密级状态：绝密(　　)　　秘密(　　)　　内部(　　)　　公开(√)

RK3399Pro RKNN API 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：	V1.7.3
	作　　者：	HPC
	完成日期：	2022-08-20
	审　　核：	熊伟
	完成日期：	2022-08-20

瑞芯微电子股份有限公司

Fuzhou Rockchips Semiconductor Co . , Ltd

更新记录

版本	修改人	修改日期	修改说明	核定人
V0.9.1	杜坤明	2018-11-27	初始版本	卓鸿添
V0.9.2	杜坤明	2018-12-19	主要修改 input 和 output 的 API 定义	卓鸿添
V0.9.3	杜坤明	2019-01-24	增加 v0.9.1 到 v0.9.2 的 API 迁移说明	卓鸿添
V0.9.4	杜坤明	2019-03-11	修复 channel_mean 没有生效的问题	卓鸿添
V0.9.6	杜坤明	2019-05-14	增加 rknn_init2 函数	卓鸿添
V0.9.7	杜坤明	2019-06-13	增加 x86 linux 的版本	卓鸿添
V0.9.8	杜坤明	2019-06-26	1. 更新 Linux X86 Demo 章节 2. 增加 rknn_batch_size>1 的支持。 3. 加入设备 ID 列表查询功能	卓鸿添
V0.9.9	杜坤明	2019-07-16	1. 增加多 input 的支持 2. 修复输入通道数大于 3 时导致推理错误的问题 3. 更改文档名称	卓鸿添
V1.2.0	杜坤明	2019-09-17	统一版本号到 V1.2.0	卓鸿添
V1.3.0	杜坤明	2019-11-27	1. 增加 multi-channels-mean 支持 2. 统一版本号到 V1.3.0	卓鸿添
V1.3.2	杜坤明	2020-04-02	1. 修复 API 版本号缺失问题 2. 更新多 Context 支持 3. 修复 int8/uint8/int16 的数据转换溢出问题 4. 为 fp16 转换加入 AVX 指令支持（x86） 5. 加速多线程中的模型加载 6. 更新版本号到 V1.3.2	卓鸿添
V1.3.3	杜坤明	2020-05-14	1. 增加 PX30 host 支持 2. 更新版本号到 V1.3.3	卓鸿添

版本	修改人	修改日期	修改说明	核定人
V1.4.0	杜坤明	2020-09-10	1. 增加 multi-scale 支持 2. 优化输入数据预处理 3. 更新版本号到 V1.4.0	卓鸿添
V1.6.0	杜坤明	2021-1-22	1. 增加 rknn_api.h 错误码 2. 更新版本号到 V1.6.0	熊伟
V1.6.1	HPC	2021-3-15	1. 修复输入数据预处理错误 2. 更新版本号到 V1.6.1	熊伟
V1.7.0	HPC	2021-8-5	1. 更新版本号到 V1.7.0	熊伟
V1.7.3	HPC	2022-08-20	1. 新增概述章节，简介 RKNN API 功能，适用硬件平台和系统依赖； 2. 更新 Example 使用说明； 3. 细化 RKNN API 使用流程； 4. 增加 NPU 固件更新说明。	熊伟

目 录

1 概述

1.1 主要功能说明

RKNN SDK 为带有 NPU 的 RK3399Pro 平台提供编程接口，帮助用户部署 RKNN Toolkit 导出的 RKNN 模型，加速 AI 应用的落地。

1.2 硬件平台

本文档适用如下硬件平台：

- RK3399Pro

1.3 系统依赖说明

1.3.1 Linux 平台

RKNN API SDK 适用于 Buildroot / Debian9 / Debian10 等系统。

1.3.2 Android 平台

RKNN API SDK 适用于 Android8.1 及以上系统使用。

2 RKNN SDK 工程简介

2.1 RKNN API 库

RKNN SDK 所提供的库和头文件位于<sdk>/rknn_api/librknn_api/目录下。发者可以在自己的应用中引用相应头文件和动态连接库即可通过 SDK 提供的接口使用 NPU。

需要注意的是 RK3399Pro 平台和 RK1808 平台所用的 RKNN C API 接口是兼容的，两者开发的应用程序可以很方便的移植。但是使用过程中，要注意区分两个平台的 librknn_api.so。如果错误使用不同平台的 librknn_api.so，可能导致应用无法正常运行。开发者可以通过以下方法来区分 librknn_api.so 对应的平台：

```
# RK1808 librknn_api.so 过滤 version 关键字时显示的是自身的版本号
$ strings librknn_api.so |grep version
librknn_api version 1.7.1 (2a83bcc build: 2021-12-02 09:46:02)

# RK3399Pro ARM64 librknn_api.so 在过滤 version 时显示的是 transfer 的版本
$ strings librknn_api.so |grep version
rknn_get_sdk_version
Transfer version 2.1.0 (b5861e7@2020-11-23T11:50:51)
.gnu.version
.gnu.version_r
# RK3399Pro 平台可以通过 rknn_query 接口查询 API 版本号，或者通过 rknn_api.h 头文件
#  查看版本号
```

注：RK1808 平台所用的 librknn_api.so 请从以下工程获取：https://github.com/rockchip-linux/rknpu。

2.2 Example 使用说明

SDK 提供了 Linux/Android 平台的 MobileNet 图像分类、MobileNet SSD 目标检测、YoloV5 目标检测、批量推理和输入透传相关的 C Demo，位于<sdk>/rknn_api/examples/c_demos 目录中。同时，SDK 还提供了 Android 平台的 rk_ssd_demo app，位于<sdk>/rknn_api/examples/android_apps/rk_ssd_demo/目录中。

