密级状态：绝密(　　)　　秘密(　　)　　内部(　　)　　公开(√)

RKNN-Toolkit2 用户使用指南

（技术部，图形计算平台中心）

文件状态： [　] 正在修改 [√] 正式发布	当前版本：	V1.1.0
	作　　者：	HPC
	完成日期：	2021-6-30
	审　　核：	熊伟
	完成日期：	2021-6-30

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd

更新记录

版本	修改人	修改日期	修改说明	核定人
V0.5.0	HPC	2020-12-18	初始版本	熊伟
V0.6.0	HPC	2021-2-24	1. 更新混合量化章节 2. 更新 RKNN 对象初始化参数 3. 更新 Caffe 加载 API	熊伟
V0.7.0	HPC	2021-3-30	1. 更新 mmse 量化支持 2. 更新部分接口说明 3. 更新量化算法的说明与建议	熊伟
V1.0.0	HPC	2021-4-21	1. 更新 config 接口定义（删除 reorder_channel，新增 quant_img_RGB2BGR 和 custom_string） 2. 添加 list_devices 和 get_sdk_version 接口说明 3. 添加 eval_perf 和 eval_memory 接口说明	熊伟
V1.1.0	HPC	2021-6-30	1. 更新混合量化接口说明 2. 更新 mmse 接口说明 3. 更新量化精度分析接口说明 4. 添加精度问题排查章节	熊伟

版本	修改人	修改日期	修改说明	核定人

目 录

1 概述

1.1 主要功能说明

RKNN-Toolkit2 是为用户提供在 PC、Rockchip NPU 平台上进行模型转换、推理和性能评估的开发套件，用户通过该工具提供的 Python 接口可以便捷地完成以下功能：

1) 模型转换：支持 Caffe、TensorFlow、TensorFlow Lite、ONNX、DarkNet、PyTorch 等模型转为 RKNN 模型，并支持 RKNN 模型导入导出，RKNN 模型能够在 Rockchip NPU 平台上加载使用。
2) 量化功能：支持将浮点模型量化为定点模型，目前支持的量化方法为非对称量化（asymmetric_quantized-8 及 asymmetric_quantized-16），并支持混合量化功能。asymmetric_quantized-16 目前版本暂不支持。
3) 模型推理：能够在 PC 上模拟 Rockchip NPU 运行 RKNN 模型并获取推理结果；或将 RKNN 模型分发到指定的 NPU 设备上进行推理并获取推理结果。
4) 性能评估：将 RKNN 模型分发到指定 NPU 设备上运行，以评估模型在实际设备上运行时的性能。
5) 内存评估：评估模型运行时的内存的占用情况。使用该功能时，必须将 RKNN 模型分发到 NPU 设备中运行，并调用相关接口获取内存使用信息。
6) 量化精度分析：该功能将给出模型量化前后每一层推理结果与浮点模型推理结果的余弦距离，以便于分析量化误差是如何出现的，为提高量化模型的精度提供思路。

注：部分功能受限于对操作系统或芯片平台的依赖，在某些操作系统或平台上无法使用。当前版本对各操作系统（平台）的功能支持情况列表如下：

	Ubuntu 18.04	Windows 7/10	Debian 9/10 (aarch64)	MacOS Mojave / Catalina
模型转换	支持			
量化	支持			
模型推理	支持			
性能评估	支持			
内存评估	支持			
多输入	支持			
批量推理	支持(部分)			
设备查询	支持			
SDK 版本查询	支持			
量化精度分析	支持			
可视化功能				
模型优化开关	支持			

1.2 适用芯片

RKNN-Toolkit2 当前版本所支持芯片的型号如下：

- RK3566
- RK3568

1.3 适用系统

RKNN-Toolkit2 是一个跨平台的开发套件，已支持的操作系统如下：

- Ubuntu 18.04（x64）及以上

2 系统依赖说明

使用本开发套件时需要满足以下运行环境要求：

表 1 运行环境

操作系统版本	Ubuntu18.04（x64）及以上
Python 版本	3.6
Python 库依赖	numpy==1.16.6 onnx==1.7.0 onnxoptimizer==0.1.0 onnxruntime==1.6.0 tensorflow==1.14.0 tensorboard==1.14.0 protobuf==3.12.0 torch==1.6.0 torchvision==0.7.0 psutil==5.6.2 ruamel.yaml==0.15.81 scipy==1.2.1 tqdm==4.27.0 requests==2.21.0 tflite==2.3.0 opencv-python==4.4.0.46 PuLP==2.4 scikit_image==0.17.2

注：

1. 本文档主要以 Ubuntu 18.04 / Python3.6 为例进行说明。其他操作系统请参考《Rockchip_Quick_Start_RKNN_Toolkit2_CN.pdf》。

3 使用说明

3.1 安装

目前提供两种方式安装 RKNN-Toolkit2：一是通过 Python 包安装与管理工具 pip 进行安装；二是运行带完整 RKNN-Toolkit2 工具包的 docker 镜像。下面分别介绍这两种安装方式的具体步骤。

3.1.1 通过 pip install 命令安装

1. 创建 virtualenv 环境（如果系统中同时有多个版本的 Python 环境，建议使用 virtualenv 管理 Python 环境）

```
sudo apt install virtualenv
sudo apt-get install python3 python3-dev python3-pip
sudo apt-get install libxslt1-dev zlib1g zlib1g-dev libglib2.0-0 \
libsm6 libgl1-mesa-glx libprotobuf-dev gcc

virtualenv -p /usr/bin/python3 venv
source venv/bin/activate
```

2. 安装依赖库

```
pip3 install -r doc/requirements.txt
```

3. 安装 RKNN-Toolkit2

```
pip3 install package/rknn_toolkit2*.whl
```

请根据不同的 python 版本及处理器架构，选择不同的安装包文件（位于 package/目录）：

- **Python3.6 for x86_64**：rknn_toolkit2-x.x.x-cp36-cp36m-linux_x86_64.whl

3.1.2 通过 Docker 镜像安装

在 docker 文件夹下提供了一个已打包所有开发环境的 Docker 镜像，用户只需要加载该镜像即可直接上手使用 RKNN-Toolkit2，使用方法如下：

1、 安装 Docker

请根据官方手册安装 Docker（https://docs.docker.com/install/linux/docker-ce/ubuntu/）。

2、 加载镜像

执行以下命令加载镜像：

```
docker load --input rknn-toolkit2-x.x.x-docker.tar.gz
```

加载成功后，执行“docker images”命令能够看到 rknn-toolkit2 的镜像，如下所示：

```
REPOSITORY      TAG       IMAGE ID       CREATED        SIZE
rknn-toolkit2   x.x.x     4f6bae6686d8   1 hours ago    5.34GB
```

3、 运行镜像

执行以下命令运行 docker 镜像，运行后将进入镜像的 bash 环境。

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb rknn-toolkit2:x.x.x /bin/bash
```

将代码映射进 Docker 环境可通过附加“-v <host src folder>:<image dst folder>”参数，例如：

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb -v /home/rk/test:/test
rknn-toolkit2:x.x.x /bin/bash
```

4、 运行 demo

```
cd /example/tflite/mobilenet_v1
python3 test.py
```

3.2 RKNN-Toolkit2 的使用

以下详细给出各使用场景下 RKNN Toolkit2 的使用流程。

3.2.1 场景一：模型运行在模拟器上

该场景下，RKNN Toolkit2 运行在 PC 上，通过模拟器运行模型。

根据模型类型的不同，该场景又分为两个子场景：一是模型为非 RKNN 模型，即 Caffe、TensorFlow、TensorFlow Lite、ONNX、DarkNet、PyTorch 等模型；二是 RKNN 模型，Rockchip 的专有模型，文件后缀为“rknn”。

3.2.1.1 运行非 RKNN 模型

运行非 RKNN 模型与 RKNN 模型的最大区别在于，进行模型推理或模型性能/内存评估前，需要先将非 RKNN 模型转成 RKNN 模型。该场景下 RKNN Toolkit2 的完整使用流程如下图所示：

