密级状态：绝密(　　)　　秘密(　　)　　内部(　　)　　公开(√)

RKNN API For RK356X User Guide

（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：	1.1.0
	作　　者：	HPC
	完成日期：	2021-08-13
	审　　核：	熊伟
	完成日期：	2021-08-13

瑞芯微电子股份有限公司

Rockchips　Semiconductor　Co . , Ltd

更新记录

版本	修改人	修改日期	修改说明	核定人
v0.6.0	HPC	2021-3-1	初始版本	熊伟
v0.7.0	HPC	2021-4-22	删除输入通道转换流程说明	熊伟
v1.0.0	HPC	2021-4-30	正式发布版本	熊伟
v1.1.0	HPC	2021-8-13	1. 增加 rknn_tensor_mem_flags 标志 2. 增加输入/输出 tensor 原生属性的查询命令 3. 增加 NC1HWC2 的内存布局	熊伟

目 录

1 主要功能说明

RKNN SDK 为带有 NPU 的 RK3566/RK3568 芯片平台提供编程接口，能够帮助用户部署使用 RKNN-Toolkit2 导出的 RKNN 模型，加速 AI 应用的落地。

2 硬件平台

本文档适用如下硬件平台：

RK3566、RK3568

注：文档部分地方使用 RK356X 来表示 RK3566/RK3568。

3 使用说明

3.1 RKNN SDK 开发流程

在使用 RKNN SDK 之前，用户首先需要使用 RKNN-Toolkit2 工具将用户的模型转换为 RKNN 模型。

得到 RKNN 模型文件之后，用户可以选择使用 C 接口在 RK3566/RK3568 平台开发应用，后续章节将说明如何在 RK3566、RK3568 平台上基于 RKNN SDK 进行开发。

3.2 RKNN Linux 平台开发说明

3.2.1 Linux 平台 RKNN API 库

对于 RK3566/RK3568，SDK 库文件为<sdk>/rknpu2/下的 librknn_api.so

3.2.2 EXAMPLE 使用说明

SDK 提供了 Linux 平台的 MobileNet 图像分类、SSD 目标检测示例。这些 Demo 能够为客户

基于 RKNN SDK 开发自己的 AI 应用提供参考。Demo 代码位于<sdk>/rknpu2/examples 目录。下面以 rknn_mobilenet_demo 为例来讲解如何快速上手运行。

1） 编译 Demo

```
cd examples/rknn_mobilenet_demo
#设置 build-linux.sh 下的 GCC_COMPILER 为正确的编译器路径
./build-linux.sh
```

2） 部署到 RK3566 或 RK3568 设备

```
adb push install/rknn_mobilenet_demo_Linux /userdata/
```

3） 运行 Demo

```
adb shell
cd /userdata/rknn_mobilenet_demo_Linux/
export LD_LIBRARY_PATH=./lib
./rknn_mobilenet_demo model/mobilenet_v1.rknn model/dog_224x224.jpg
```

3.3 RKNN ANDROID 平台开发说明

3.3.1 ANDROID 平台 RKNN API 库

Android 平台有两种方式来调用 RKNN API

1） 应用直接链接 librknnrt.so

2） 应用链接 Android 平台 HIDL 实现的 librknn_api_android.so

对于需要通过 CTS/VTS 测试的 Android 设备可以使用基于 Android 平台 HIDL 实现的 RKNN API。如果不需要通过 CTS/VTS 测试的设备建议直接链接使用 librknnrt.so，对各个接口调用流程的链路更短，可以提供更好的性能。

对于使用 Android HIDL 实现的 RKNN API 的代码位于 RK356x Android 系统 SDK 的 vendor/rockchip/hardware/interfaces/neuralnetworks 目录下。当完成 Android 系统编译后，将会生成一些 NPU 相关的库（对于应用只需要链接使用 librknn_api_android.so 即可），如下所示：

```
/system/lib/librknn_api_android.so
/system/lib/librknnhal_bridge.rockchip.so
/system/lib64/librknn_api_android.so
/system/lib64/librknnhal_bridge.rockchip.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0-adapter-helper.so
/vendor/lib64/librknnrt.so
/vendor/lib64/hw/rockchip.hardware.neuralnetworks@1.0-impl.so
```

也可以使用如下命令单独重新编译生成以上的库

```
mmm vendor/rockchip/hardware/interfaces/neuralnetworks/ -j8
```