2.2.1 C Demo 快速上手

以 rknn_mobilenet_demo 为例，该 demo 的编译、使用流程如下：

1. 编译 demo

```
cd <sdk>/rknn_api/examples/c_demos/rknn_mobilenet_demo
# 根据所用 RK3399Pro 的固件选择编译脚本
# 如果使用 Buildroot 或 Debian 等系统，请使用 build_linux.sh
# 如果使用 Android 系统，请使用 build_android.sh 脚本
# 对于 build_linux.sh 脚本，请修改脚本中 GCC_COMPILER，指向目标交叉编译器
# 对于 build_android.sh 脚本，请修改脚本中 ANDROID_NDK_PATH，指向 NDK 路径

./build_linux.sh
# 或
./build_android.sh
```

2. 部署到 RK3399Pro 设备

```
# Linux 固件
adb push install/rknn_mobilenet_demo_Linux /userdata

# Android 固件
adb shell root
adb shell remount
adb push install/rknn_mobilenet_demo_Android /usrdata/local
```

3. 运行 demo

```
# Linux 固件
cd /usrdata/rknn_mobilenet_demo_Linux

# Android 固件
cd /usrdata/local/rknn_mobilenet_demo_Android

./run_demo.sh
```

2.2.2 Android 应用快速上手

RKNN SDK 当前提供了一个针对摄像头视频流的目标检测应用 rk_ssd_demo。该应用将摄像头拍摄到的视频流显示在屏幕中，并将检测到的物体在视频中用方框标记。

该应用使用 ssd_inception_v2 作为目标检测模型，通过 JNI 的方式调用 RKNN C API 以使用

RK3399Pro 自带的 NPU 进行模型推理。

该应用的使用步骤如下：

1. 在 Android Studio 软件中加载该工程，调整工程中的 ndk 路径、Android Gradle Plugin Version（建议用 4.2.2）和 Gradle Version（建议用 6.7.1）等信息。
2. 编译 release 版 APK，具体的编译方法请参考 Android Studio 的相关文档。
3. 在 RK3399Pro 开发板上安装该应用，并在应用权限中打开该应用所需的摄像头、存储等权限。
4. 连接摄像头，打开 APP。

3 RKNN SDK 使用说明

3.1 调用流程

RKNN API 典型的调用流程如下图所示：

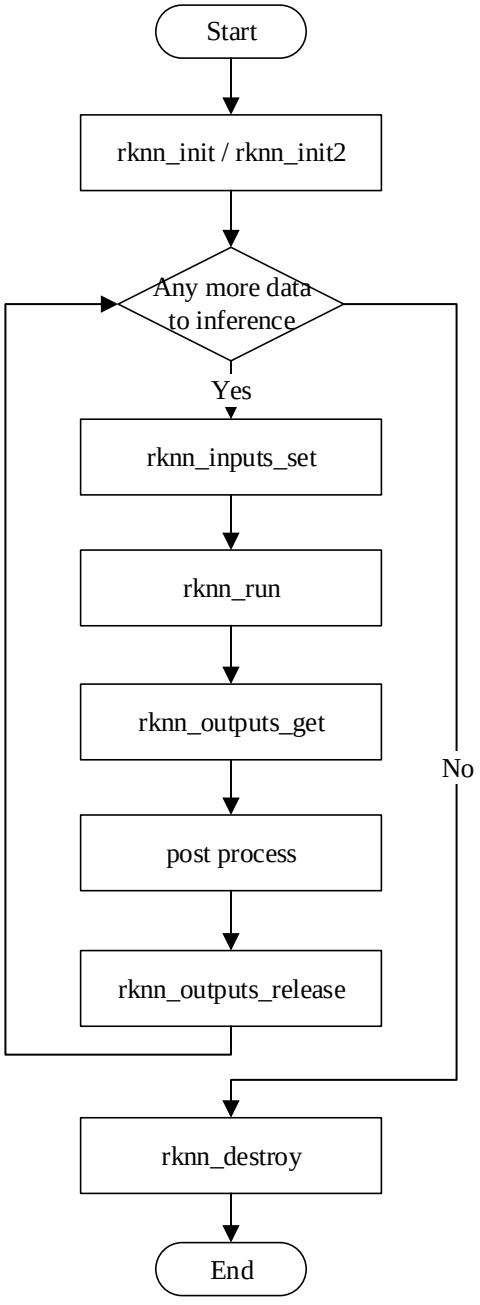

图 3-1 RKNN API 典型调用流程

该流程具体说明如下：

1. 将 RKNN 模型文件读到内存中，如 mobilenet_v1.rknn。

2. 调用 rknn_init 接口初始化上下文并将 RKNN 模型加载到 NPU 中。示例代码如下：

```
rknn_context ctx = 0;
ret = rknn_init(&ctx, model, model_len, RKNN_FLAG_PRIOR_MEDIUM);
if(ret < 0) {
    printf("rknn_init fail! ret=%d\n", ret);
    goto Error;
}
```

其中，ctx 为上下文对象；model 为 RKNN 模型在内存中的指针；model_len 为模型大小；RKNN_FLAG_PRIOR_MEDIUM 为优先级标志位。（其他标志位详见 rknn_init 或 rknn_init2 接口中的说明）

当 RK3399Pro 接有其他 NPU 设备（如 RK1808 计算棒）时，需要指定运行 RKNN 模型的设备 ID。此时需要使用 rknn_init2 接口，并在 extend 参数中指定设备 ID。

3. RKNN 模型的输入和输出节点属性（例如数据类型）可能和原始模型不同，因此需要通过 rknn_query 这个接口获取输入和输出节点的属性，示例代码如下：

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
if(ret < 0) {
    printf("rknn_query fail! ret=%d\n",ret);
    goto Error;
}
```

以上接口用于获取输入和输出的个数，存储在 io_num.n_input 和 io_num.n_output 中。

获取输出节点属性的示例代码如下：

```
rknn_tensor_attr output0_attr;
output0_attr.index = 0;
ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &output0_attr, sizeof(output0_attr));
if(ret < 0) {
    printf("rknn_query fail! ret=%d\n",ret);
    goto Error;
}
```

上述接口用于获取某个输出节点的属性，其中 rknn_tensor_attr 中的 index 属性一定要填写，且该属性的值不能大于等于前面查到的 output 个数。（该结构的详细定义请参考数据结构 rknn_tensor_attr 中的说明）

输入节点属性的获取方法和输出节点相似。

4. 根据 RKNN 模型的输入属性和输入数据的具体格式，调用 rknn_input_set 设置模型的输入。示例代码如下：

