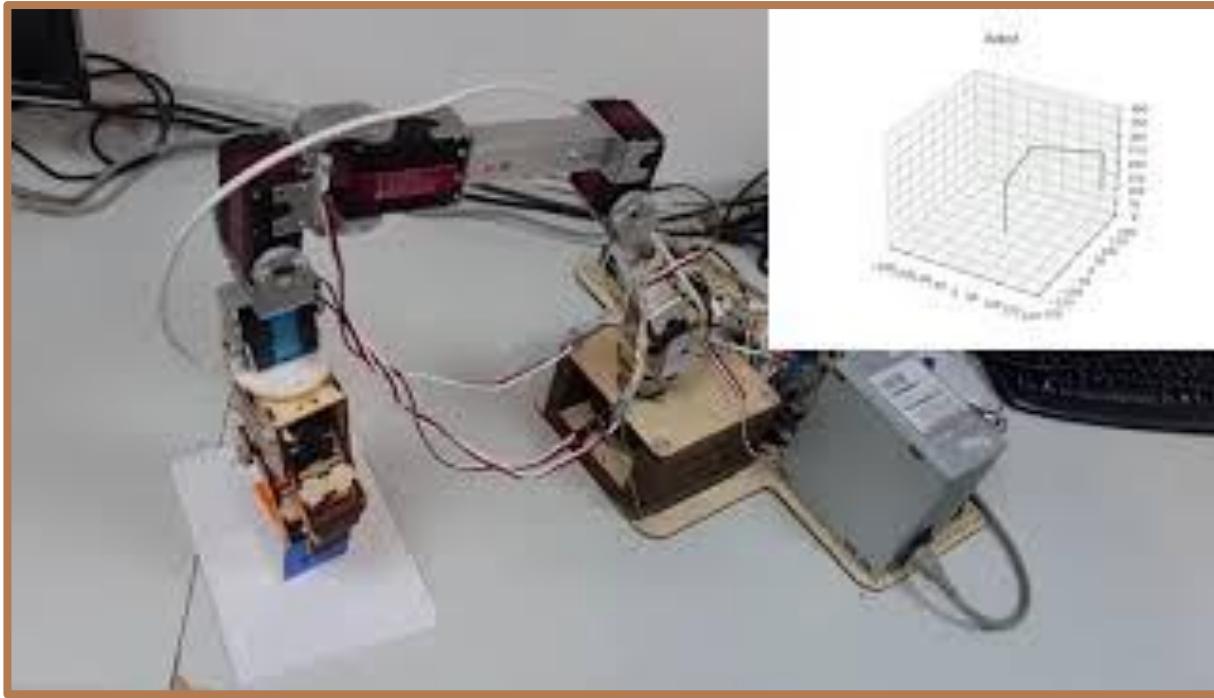


3. Перемещение манипулятора

```
def spi_send(self, txData):  
    N = 40  
    spi = spidev.SpiDev()  
    spi.open(0, 0)  
    spi.max_speed_hz = 500000  
    txData = self.list_int_to_bytes(txData)  
    txData = txData+[0]*(N-len(txData))  
    rxData = []  
    _ = spi.xfer2([240])  
    for i in range(40):  
        rxData.append(spi.xfer2([txData[i]])[0])  
    spi.close()  
    return rxData
```

\* В txData передаются инструкции, описанные в документации манипулятора

# DIY-манипулятор (Демонстрация)



# Jetson Nano и путь к запуску YOLOV8

- 128 CUDA ядер
- Линукс 
- Шина GPIO
- Частота CPU 1.47 GHz
- Частота GPU 0.74 GHz



## JetPack от производителя

1. Нет настроенной CUDA
2. Невозможно установить ultralytics
3. Устаревшая версия питона

