



NPU 快速入门 开发指南

版本号: 1.2
发布日期: 2024.11.13

版本历史

版本号	日期	制/修订人	内容描述
1.0	2024.03.12	AWA1382	初始版本
1.1	2024.04.11	AWA1382	增加 MR536、T536 平台说明
1.2	2024.11.13	AWA1382	1) 增加 A733 平台说明； 2) 更新 T527 平台说明； 3) 增加 SDK 产品包介绍一章。



目 录

1 前言	1
1.1 文档简介	1
1.2 目标读者	1
1.3 适用范围	1
1.3.1 名词解释	1
1.4 文档约定	2
2 SDK 产品包介绍	3
2.1 参考文档	3
2.2 常用工具	4
2.3 驱动与中间件	4
2.4 模型示例与端侧示例程序	4
3 开发环境准备与检查	6
4 部署示例程序	7
4.1 模型转换量化	7
4.2 编译示例程序	11
4.2.1 Linux 编译	12
4.2.2 Android 编译	13
4.2.3 运行示例程序	13
5 Docker 中部署示例程序	15
5.1 Ubuntu 机器操作	15
5.2 Docker 容器内操作	15
5.2.1 模型量化转换	15
5.2.2 编译示例程序	16
5.2.2.1 Linux 编译	16
5.2.2.2 Android 编译	16
5.3 运行示例程序	17
6 常见问题	19

1 前言

1.1 文档简介

本文档对 NPU SDK 产品包进行介绍，以及面向零基础用户介绍如何快速在 PC 上完成模型转换，并部署到板端。

1.2 目标读者

本文档（本指南）主要适用于以下人员：

- 技术支持工程师
- AI 软件开发工程师
- AI 算法开发工程师

1.3 适用范围

硬件平台：V85x、R853、MR527、T527、AI985、MR536、T536、A733

软件平台：Tina 系统、Android13 及以上系统

1.3.1 名词解释

术语	解释说明
NPU	Neural Network Processing Unit，神经网络处理器。
NBG	Network Binary Graph，驱动可识别的网络二进制文件。
pcq	perchannel quantize，逐通道量化。

1.4 文档约定

本文中出现的符号如下：

注意

- 提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。



2 SDK 产品包介绍

本节简要介绍 SDK 产品包的组成及使用方法。包括如下内容：

类型	获取方式
文档	一号通发布，Software 软件类文档/基础组件开发指南
工具	工具不同版本的下载链接见《NPU_环境部署_参考指南》附录。
驱动源码	Linux5.4 (Android/tina) 及以上：位于 SDK， Android/longan/bsp/drivers/npu 或 Tina/bsp/driver/npu Linux4.9：位于 SDK，lichee/linux-4.9/drivers/char/sunxi_nna_xxx
中间件	Android、Linux：位于 ai-sdk，ai-sdk/viplite(unified) V85x/R853：位于 SDK，package/allwinner/libsdk-viplite-driver/sdk_release
模型示例	位于 ai-sdk，ai-sdk/models
端侧示例程序	位于 ai-sdk，ai-sdk/examples

说明

不同平台的 ai-sdk 仓库在不同位置：

V85x/R853：位于 SDK，package/allwinner/ai-sdk，部分资料需额外下载，下载链接见通过一号通的Release发布声明/文档包指引/npu-deploy-toolchain.txt。

MR527/AI985：位于 SDK，openwrt/package/allwinner/vision/ai-sdk。

MR536：位于 SDK，platform/allwinner/vision/ai-sdk（编译在openwrt/package/allwinner/vision/ai-sdk进行）。

T527 AIOT Linux/T536：位于 SDK，platform/allwinner/vision/ai-sdk（编译在buildroot/buildroot-xxxxxx进行）。

T527/A733 Android：位于 SDK，vendor/aw/ai-sdk。

2.1 参考文档

目前在一号通提供的指导文档、说明文档汇总如下：

功能分类	文档	内容
环境搭建	NPU_开发环境搭建_参考指南	指导安装 NPU 工具
模型部署	NPU_快速入门_开发指南	介绍产品包，及指导基于常用模型快速跑通示例和端侧程序
模型部署	NPU_模型部署_开发指南	介绍系统及详细说明模型部署、工具使用
模型部署	NPU_算子支持列表	提供算子支持列表及各框架支持情况
模型部署	NPU_RuntimeAPI_参考指南	详细说明端侧应用程序的 API
NPU 测试	[型号]_NPU_常见网络性能_测试报告	常见模型性能参考

2.2 常用工具

NPU 开发工具主要包括以下几个：

- **Acuity Toolkit**：模型转换工具，用于模型导入、优化、量化、推理以及导出等。
- **VivanteIDE**：IDE 仿真工具，用于 PC 侧的模型仿真验证和 Profile 性能仿真分析。
- **Docker 镜像包**：集成平台部署 AI 算法所需的软件、工具包。
- **NBInfo**：NBG 分析工具，用于 PC 端查看分析转换后的驱动可识别的模型文件。
- **vpm_run**：模型运行工具，用于端侧运行任意模型，通用程序。