```
rknn_input inputs[1];
inputs[0].index = input_index;
inputs[0].buf = img.data;
inputs[0].size = img_width * img_height * img_channels;
inputs[0].pass_through = FALSE;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].fmt = RKNN_TENSOR_NHWC;
ret = rknn_inputs_set(ctx, 1, inputs);
if(ret < 0) {
      printf("rknn_input_set fail! ret=%d\n", ret);
      goto Error;
}
```

首先，创建一个 rknn_input 数组（示例代码假设模型只有一个输入），并依次填写数组中所有成员的属性值。

其中：

index：RKNN 模型输入节点的索引。

buf：CPU 可以访问的数据指针，例如由 Camera 产生的图像数据。

size：输入数据 buffer 的大小。

pass_through：输入数据是否直接传给 RKNN 模型。

TRUE： 如果应用程序传入的输入数据属性（主要是数据类型，数据排列方式，量化参数）和 rknn_query 接口查询得到的模型输入属性一致，则可以将该变量设为 TRUE（此时不需要设置 type 和 fmt）。在这种模式下，rknn_inputs_set 接口直接将应用程序传入的数据 Buffer 透传给 RKNN 模型的输入节点。这种模式用于已知 RKNN 模型的输入属性，且已经将原始输入数据做过相应的预处理。

FALSE： 如果应用程序传入的输入数据属性和 rknn_query 接口查询得到的模型输入属性不一致，则需要将该变量设为 FASLE，同时下面的 type 和 fmt 也需要根据应用程序传入的数据 Buffer 进行设置。在这种模式下，rknn_inputs_set 函数会自动进行数据类型，排列格式的转换以及量化（或反量化）的处理。注意，

目前这种模式下不支持用户传入使用动态定点量化(DFP)或非对称量化（AFFINE ASYMMETRIC）的输入数据。

type：输入数据的数据类型，例如 RGB888 的数据，其类型为 RKNN_TENSOR_UINT8。

fmt：输入数据的排列格式，NHWC 或 NCHW，一般 Camara 或者 OpenCV 获取的数据，其排列方式为 RKNN_TENSOR_NHWC。

5. 设置完输入数据后，调用 rknn_run 接口进行模型推理，该函数正常情况下会立即返回，并不会阻塞。但如果连续 3 次的推理结果都没有被应用程序通过 rknn_outputs_get 获取时，该接口会阻塞，直至 rknn_outputs_get 被调用。示例代码如下：

```
ret = rknn_run(ctx, NULL);
if(ret < 0) {
    printf("rknn_run fail! ret=%d\n", ret);
    goto Error;
}
```

6. 执行完 rknn_run,应用程序应该调用 rknn_outputs_get 接口等待推理完成，该函数会阻塞直到推理完成，推理完成后可以获取推理的结果。示例代码如下：

```
rknn_output outputs[1];
outputs[0].want_float = TRUE;
outputs[0].is_prealloc = FALSE;
ret = rknn_outputs_get(ctx, 1, outputs, NULL);
if(ret < 0) {
    printf("rknn_outputs_get fail! ret=%d\n", ret);
    goto Error;
}
```

首先，先创建 rknn_output 数组（示例代码假设模只有一个输出）。rknn_output 结构体中的 want_float 和 is_prealloc 这两个成员变量必须赋值。

want_float：由于 RKNN 模型的输出可能与原始模型的输出属性不一致。通常情况下，RKNN 模型输出节点的数据类型为 UINT8 或 FP16,如果用户希望获得的是 FP32 的浮点数据，则可以将该属性置为 TRUE；如果希望获得的是 RKNN 模型的原始输出数据，则置为 FALSE 即可。

is_prealloc：如果应用程序没有为输出数据分配内存，需要将该预分配标志设为 FALSE，此时 outputs[0]结构体中的其余成员变量不需要赋值，NPU 的运行时组件会自动为所有输出节点分配内存，并将内存地址填写在 outputs[0]的 buf 属性中。

index：对应输出节点的索引。

buf：输出节点数据 Buffer 的地址。如果输出数据的内存由应用程序分配，则应该将内存的地址填写在该属性中。否则不需要填写，NPU 运行时组件会将内部分配的输出数据 Buffer 地址填写在该属性中。

size：输出节点数据 Buffer 的大小。

输出节点的其他属性可以通过 rknn_query 查询得到。此处需要注意的是，如果输出数据的内存是由 NPU 运行时组件自动分配的，则需要调用 rknn_outputs_release 接口释放相应内存，如果该内存由应用程序自行分配，则应用程序需要自行释放以避免内存泄漏。

输出数据内存由应用程序自行释放时的示例代码如下：

```
rknn_output outputs[1];
outputs[0].want_float = TRUE;
outputs[0].is_prealloc = TRUE;
outputs[0].index = 0;
outputs[0].buf = output0_buf;
outputs[0].size = output0_attr.n_elems * sizeof(float);
ret = rknn_outputs_get(ctx, 1, outputs, NULL);
if(ret < 0) {
      printf("rknn_outputs_get fail! ret=%d\n", ret);
      goto Error;
}
```

注意，在应用程序分配内存时，需要根据输出节点的属性以及 want_float 的值来确定 Buffer 的大小。当 want_float 设为 FALSE 时，Buffer 的大小就是查询得到的输出属性的 size 值，否则该 Buffer 的大小等于 outputs0_attr.**n_elems * sizeof(float)**。

7. 使用 rknn_outputs_get 接口获取的输出数据不再需要使用时，请调用 rknn_outputs_release 接口释放相应数据，否则会造成内存泄漏。示例代码如下：