## Неофициальная сборка

1. Есть настроенная CUDA
2. Установленный Python 3.8



<https://habr.com/ru/articles/823043>  
Статья про jetson

## Физические решения



Корпус



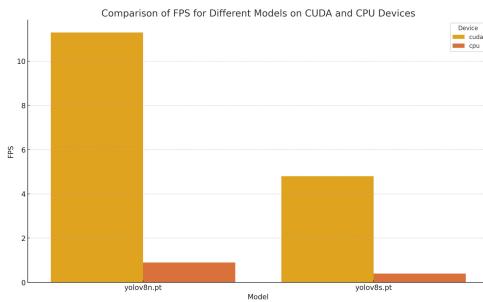
Активная работа

Режим простоя



# Программные решения

1. Был установлен сторонний образ, в котором:
  - a. Настроенная CUDA v10
  - b. Установленный python 3.8
  - c. OpenCV скомпилированная под cuda
2. Установлен ultralytics
3. Отключен графический интерфейс



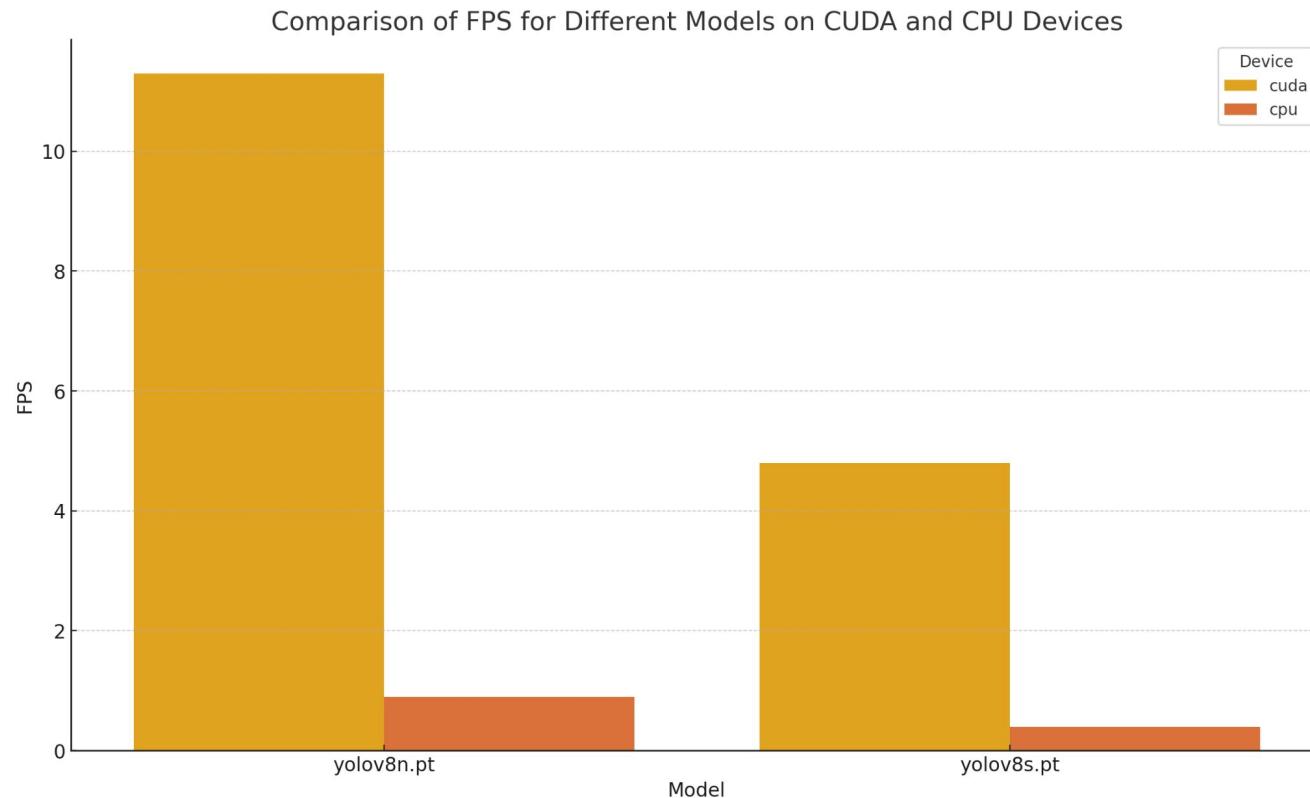
yolov8n-модель работает на CUDA со скоростью 11 FPS

```
Libraries
CUDA: 10.2.300
cuDNN: 8.2.1.32
TensorRT: 8.0.1.6
VPI: 1.1.15
Vulkan: 1.2.141
OpenCV: 4.8.0 with CUDA: YES
```