2.3 驱动与中间件

NPU 开发可以参考《NPU_模型部署_开发指南》的《NPU 驱动配置》和《驱动问题》，共有两套 NPU 模块驱动：

- **aw_nna_vip/sunxi_nna_vip**：VIPLite 版本 NPU 驱动。
- **aw_nna_galcore/sunxi_nna_galcore**：Unified 版本 NPU 驱动。

NPU 模块运行例程依赖中间件，其中：

- **ai-sdk/viplite-android**：Android 环境 VIPLite 版本 NPU 中间件。
- **ai-sdk/viplite-tina**：Linux 环境 VIPLite 版本 NPU 中间件。
- **ai-sdk/unified-android**：Android 环境 Unified 版本 NPU 中间件。
- **ai-sdk/unified-tina**：Linux 环境 Unified 版本 NPU 中间件。

说明

VIPLite 版本的中间件，在 v2.0.0 之前，包含 **libVIPLite.so** 和 **libVIPuser.so**。v2.0.0 之后，包含 **libNBGlinker.so** 和 **libVIPhal.so**。库名字有所差异，接口 API 基本不变，API 具体用法可参考《NPU_RuntimeAPI_参考指南》。

2.4 模型示例与端侧示例程序

模型示例放在 **ai-sdk/models**，给出不同深度学习模型格式的模型原型和数据集，可使用工具进行转换、部署。包括以下模型：

模型	平台	备注
lenet	caffe	
lstm_mnist	tensorflow	lstm

模型	平台	备注
mobilenet_v1_1.0_224_quant	tflite	已量化
yolov3_tiny	darknet	
MobileNetV2_Imagenet	keras	
squeezenet1_0	pytorch	
inception_v1	onnx	
yolov5s-sim	onnx	
deepspeech2	tensorflow	

端侧示例程序放在ai-sdk/examples，给出不同功能的程序源码，可用 SDK 的工具链进行编译，放在端侧运行测试。包括以下程序：

功能分类	例程	备注
基础例程	vpm_run	VIPLite 版本 NPU 驱动使用的基础例程
基础例程	vnn_run	Unified 版本 NPU 驱动使用的基础例程
目标检测	yolov3	yolov3 模型部署 demo
目标检测	yolov5	yolov5 模型部署 demo
实例分割	yolact	yolact 模型部署 demo
深度检测	struct2depth	struct2depth 模型部署 demo
图像分类	resnet50	Resnet50 模型部署 demo
图像分类	lenet	Lenet 模型部署 demo
OCR 识别	chineseOCR	chineseOCR 模型部署 demo
人体姿态	head_pose	head_pose 模型部署 demo
多线程例程	multi_thread	NPU 多线程运行 demo
零拷贝例程	shared_buffer	零拷贝，即其他模块和 NPU 共享 buffer 的 demo
共享权重例程	share_weight	共享权重的 demo
自定义 Lut 例程	custom_lut	自定义 Lut 算子的 demo

3 开发环境准备与检查

根据《NPU_开发环境部署_参考指南》，部署 PC 侧的 ubuntu 环境，使用 Docker 镜像环境或自行安装两个工具均可。

首先，使用 `pegasus --help` 命令检查 Acuity Toolkit 是否可用，打印信息如下：

Pegasus commands.

positional arguments:

{import,export,generate,prune,inference,quantize,train,dump,measure,help}

import Import models.

export Export models.

generate Generate metas.

prune prune models.

inference Inference model and get result.

quantize Quantize model.

train Train model.

dump Dump model activations.

measure Get amount of calculation, parameter and activation.

help Print a synopsis and a list of commands.

optional arguments:

-h, --help show this help message and exit

然后，使用 `echo` 命令检查 `ACUITY_PATH` 环境变量是否配置，方法如下：

```
AwExdroid-AI:~/ai-sdk$ echo ${ACUITY_PATH}
/root/Verisilicon_Tool_Acuity_Toolkit/acuity-toolkit-binary-x.x.x/bin/
```

最后，使用 `echo` 命令检查 `VIV_SDK` 环境变量是否配置，方法如下：

```
AwExdroid-AI:~/ai-sdk$ echo ${VIV_SDK}
/root/Vivante_IDE/VivanteIDEx.x.x/cmdtools
```

本文所用示例已集成到 `ai-sdk/models`、`ai-sdk/examples` 和 Docker 镜像包中。ai-sdk 获取方式见《SDK 产品包介绍》一节。