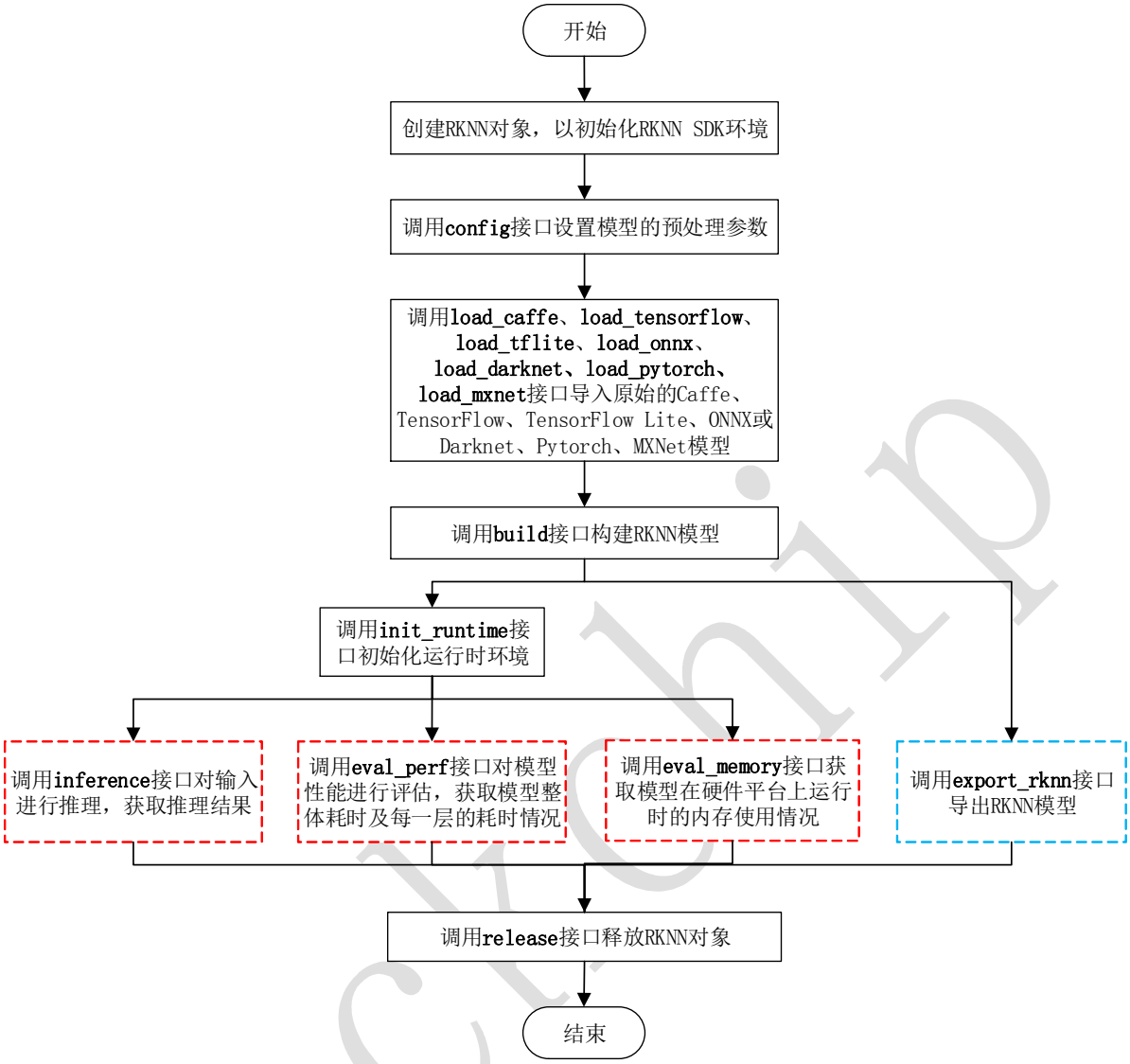

图 3-2-1-1 PC 上运行非 RKNN 模型时工具的使用流程

注：

1. 以上步骤请按顺序执行。
2. 蓝色框标注的步骤导出的 RKNN 模型可以通过 load_rknn 接口导入并使用。
3. 红色框标注的模型推理、性能评估和内存评估的步骤先后顺序不固定，根据实际使用情况决定。
4. 只有当目标平台是 Rockchip NPU 时，才可以调用 eval_perf / eval_memory 接口。

3.2.2 场景二：模型运行在与 PC 相连的 Rockchip NPU 平台上

RKNN Toolkit2 目前支持的 Rockchip NPU 平台包括 RK3566 / RK3568。

该场景下，RKNN Toolkit2 运行在 PC 上，通过 PC 的 USB 连接 NPU 设备。RKNN Toolkit2 将 RKNN 模型传到 NPU 设备上运行，再从 NPU 设备上获取推理结果、性能信息等。

首先，需要完成以下两个步骤：

1. 确保开发板的 USB OTG 连接到 PC，并且正确识别到设备，即在 PC 上调用 RKNN-Toolkit2 的 list_devices 接口可查询到相应的设备，关于该接口的使用方法，参见 3.5.15 章节。
2. 调用 init_runtime 接口初始化运行环境时需要指定 target 参数和 device_id 参数。其中 target 参数表明硬件类型，当前版本可选值为“rk3566”、“rk3568”。当 PC 连接多个设备时，还需要指定 device_id 参数，即设备编号，设备编号可通过 list_devices 接口查询，示例如下：

```
all device(s) with adb mode:
VD46C3KM6N
```

初始化运行时环境**代码示例**如下：

```
# RK3566
ret = init_runtime(target='rk3566', device_id='VGEJY9PW7T')

# RK3568
ret = init_runtime(target='rk3568', device_id='515e9b401c060c0b')
```

3.2.2.1 运行非 RKNN 模型

当模型为非 RKNN 模型（Caffe、TensorFlow、TensorFlow Lite、ONNX、DarkNet、PyTorch 等模型）时，RKNN-Toolkit2 工具的使用流程及注意事项同场景一里的子场景一（见 3.2.1.1 章节）。

3.2.2.2 运行 RKNN 模型

运行 RKNN 模型时，用户不需要设置模型预处理参数，也不需要构建 RKNN 模型，其使用流程如下图所示：

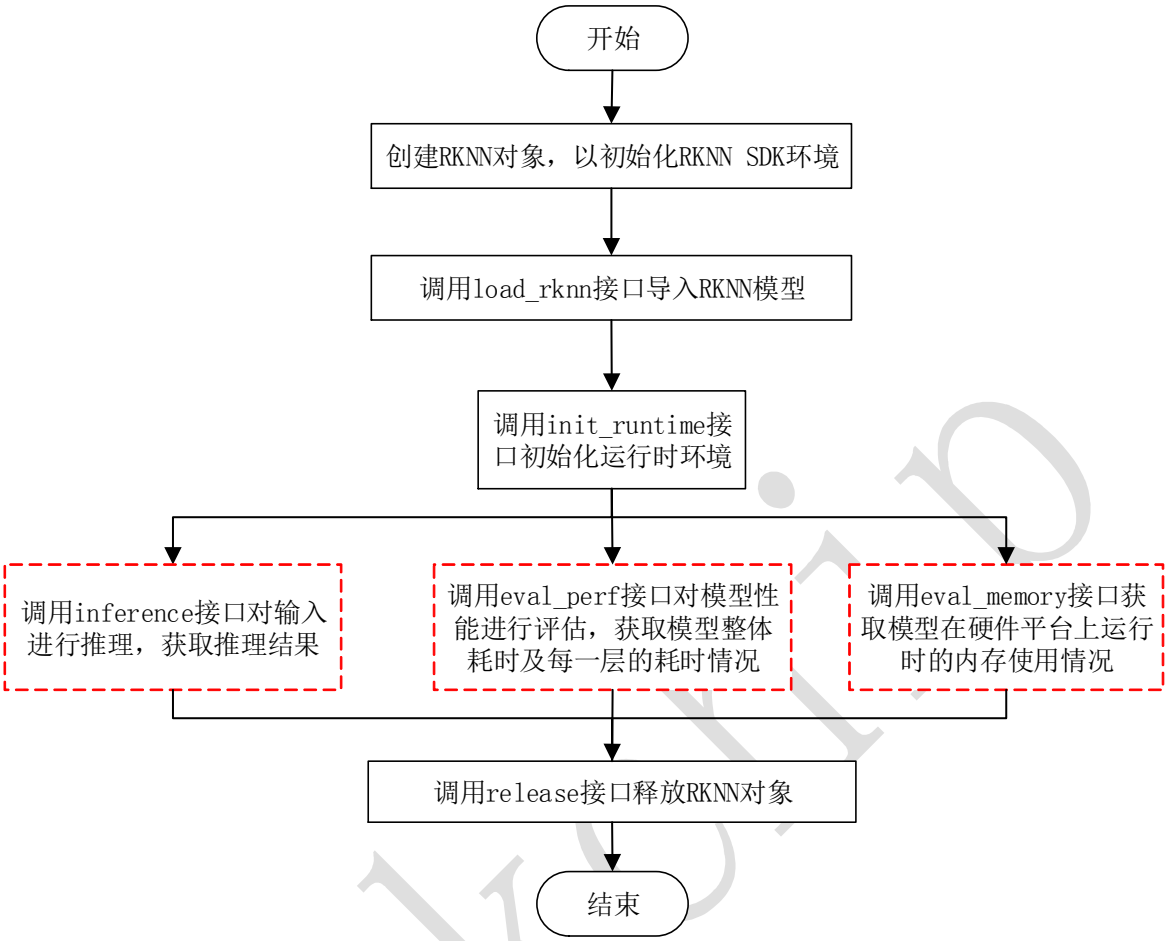

图 3-2-2-2 PC 上运行 RKNN 模型时工具的使用流程

注：

1. 以上步骤请按顺序执行。
2. 红色框标注的模型推理、性能评估和内存评估的步骤先后顺序不固定，根据实际使用情况决定。
3. 调用 inference / eval_perf / eval_memory 接口获取内存使用情况时，模型必须运行在硬件平台上。
4. 通过 load_rknn 导入的方式仅用于硬件平台相关功能的使用，无法使用如精度分析 accuracy_analysis 等功能。