Сторонний образ



# Результат



# Анализ производительности моделей

```
Server ip>localhost
Server port>8001
System Name>i7_12700H_FULL
Try to find test.py and connect to data-server
File loaded
Current machine data:
platform      : Linux
release       : 6.7.1-arch1-1
version        : #1 SMP PREEMPT_DYNAMIC Sun, 21 Jan 2024 22:14:10 +0000
architecture   : x86_64
cpu_cores     : 14
cpu_all       : 28
ram           : 15.317218780517578 GB
cpu_name      : 12th Gen Intel(R) Core(TM) i7-12700H
Got models list
Do you want to test base models? (y - yes, other - not): y
Do you want to test openvino intel models? (y - yes, other - not): y
Do you want to test ncnn models? (y - yes, other - not): y
Do you want to test ncnn_lite models? (y - yes, other - not): y
Do you want to test tflite models? (y - yes, other - not): y
Do you want to test tflite_lite models? (y - yes, other - not): y
Do you want to test onnx_lite models? (y - yes, other - not): y
Do you want to test onnx models? (y - yes, other - not): y
Do you want to test int8 models? (y - yes, other - not): y
We need to download videos: adekvat.mp4 (1)
All models and videos downloaded.
Going to test: 79 models
yolov8n.pt
Testing model: data/models/yolov8n.pt with video: data/videos/adekvat.mp4
Warmup finished
100%|██████████| 199/200 [00:38<00:00,  5.13it/s]
Benchmark finished
Model validated on coco8.yaml
100%|██████████| 199/200 [00:39<00:00,  5.10it/s]
Model test results:
{"inference_time": 187.77700901031494, "inference_time_1": 187.8, "inference_time_min": 176.90491676330566, "inference_time_max": 225.9984016418457, "fps": 5.3, "half": 0, "int8": 0, "runtime": "BASE", "map50": None, "map75": None, "device": "cpu", "warmup_min_inf_time": 122.60580062866211, "warmup_max_inf_time": 204.34117317199707}
Tested: 1/79
yolov8s.pt
Testing model: data/models/yolov8s.pt with video: data/videos/adekvat.mp4
Warmup finished
52%|██████████| 105/200 [00:56<00:50,  1.87it/s]
```