当前请确保 vendor/rockchip/hardware/interfaces/neuralnetworks 目录更新包含以下提交：

```
commit 3df48976433d677edb3944132775450672c8d37c (HEAD)
Author: Huacong Yang <will.yang@rock-chips.com>
Date:   Thu Apr 29 15:35:25 2021 +0800

    neuralnetworks: update RKNN SDK 1.0

    Change-Id: I0c9f19461a4b356b0b0a81b307c4ad33bf61d4cf
    Signed-off-by: Huacong Yang <will.yang@rock-chips.com>
```

3.3.2 EXAMPLE 使用说明

目前 SDK 提供了 MobileNet 图像分类、SSD 目标检测示例。Demo 代码位于 <sdk>/rknpu2/examples 目录。用户可以使用 NDK 编译 Android 命令行中执行的 demo。下面以 rknn_mobilenet_demo 为例来讲解在 Android 平台上该 demo 如何使用：

1）编译 Demo

```
cd examples/rknn_mobilenet_demo
#设置 build-android.sh 下的 ANDROID_NDK_PATH 为正确的 NDK 路径
./build-android.sh
```

2）部署到 RK3566 或 RK3568 设备

```
adb push install/rknn_mobilenet_demo_Android /data/
```

3）运行 Demo

```
adb shell
cd /data/rknn_mobilenet_demo_Android/
export LD_LIBRARY_PATH=./lib
./rknn_mobilenet_demo model/mobilenet_v1.rknn model/dog_224x224.jpg
```

以上 Demo 默认使用 librknnrt.so，如果开发者需要使用 librknn_api_android.so，可以将对应 Demo 下的 CMakeLists.txt 中链接 librknn_api.so 的地方修改为 librknn_api_android.so 即可，再按上述步骤编译运行。

3.4 RKNN C API

3.4.1 API 流程说明

目前在 RK356X 上有两组 API 可以使用，分别是通用 API 接口和零拷贝流程的 API 接口。两组 API 的主要区别在于，通用接口每次更新帧数据，需要将外部模块分配的数据拷贝到 NPU 运行时的输入内存，而零拷贝流程的接口会创建内存信息结构体，这些内存信息可以传给 NPU 和 RGA 等模块使用，减少了内存拷贝的花销，用户需要自行管理这些内存信息结构体。

对于通用 API 接口，首先初始化 rknn_input 结构体，帧数据包含在该结构体中，使用 rknn_inputs_set 函数设置模型输入，等待推理结束后，使用 rknn_outputs_get 函数获取推理的输出，进行后处理。在每次推理前，更新帧数据。通用 API 调用流程如图 3-1 所示，黄色字体流程表示用户行为。