3.2.3 场景三：模型运行在 RK356x Linux 开发板上

目前版本该功能暂不支持。

这种场景下，RKNN-Toolkit2 直接安装在 RK356x Linux 系统中。构建或导入的 RKNN 模型直接在 RK356x 上运行，以获取模型实际的推理结果或性能信息。

对于 RK356x 开发板，RKNN-Toolkit2 工具的使用流程取决于模型种类，如果模型类型是非 RKNN 模型，则使用流程同场景一中的子场景一（见 3.2.1.1 章节）；否则使用流程同子场景二（见 3.2.2.2 章节）。

3.3 混合量化

RKNN-Toolkit2 提供的量化功能可以在提高模型推理速度的基础上尽量保证模型精度，但是仍有某些特殊模型在量化后出现精度下降较多的情况。为了让在性能和精度之间做更好的平衡，RKNN-Toolkit2 提供了混合量化功能，用户可以通过分析手动指定各层是否进行量化，量化参数也可以进行修改。

注：

1. examples/functions 目录下提供了一个混合量化的例子 hybrid_quant，可以参考该例子进行混合量化的实践。

3.3.1 混合量化功能用法

目前混合量化功能支持如下用法：

1. 将指定的量化层改成非量化层（如用 float16 进行计算），因 NPU 上非量化算力较低，所以推理速度会有一定降低。

3.3.2 混合量化配置文件

在使用混合量化功能时，第一步是生成混合量化配置文件，本节对该配置文件进行简要介绍。

当调用混合量化接口 hybrid_quantization_step1 后，会在当前目录下生成名为 {model_name}.quantization.cfg 的配置文件。配置文件格式如下：

```
custom_quantize_layers: {}
quantize_parameters:
    Preprocessor/sub:0:
        qtype: asymmetric_quantized
        qmethod: layer
        dtype: int8
        min:
        -   -1.0
        max:
        -   1.0
        scale:
        -   0.00784313725490196
        zero_point:
        -   0
        ori_min:
        -   -1.0
        ori_max:
        -   1.0
.....
```

配置文件正文第一行是一个自定义量化操作数的字典，用户可将操作数名和相应的量化类型（可选值为 float16 / int16）添加到这里。 int16 暂未支持。

之后是模型中每个操作数的量化参数，每一个操作数都是一个字典。每个字典的 key 即 tensor_name，字典的 value 即量化参数，如果没有经过量化，则 dtype 值为 float16。

3.3.3 混合量化使用流程

使用混合量化功能时，具体分四步进行。

第一步，加载原始模型，生成量化配置文件和模型结构文件和模型配置文件。具体的接口调用流程如下：

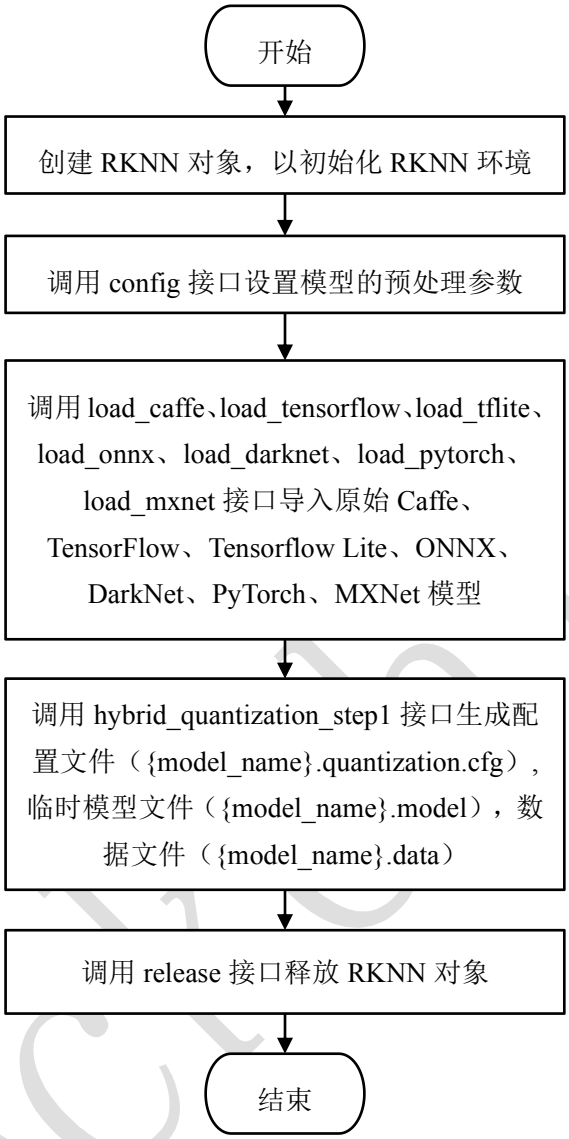

图 3-3-3-1 混合量化第一步接口调用流程

第二步，修改第一步生成的量化配置文件。

- 如果是将某些量化层改成非量化层，则找到不要量化的层的输出操作数（如果该层的输出操作数有多个，设置第一个或任意一个即可），将这些操作数加到 custom_quantize_layers 字典中，值为 float16。

第三步，生成 RKNN 模型。具体的接口调用流程如下：

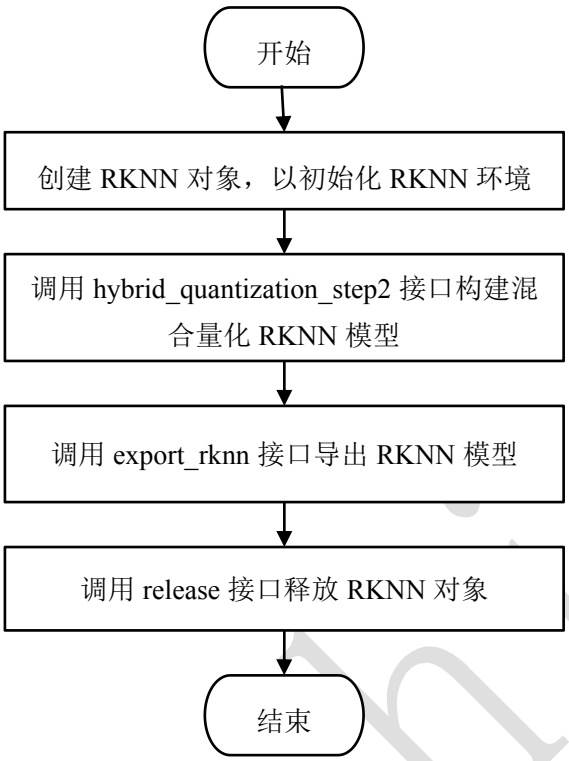

图 3-3-3-2 混合量化第三步接口调用流程

第四步，使用上一步生成的 RKNN 模型进行推理。

3.4 示例

以下是加载 TensorFlow Lite 模型的示例代码（详细参见 example/tflite/mobilenet_v1 目录），如果在 PC 上执行这个例子，RKNN 模型将在模拟器上运行：

```
import numpy as np
import cv2
from rknn.api import RKNN


def show_outputs(outputs):
    output = outputs[0][0]
    output_sorted = sorted(output, reverse=True)
    top5_str = 'mobilenet_v1\n-----TOP 5-----\n'
    for i in range(5):
        value = output_sorted[i]
        index = np.where(output == value)
        for j in range(len(index)):
            if (i + j) >= 5:
                break
```

```
                if value > 0:
                    topi = '{}: {}\n'.format(index[j], value)
                else:
                    topi = '-1: 0.0\n'
                top5_str += topi
        print(top5_str)


if __name__ == '__main__':

    # Create RKNN object
    rknn = RKNN()

    # pre-process config
    print('--> config model')
    rknn.config(mean_values=[128, 128, 128], std_values=[128, 128, 128])
    print('done')

    # Load tensorflow model
    print('--> Loading model')
    ret = rknn.load_tflite(model='mobilenet_v1_1.0_224.tflite')
    if ret != 0:
        print('Load mobilenet_v1 failed!')
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
    if ret != 0:
        print('Build mobilenet_v1 failed!')
        exit(ret)
    print('done')

    # Export rknn model
    print('--> Export RKNN model')
    ret = rknn.export_rknn('./mobilenet_v1.rknn')
    if ret != 0:
        print('Export mobilenet_v1.rknn failed!')
        exit(ret)
    print('done')

    # Set inputs
    img = cv2.imread('./dog_224x224.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = np.expand_dims(img, 0)

    # init runtime environment
    print('--> Init runtime environment')
    ret = rknn.init_runtime()
```

```
    if ret != 0:
        print('Init runtime environment failed')
        exit(ret)
    print('done')

    # Inference
    print('--> Running model')
    outputs = rknn.inference(inputs=[img])
    show_outputs(outputs)
    print('done')

    rknn.release()
```

其中 dataset.txt 是一个包含测试图片路径的文本文件，例如我们在 example/tflite/mobilent_v1 目录下有一张 dog_224x224.jpg 的图片，那么对应的 dataset.txt 内容如下：

```
dog_224x224.jpg
```

demo 运行模型预测时输出如下结果：

```
-----TOP 5-----
[156]: 0.8544921875
[155]: 0.080322265625
[205]: 0.0129241943359375
[284]: 0.0084075927734375
[194]: 0.0025787353515625
```