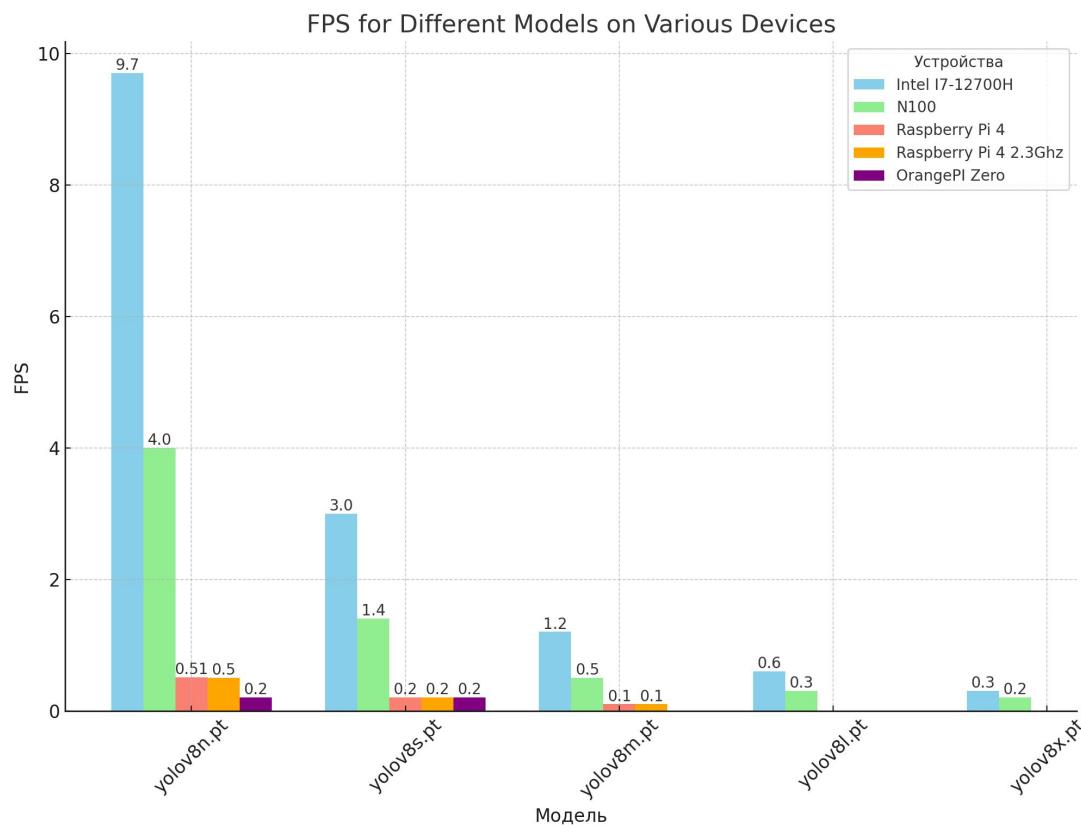
Полная аналитика



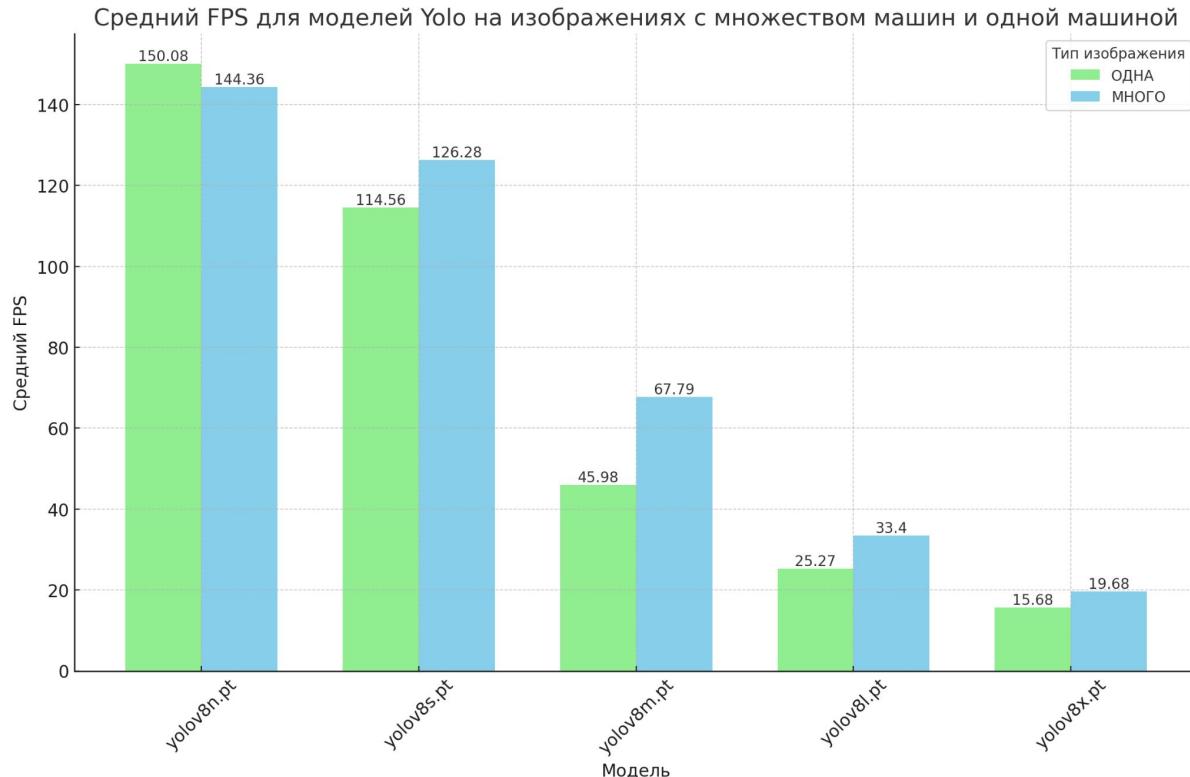
GitHub репозиторий



