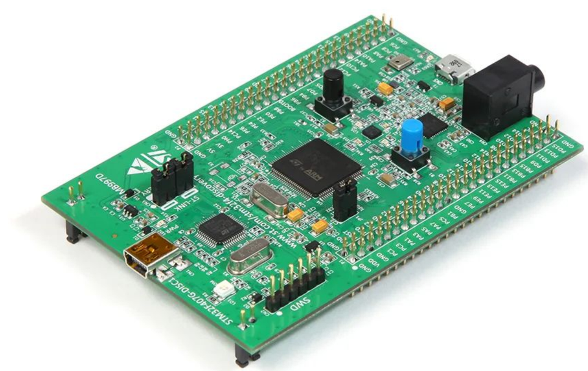
Интеграция фреймворка NCNN в RTOS Embox для задач детекции объектов

Кривенко Дмитрий, студент МФТИ

ФПМИ МФТИ

Мотивация

Нейросетевой inference прямо на устройстве: минимальный размер, минимум зависимостей и предсказуемые задержки — без тяжёлой ОС и облака.

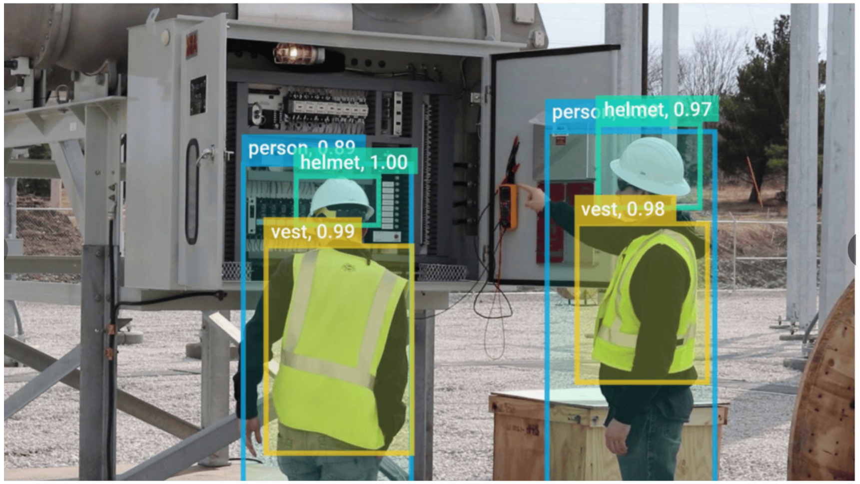


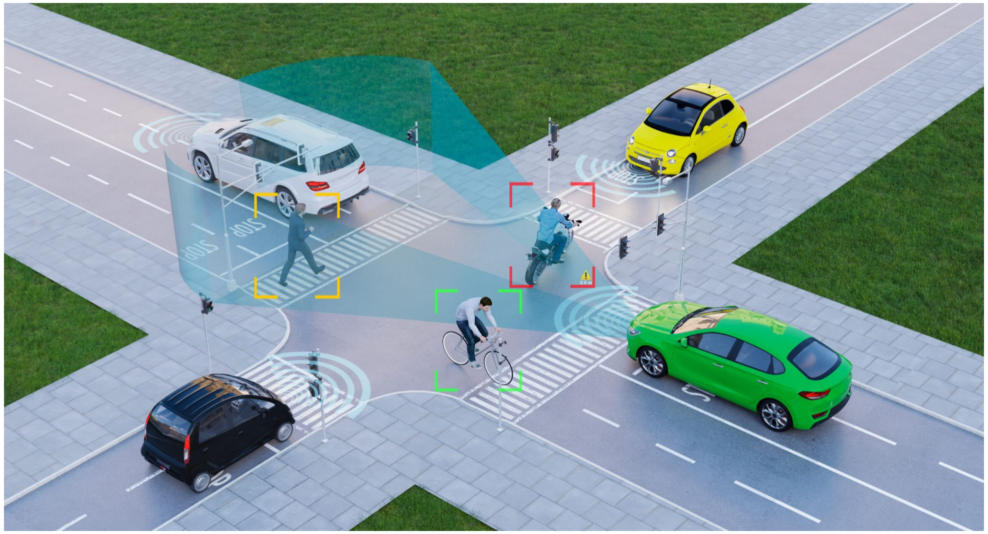
Преимущество перед облаком

- Предсказуемые задержки и контроль выполнения
- Данные остаются локально
- Работает при нестабильной или отсутствующей сети

Где нужен on-device?