3.5 API 详细说明

3.5.1 RKNN 初始化及对象释放

在使用 RKNN Toolkit2 的所有 API 接口时，都需要先调用 RKNN()方法初始化 RKNN 对象，不再使用该对象时通过调用该对象的 release()方法进行释放。

初始化 RKNN 对象时，可以设置 ***verbose*** 和 ***verbose_file*** 参数，以打印详细的日志信息。其中 verbose 参数指定是否要在屏幕上打印详细日志信息；如果设置了 verbose_file 参数，且 verbose 参数值为 True，日志信息还将写到该参数指定的文件中。

举例如下：

```
# 将详细的日志信息输出到屏幕，并写到 mobilenet_build.log 文件中
```

```
rknn = RKNN(verbose=True, verbose_file='./mobilenet_build.log')
# 只在屏幕打印详细的日志信息
rknn = RKNN(verbose=True)
…
rknn.release()
```

3.5.2 RKNN 模型配置

在构建 RKNN 模型之前，需要先对模型进行通道均值、量化图片 RGB2BGR 转换、量化类型等的配置，这些操作可以通过 config 接口进行配置。

API	**config**
描述	设置模型参数。
参数	batch_size：批处理大小，默认值为 100。量化时将根据该参数决定每一批次参与运算的数据量，以校正量化结果。如果 dataset 中的数据量小于 batch_size，则该参数值将自动调整为 dataset 中的数据量。
	mean_values：输入的均值。参数格式是一个列表，列表中包含一个或多个均值子列表，多输入模型对应多个子列表，每个子列表的长度与该输入的通道数一致，例如[[128,128,128]]，表示一个输入的三个通道的值减去 128。如果 quant_img_RGB2BGR 设置成 True，则优先做 RGB2BGR 转换，再做减均值。
	std_values：输入的归一化值。参数格式是一个列表，列表中包含一个或多个归一化值子列表，多输入模型对应多个子列表，每个子列表的长度与该输入的通道数一致，例如[[128,128,128]]，表示设置一个输入的三个通道的值减去均值后再除以 128。如果 quant_img_RGB2BGR 设置成 True，则优先做 RGB2BGR 转换，再减均值和除以归一化值。
	epochs：量化时的迭代次数，每迭代一次，就选择 batch_size 指定数量的图片进行量化校正。默认值为-1，此时会根据 dataset 中的图片数量自动计算迭代次数以最大化利用数据集中的数据。目前版本该参数暂不支持。
	quant_img_RGB2BGR：表示在加载量化图像时是否需要先做 RGB2BGR 的操作。默认值为 False，如果有多个输入，则用列表包含起来，如[True, True, False]。

	该配置一般用在 Caffe 的模型上，Caffe 的模型训练时大多会对先数据集图像进行 RGB2BGR 转换，此时需要将该配置设为 True。 另外，该配置只对量化图像格式为 jpg/jpeg/png/bmp 有效，npy 格式读取时会忽略该配置，因此当模型输入为 BGR 时，npy 也需要为 BGR 格式。 **该配置仅用于在量化阶段（build 接口）读取量化图像或量化精度分析（accuracy_analysis 接口），并不会记录在最终的 RKNN 模型中，因此如果模型的输入为 BGR，则在调用 toolkit 的 inference 或 C-API 的 run 函数之前，需要保证传入的图像数据也为 BGR 格式。**
	quantized_dtype：量化类型，目前支持的量化类型有 asymmetric_quantized-8、asymmetric_quantized-16，默认值为 asymmetric_quantized-8。asymmetric_quantized-16 目前版本暂不支持。
	quantized_algorithm：计算每一层的量化参数时采用的量化算法，目前支持的量化算法有：normal 及 mmse，默认值为 normal。 Normal 量化算法的特点是速度较快，推荐量化数据量一般为 20-100 张左右，更多的数据量下精度未必会有进一步提升。 Mmse 量化算法由于采用暴力迭代的方式，速度较慢，但通常会比 normal 具有更高的精度，推荐量化数据量一般为 20-50 张左右，用户也可以根据量化时间长短对量化数据量进行适当增减。
	quantized_method：目前支持 layer 或者 channel。 layer：每层的 weight 只有一套量化参数； channel：每层的 weight 的每个通道都有一套量化参数； 通常情况下 channel 会比 layer 精度更高，默认值为 layer。
	optimization_level：模型优化等级。通过修改模型优化等级，可以关掉部分或全部模型转换过程中使用到的优化规则。该参数的默认值为 3，打开所有优化选项。值为 2 或 1 时关闭一部分可能会对部分模型精度产生影响的优化选项，值为 0 时关闭所有优化选项。

	target_platform：指定 RKNN 模型是基于哪个目标芯片平台生成的。目前支持 RK3566 / RK3568。该参数的值大小写不敏感。
	custom_stirng：添加自定义字符串信息到 rknn 模型，可以在 runtime 时通过 query 查询到该信息。
返回值	无

举例如下：

```
# model config
rknn.config(mean_values=[[103.94, 116.78, 123.68]],
            std_values=[[58.82, 58.82, 58.82]],
            quant_img_RGB2BGR=True, target_platform='rk3566')
```

3.5.3 模型加载

RKNN-Toolkit2 目前支持 Caffe、TensorFlow、TensorFlow Lite、ONNX、DarkNet、PyTorch 等模型的加载转换，这些模型在加载时需调用对应的接口，以下为这些接口的详细说明。

3.5.3.1 Caffe 模型加载接口

API	**load_caffe**
描述	加载 caffe 模型。
参数	model：caffe 模型文件（.prototxt 后缀文件）所在路径。
	proto：caffe 模型的格式（可选值为'caffe'或'lstm_caffe'）。为了支持 RNN 模型，增加了相关网络层的支持，此时需要设置 caffe 格式为'lstm_caffe'。'lstm_caffe'目前版本暂不支持。
	blobs：caffe 模型的二进制数据文件（.caffemodel 后缀文件）所在路径。该参数值可以为 None，RKNN Toolkit2 将随机生成权重等参数。
	input_name：caffe 模型存在多输入时，可以通过该参数指定输入层名的顺序，形如['input1','input2','input3']，注意名字需要与模型输入名一致；也可不设定，按 caffe 模型文件（.prototxt 后缀文件）自动给定。

返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前路径加载 mobilenet_v2 模型
ret = rknn.load_caffe(model='./mobilenet_v2.prototxt',
                      proto='caffe',
                      blobs='./mobilenet_v2.caffemodel',
                      inputname=['input1'])
```

3.5.3.2 TensorFlow 模型加载接口

API	**load_tensorflow**
描述	加载 TensorFlow 模型。
参数	tf_pb：TensorFlow 模型文件（.pb 后缀）所在路径。
	inputs：模型输入节点（层名），支持多个输入节点。所有输入节点名放在一个列表中。
	input_size_list：每个输入节点对应的图片的尺寸和通道数。如示例中的 mobilenet-v1 模型，其输入节点对应的输入尺寸是[[1, 224, 224, 3]]。
	outputs：模型的输出节点（操作数名），支持多个输出节点。所有输出节点名放在一个列表中。
	predef_file：为了支持一些控制逻辑，需要提供一个 npz 格式的预定义文件。可以通过以下方法生成预定义文件：np.savez('prd.npz', [placeholder name]=prd_value)。如果“placeholder name”中包含'/'，请用'#'替换。目前版本该参数暂不支持。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 ssd_mobilenet_v1_coco_2017_11_17 模型
ret = rknn.load_tensorflow(
        tf_pb='./ssd_mobilenet_v1_coco_2017_11_17.pb',
```

```
            inputs=['FeatureExtractor/MobilenetV1/MobilenetV1/Conv2d_0
                    /BatchNorm/batchnorm/mul_1'],
            outputs=['concat', 'concat_1'],
            input_size_list=[[1, 300, 300, 3]])
```

3.5.3.3 TensorFlow Lite 模型加载接口

API	**load_tflite**
描述	加载 TensorFlow Lite 模型。
参数	model：TensorFlow Lite 模型文件（.tflite 后缀）所在路径。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 mobilenet_v1 模型
ret = rknn.load_tflite(model = './mobilenet_v1.tflite')
```

3.5.3.4 ONNX 模型加载

API	**load_onnx**
描述	加载 ONNX 模型。
参数	model：ONNX 模型文件（.onnx 后缀）所在路径。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 arcface 模型
ret = rknn.load_onnx(model = './arcface.onnx')
```

3.5.3.5 DarkNet 模型加载接口

API	**load_darknet**

描述	加载 DarkNet 模型。
参数	model：DarkNet 模型文件（.cfg 后缀）所在路径。
	weight：权重文件（.weights 后缀）所在路径。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 yolov3-tiny 模型
ret = rknn.load_darknet(model = './yolov3-tiny.cfg',
                        weight='./yolov3.weights')
```

3.5.3.6 PyTorch 模型加载接口

API	**load_pytorch**
描述	加载 PyTorch 模型。
参数	model：PyTorch 模型文件（.pt 后缀）所在路径，而且需要是 torchscript 格式的模型。必填参数。
	input_size_list：每个输入节点对应的图片的尺寸和通道数。例如[[1,1,224,224],[1,3,224,224]]表示有两个输入，其中一个输入的 shape 是[1,1,224,224]，另外一个输入的 shape 是[1,3,224,224]。必填参数。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 resnet18 模型
ret = rknn.load_pytorch(model = './resnet18.pt',
                        input_size_list=[[1,3,224,224]])
```