4 部署示例程序

本章将介绍如何快速在开发板上运行示例程序。

4.1 模型转换量化

在ai-sdk/models中，有各个框架模型的转换示例。下面以 yolov5s 模型为例，介绍部署脚本的使用方法。

yolov5s-sim 的目录结构如下：

```
AwExdroid-AI:~/ai-sdk/models$ tree yolov5s-sim
yolov5s-sim/
├── channel_mean_value.txt      #用于配置通道mean和scale的文件
├── convert_export.sh          #快速部署脚本1
├── dataset.txt                #记录量化校正的所有数据的路径
├── images                    #用于量化校正的数据集
│   ├── COCO_train2014_000000000529.jpg
│   ├── COCO_train2014_000000001183.jpg
│   ├── COCO_train2014_000000002349.jpg
│   ├── COCO_train2014_000000003685.jpg
│   ├── COCO_train2014_000000004463.jpg
│   └── dog.jpg
├── inference_compare.sh      #快速部署脚本2
├── inputs_outputs.txt        #用于配置输入输出信息的文件
└── yolov5s-sim.onnx          #yolov5s的onnx模型文件，已把输入尺寸固定为(1,3,640,640)
```

提供两种脚本进行部署，通用脚本（其他模型通用）和快速部署脚本。不管使用哪种方式，固定 shape 的模型生成、命令详细解释、多输入模型的配置等请参考《NPU_模型部署_开发指南》的《NPU 使用说明》一章。若量化后精度不满足需求，均需要手动调整量化描述文件进行精度优化，请参考《NPU_模型部署_开发指南》的《量化精度调优》一章。

第一种，通用脚本部署。

脚本请从ai-sdk/scripts目录中获取。

```
AwExdroid-AI:~/ai-sdk/scripts$ tree
├── pegasus_dump.sh
├── pegasus_export_ovx_nbg.sh
├── pegasus_export_ovx.sh
├── pegasus_import.sh
├── pegasus_inference.sh
└── pegasus_quantize.sh
```

建议把原始模型目录放置在ai-sdk/models下，在 models 目录下执行source env.sh。而且保证模型的名字和模型目录名一致。

```
AwExdroid-AI:~/ai-sdk/models$ source env.sh
AwExdroid-AI:~/ai-sdk/models$ ls -l
-rwxr--r-- 1 1010 1012 2211 Jan 10 09:00 env.sh
drwxr-xr-x 3 1010 1012 4096 Mar 6 06:04 lenet
lrwxrwxrwx 1 root root 26 Mar 11 05:43 pegasus_dump.sh -> ../scripts/pegasus_dump.sh
lrwxrwxrwx 1 root root 32 Mar 11 05:43 pegasus_export_ovx.sh -> ../scripts/pegasus_export_ovx.sh
lrwxrwxrwx 1 root root 35 Mar 11 05:43 pegasus_export_ovx_nbg.sh -> ../scripts/pegasus_export_ovx_nbg.sh
lrwxrwxrwx 1 root root 28 Mar 11 05:43 pegasus_import.sh -> ../scripts/pegasus_import.sh
lrwxrwxrwx 1 root root 31 Mar 11 05:43 pegasus_inference.sh -> ../scripts/pegasus_inference.sh
lrwxrwxrwx 1 root root 30 Mar 11 05:43 pegasus_quantize.sh -> ../scripts/pegasus_quantize.sh
drwxrwxr-x 2 1010 1012 4096 Mar 8 17:05 yolov5s-sim
... ..
```

按照以下步骤进行，生成 uint8 量化的 NBG 文件。

```
~/ai-sdk/models$ ./pegasus_import.sh yolov5s-sim
~/ai-sdk/models$ vim yolov5s-sim/yolov5s-sim_inputmeta.yml #修改scale为0.00392157
~/ai-sdk/models$ ./pegasus_quantize.sh yolov5s-sim uint8
~/ai-sdk/models$ ./pegasus_inference.sh yolov5s-sim uint8
~/ai-sdk/models$ ./pegasus_export_ovx_nbg.sh yolov5s-sim uint8 VIP9000PICO_PID0XEE ${VIV_SDK} #注意调整第三个参数
```

说明

1. V85x、R853s 平台的型号为 VIP9000PICO_PID0XEE; MR527、AI985、T527 平台的型号为 VIP9000NANOSI_PLUS_PID0X10000016; MR536、T536、A733 平台的型号为 VIP9000NANODI_PLUS_PID0X1000003B。

步骤简要说明：

1. yolov5s-sim.onnx 文件，是基于官方的 yolov5s.onnx，把输入尺寸固定为 (1,3,640,640)；
2. inputs_outputs.txt 文件，是基于经验，去掉 yolov5s 的 output 节点，所以输出只有 3 个；
3. pegasus_import.sh 模型名称，进行模型导入，转化出模型的结构、权重、输入配置、输出配置；
4. 自行手动修改 yolov5s-sim_inputmeta.yml，是基于 yolov5s 算法原理，进行 scale 修正；
5. pegasus_quantize.sh 模型名称量化类型 [uint8, int16, pcq]，进行模型量化
6. pegasus_inference.sh 模型名称量化类型，进行模型推理
7. pegasus_export_ovx_nbg.sh 模型名称量化类型 NPU 型号 SDK 工具目录，进行模型导出