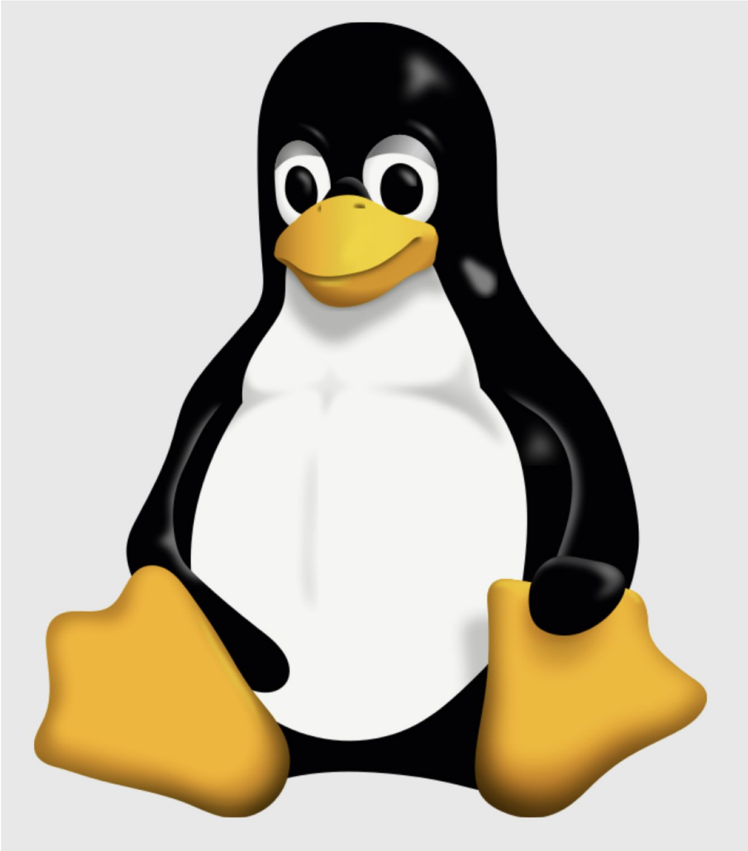
- ADAS (помощь водителю)
- Инфраструктура (обнаружение посторонних предметов)
- Безопасность (человек в опасной зоне/охрана труда)





Почему не Linux?

- Размер и сложность стека
- Менее предсказуемые задержки
- Контроль конфигурации



Embox RTOS

- Лёгкая модульная операционная система реального времени для встраиваемых и ресурсно-ограниченных платформ.
- Поддерживает POSIX-подобный интерфейс
- Конфигурируется «под задачу» - можно собрать образ только с нужными драйверами и библиотеками.
- В открытой литературе Embox почти не рассматривается как платформа для нейросетевых вычислений.

NCNN

- Открытый C++-фреймворк инференса нейросетей, ориентированный на мобильные и embedded-устройства.
- Минимальный рантайм: можно собрать статически и использовать без тяжёлой экосистемы.
- Поддерживает свёрточные сети и детекторы, включая модели семейства YOLO.
- В основном применяется на Android и десктопах; связка NCNN + RTOS (и особенно Embox) практически не описана

Другие решения

- TensorFlow Lite
- OpenCV DNN
- ONNX Runtime

YOLO

Детекция объектов в реальном времени

- One-stage detector: на одном проходе выдаёт bounding boxes + классы + confidence
- Хорошее соотношение точность/скорость
- Семейство моделей разных размеров (nano/small/...)
- 80 классов

Интеграция Embox + NCNN

- NCNN подключён как библиотека Embox
- Сборка внешнего проекта на этапе build
- Заголовки и toolchain-зависимости описаны явно
- Демо-приложение линкуется с libncnn статически

```
package third_party.lib.ncnn


@BuildArtifactPath(
  cppflags="-I$(EXTERNAL_BUILD_DIR)/third_party/lib/ncnn/libncnn/install/src -I$(EXTERNAL_BUIL
)
module ncnn_headers {}

@Build(stage=1, script="$(EXTERNAL_MAKE)")
@BuildDepends(third_party.lib.ncnn.ncnn_headers)
@BuildDepends(third_party.compiler_headers)
@BuildDepends(embox.cxx.LibCxx)
static module libncnn {

    source
        "^BUILD/extbld/third_party/lib/ncnn/libncnn/install/src/allocator.cpp",
        "^BUILD/extbld/third_party/lib/ncnn/libncnn/install/src/benchmark.cpp",
        "^BUILD/extbld/third_party/lib/ncnn/libncnn/install/src/blob.cpp",
        "^BUILD/extbld/third_party/lib/ncnn/libncnn/install/src/c_api.cpp",
        "^BUILD/extbld/third_party/lib/ncnn/libncnn/install/src/command.cpp",
        "^BUILD/extbld/third_party/lib/ncnn/libncnn/install/src/datareader.cpp",
        "^BUILD/extbld/third_party/lib/ncnn/libncnn/install/src/expression.cpp",