3.5.3.7 MXNet 模型加载接口

目前版本该功能暂不支持。

API	**load_mxnet**
描述	加载 MXNet 模型。
参数	symbol：MXNet 模型的网络结构文件，后缀是 json。必填参数。
	params：MXNet 模型的参数文件，后缀是 params。必填参数。
	input_size_list：每个输入节点对应的图片的尺寸和通道数。例如[[1,1,224,224],[1,3,224,224]]表示有两个输入，其中一个输入的 shape 是[1,1,224,224]，另外一个输入的 shape 是[1,3,224,224]。必填参数。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 resnext50 模型
ret = rknn.load_mxnet(symbol='resnext50_32x4d-symbol.json',
                      params='resnext50_32x4d-4ecf62e2.params',
                      input_size_list=[[1,3,224,224]] )
```

3.5.4 构建 RKNN 模型

API	**build**
描述	依照加载的模型结构及权重数据，构建对应的 RKNN 模型。
参数	do_quantization：是否对模型进行量化，值为 True 或 False。
	dataset：量化校正数据的数据集。目前支持文本文件格式，用户可以把用于校正的图片（jpg 或 png 格式）或 npy 文件路径放到一个.txt 文件中。文本文件里每一行一条路径信息。如： a.jpg

	b.jpg 或 a.npy b.npy 如有多个输入，则每个输入对应的文件用空格隔开，如： a.jpg a2.jpg b.jpg b2.jpg 或 a.npy a2.npy b.npy b2.npy 注：量化图片建议选择与预测场景较吻合的图片。
	rknn_batch_size：模型的输入 Batch 参数调整，默认值为 1。如果大于 1，则可以在一次推理中同时推理多帧输入图像或输入数据，如 MobileNet 模型的原始 input 维度为[1, 224, 224, 3]，output 维度为[1, 1001]，当 rknn_batch_size 设为 4 时，input 的维度变为[4, 224, 224, 3]，output 维度变为[4, 1001]。 **注：** **1. rknn_batch_size 的调整并不会提高一般模型在 NPU 上的执行性能，但却会显著增加内存消耗以及增加单帧的延迟。** **2. rknn_batch_size 的调整可以降低超小模型在 CPU 上的消耗，提高超小模型的平均帧率。（适用于模型太小，CPU 的开销大于 NPU 的开销）。** **3. rknn_batch_size 的值建议小于 32，避免内存占用太大而导致推理失败。** **4. rknn_batch_size 修改后，模型的 input/output 维度会被修改，使用 inference 推理模型时需要设置相应的 input 的大小，后处理时，也需要对返回的 outputs 进行处理。**
返回值	0：构建成功
	-1：构建失败

举例如下：

```
# 构建 RKNN 模型，并且做量化
ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
```

3.5.5 导出 RKNN 模型

通过本工具构建的 RKNN 模型通过该接口可以导出存储为 RKNN 模型文件，用于模型部署。

API	**export_rknn**
描述	将 RKNN 模型保存到指定文件中（.rknn 后缀）。
参数	export_path：导出模型文件的路径。
返回值	0：导出成功
	-1：导出失败

举例如下：

```
……
# 将构建好的 RKNN 模型保存到当前路径的 mobilenet_v1.rknn 文件中
ret = rknn.export_rknn(export_path = './mobilenet_v1.rknn')
……
```

3.5.6 加载 RKNN 模型

API	**load_rknn**
描述	加载 RKNN 模型。加载后的模型仅限于连接 NPU 硬件进行推理或获取性能数据等。不能用于模拟器或精度分析等。
参数	path：RKNN 模型文件路径。
	load_model_in_npu：是否直接加载 npu 中的 rknn 模型。其中 path 为 rknn 模型在 npu 中的路径。只有当 RKNN-Toolkit2 运行在连有 NPU 设备的 PC 上时才可以设为 True。默认值为 False。目前版本该参数暂不支持。
返回值	0：加载成功
	-1：加载失败

举例如下：

```
# 从当前路径加载 mobilenet_v1.rknn 模型
ret = rknn.load_rknn(path='./mobilenet_v1.rknn')
```

3.5.7 初始化运行时环境

在模型推理或性能评估之前，必须先初始化运行时环境，明确模型的运行平台（具体的目标硬件平台或软件模拟器）。

API	**init_runtime**
描述	初始化运行时环境。确定模型运行的设备信息（硬件平台信息、设备 ID）；性能评估时是否启用 debug 模式，以获取更详细的性能信息。
参数	target：目标硬件平台，支持“rk3566”/“rk3568”。默认为 None，即在 PC 使用工具时，模型在模拟器上运行。
	device_id：设备编号，如果 PC 连接多台设备时，需要指定该参数，设备编号可以通过“***list_devices***”接口查看。默认值为 None。
	perf_debug：进行性能评估时是否开启 debug 模式。在 debug 模式下，可以获取到每一层的运行时间，否则只能获取模型运行的总时间。默认值为 False。
	eval_mem：是否进入内存评估模式。进入内存评估模式后，可以调用 eval_memory 接口获取模型运行时的内存使用情况。默认值为 False。
	async_mode：是否使用异步模式。调用推理接口时，涉及设置输入图片、模型推理、获取推理结果三个阶段。如果开启了异步模式，设置当前帧的输入将与推理上一帧同时进行，所以除第一帧外，之后的每一帧都可以隐藏设置输入的时间，从而提升性能。在异步模式下，每次返回的推理结果都是上一帧的。该参数的默认值为 False。 目前版本该参数暂不支持。
返回值	0：初始化运行时环境成功
	-1：初始化运行时环境失败

举例如下：

```
# 初始化运行时环境
ret = rknn.init_runtime(target='rk3566', device_id='012345789AB')
if ret != 0:
    print('Init runtime environment failed')
    exit(ret)
```

3.5.8 模型推理

在进行模型推理前，必须先构建或加载一个 RKNN 模型。

API	**inference**
描述	对当前模型进行推理，返回推理结果。 如果 RKNN-Toolkit2 运行在 PC 上，且初始化运行环境时设置 target 为 Rockchip NPU 设备，得到的是模型在硬件平台上的推理结果。 如果 RKNN-Toolkit2 运行在 PC 上，且初始化运行环境时没有设置 target，得到的是模型在模拟器上的推理结果。
参数	inputs：待推理的输入，如经过 cv2 处理的图片。格式是 ndarray list。
	data_format：数据模式，可以填以下值：“nchw”，“nhwc”。默认值为“nhwc”。
	inputs_pass_through：将输入透传给 NPU 驱动。非透传模式下，在将输入传给 NPU 驱动之前，工具会对输入进行减均值、除方差等操作；而透传模式下，不会做这些操作。这个参数的值是一个数组，比如要透传 input0，不透传 input1，则这个参数的值为[1, 0]。默认值为 None，即对所有输入都不透传。
返回值	results：推理结果，类型是 ndarray list。

举例如下：

对于分类模型，如 mobilenet_v1，代码如下（完整代码参考 example/tflite/mobilent_v1）：

```
# 使用模型对图片进行推理，得到 TOP5 结果
……
outputs = rknn.inference(inputs=[img])
show_outputs(outputs)
……
```

输出的 TOP5 结果如下：

```
-----TOP 5-----
[156]: 0.85107421875
[155]: 0.09173583984375
[205]: 0.013587951660156 25
[284]: 0.006465911865234375
[194]: 0.002239227294921875
```

对于目标检测的模型，如 ssd_mobilenet_v1，代码如下（完整代码参考 example/tensorflow/ssd_mobilenet_v1）：