```
rknn_outputs_release(ctx, 1, outputs);
```

该函数的传参方式与 rknn_outputs_get 类似。

需要注意的是，不管 rknn_outputs_get 传入的 rknn_output[x].is_prealloc 是 TRUE 还是 FALSE 都需要调用该函数对 output 进行最终的释放。

8. 需要还有数据需要推理，可跳回步骤 4 进行下一次推理。

9. 程序退出前，需要调用 rknn_destroy 接口卸载 RKNN 模型并销毁上下文。示例代码如下：

```
rknn_destroy(ctx);
```

完整代码请参见 SDK/rknn_api/exampels/c_demos 或 android_apps 中各个示例 src 中的代码。

3.2 API 内部处理流程

在推理 RKNN 模型时，原始数据需要经过输入处理、NPU 推理、输出处理三大流程。在典型的图片推理场景中，假设输入数据 data 是 3 通道的图片且排列顺序为 NHWC，运行时（runtime）对数据处理的流程如图 3-2-1 所示。在 API 层面上，当输入的 pass_through 属性（详见 rknn_input 结构体说明）设成 FALSE 时 rknn_inputs_set 接口包含了颜色通道转换、归一化、量化、NHWC 转成 NCHW 的过程；当输出的 want_float 属性（详见 rknn_output 结构体说明）设成 TRUE 时，rknn_outputs_get 接口包含了反量化的过程。

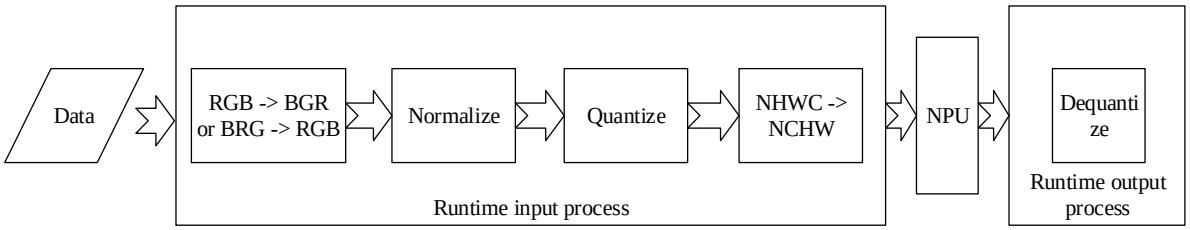

图 3-2 完整的图片数据处理流程

在实际使用过程中，对于某些 RKNN 模型，输入处理的流程并没有全部执行。例如，当输入数据不是 3 通道图像，或者 RKNN Toolkit 导出模型时 config 接口配置的 reorder 参数值为”0 1 2”时，输入处理流程中的颜色通道转换并不会被实际执行。当 RKNN 模型输入 tensor 的属性是 NHWC 布局时，没有 NHWC 转 NCHW 的流程。当输入的 pass_through 属性（详见 rknn_input 结构体说明）设成 TRUE 时，rknn_inputs_set 中的所有输入流程都不会执行，而是直接将输入数据透传给模型。

当输出的 want_float 属性（详见 rknn_output 结构体说明）设成 FALSE 时，rknn_outputs_get 接口不会执行反量化操作，而是返回模型输出 tensor 定义的数据类型。

3.3 量化和反量化

当输入的 pass_through 属性（详见 rknn_input 结构体说明）设成 TRUE 时，表明在 NPU 推理之前，输入数据要用户自行处理。当输出的 want_float 属性（详见 rknn_output 结构体说明）设成 FALSE 时，rknn_outputs_get 接口拿到的是 RKNN 模型中定义的输出数据类型，如果和用户后处理的数据类型不一致，则需要做相应的反量化等操作。

量化和反量化用到的量化方式、量化数据类型以及量化参数，可以通过 rknn_query 接口查询。目前 RK3399Pro 的 NPU 有非对称和动态定点两种量化方式，每种量化方式指定相应的量化数据类型。总共有以下四种数据类型和量化方式组合：

- uint8（非对称量化）
- int8（动态定点）
- int16（动态定点）
- float16（无）

通常，归一化后的数据用 32 位浮点数保存，32 位浮点数转换成 16 位浮点数请参考 IEEE-754 标准。假设归一化后的 32 位浮点数是 D，下面介绍量化流程：

1) float32 转 uint8

假设输入 tensor 的非对称量化参数是 Sq，ZP，数据 D 量化公式如下：

$$D_q = round(clamp(D / S_q + ZP, 0, 255))$$

2) float32 转 int8

假设输入 tensor 的动态定点量化参数是 fl，数据 D 量化公式如下：

$$D_q = round(clamp(D * 2^{fl}, -128, 127))$$

3) float32 转 int16

假设输入 tensor 的动态定点量化参数是 fl，数据 D 量化过程表示为下式：

$$D_q = round(clamp(D * 2^{fl}, -32768, 32767))$$

反量化流程是量化的逆过程，可以根据上述公式反推出反量化公式，这里不做赘述。

3.4 RKNN API 详细说明

RKNN API 各接口的详细说明如下。

3.4.1 rknn_init & rknn_init2

API	int rknn_init(rknn_context* context, void* model, uint32_t size, uint32_t flag) int rknn_init2(rknn_context* context, void* model, uint32_t size, uint32_t flag, rknn_init_extend* extend)
功能	创建 context 并加载 rknn 模型，并根据 flag 执行特定的初始化行为。
参数	rknn_context* context： context 对象指针。用于返回创建的 context 对象。
	void* model： 指向 rknn 模型的指针。
	uint32_t size： rknn 模型的大小。
	uint32_t flag： 扩展 flag： **RKNN_FLAG_PRIOR_HIGH**：创建高优先级的 Context。 **RKNN_FLAG_PRIOR_MEDIUM**：创建中优先级的 Context。 **RKNN_FLAG_PRIOR_LOW**：创建低优先级的 Context。 **RKNN_FLAG_ASYNC_MASK**：打开异步模式。打开之后，rknn_outputs_get 将不会阻塞太久，因为它直接返回的上一帧的推理结果（第一帧的推理结果除外），这将显著提高单线程模式下的推理帧率，但代价是 rknn_outputs_get 返回的不是当前帧的推理结果。但当 rknn_run 和 rknn_outputs_get 不在同一个线程时，则无需打开该异步模式。 **RKNN_FLAG_COLLECT_PERF_MASK**：打开性能收集调试开关。打开之后能够通过 rknn_query 接口查询网络每层运行时间。需要注意，该标志被设置后，因为需要同步每层的执行操作，所以推理一帧的总耗时会比不使用 RKNN_FLAG_COLLECT_PERF_MASK 标志时更长。
	rknn_init_extend* extend: 扩展信息的指针，如用于设置或获取当前 init 的信息，如设置设备的 ID 号 device_id（详见 rknn_api.h 的 rknn_init_extend 定义）。如不用，可赋 NULL。
返回值	int 错误码（见 rknn 返回值错误码）。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0);
```