生成文件情况如下：

```
AwExdroid-AI:~/ai-sdk/models$ tree yolov5s-sim
yolov5s-sim
├── channel_mean_value.txt
├── convert_export.sh
├── dataset.txt
├── entropy.txt      #熵值文件
├── images
├── inf              #仿真后的输入输出tensor文件
├── yolov5s-sim_uint8 #后缀uint8是量化类型
│   ├── iter_0_attach_Transpose_Transpose_214_out0_0_out0_1_3_80_80_85.tensor #214节点，输出tensor文件
│   ├── iter_0_attach_Transpose_Transpose_326_out0_1_out0_1_3_40_40_85.tensor #326节点，输出tensor文件
│   ├── iter_0_attach_Transpose_Transpose_438_out0_2_out0_1_3_20_20_85.tensor #438节点，输出tensor文件
│   └── iter_0_images_208_out0_1_3_640_640.tensor #images节点，输入的tensor文件
├── inference_compare.sh
└── inputs_outputs.txt
```

```

├── wksp
│   ├── yolov5s-sim_uint8
│   └── yolov5s-sim_uint8_nbg_unify #用于IDE工具或Unified版本驱动的工程代码
│       ├── BUILD
│       ├── main.c
│       ├── makefile.linux
│       ├── nbg_meta.json
│       ├── network_binary.nb #用于设备端的NBG模型文件
│       ├── vnn_global.h
│       ├── vnn_post_process.c
│       ├── vnn_post_process.h
│       ├── vnn_pre_process.c
│       ├── vnn_pre_process.h
│       ├── vnn_yolov5ssimuint8.c
│       ├── vnn_yolov5ssimuint8.h
│       ├── yolov5ssimuint8.2012.vcxproj
│       └── yolov5ssimuint8.vcxproj
├── yolov5s-sim.data #网络权重文件
├── yolov5s-sim_inputmeta.yml #输入描述文件
├── yolov5s-sim.json #网络结构文件
├── yolov5s-sim.onnx
├── yolov5s-sim_postprocess_file.yml #输出描述文件
└── yolov5s-sim_uint8.quantize #量化描述文件

```

第二种，快速部署脚本部署。

```

#进入模型目录
AwExdroid-AI:~/ai-sdk/models$cd yolov5s-sim

#运行转换导出脚本
#用法: ./convert_export.sh <model_name> <quantize_type> <platform>
#quantize_type: uint8, int16, pcq
#platform: v85x,r853,mr527,t527,ai985,mr536,t536
AwExdroid-AI:~/ai-sdk/models/yolov5s-sim$ ./convert_export.sh yolov5s-sim uint8 v85x
AwExdroid-AI:~/ai-sdk/models/yolov5s-sim$ ./inference_compare.sh yolov5s-sim uint8

```

步骤简要说明（倾斜部分，为与第一种不同的地方）：

1. yolov5s-sim.onnx 文件，是基于官方的 yolov5s.onnx，把输入尺寸固定为 (1,3,640,640)；
2. inputs_outputs.txt 文件，是基于经验，去掉 yolov5s 的 output 节点，所以输出只有 3 个；
3. channel_mean_value.txt，是基于 yolov5s 算法原理，进行 mean 和 scale 修正（脚本自动识别）；
4. ./convert_export.sh 模型名称量化类型硬件平台，进行模型转换导出。
5. ./inference_compare.sh 模型名称量化类型，进行仿真推理对比

生成文件情况如下：

与第一种脚本类似，只是 inf 目录下多了 yolov5s-sim_non-quantized 目录，用于 tensor 对比。

```

AwExdroid-AI:~/ai-sdk/models/yolov5s-sim$ tree
.
├── channel_mean_value.txt
├── convert_export.sh
├── dataset.txt
├── entropy.txt #熵值文件
└── images