```
# 使用模型对图片进行推理，得到目标检测结果
……
outputs = rknn.inference(inputs=[image])
……
```

输出的结果经过后处理后输出如下图片（物体边框的颜色是随机生成的，所以每次运行这个 example 得到的边框颜色会有所不同）：

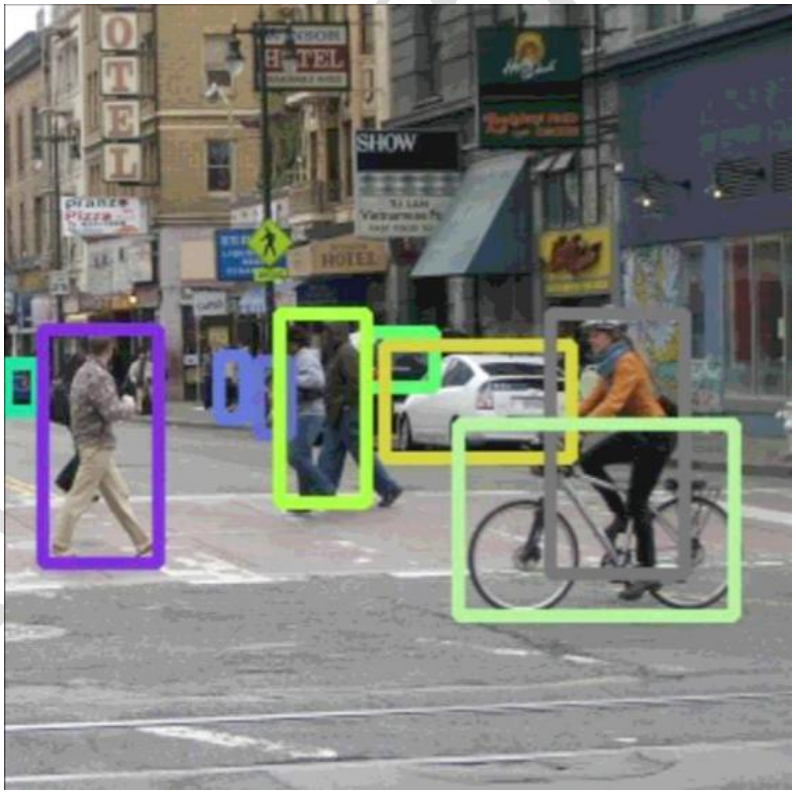

图 3-5-8-1 ssd_mobilenet_v1 inference 结果

3.5.9 评估模型性能

API	**eval_perf**
描述	评估模型性能。 模型必须运行在与 PC 连接的 RK3566 / RK3568 上。如果初始化运行环境时设置 perf_debug 为 False，则获得的是模型在硬件上运行的总时间；如果设置 perf_debug 为 True，除了返回总时间外，还将返回每一层的耗时情况。

返回值	perf_result：性能信息。类型为字典。在硬件平台上运行，且初始运行环境时设置 perf_debug 为 False 时，得到的字典只有一个字段‘total_time’，示例如下：
	{
	‘total_time’: 1000
	}
	其他场景下，得到的性能信息字典多一个‘layers’字段，这个字段的值也是一个字典，这个字典以每一层的 ID 作为 key，其值是一个包含‘name’（层名）、‘operation’（操作符）、‘target’（该层的执行设备），‘time’（该层耗时）等信息的字典。
	举例如下：
	{
	'total_time', 13870,
	'layers', {
	'0': {
	'name': 'convolution0',
	'operation': 'ConvRelu',
	'target': 'NPU',
	'time': 362
	}
	'1': {
	'name': "convolution1",
	'operation': 'ConvRelu',
	'target': 'NPU',
	'time': 158
	}
	}
	}

举例如下：

```
# 对模型性能进行评估
```

```
……
rknn.eval_perf(inputs=[image], is_print=True)
……
```

如 example/tflite/mobilenet_v1，其性能评估结果打印如下（不同版本的工具结果可能略有不同）：

```
==============================================================================
                                Performance
        #### The performance result is just for debugging, ####
        #### may worse than actual performance!              ####
==============================================================================
Layer ID    Name                                                    Operator          Target    Time(us)
0           InputOperator:input                                     InputOperator     CPU       14
1           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_0/Relu6         ConvClip          NPU       316
2           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_1_depthwise/Relu6   ConvClip     NPU       329
3           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_1_pointwise/Relu6   ConvClip     NPU       510
4           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_2_depthwise/Relu6   ConvClip     NPU       324
5           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_2_pointwise/Relu6   ConvClip     NPU       192
6           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_3_depthwise/Relu6   ConvClip     NPU       233
7           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_3_pointwise/Relu6   ConvClip     NPU       227
8           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_4_depthwise/Relu6   ConvClip     NPU       143
9           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_4_pointwise/Relu6   ConvClip     NPU       132
10          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_5_depthwise/Relu6   ConvClip     NPU       142
11          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_5_pointwise/Relu6   ConvClip     NPU       193
12          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_6_depthwise/Relu6   ConvClip     NPU       71
13          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_6_pointwise/Relu6   ConvClip     NPU       99
14          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_7_depthwise/Relu6   ConvClip     NPU       79
15          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_7_pointwise/Relu6   ConvClip     NPU       171
16          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_8_depthwise/Relu6   ConvClip     NPU       78
17          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_8_pointwise/Relu6   ConvClip     NPU       196
18          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_9_depthwise/Relu6   ConvClip     NPU       78
19          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_9_pointwise/Relu6   ConvClip     NPU       195
20          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_10_depthwise/Relu6  ConvClip     NPU       79
21          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_10_pointwise/Relu6  ConvClip     NPU       170
22          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_11_depthwise/Relu6  ConvClip     NPU       78
23          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_11_pointwise/Relu6  ConvClip     NPU       170
24          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_12_depthwise/Relu6  ConvClip     NPU       62
25          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_12_pointwise/Relu6  ConvClip     NPU       232
26          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_13_depthwise/Relu6  ConvClip     NPU       169
27          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_13_pointwise/Relu6  ConvClip     NPU       494
28          Conv:MobilenetV1/Logits/AvgPool_1a/AvgPool              Conv              NPU       182
29          Conv:MobilenetV1/Logits/Conv2d_1c_1x1/BiasAdd           Conv              NPU       206
30          Softmax:MobilenetV1/Predictions/Reshape_1               Softmax           CPU       335
31          Reshape:MobilenetV1/Logits/SpatialSqueeze               Reshape           CPU       99
32          OutputOperator:MobilenetV1/Predictions/Reshape_1        OutputOperator    CPU       40
Total Time(us): 6038
FPS: 165.62
==============================================================================
```

3.5.10 获取内存使用情况

API	**eval_memory**
描述	获取模型在硬件平台运行时的内存使用情况。 模型必须运行在与 PC 连接的 RK3566 / RK3568 上。
参数	is_print：是否以规范格式打印内存使用情况，默认值为 True。

返回值	memory_detail：内存使用情况，类型为字典。
	内存使用情况按照下面的格式封装在字典中：
	{
	'total_weight_allocation': 4312608
	'total_internal_allocation': 1756160,
	'total_model_allocation': 6068768
	}
	● 'total_weight_allocation' 字段表示模型中权重的内存占用。
	● 'total_internal_allocation' 字段表示模型中中间 tensor 内存占用。
	● 'total_model_allocation' 表示模型的内存占用，即权重和中间 tensor 的内存占用之和。

举例如下：

```
# 对模型内存使用情况进行评估
……
memory_detail = rknn.eval_memory()
……
```

如 example/tflite 中的 mobilenet_v1，它在 RK3566 上运行时内存占用情况如下：

```
==============================================================
                Memory Profile Info Dump
==============================================================
NPU model memory detail(bytes):
    Total Weight Memory: 4.11 MiB
    Total Internal Tensor Memory: 1.67 MiB
    Total Memory: 5.79 MiB

INFO: When evaluating memory usage, we need consider
the size of model, current model size is: 4.33 MiB
==============================================================
```

3.5.11 查询 SDK 版本

API	**get_sdk_version**
描述	获取 SDK API 和驱动的版本号。 **注：使用该接口前必须完成模型加载和初始化运行环境，且该接口只能在硬件平台 RK3566 / RK3568 上使用。**
参数	无
返回值	sdk_version：API 和驱动版本信息，类型为字符串。

举例如下：

```
# 获取 SDK 版本信息
……
sdk_version = rknn.get_sdk_version()
……
```

返回的 SDK 信息如下：

```
=============================================
RKNN VERSION:
  RKNNAPI:      API: 1.6.1 (de5c7ec build: 2021-04-25 10:21:45)
  RKNNAPI:      DRV: 1.6.1 (de5c7ec build: 2021-04-25 10:14:11)
=============================================
```

3.5.12 混合量化

3.5.12.1 hybrid_quantization_step1

使用混合量化功能时，第一阶段调用的主要接口是 hybrid_quantization_step1，用于生成临时模型文件（{model_name}.model）、数据文件（{model_name}.data）和量化配置文件（{model_name}.quantization.cfg）。接口详情如下：

API	**hybrid_quantization_step1**
描述	根据加载的原始模型，生成对应的临时模型文件、配置文件和量化配置文件。

参数	dataset：量化校正数据的数据集。目前支持文本文件格式，用户可以把用于校正的图片（jpg 或 png 格式）或 npy 文件路径放到一个.txt 文件中。文本文件里每一行一条路径信息，如： a.jpg b.jpg 或 a.npy b.npy
	proposal：产生混合量化的配置建议值。
	proposal_dataset_size：proposal 使用的 dataset 的张数。因为 proposal 功能比较耗时，所以默认只使用 1 张，也就是 dataset 里的第一张。
返回值	0：成功
	-1：失败

举例如下：

```
# Call hybrid_quantization_step1 to generate quantization config
……
ret = rknn.hybrid_quantization_step1(dataset='./dataset.txt')
……
```

3.5.12.2 hybrid_quantization_step2

用于使用混合量化功能时生成 RKNN 模型，接口详情如下：

API	**hybrid_quantization_step2**
描述	接收临时模型文件、配置文件、量化配置文件和校正数据集作为输入，生成混合量化后的 RKNN 模型。
参数	model_input：第一步生成的临时模型文件，形如“{model_name}.model”。数据类型为字符串，必填参数。

	data_input：第一步生成的配置文件，形如“{model_name}.data”。数据类型为字符串，必填参数。
	model_quantization_cfg：经过修改后的模型量化配置文件，形如“{model_name}.quantization.cfg”。数据类型为字符串，必填参数。
返回值	0：成功
	-1：失败

举例如下：