3.4.2 rknn_destroy

API	int rknn_destroy(rknn_context context)
功能	卸载 rknn 模型并销毁 context 及其相关资源。
参数	rknn_context context： context 的对象。
返回值	int 错误码（见 rknn 返回值错误码）。

示例代码如下：

```
int ret = rknn_destroy (ctx);
```

3.4.3 rknn_query

API	int rknn_query(rknn_context context, rknn_query_cmd cmd, void* info, uint32_t size)
功能	查询模型与 SDK 的相关信息。
参数	rknn_context context： context 的对象。
	rknn_query_cmd cmd： 查询命令。
	void* info： 存放返回结果的结构体变量。
	uint32_t size： info 对应的结构体变量的大小。
返回值	int 错误码（见 rknn 返回值错误码）

当前 SDK 支持的查询命令如下表所示：

表 3-1 RKNN SDK 支持的查询命令

查询命令	返回结果结构体	功能
RKNN_QUERY_IN_OUT_NUM	rknn_input_output_num	查询 input 和 output 的 Tensor 个数。
RKNN_QUERY_INPUT_ATTR	rknn_tensor_attr	查询 Input Tensor 属性。
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	查询 Output Tensor 属性。
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	查询网络各层运行时间。 该查询需要在 rknn_init 的 flag ‘与’上 RKNN_FLAG_COLLECT_PERF_MASK，否则获取不到详细的各层性能信息。另外，RKNN_QUERY_PERF_DETAIL 查询返回的 rknn_perf_detail 结构体的 perf_data 成员不需要用户进行主动释放。 同时该查询**需要在 rknn_outputs_get 函数调用后才能返回正确的查询结果**。
RKNN_QUERY_PERF_RUN	rknn_perf_run	查询单帧推理的硬件执行时间。 **同时该查询需要在 rknn_outputs_get 函数调用后才能返回正确的查询结果。**
RKNN_QUERY_SDK_VERSION	rknn_sdk_version	查询 SDK 版本。

接下来的小节将依次给出各个查询命令的使用方法。

3.4.3.1 查询输入和输出个数

传入 RKNN_QUERY_IN_OUT_NUM 命令可以查询模型 Input 和 Output 的 Tensor 个数。其中需要先创建 rknn_input_output_num 结构体对象。

示例代码如下：

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
printf("model input num: %d, output num: %d\n", io_num.n_input, io_num.n_output);
```

3.4.3.2 查询输入节点的 Tensor 属性

传入 RKNN_QUERY_INPUT_ATTR 命令可以查询模型输入节点的 Tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象。

示例代码如下：

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
    input_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_INPUT_ATTR, &(input_attrs[i]),
                     sizeof(rknn_tensor_attr));
}
```

3.4.3.3 查询输出节点的 Tensor 属性

传入 RKNN_QUERY_OUTPUT_ATTR 命令可以查询模型输出节点的 Tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象。

示例代码如下：

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
    output_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &(output_attrs[i]),
                     sizeof(rknn_tensor_attr));
}
```

3.4.3.4 查询网络各层运行时间

如果在 rknn_init 函数调用时有设置 RKNN_FLAG_COLLECT_PERF_MASK 标志，那么在执行 rknn_outputs_get 调用完成之后，可以传入 RKNN_QUERY_PERF_DETAIL 命令来查询网络每层运行时

间。其中需要先创建 rknn_perf_detail 结构体对象。

另外，RKNN_QUERY_PERF_DETAIL 查询返回的 rknn_perf_detail 结构体的 perf_data 成员**不需要用户进行主动释放**。

该查询需要在 rknn_outputs_get 函数调用后才能返回正确的查询结果。

示例代码如下：

```
rknn_perf_detail perf_detail;
ret = rknn_query(ctx, RKNN_QUERY_PERF_DETAIL, &perf_detail,
sizeof(rknn_perf_detail));
printf("%s", perf_detail.perf_data);
```

3.4.3.5 查询单帧推理的时间

传入 RKNN_QUERY_PERF_RUN 命令可以查询单帧推理的硬件执行时间。其中需要先创建 rknn_perf_run 结构体对象。

同时该查询需要在 rknn_outputs_get 函数调用后才能返回正确的查询结果。

示例代码如下：

```
rknn_perf_run perf_run;
ret = rknn_query(ctx, RKNN_QUERY_PERF_RUN, &perf_run,
                 sizeof(rknn_perf_run));
printf("%ld", perf_run.run_duration);
```

3.4.3.6 查询 SDK 版本

传入 RKNN_QUERY_SDK_VERSION 命令可以查询 RKNN API 以及 Driver 的版本。其中需要先创建 rknn_sdk_version 结构体对象。

示例代码如下：