```

```

├── inf          #仿真后的输入输出tensor文件
│   ├── yolov5s-sim_non-quantized #后缀non-quantized是未量化，即inference时传入float参数
│   │   ├── iter_0_attach_Transpose_Transpose_214_out0_0_out0_1_3_80_80_85.tensor #214节点，输出tensor文件
│   │   ├── iter_0_attach_Transpose_Transpose_326_out0_1_out0_1_3_40_40_85.tensor #326节点，输出tensor文件
│   │   ├── iter_0_attach_Transpose_Transpose_438_out0_2_out0_1_3_20_20_85.tensor #438节点，输出tensor文件
│   │   └── iter_0_images_208_out0_1_3_640_640.tensor #images节点，输入的tensor文件
│   └── yolov5s-sim_uint8 #后缀uint8是量化类型
│       ├── iter_0_attach_Transpose_Transpose_214_out0_0_out0_1_3_80_80_85.tensor #214节点，输出tensor文件
│       ├── iter_0_attach_Transpose_Transpose_326_out0_1_out0_1_3_40_40_85.tensor #326节点，输出tensor文件
│       ├── iter_0_attach_Transpose_Transpose_438_out0_2_out0_1_3_20_20_85.tensor #438节点，输出tensor文件
│       └── iter_0_images_208_out0_1_3_640_640.tensor #images节点，输入的tensor文件
├── inference_compare.sh
├── inputs_outputs.txt
├── wksp
│   ├── yolov5s-sim_uint8
│   └── yolov5s-sim_uint8_nbg_unify #用于IDE工具或Unified版本驱动的工程代码
│       ├── BUILD
│       ├── main.c
│       ├── makefile.linux
│       ├── nbg_meta.json
│       ├── network_binary.nb #用于设备端的NBG模型文件
│       ├── vnn_global.h
│       ├── vnn_post_process.c
│       ├── vnn_post_process.h
│       ├── vnn_pre_process.c
│       ├── vnn_pre_process.h
│       ├── vnn_yolov5ssimuint8.c
│       ├── vnn_yolov5ssimuint8.h
│       ├── yolov5ssimuint8.2012.vcxproj
│       └── yolov5ssimuint8.vcxproj
├── yolov5s-sim.data #网络权重文件
├── yolov5s-sim_inputmeta.yml #输入描述文件
├── yolov5s-sim.json #网络结构文件
├── yolov5s-sim.onnx
├── yolov5s-sim_postprocess_file.yml #输出描述文件
└── yolov5s-sim_uint8.quantize #量化描述文件

```

打印信息分析如下：

与第一种脚本类似，但是最后会打印执行过的完整命令（供学习），以及 uint8 与 golden tensor（non-quantized）的相似度。

#convert_export.sh最后的打印：

CMD History:

CMD:pegasus import onnx --model yolov5s-sim.onnx --output-model yolov5s-sim.json --output-data yolov5s-sim.data --inputs images --input-size-list '3,640,640' --outputs '350 498 646'

CMD:pegasus generate inputmeta --model yolov5s-sim.json --separated-database --input-meta-output yolov5s-sim_inputmeta.yml

CMD:pegasus generate postprocess-file --model yolov5s-sim.json --postprocess-file-output yolov5s-sim_postprocess_file.yml

CMD:pegasus quantize --model yolov5s-sim.json --model-data yolov5s-sim.data --device CPU --with-input-meta yolov5s-sim_inputmeta.yml --compute-entropy --rebuild --model-quantize yolov5s-sim_uint8.quantize --quantizer asymmetric_affine --qtype uint8

CMD:pegasus export ovxlib --model yolov5s-sim.json --model-data yolov5s-sim.data --dtype quantized --model-quantize yolov5s-sim_uint8.quantize --batch-size 1 --save-fused-graph --target-ide-project 'linux64' --optimize VIP9000PICO_PID0XEE --viv-sdk /root/Vivante_IDE/VivanteIDE5.8.2/cmdtools --pack-nbg-unify --with-input-meta yolov5s-sim_inputmeta.yml --postprocess-file yolov5s-sim_postprocess_file.yml --output-path ./wksp/yolov5s-sim_uint8/yolov5s-sim_uint8

```
#inference_compare.sh最后的打印：
#四个tensor的对比（第四个是输入tensor，所以是相同的）
using ./inf/yolov5s-sim_non-quantized for gloden tensor
using ./inf/yolov5s-sim_uint8 for compare tensor
==== compute tensor similarity for iter_0_attach_Transpose_Transpose_214_out0_0_out0_1_3_80_80_85.tensor ====
euclidean_distance 373.8933
cos_similarity 0.998559
==== compute tensor similarity for iter_0_attach_Transpose_Transpose_326_out0_1_out0_1_3_40_40_85.tensor ====
euclidean_distance 174.34483
cos_similarity 0.998785
==== compute tensor similarity for iter_0_attach_Transpose_Transpose_438_out0_2_out0_1_3_20_20_85.tensor ====
euclidean_distance 88.084885
cos_similarity 0.998935
==== compute tensor similarity for iter_0_images_208_out0_1_3_640_640.tensor ====
euclidean_distance 0.946635
cos_similarity 0.999999
=====End compute for iter_0_images_208_out0_1_3_640_640.tensor=====
CMD History:
CMD:pegasus inference --model yolov5s-sim.json --model-data yolov5s-sim.data --dtype quantized --model-quantize
yolov5s-sim_uint8.quantize --iterations 1 --device CPU --output-dir ./inf/${NAME}_uint8 --postprocess-file yolov5s-
sim_postprocess_file.yml --with-input-meta yolov5s-sim_inputmeta.yml
CMD:python3 /root/acuity-toolkit-binary-6.21.1/bin/tools/compute_tensor_similarity.py ./inf/yolov5s-sim_uint8/
iter_0_images_208_out0_1_3_640_640.tensor ./inf/yolov5s-sim_non-quantized/
iter_0_images_208_out0_1_3_640_640.tensor
```