```

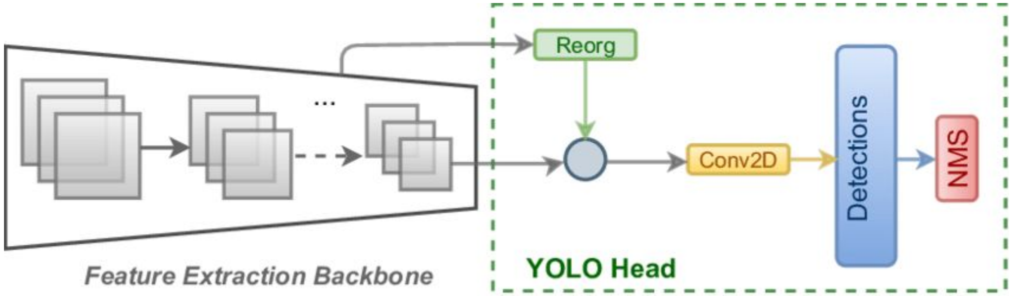
Интеграция Embox + NCNN

- Параметры/веса модели размещены в образе ФС
- Загрузка в рантайме через стандартный файловый интерфейс

```
package embox.cmd.ncnn


@LinkerSection(text="ncnn_yolo_demo_test_text",rodata="ncnn_yolo_de
@App
@AutoCmd
@CxxCmd(name="ncnn_yolo_demo", help="YOLOv8n NCNN demo on Embox")
@Build(stage=2)
@BuildDepends(third_party.lib.ncnn.libncnn)
module ncnn_yolo_demo {

    source
        "main.cpp",
        "dfl_decode.cpp",
        "image_io.cpp",
        "nms.cpp",
        "stb_image_impl.cpp",
        "yolov8n_ncnn_model/yolov8n.id.h",
        "yolov8n_ncnn_model/yolov8n.mem.h"

    depends third_party.lib.ncnn.libncnn
}
```

Пайплайн инференса

- Вход: изображение
- **Предобработка**: resize → преобразование формата → нормализация
- Упаковка в ncnn::Mat → inference (ncnn::Net)
- **Постобработка**: decode → фильтр по confidence → NMS → список детекций через порог уверенности



Демонстрация работы

```
root@embox:/#ncnn_yolo_demo ./data/photos/cat_and_dog.png
[1] img=./data/photos/cat_and_dog.png S=320
[1] ok: 998x665 bytes=1991010
[2] load model...
[2] ptrs: pparam=0x003afec8 pmodel=0x003b1fd0
[2] rc: param=0 model=0
[3] preprocess(letterbox+norm)...
[3] lb bytes=307200 nw=320 nh=213 pad=(0,53) scale=0.320641
[3] in mat: w=320 h=320 c=3 dims=3 data=0x035ed040
[4] infer(input+extract)...
[4] rc: input=0 extract=0 out(w=2100 h=84 c=1 dims=2 data=0x044d2040)
[5] decode+nms...
[5] dets=11 (C=84)
[5] nms keep=2
  dog cls=16 conf=0.746 x=2.4 y=53.6 w=316.6 h=184.7
  cat cls=15 conf=0.280 x=0.1 y=71.0 w=185.1 h=167.9
```

```
root@embox:/#ncnn_yolo_demo ./data/photos/cat.png
[1] img=./data/photos/cat.png S=320
[1] ok: 1439x977 bytes=4217709
[2] load model...
[2] ptrs: pparam=0x003afec8 pmodel=0x003b1fd0
[2] rc: param=0 model=0
[3] preprocess(letterbox+norm)...
[3] lb bytes=307200 nw=320 nh=217 pad=(0,51) scale=0.222377
[3] in mat: w=320 h=320 c=3 dims=3 data=0x035ed040
[4] infer(input+extract)...
[4] rc: input=0 extract=0 out(w=2100 h=84 c=1 dims=2 data=0x04933040)
[5] decode+nms...
[5] dets=10 (C=84)
[5] nms keep=1
  cat cls=15 conf=0.928 x=0.3 y=56.4 w=298.0 h=191.2
```

```
root@embox:/#ncnn_yolo_demo ./data/photos/dog.png
[1] img=./data/photos/dog.png S=320
[1] ok: 2202x1298 bytes=8574588
[2] load model...
[2] ptrs: pparam=0x003afec8 pmodel=0x003b1fd0
[2] rc: param=0 model=0
[3] preprocess(letterbox+norm)...
[3] lb bytes=307200 nw=320 nh=189 pad=(0,65) scale=0.145322
[3] in mat: w=320 h=320 c=3 dims=3 data=0x035ed040
[4] infer(input+extract)...
[4] rc: input=0 extract=0 out(w=2100 h=84 c=1 dims=2 data=0x0514c040)
[5] decode+nms...
[5] dets=10 (C=84)
[5] nms keep=1
  dog cls=16 conf=0.942 x=50.1 y=69.4 w=223.2 h=181.8
```

Ресурсы и ограничения

- Размер демонстрационной сборки с YOLOv8n - 30Мб
- Значимая часть памяти это веса модели и картинки для демонстрации
- Требует платформы с внешней памятью и достаточным RAM
- Недостижима для классических микроконтроллеров, где флеш-память и RAM измеряются единицами мегабайт и ниже.

Спасибо за внимание!