```
rknn_sdk_version version;
ret = rknn_query(ctx, RKNN_QUERY_SDK_VERSION, &version,
                       sizeof(rknn_sdk_version));
printf("api version: %s\n", version.api_version);
printf("driver version: %s\n", version.drv_version);
```

3.4.4 rknn_inputs_set

API	int rknn_inputs_set(rknn_context context, uint32_t n_inputs, rknn_input inputs[])
功能	设置 inputs 的 buffer 以及参数。 Buffer 及参数需存储在 rknn_input 中。该函数能够支持多个 input，其中每个 input 是 rknn_input 结构体对象，在传入之前用户需要设置该对象。
参数	rknn_context context： context 的对象。
	uint32_t n_inputs： inputs 的个数。
	rknn_input inputs[]： inputs 的数组指针，数组每个元素是 rknn_input 结构体对象。
返回值	int 错误码（见 rknn 返回值错误码）

示例代码如下：

```
rknn_input inputs[1];
memset(inputs, 0, sizeof(inputs));
inputs[0].index = 0;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].size = img_width*img_height*img_channels;
inputs[0].pass_through = FALSE;
inputs[0].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;

ret = rknn_inputs_set(ctx, 1, inputs);
```

3.4.5 rknn_run

API	int rknn_run(rknn_context context, rknn_run_extend* extend)
功能	执行一次模型推理，调用之前需要先通过 rknn_inputs_set 函数设置 input 数据。 该函数正常不会阻塞，但是当有超过 3 次推理结果没有通过 rknn_outputs_get 获取时则会阻塞，直至 rknn_outputs_get 被调用。
参数	rknn_context context：context 的对象。
	rknn_run_extend* extend：扩展信息的指针，用于设置或输出当前 rknn_run 对应的帧的信息，如 frame_id（详见 rknn_api.h 的 rknn_run_extend 定义）。如不用，可赋 NULL。
返回值	int 错误码（见 rknn 返回值错误码）

示例代码如下：

```
ret = rknn_run(ctx, NULL);
```

3.4.6 rknn_outputs_get

API	int rknn_outputs_get(rknn_context context, uint32_t n_outputs, rknn_output outputs[], rknn_output_extend* extend)
功能	等待推理操作结束并获取 outputs 结果。 该函数能够一次获取多个 output 数据。其中每个 output 是 rknn_output 结构体对象，在函数调用之前需要依次创建并设置每个 rknn_output 对象。另外，在推理结束前该函数会一直阻塞（除非有异常出错）。output 结果最后会被存至 outputs[]数组。 对于 output 数据的 buffer 存放可以采用两种方式：一种是用户自行申请和释放，此时 rknn_output 对象的 is_prealloc 需要设置为 TRUE，并且将 buf 指针指向用户申请的 buffer；另一种是由 rknn 来进行分配，此时 rknn_output 对象的 is_prealloc 设置为 FALSE 即可，函数执行之后 buf 将指向 output 数据。
参数	rknn_context context： context 的对象。
	uint32_t n_outputs： outputs 数组的个数，该个数要与 rknn 模型的 output 个数一致。（rknn 模型的 output 个数可以通过 rknn_query 查询得到。）
	rknn_output outputs[]： output 数据的数组，其中数组每个元素为 rknn_output 结构体对象，代表模型的一个 output。
	rknn_output_extend* extend： 扩展信息的指针，用于输出当前 output 对应的帧的信息，如 frame_id（详见 rknn_api.h 的 rknn_output_extend 定义）。如不用，可赋 NULL。
返回值	int 错误码（见 rknn 返回值错误码）

示例代码如下：

```
rknn_output outputs[io_num.n_output];
memset(outputs, 0, sizeof(outputs));
for (int i = 0; i < io_num.n_output; i++) {
    outputs[i].want_float = TRUE;
    outputs[i].is_prealloc = FALSE;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

3.4.7 rknn_outputs_release

API	int rknn_outputs_release(rknn_context context, uint32_t n_ouputs, rknn_output outputs[])
功能	释放由 rknn_outputs_get 获取的 outputs。 在 outputs 不再使用时需要调用该函数进行 outputs 的释放（不管 rknn_output[x].is_prealloc 是 TRUE 还是 FALSE 都需要调用该函数进行最终的释放）。 该函数被调用后，当 rknn_output[x].is_prealloc = FALSE 时，由 rknn_outputs_get 获取的 rknn_output[x].buf 地址也会被自动释放；当 rknn_output[x].is_prealloc = ture 时，rknn_output[x].buf 则需要用户自己主动释放。
参数	rknn_context context： context 的对象
	uint32_t n_outputs：outputs 数组的个数，该个数要与 rknn 模型的 output 个数一致。（rknn 模型的 output 个数可以通过 rknn_query 查询得到）
	rknn_output outputs[]： outputs 的数组指针
返回值	int 错误码（见 rknn 返回值错误码）

示例代码如下

```
ret = rknn_outputs_release(ctx, io_num.n_output, outputs);
```

3.4.8 rknn_find_devices

API	int rknn_find_devices(rknn_devices_id* pdevs)
功能	查找连接到 Host 的设备信息。
参数	rknn_devices_id* pdevs： 设备信息的结构体指针
返回值	int 错误码（见 rknn 返回值错误码）

示例代码如下：

```
rknn_devices_id devids;
ret = rknn_find_devices (&devids);
printf("n_devices = %d\n", devids.n_devices);
for(int i=0; i<devids.n_devices; i++) {
      printf("%d:    type=%s, id=%s\n", i, devids.types[i], devids.ids[i]);
}
```