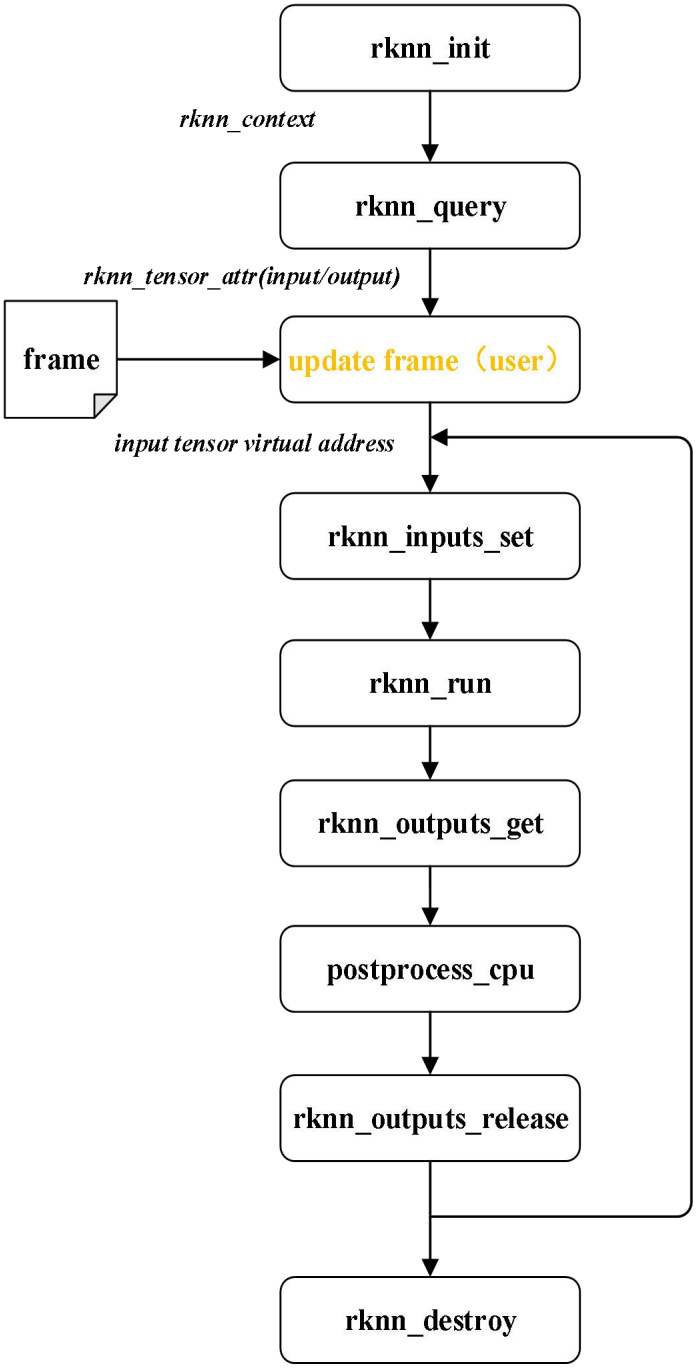

图 3-1 通用 API 接口调用流程

对于零拷贝 API 接口，在分配内存后使用内存信息初始化 rknn_tensor_memory 结构体，在推理前创建并设置该结构体，并在推理后读取该结构体中的内存信息。根据用户是否需要自行分配模型的模块内存（输入/输出/权重/中间结果）和内存表示方式（文件描述符/物理地址等）差异，有下列三种典型的零拷贝调用流程，如图 3-2 至图 3-4 所示，红色部分表示专为零拷贝加入的接口和数据结构，斜体表示接口调用之间传递的数据结构。