```
# Call hybrid_quantization_step2 to generate hybrid quantized RKNN model
……
ret = rknn.hybrid_quantization_step2(
        model_input='./ssd_mobilenet_v2.model',
        data_input='./ssd_mobilenet_v2.data',
        model_quantization_cfg='./ssd_mobilenet_v2.quantization.cfg',
        )
……
```

3.5.13 量化精度分析

该接口的功能是进行浮点、量化推理并产生每层的数据，用于量化精度分析。

API	**accuracy_analysis**
描述	推理并产生快照，也就是 dump 出每一层的 tensor 数据。会 dump 出包括 fp32 和 quant 两种数据类型的快照，用于计算量化误差。 **注：** 1. **该接口只能在 build 或 hybrid_quantization_step2 之后调用，并且原始模型应该为非量化的模型，否则会调用失败。** 2. **该接口使用的量化方式与 config 中指定的一致。**
参数	inputs：图像（jpg / png / bmp / npy 等）路径 list。
	output_dir：输出目录，所有快照都保存在该目录（默认输出目录名为‘snapshot’）。 如果没有设置 target，在 output_dir 下会输出： ● simulator 目录：保存整个量化模型在 simulator 上完整运行时每一层的结果（已

	转成 float32）； ● golden 目录：保存整个浮点模型在 simulator 上完整跑下来时每一层的结果; ● error_analysis.txt：记录在 simulator 上量化模型逐层运行时每一层的结果与浮点模型逐层运行时每一层的结果的余弦距离（entire_error cosine），以及量化模型取上一层的浮点结果作为输入时，输出与浮点模型的余弦距离（per_layer_error cosine） 如果有设置 target，则在 output_dir 里还会多输出： ● runtime 目录：保存整个量化模型在 NPU 上完整运行时每一层的结果（已转成 float32） ● error_analysis.txt：在上述记录的内容的基础上，还会记录量化模型在 simulator 上逐层运行时每一层的结果与 NPU 上逐层运行时每一层的结果的余弦距离（entire_error cosine）
	target：目标硬件平台，支持“rk3566”/“rk3568”，默认为 None。 如果设置了 target，则会获取 NPU 运行时每一层的结果，并进行精度的分析。
	device_id：设备编号，如果 PC 连接多台设备时，需要指定该参数，设备编号可以通过“list_devices”接口查看。默认值为 None。
返回值	0：成功
	-1：失败

举例如下：

```
……

    # Create RKNN object
    rknn = RKNN(verbose=True)

    print('--> config model')
    rknn.config(mean_values=[128, 128, 128], std_values=[128, 128, 128], )
    print('done')

    # Load model
    print('--> Loading model')
    ret = rknn.load_tensorflow(tf_pb='mobilenet_v1.pb',
```

```
                                    inputs=['input'],
                                    outputs=['MobilenetV1/Logits/SpatialSqueeze'],
                                    input_size_list=[[1, 224, 224, 3]])
    if ret != 0:
        print('Load mobilenet_v1 failed!')
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=True, dataset='dataset.txt')
    if ret != 0:
        print('build mobilenet_v1 failed!')
        exit(ret)
    print('done')

    print('--> Accuracy analysis')
    rknn.accuracy_analysis(inputs=['./dog_224x224.jpg'])

……
```

3.5.14 注册自定义算子

目前版本该功能暂不支持。

3.5.15 获取设备列表

API	**list_devices**
描述	列出已连接的 RK3566 / RK3568。 注：目前设备连接模式有两种：ADB 和 NTB。其中 RK3566 / RK3568 支持 ADB/NTB 模式。多设备连接时请确保他们的模式都是一样的。
参数	无
返回值	返回 adb_devices 列表和 ntb_devices 列表，如果设备为空，则返回空列表。

举例如下：