可见三个输出的 cos 相似度都比较高。

4.2 编译示例程序

在ai-sdk/examples中，有不同模型、不同功能的端侧示例程序。下面以 yolov5s 模型为例，介绍编译方法。

yolov5s 的目录结构如下：

```
AwExdroid88:~/ai-sdk/examples/yolov5$ tree src/
|__ input_data
|   |__ dog_640_640.jpg #输入数据：已缩小到模型输入的尺寸
|   |__ dog.jpg        #输入数据：原尺寸
|   |__ main.c          #NPU运行实现
|   |__ Makefile
|   |__ model           #NBG模型，各平台不通用，请选择正确平台的NBG模型，或使用上文方法转换的进行替换
|   |   |__ v2
|   |   |   |__ yolov5.nb
|   |   |__ v3
|   |   |   |__ yolov5.nb
|   |__ yolov5_post_process.cpp #后处理实现
|   |__ yolov5_post_process.h
|   |__ yolov5_pre_process.cpp #预处理实现
|   |__ yolov5_pre_process.h
```

⚠ 注意

MR527、AI985、T527 Linux 平台使用 v2 模型，MR536、T536、A733 平台使用 v3 模型。

4.2.1 Linux 编译

使用 Tina 预置编译

V85x、R853 平台：

```
#在SDK根目录执行以下命令
make menuconfig
Allwinner --->
ai-sdk selection --->
<*> yolov5..... yolov5 demo

make -j32
#输出文件在~/sdk_dir/out/xxx/compile_dir/target/ai-sdk/yolov5
```

MR527、AI985、MR536 平台：

```
#在SDK根目录执行以下命令
make menuconfig
Allwinner --->
Vision --->
<*> ai-sdk-viplite..... allwinner npu viplite framework --->

make -j32
#输出文件在~/sdk_dir/out/xxx/openwrt/build_dir/target/ai-sdk/ipkg.../etc/npu/yolov5
#安装到设备在/etc/npu/yolov5
```

make 进行全局编译一次后，后续修改示例程序源码，可以进入该目录，使用mm-B命令单独编译 yolov5。

⚠ 注意

MR536 平台的源码在platform/allwinner/vision/ai-sdk，需在openwrt/package/allwinner/vision/ai-sdk路径进行mm。

T527 Linux、T536 平台：

```
#在SDK根目录执行以下命令
./build.sh buildroot_menuconfig
Target packages --->
allwinner platform private package select --->
ai-sdk --->
[*] ai sdk
[*] ai sdk viplite lib and vpm_run
[*] ai sdk viplite yolov5
cd buildroot/buildroot-xxxxxx
make ai-sdk-rebuild
#输出文件在~/sdk_dir/out/xxx/xxx/buildroot/buildroot/target/etc/npu/yolov5
#安装到设备在/etc/npu/yolov5
```

4.2.2 Android 编译

```
AwExdroid88:~/Android_sdk/vendor/aw/ai-sdk/examples/yolov5$ mm

#输出文件分别在以下路径
#~/Android_sdk/out/target/product/xxx/vendor/etc/models/yolov5.nb
#~/Android_sdk/out/target/product/xxx/vendor/etc/input_data/dog.jpg
#~/Android_sdk/out/target/product/xxx/vendor/bin/yolov5
#~/Android_sdk/out/target/product/xxx/vendor/bin/yolov5.sh

#安装到设备在/vendor/bin/yolov5
```

4.2.3 运行示例程序

把二进制文件 yolov5、测试图片 dog.jpg 和模型文件 yolov5s-sim_uint8.nb 推到板端，执行以下命令：

```
./yolov5 yolov5s-sim_uint8.nb dog.jpg

#运行结果见result.png
```

若编译时进行预置，也可以直接执行 yolov5.sh 查看效果：

```
# cat /vendor/bin/yolov5.sh
#!/bin/sh

yolov5 /vendor/etc/models/yolov5.nb /vendor/etc/input_data/dog.jpg
```

yolov5s-sim_uint8.nb 文件即模型导出时生成在wkspace/yolov5s-sim_uint8_nbg_unify/network_binary.nb。