3.5RKNN 数据结构定义

3.5.1 rknn_input_output_num

结构体 rknn_input_output_num 表示 input 和 output 的 Tensor 个数，其结构体成员变量如下表所示：

表 3-2 rknn_input_output_num 结构体成员

成员变量	数据类型	含义
n_input	uint32_t	Input Tensor 个数
n_output	uint32_t	Output Tensor 个数

3.5.2 rknn_tensor_attr

结构体 rknn_tensor_attr 表示模型的 Tensor 的属性，结构体的定义如下表所示：

表 3-3 rknn_tensor_attr 结构体成员

成员变量	数据类型	含义
index	uint32_t	表示 input 或 output 的 Tensor 的索引。 当使用 rknn_query 查询前，需要设置该参数。
n_dims	uint32_t	Tensor 维度个数。
dims	uint32_t[]	Tensor 各维度值。
name	char[]	Tensor 名称。
n_elems	uint32_t	Tensor 数据元素个数。
size	uint32_t	Tensor 数据所占内存大小。
fmt	rknn_tensor_format	Tensor 维度的格式，有以下格式： RKNN_TENSOR_NCHW RKNN_TENSOR_NHWC
type	rknn_tensor_type	Tensor 数据类型，有以下数据类型： RKNN_TENSOR_FLOAT32 RKNN_TENSOR_FLOAT16 RKNN_TENSOR_INT8 RKNN_TENSOR_UINT8 RKNN_TENSOR_INT16
qnt_type	rknn_tensor_qnt_type	Tensor 量化类型，有以下的量化类型： RKNN_TENSOR_QNT_NONE：未量化； RKNN_TENSOR_QNT_DFP：动态定点量化； RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC：非对称量化。
fl	int8_t	动态定点量化类型的参数。
zp	uint32_t	非对称量化类型的参数。
scale	float	非对称量化类型的参数。

3.5.3 rknn_input

结构体 rknn_input 表示模型的一个数据 input，用来作为参数传入给 rknn_inputs_set 函数。结构体的定义如下表所示：

表 3-4 rknn_input 结构体成员

成员变量	数据类型	含义
index	uint32_t	该 input 的索引。
buf	void*	input 数据 Buffer 的指针。
size	uint32_t	input 数据 Buffer 所占内存大小。
pass_through	uint8_t	input 数据直通模式。 **TRUE**：input 数据不做任何转换直接传至 rknn 模型的 input 节点，因此下面的 type 和 fmt 不需要进行设置。 **FALSE**：input 数据会根据下面的 type 和 fmt 转换成跟模型的 input 节点一致的数据，因此下面的 type 和 fmt 需要进行设置。
type	rknn_tensor_type	input 数据的类型。
fmt	rknn_tensor_format	input 数据的格式。

3.5.4 rknn_output

结构体 rknn_output 表示模型的一个数据 output，用来作为参数传入给 rknn_outputs_get 函数。结构体的定义如下表所示：

表 3-5 rknn_output 结构体成员

成员变量	数据类型	含义
want_float	uint8_t	标识是否需要将 output 数据转为 float 类型的 output。
is_prealloc	uint8_t	标识存放 output 数据的 Buffer 是否是预分配。
index	uint32_t	该 output 的索引。
buf	void*	output 数据 Buffer 的指针。
size	uint32_t	output 数据 Buffer 所占内存大小。

is_prealloc 为 FALSE 时，在 rknn_outputs_ge 函数执行后，结构体对象的 index/buf/size 成员将会被赋值，因此这三个成员变量不需要预先赋值。

is_prealloc 为 TRUE 时，结构体对象的 index/buf/size 需要预先被赋值，否则 rknn_outputs_get 函数调用会失败并报错。

3.5.5 rknn_perf_detail

结构体 rknn_perf_detail 表示模型的性能详情，结构体的定义如下表所示：

表 3-6 rknn_perf_detail 结构体成员

成员变量	数据类型	含义
perf_data	char*	性能详情包含网络每层运行时间，能够直接打印出来查看。
data_len	uint64_t	存放性能详情的字符串数组的长度。

3.5.6 rknn_perf_run

结构体 rknn_perf_run 表示模型的单次推理的执行时间，结构体的定义如下表所示：

表 3-7 rknn_perf_run 结构体成员

成员变量	数据类型	含义
run_duration	int64_t	模型的单次推理的硬件执行时间，单位 us。

3.5.7 rknn_init_extend

结构体 rknn_init_extend 表示 rknn_init 的扩展信息，用来作为参数传入给 rknn_init 函数，结构体的定义如下表所示：

表 3-8 rknn_init_extend 结构体成员

成员变量	数据类型	含义
device_id	char*	输入参数，用于选择当前连接的设备。如“0123456789ABCDEF”，该设备 id 可以通过 adb devices 进行查询。如果当前只有一个连接的设备，可以简单赋 nullptr 即可。

3.5.8 rknn_run_extend

结构体 rknn_run_extend 表示 rknn_run 的扩展信息,用来作为参数传入给 rknn_run 函数，结构体的定义如下表所示：

表 3-9 rknn_run_extend 结构体成员

成员变量	数据类型	含义
frame_id	uint64_t	返回参数，表示当前 run 的帧 id。该 id 与 rknn_output_extend.frame_id 一一对应，在 rknn_run 和 rknn_outputs_get 处于不同线程的情况下，可以用来确定帧的对应关系。

3.5.9 rknn_output_extend

结构体 rknn_output_extend 表示 rknn_outputs_get 的扩展信息，用来作为参数传入给

rknn_outputs_get 函数，结构体的定义如下表所示：

表 3-10 rknn_output_extend 结构体成员

成员变量	数据类型	含义
frame_id	uint64_t	返回参数，表示当前 output 的帧 id。该 id 与 rknn_run_extend.frame_id 一一对应，在 rknn_run 和 rknn_outputs_get 处于不同线程的情况下，可以用来确定帧的对应关系。

3.5.10 rknn_sdk_version

结构体 rknn_sdk_version 用来表示 RKNN SDK 的版本信息，结构体的定义如下：

表 3-11 rknn_sdk_version 结构体成员

成员变量	数据类型	含义
api_version	char[]	rknn api 的版本信息。
drv_version	char[]	rknn api 所基于的驱动版本信息。

3.5.11 rknn_devices_id

结构体 rknn_devices_id 用来表示设备列表信息，结构体的定义如下：

表 3-12 rknn_device_id 结构体成员

成员变量	数据类型	含义
n_devices	uint32_t	设备的个数。
types	char[][]	设备类型的列表。
ids	char[][]	设备 ID 的列表