# Базовые COCO модели Yolov8



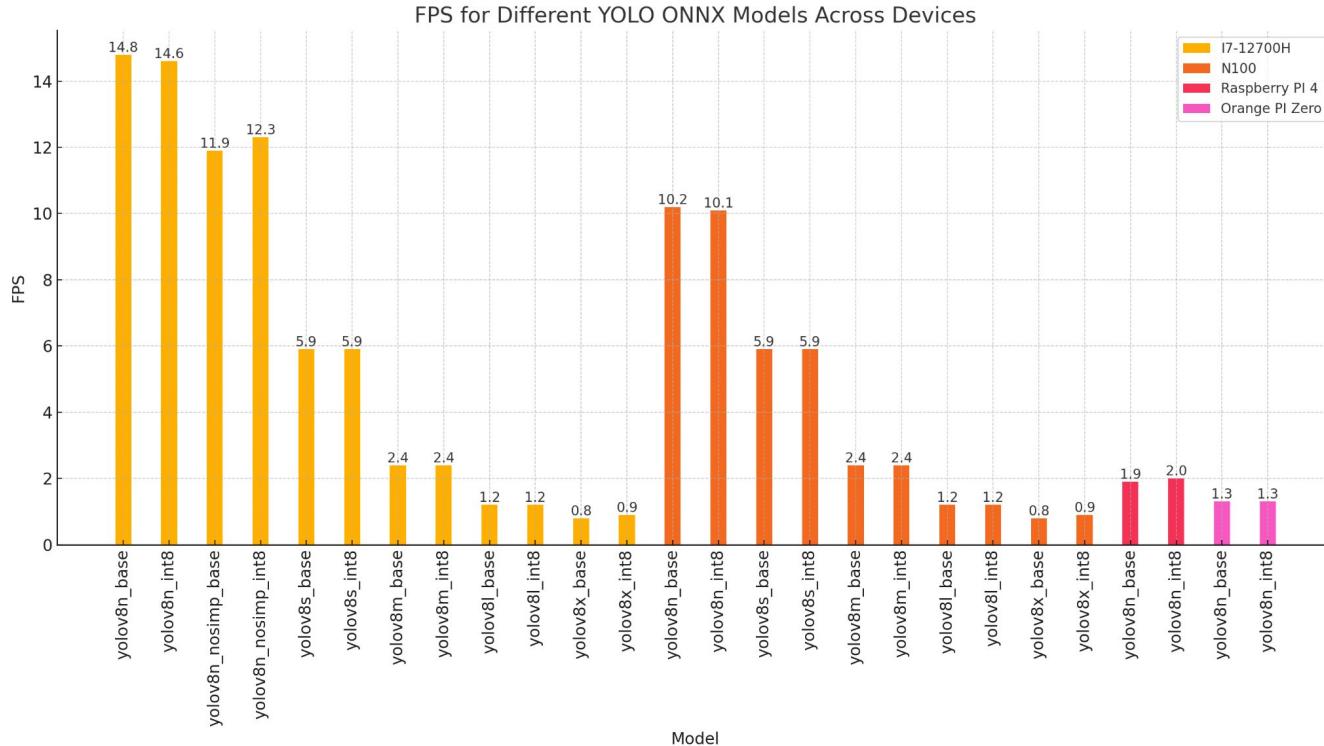
# Влияние количества объектов на скорость