Android 设备可推到/data目录，容量比较小的 Tina 设备可插入 SD 卡再推到/mnt/extsd目录中。

运行打印输出分析如下：

```
./yolov5s_nbg input
viplite init OK.
Viplite driver version=0x00010d00...
VIP cid=0xee, device_count=1
* device[0] core_count=1
awnn_init total: 182.94 ms.
vip_create_network network_binary.nb: 18.83 ms.

#输入信息
input 0 dim 640 640 3 1, data_format=2, name=input[0], elements=0, scale=0.003922, zero_point=0
create input buffer 0: 1228800

#输出信息
output 0 dim 85 80 80 3 1, data_format=2, name=uid_5_out_0, elements=1632000, scale=0.085724, zero_point=210
create output buffer 0: 1632000
output 1 dim 85 40 40 3 1, data_format=2, name=uid_4_out_0, elements=408000, scale=0.071622, zero_point=204
create output buffer 1: 408000
output 2 dim 85 20 20 3 1, data_format=2, name=uid_3_out_0, elements=102000, scale=0.071997, zero_point=196
create output buffer 2: 102000
memory pool size=5524480 bytes
```



```
load_param network_binary.nb: 2.44 ms.
prepare network network_binary.nb: 30.11 ms.
set network io network_binary.nb: 0.01 ms.
awnn_create total: 52.78 ms.
yolov5s_preprocess.cpp run.
memcpy(0xb410e000, 0xb2ff0010, 1228800) load_input_data: 1.54 ms.
vip_flush_buffer input: 0.02 ms.
awnn_set_input_buffers total: 2.06 ms.
vip_run_network: 128.42 ms.
vip_flush_buffer output: 0.02 ms.
int8/uint8 1632000 memcpy: 11.80 ms.
int8/uint8 408000 memcpy: 2.91 ms.
int8/uint8 102000 memcpy: 0.56 ms.
tensor to fp: 51.69 ms.
awnn_run total: 181.01 ms.
yolov5s_postprocess.cpp run.
```

#检测结果

```
detection num: 3
16: 82%, [ 111, 243, 256, 601], dog
7: 69%, [ 390, 83, 576, 194], truck
1: 45%, [ 83, 143, 468, 467], bicycle
awnn_destroy total: 2.41 ms.
awnn_uninit total: 35.26 ms.
```



5 Docker 中部署示例程序

5.1 Ubuntu 机器操作

进入 Docker 容器

```
sudo docker run -it --privileged -v /home/${USER}/docker_data:/workspace ubuntu-npu:v1.x /bin/bash

#或者使用容器ID进入
sudo docker ps -a
sudo docker exec -it 容器ID /bin/bash
```

将 mobilenet_v2_ssd 例程放在 docker_data 目录（该目录已挂载到 docker 容器），路径如下：

```
.
├── board-demo
├── 3rdparty
├── common
├── mobilenet_v2_ssd_demo
├── model-convert
└── mobilenet_v2_ssd
```

5.2 Docker 容器内操作

5.2.1 模型量化转换

```
cd /workspace/model-convert/mobilenet_v2_ssd/

# 导入
./pegasus_import.sh mb2-ssd-lite-sim
# 留意生成的两个yaml文件，提供的例程中已修改；
# 1.xxx_inputmeta.yml，修改校准文件的路径path，根据算法输入的预处理修改mean，scale的值；
# 2.xxx_postprocess_file.yml，add_postproc_node参数设为true，推理结果由NPU内部做反量化浮点输出；

# 量化
./pegasus_quantize.sh mb2-ssd-lite-sim uint8

# 仿真
./pegasus_inference.sh mb2-ssd-lite-sim uint8

# 导出nb模型
./pegasus_export_ovx_nbg_x527.sh mb2-ssd-lite-sim uint8
# 生成的模型文件为./wksp/mb2-ssd-lite-sim_uint8_nbg_unify/mb2-ssd-lite-sim_xxx.nb
```

5.2.2 编译示例程序

5.2.2.1 Linux 编译

提供的示例可以直接在 docker 环境下编译，需检查交叉编译工具链是否正确；

其中示例提供的 libVIP*.so 库作为编译链接使用，板端运行以板端的 so 库为准；

```
cd /workspace/board-demo/mobilenet_v2_ssd_demo/

cd ../0-toolchains
#解压选择硬件匹配的交叉编译工具链，下面以MR527平台使用为例
tar xvf gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu.tar.xz

# 解压opencv库压缩包
cd ../3rdparty/opencv/
# armhf, 例如v85x平台使用
unzip arm-linux-gnueabi-hf-aw.zip
# aarch64, 例如MR527平台使用
unzip aarch64-linux-sunxi-glibc.zip

# 重新进入demo目录
cd /workspace/board-demo/mobilenet_v2_ssd_demo/

#使用其它路径的工具链，需打开CMakeList.txt文件修改对应的交叉编译工具链路径
vim CMakeLists.txt

# 编译
mkdir build
cd build
cmake ..
make

# 将可执行文件推到板端运行，建议推到tf卡目录，空间充足
# 将nb模型文件、输入图片push到与执行文件相同目录
# ./mbv2-ssd-demo -h 查看执行示例帮助

# 例如
./mbv2-ssd-demo -b model/mbv2_ssd_xxx.nb -i 000012.jpg
```

5.2.2.2 Android 编译

下载 Android NDK，下载地址：<https://developer.android.google.cn/ndk/downloads/index?hl=th>