```
from rknn.api import RKNN

if __name__ == '__main__':
    rknn = RKNN()
    rknn.list_devices()
```

```
    rknn.release()
```

返回的设备列表信息如下（这里有两个 RK1808 开发板，它们的连接模式都是 adb）：

```
**************************
all device(s) with adb mode:
VD46C3KM6N
**************************
```

注：使用多设备时，需要保证它们的连接模式都是一致的，否则会引起冲突，导致设备连接失败。

3.6 精度问题排查

模型精度问题排除一般从两个方面进行排查，一方面是 PC 仿真精度排查，另一方面是板端运行时精度排查。PC 仿真结果正确是板端运行时正确的前提，所以推荐用户在处理 rknn 模型精度问题时，优先保证 PC 仿真结果正确，再进行板端运行时精度问题排查。因此我们将针对 PC 仿真精度排查以及板端运行时精度排查两个方面给出精度问题排查的建议以及处理方案。

另外，判断精度可以简单的使用余弦距离作为基本的判断，但这不等于最终的模型量化精度，因此仅仅作为参考，判断精度情况还是得最终跑数据集进行验证。当然，在下述精度问题排查过程中，可以先简单使用余弦距离来作为精度是否有改善的依据。

3.6.1 PC 仿真精度排查

PC 仿真结果的正确是板端模型推理正确的前提，因此需要确保 PC 上 simulator 推理正确。rknn-toolkit2 提供了模型是否进行量化的选择，因此本部分分别对“fp16 模型”以及“量化模型”进行精度分析。由于“fp16 模型”的结果正确是“量化模型”精度正确的前提，因此当存在“量化模型”精度问题时，一般推荐用户能够优先验证“fp16 模型”的正确性。下面将分别对“fp16 模型”以及“量化模型”的精度问题的排查策略进行详细阐述。

3.6.1.1 “fp16 模型”精度问题排查

“fp16 模型”的结果正确是保证后续“量化模型”精度正确的前提，用户只需要在使用 rknn

的 build 接口时，将 do_quantization 参数设置为 False，即可以将原始模型转换为“fp16 模型”。如果“fp16 模型”输出结果错误，则需进行以下排查：

1）配置问题

模型的配置信息主要集中在 rknn 的 config 这个接口里，同时在其他 rknn 的 API 里也有少数的配置信息，但并不是每个配置信息都会引起精度问题，主要会引起“fp16 模型”精度问题的参数如下：

mean_values / std_values：模型的归一化参数，这边要确保其和原始模型使用的参数相同。

input_size_list：load_tensorflow / load_pytorch 的输入节点 shape 信息，如果配置错误也会导致错误的推理结果。

inputs / outputs：load_tensorflow 的输入和输出节点名称，同样如果配置错误也会导致错误的推理结果。

inference 接口参数：rknn 的 inference 接口的输入参数，主要包括 inputs 和 data_format。一般在 python 环境下，图像数据都是通过 cv2.imread 读取的，此时需要注意是 cv2.imread 读取的图像格式为 BGR，如果原始模型的输入为 BGR（如大部分的 caffe 模型），则可以直接将读取的图像数据传给 rknn 的 inference 接口进行推理；而如果原始模型的输入为 RGB，则还需要调用 cv2.cvtColor(img, cv2.COLOR_BGR2RGB)将图像数据转为 RGB，才可以传给 rknn 的 inference 接口进行推理。另外，通过 cv2.imread 读取的图像数据的 layout 为 NHWC，因为 data_format 的默认值为 NHWC，因此不需要设置 data_format 参数。而如果模型的输入数据不是通过 cv2.imread 读取，此时用户就必须清楚知道输入数据的 layout 并设置正确的 data_format 参数，如果是图像数据，也要保证其 RGB 顺序与模型的输入 RGB 顺序一致。

参数配置的检查是很重要的环节，是很多用户发现“fp16 模型”输出结果错误的主要原因。具体步骤如下：

a. 使用原始模型在原始模型的推理框架下进行推理，如 caffe 模型就使用 caffe_bvlc 或 opencv_caffe 来进行推理，pytorch 模型就使用 pytorch 推理框架进行推理，pb 和 tflite 就使用 tensorflow 进行推理，onnx 使用 onnxruntime 进行推理等，然后将推理结果保存下来。

b. 使用原始模型在 rknn-toolkit2 的推理框架下进行推理，这边需要使用与前一步骤里同样的

输入数据，并设置 fp16 的推理方式（rknn 的 build 的 do_quantization 设为 False），同时 init_runtime 的 target 参数不要配置或设为 None，此时使用的 rknn-toolkit2 内部的 simulator 来进行推理，同样将推理的结果保存下来。

c. 对比两次推理的结果，如果结果较为一致（可以用余弦距离来判断一致性），说明上述的配置都没有问题。

d. 如果结果不一致，检查上述参数是否正确。

如果确认上述参数配置无误，结果仍然不一致，则可能是仿真端内部实现问题。

2）仿真端的内部实现问题

这部分可能与仿真端的内部 bug 有关，但是出现概率很低。一般推荐用户通过以下两种方式进行排查。

一种是在构建 rknn 对象时设置 verbose 参数为'Debug'，则打开 rknn-toolkit2 的 debug 模式，再次运行会输出 rknn 模型构建过程中的精度检查日志，根据输出日志里的“check results”的结果（余弦相似度与欧氏距离），可以判定是哪一个步骤出现问题。如果确定了该问题，最好能够将复现该结果的模型连同日志提供给瑞芯微 NPU 团队进行分析解决。这种方法使用的是 rknn-toolkit2 内部提供的接口进行错误检查，一般适用于快速定位问题，如果没有相关“check results”日志输出或仍然无法定位问题，也可以使用下一种方式。

另一种方式的前提条件是用户能获得原始模型在原始框架下的每一层输出的真值，这时可以使用精度分析接口 dump 出浮点模型的每一层 golden 结果（dump 出的 golden 结果是 rknn-toolkit2 精度分析参考的模型“真值”），并与原始框架下的模型每层输出结果进行余弦相似度对比。如果 golden 结果与原始框架下的模型首层输出结果都无法对齐（一般认为余弦相似度低于 0.99 存在少许不一致，低于 0.98 几乎可以认为该层结果就是错误的），则可能还是上一步参数配置有误，需要返回上一步进行重新排查。如果首层输出结果一致，但中间层或者最终结果不一致，则可能是 PC 仿真实现 bug 导致，此时用户具体定位到是哪一层开始出现的不一致，可以将围绕这一层对模型进行截取，并将该模型的复现文件提供瑞芯微 NPU 团队进行分析解决。

3.6.1.2 “量化模型”精度问题排查

在经过“fp16 模型”的精度验证后，排除了“fp16 模型”的错误可能，就可以对模型进行量化，并进一步对“量化模型”进行精度分析。如果在“量化模型”遇到精度问题，将主要从以下几个方面进行排查：

3.6.1.2.1 配置导致

与“fp16 模型”的配置类似，配置错误也会导致“量化模型”精度问题，在保证“fp16 模型”正确的配置基础上，仍然要对以下参数配置进行检查。

quantized_dtype：量化类型的选择，不同的量化类型的精度差异非常大，同时运行时性能的差异也非常大，一般选择折中的量化类型，如 asymmetric_quantized-8。如果选择了 asymmetric_quantized-4，可以达到最佳的运行时性能，但精度也最差，因此只适合少数对 4-bit 量化不敏感的模型使用。如果选择了 asymmetric_quantized-16，可以达到接近原始模型的精度，但运行时的性能会比较差，所以只适合用在对运行时性能不敏感，但对精度要求非常高的场景下。但是，因为一般在 NPU 中，16-bit 量化和非量化（float16）的运算性能差异不大，因此建议选择 fp16（rknn 的 build 接口的 do_quantization 设为 False）的运算方式来代替 16-bit 量化。（asymmetric_quantized-16 目前版本暂不支持，asymmetric_quantized-4 在 rk356x 下也不支持）

quant_img_RGB2BGR：表示在加载量化图像时是否需要先做 RGB2BGR 的操作，一般用于 caffe 模型，更多详细信息见 quant_img_RGB2BGR 参数说明，该参数配置错误也会导致量化精度大降。

dataset：rknn 的 build 接口的量化校正集配置。如果选择了和实际部署场景不大一致的校正集，则可能会出现精度下降的问题，或者校正集的数量过多或过少都会影响精度（一般选择 50～200 张）。

具体检查“量化模型”的参数配置问题，一般可以按如下步骤进行：

1) 直接进行“量化模型”推理，然后检查推理的结果与原始模型在原始推理框架下推理的结果进行比较，如果结果差异不是很大，则可以认为 quantized_dtype、quant_img_RGB2BGR 和 dataset 参数基本无误。

2) 如果结果差异还是较大：

   a. 如 quantized_dtype 配置为 4-bit 算法，则可以修改 quantized_dtype 为更高比特的量化算法；

   b. 如原始模型输入的图像格式是 BGR（多见于 caffe 的模型），此时可以修改 quant_img_RGB2BGR 为 True，关于模型输入的 RGB 顺序，其实从前面“fp16 模型”精度验证步骤的输入数据的处理代码中可以得知输入数据的 RGB 顺序。

   c. 可以先使用一张图像进行量化（dataset.txt 中只留一行），推理时也使用这张图像进行推理，如果此时单张图像的精度提升较多，则说明先前使用的量化校正集选择不佳，可以重新选择与部署场景较吻合的图片。

   d. 如原先只使用一张图像进行量化（dataset.txt 中只留一行），此时可以尝试使用更多的图像进行量化，可以提高到 50～200 张左右。

经过上述配置排查之后，有些模型的精度可能已经可以满足要求，有些模型可能精度还是不够，可以尝试下述章节里的方法（更改量化方法），但应该不会出现量化结果完全错误的情况，如果出现完全错误的情况，请重新检查上述的配置。

3.6.1.2.2 量化方法导致

有些模型本身对量化并不友好，这时可以尝试切换不同的量化方法和量化算法。目前量化方法主要有两种，分别是 Per-Layer / Per-Channel，对应于 rknn 的 config 接口里的 quantized_method 参数，量化算法主要也分为两种，分别是 Normal / MMSE，对应于 rknn 的 config 接口里的 quantized_algorithm 参数。步骤如下：

1) 如原先使用的是 Per-Layer 的量化方法，可以改为 Per-Channel 的量化方法，一般情况下，Per-Channel 的量化方法精度比 Per-Layer 的量化方法精度会高许多，但可能会带来执行效率的略微下降（可以忽略不计）

2) 如量化方法已改为 Per-Channel，但精度还是无法满足要求，此时可以将量化算法由 Normal 改为 MMSE，这种方式会导致量化的时间大幅增加，但会带来比 Normal 更好的精度表现。

同时运行时的性能并不会受到影响。

如果使用上述方法后，精度还是偏低，那可能是模型有些 Op 对现有的量化算法并不友好，在量化后会出现精度下降较多的情况。如：Conv 的 weight 的分布很不均匀的情况下，此时可以考虑使用混合量化来进一步改进模型精度，混合量化可以让模型中的不同 OP 使用不同的量化类型。步骤如下：

1) 先使用精度分析接口对精度进行分析，找出造成精度下降的层。
2) 使用混合量化的方法，并将怀疑的层的输出 tensor 的 name 写入混合量化的配置文件中。
3) 完成混合量化的步骤，并测试精度情况。（可以继续使用精度分析接口来看精度变化情况）

一般经过混合量化之后，模型的精度是可以提高的，如果提高不明显或不够，则可以尝试将更多的层进行混合量化，但是同时也会带来运行时速度的降低，因此混合量化需要用户自己权衡精度和速度。还有一种特殊方式是，当精度下降的 Op 处于最后一层时，也可以选择将该层 Op 的运行放在后处理中进行，同样会有效避免该层的精度问题。

至此，在排查上述的原因后，我们基本可以获得一个精度较优的量化模型，仿真端的精度问题排查就基本完成了。

3.6.2 运行时精度排查

在经过 PC 仿真端的“fp16 模型”精度验证以及“量化模型”精度验证之后，一般在模拟器上的量化精度应该都可以做到满足应用的需求，但用户可能会经常遇到在模拟器上的量化精度还不错，但是在板端通过 rknn 的 C API 编程进行推理测试时，却发现精度不够或根本不对的问题。出现这种问题的原因一般有两种，一种是调用 rknn 的 C API 编程的代码本身问题导致，例如输入数据不对、运行时参数配置不对或后处理代码错误等等；另外一种是板端的 runtime 的 bug 导致。当遇到这种问题时，我们可以先通过以下步骤来排查板端的 runtime 的问题：

1) 在配置好连板调试环境的情况下（环境配置方法在提供的 RKNN C API 压缩包中有详细说明），使用 rknn-toolkit2 转换 rknn 模型后设置 init_runtime 的 target 参数，如 target=’rk3566’，并将板子通过 usb 连接到 pc 上（参考 3.2.2 章节），然后进行连板推理并检查推理结果是

否正确。（因为 PC 仿真并没有严格模拟 NPU 硬件，所以结果可能与 PC 仿真并没有完全一致）

2) 如果步骤 1 里的推理结果与 PC 仿真结果差异较大，则可以初步确定板端的 runtime 运行该模型存在 bug，此时可以使用精度分析的接口进行 runtime 端的精度检查，只需设置 accuracy_analysis 的 target 参数即可，如 target='rk3566'，accuracy_analysis 调用完后会输出每层的精度分析结果，如果发现板端 runtime 的结果与仿真的量化真值之间有明显差异，此时可以将该分析结果以及复现的模型反馈给瑞芯微 NPU 团队进行修复。

如果上述验证没有问题，则问题出在用户调用 rknn 的 C API 进行编程的 C/C++代码本身，这时用户需要仔细检验下 rknn 的 C API 的接口配置等是否配置正确，以及模型的前处理和后处理流程是否正确（需要与 PC 仿真端的流程完全一致）。关于 rknn 的 C API 的使用与配置问题请查阅相关 rknn_api 的文档。