Поведение N модели отличается от S, M, L, X

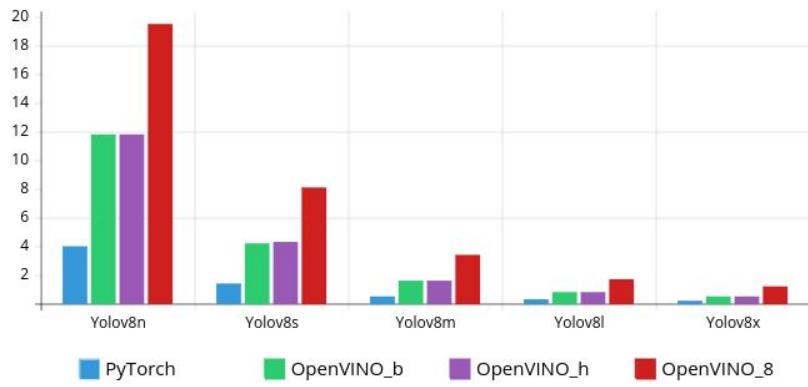


# ONNX на разных устройствах

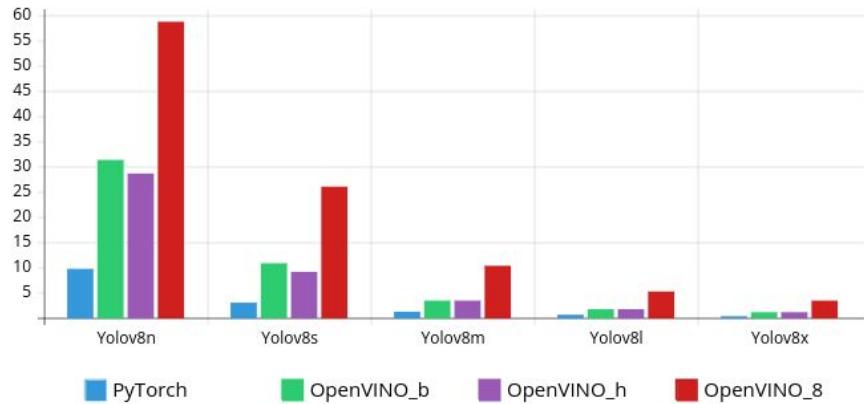


# Оптимизации под Intel

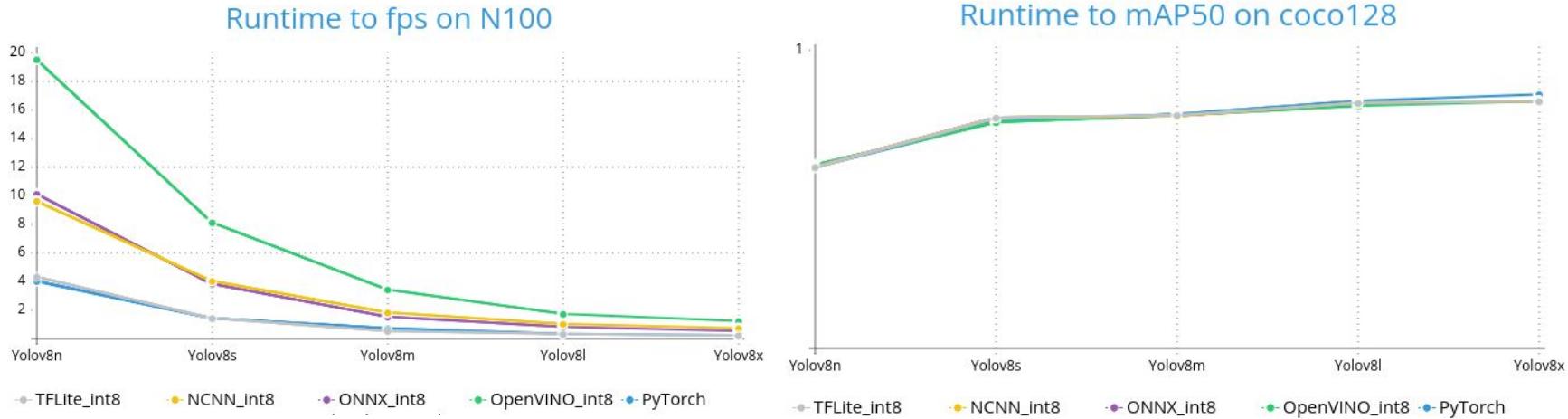
OpenVINO on N100



OpenVINO on i7-12700H

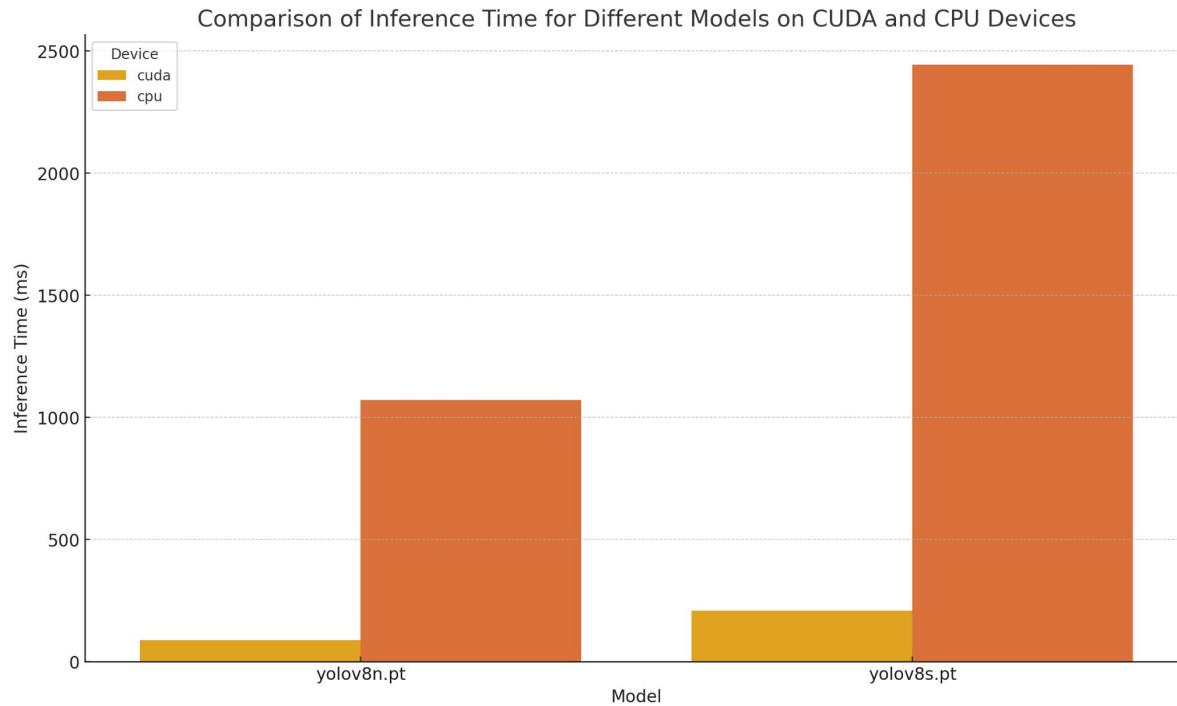


# Сравнение int8 рантаймов



# Jetson Nano - CUDA

Максимальный FPS: 14 на CUDA





[Статья про jetson](https://habr.com/ru/articles/823043)



Краткая выжимка  
информации по YOLO с  
курса



[Код для подбора с  
конвейера](https://github.com/robotx-school/June-Yolo-2024)

Короче  
говоря:



[Arm2arm](https://github.com/robotx-school/arm2arm)



[Полная аналитика  
моделей](https://habr.com/ru/articles/822917)



[GitHub репозиторий  
с кодом для тестов](https://github.com/ret7020/YoloBenchmarks)



alexeyfas



stepan\_burmistrov



ураaaaaaaaaaaaaaaaaaaaaaaa  
отдыхать  
+

ураaaaaaaaaaaaaaaaaaaaaaaa  
пицца



**Skoltech**  
Skolkovo Institute of Science and Technology  
**RobotX**  
School of robotics