将下载的 NDK 放到编译服务器目录，例如：../0-toolchains/；

请根据下载的 NDK 版本修改build_android.sh 文件；

```
#!/bin/bash

# please modify ndk path
#set -e ANDROID_NDK=/your/ndk/path/android-ndk-r25b
```

```
#if [ -z ${ANDROID_NDK} ]
#then
# ANDROID_NDK=/your/ndk/path/android-ndk-r25b
#fi

CUR_DIR=$(cd $(dirname $0) && pwd)

# aarch64
BUILD_DIR=${CUR_DIR}/build_aarch64_android

if [[ ! -d "${BUILD_DIR}" ]]; then
  mkdir -p ${BUILD_DIR}
fi

cd ${BUILD_DIR}

cmake -DCMAKE_TOOLCHAIN_FILE="$ANDROID_NDK/build/cmake/android.toolchain.cmake" \
  -DCMAKE_MAKE_PROGRAM="$ANDROID_NDK/prebuilt/linux-x86_64/bin/make" \
  -DANDROID_ABI="arm64-v8a" \
  -DANDROID_PLATFORM=android-24 ..

make
```

运行编译脚本./build_android.sh。

5.3 运行示例程序

把二进制文件 mbv2-ssd-demo、测试图片和模型文件 mbv2_ssd_xxx.nb 推到板端，执行以下命令：

```
./mbv2-ssd-demo -b model/mbv2_ssd_xxx.nb -i 000012.jpg
#运行结果见ssd_out.png
```

Android 设备可推到/data目录，容量比较小的 Tina 设备可插入 SD 卡再推到/mnt/extsd目录中。

运行打印输出分析如下：

```
model_file=mb2-ssd-lite-sim_uint8_85x.nb, input=000012.jpg, loop_count=1, malloc_mbyte=10
#驱动版本
npu driver version=0x00010d00...

#输入信息
input 0 dim 300 300 3 1, data_format=2, quant_format=2, name=input[0], scale=0.007812, zero_point=127

#输出信息
output 0 dim 21 3000 1 0, data_format=1, name=uid_3_sub_uid_1_out_0, none-quant
output 1 dim 4 3000 1 0, data_format=2, name=uid_2_out_0, scale=0.006623, zero_point=47
nbg name=mb2-ssd-lite-sim_uint8_85x.nb, size: 3463232.
create network 0: 7471 us.
prepare network: 7169 us.
mobilenet_v2 ssd preprocess.cpp run.
network: 0, loop count: 1
run time for this network 0: 16124 us.
get output finished.
mbv2_ssd_postprocess.cpp run.
```

```
#检测结果  
car = 0.98633 at 162 94 179 x 170  
postprocess time : 0.047 Sec  
destory npu finished.  
~NpuUint.
```



6 常见问题

1) yolov5 检测结果为 0

```
yolov5s_postprocess.cpp run.  
#检测数量为0  
detection num: 0  
awnn_destroy total: 4.30 ms.
```

可能原因：NBG 模型文件的输出与运行程序不一致，运行程序是 3 个输出。看以下打印可检查，可见 NBG 模型文件是四个输出。

```
output 0 dim 85 25200 1, data_format=2, name=uid_7_out_0, elements=2142000, scale=2.498428, zero_point=0  
create output buffer 0: 2142000  
output 1 dim 85 80 80 3 1, data_format=2, name=uid_6_out_0, elements=1632000, scale=0.085724, zero_point=210  
create output buffer 1: 1632000  
output 2 dim 85 40 40 3 1, data_format=2, name=uid_5_out_0, elements=408000, scale=0.071622, zero_point=204  
create output buffer 2: 408000  
output 3 dim 85 20 20 3 1, data_format=2, name=uid_4_out_0, elements=102000, scale=0.071997, zero_point=196  
create output buffer 3: 102000
```

解决方法：检查 inputs_outputs.txt 文件，是否配置 --outputs '350 498 646' 三个输出，否则请修正。

2) yolov5 检测结果混乱

```
awnn_run total: 180.80 ms.  
yolov5s_postprocess.cpp run.  
detection num: 921  
32: 100%, [ 308, 37, 312, 38], sports ball  
50: 100%, [ 173, 18, 173, 27], broccoli  
50: 100%, [ 172, 20, 173, 26], broccoli  
50: 100%, [ 620, 34, 621, 40], broccoli  
50: 100%, [ 172, 18, 173, 27], broccoli
```

可能原因：scale 未调整，量化结果不对，导致结果误差大。

解决方法：检查 yolov5s-sim_inputmeta.yml 的 scale 是否为 0.00392157，否则请修正。

3) 运行报错

```
./mbv2-ssd-demo: line 1: syntax error: unterminated quoted string
```

可能原因：编译运行程序的工具链和端侧设备的工具链不一致。

解决方法：检查 CMakeLists.txt




著作权声明

版权所有 © 2024 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、、**全志科技**、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。