3.5.12 RKNN 返回值错误码

RKNN API 函数的返回值错误码定义如下表所示

表 3-13 RKNN 错误码对照表

错误码	错误详情
RKNN_SUCC	执行成功
RKNN_ERR_FAIL	执行出错
RKNN_ERR_TIMEOUT	执行超时
RKNN_ERR_DEVICE_UNAVAILABLE	NPU 设备不可用
RKNN_ERR_MALLOC_FAIL	内存分配失败
RKNN_ERR_PARAM_INVALID	传入参数错误
RKNN_ERR_MODEL_INVALID	传入的 RKNN 模型无效
RKNN_ERR_CTX_INVALID	传入的 rknn_context 无效
RKNN_ERR_INPUT_INVALID	传入的 rknn_input 对象无效
RKNN_ERR_OUTPUT_INVALID	传入的 rknn_output 对象无效
RKNN_ERR_DEVICE_UNMATCH	版本不匹配
RKNN_ERR_INCOMPATILE_PRE_COMPILE_MODEL	不兼容的预编译模型
RKNN_ERR_INCOMPATILE_OPTIMIZATION_LEVEL_VERSION	不兼容的优化等级
RKNN_ERR_TARGET_PLATFORM_UNMATCH	不兼容的目标硬件平台

4 NPU 固件说明

4.1 NPU 固件目录说明

RK3399Pro 的 NPU 驱动被封装在 NPU 的 boot.img 文件中。RK3399Pro 更新 NPU 驱动时，只要替换相应的 boot.img 等文件即可。

不同的 RK3399Pro 开发板通过不同的方式（PCIE 和 USB 3.0）和 NPU 通信，所使用的 NPU 固件也不同。

固件目录如下：

```
npu_firmware/
├──── npu_fw
└──── npu_fw_pcie
```

- npu_fw_pcie: 适用于 PCIE 接口的 NPU 固件，包括 boot.img, MiniLoaderAll.bin, trust.img, uboot.img 等。
- npu_fw: 适用于 USB 接口的 NPU 固件，包括 boot.img, MiniLoaderAll.bin, trust.img, uboot.img 等。

注：可以通过 NPU CONNECTION 引脚的短接方式简单判断当前开发板 NPU 使用的是哪种接口方式。如果是 1、2 短接，则用的是 USB3.0，如果是 2、3 短接，则用的是 PCIE。NPU CONNECTION 位置如下图所示：

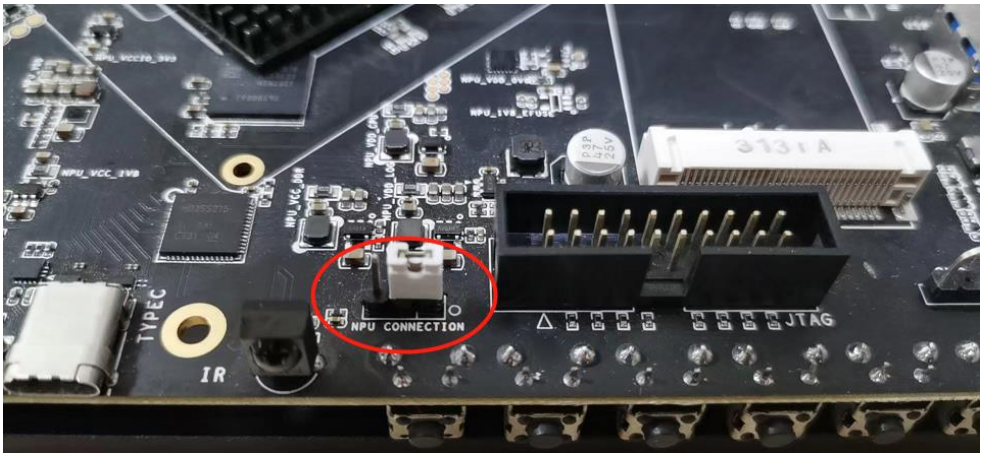

图 4-1 NPU CONNECTION 位置示意图

如果没有 NPU CONNECTION 引脚，则一般是使用 USB 的方式进行连接。

4.2 更新 NPU 驱动

在 RK3399Pro 上更新 NPU 驱动是通过更新 NPU 相关的 root.img 等文件实现的。具体的更新方法如下：

- 更新 PCIE 接口 NPU 驱动:

```
# 如果是 Android 系统的固件，在更新前先获取 root 和读写权限，如果是 Linux
# 系统的固件，跳过这两条命令
adb shell root
adb shell remount
# 更新 boot.img 等
adb push npu_firmware/npu_pcie_fw/* /vendor/etc/npu_fw/
adb shell reboot
```

- 更新 USB 接口 NPU:

```
# 如果是 Android 系统的固件，在更新前先获取 root 和读写权限，如果是 Linux
# 系统的固件，跳过这两条命令
adb shell root
adb shell remount
# 更新 boot.img 等
adb push npu_firmware/npu_fw/* /vendor/etc/npu_fw/
adb shell reboot
```

注意：不同的 RK3399Pro 固件，其 npu_fw 的路径可能不同，在更新 boot.img 等文件前建议先确认下该文件夹的位置。

5 附录

5.1 参考文档

RKNN Toolkit 使用指南：《Rockchip_User_Guide_RKNN_Toolkit_CN.pdf》

RKNN Toolkit Lite 使用指南：《Rockchip_User_Guide_RKNN_Toolkit_Lite_CN.pdf》

以上文档均存放在 rknn-toolkit/doc 目录中，请访问以下链接查阅：

https://github.com/rockchip-linux/rknn-toolkit/tree/master/doc

如果要使用适用于 RK1808 的 RKNN C API，请参考以下工程：

https://github.com/rockchip-linux/rknpu

5.2 问题反馈渠道

请通过 RKNN QQ 交流群，Github Issue 或瑞芯微 redmine 将问题反馈给 Rockchip NPU 团队。

RKNN QQ 交流群：1025468710

Github issue: https://github.com/rockchip-linux/rknn-toolkit/issues

Rockchip Redmine: https://redmine.rock-chips.com/

注：Redmine 账号需要通过销售或业务人员开通。如果是第三方开发板，请先找第三方厂方反馈问题。