1）输入/输出内存由运行时分配

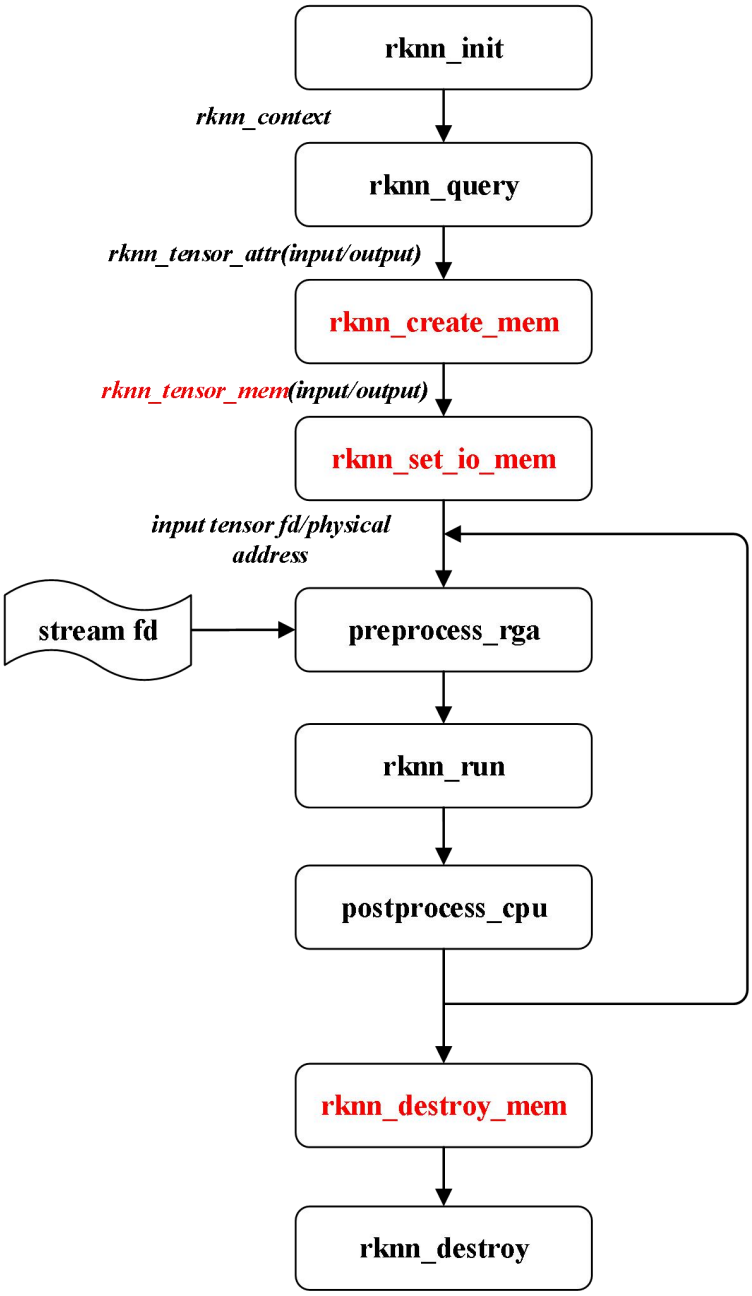

图 3-2 零拷贝 API 接口调用流程（输入/输出内部分配）

如图 3-2 所示，rknn_create_mem 接口创建的输入/输出内存信息结构体包含了文件描述符成员和物理地址，RGA 的接口使用到 NPU 分配的内存信息， preprocess_rga 表示 RGA 的接口，stream_fd 表示 RGA 的接口输入源的内存数据，postprocess_cpu 表示后处理的 CPU 实现。

2）输入/输出内存由外部分配

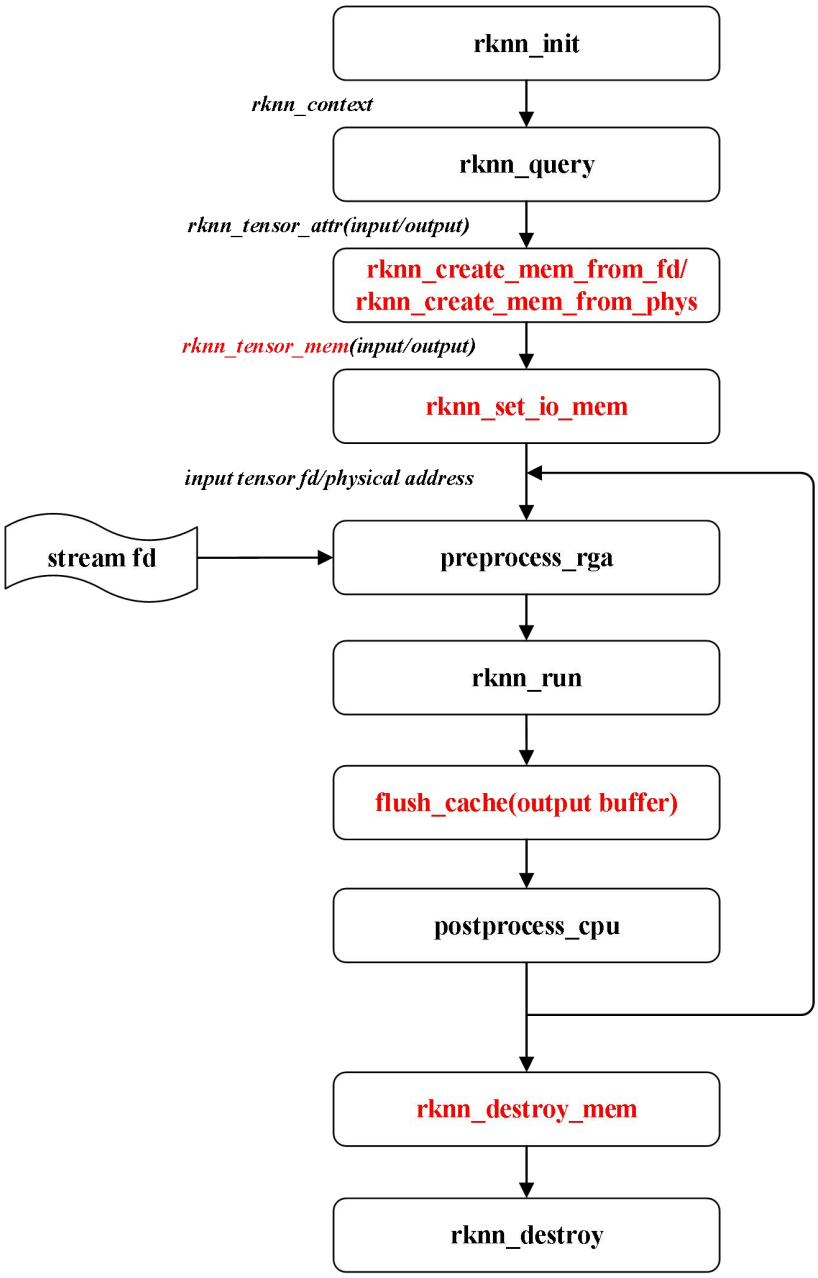

图 3-3 零拷贝 API 接口调用流程（输入/输出外部分配）

如图 3-3 所示，flush_cache 表示用户需要调用与分配的内存类型关联的接口来刷新输出缓存。

3）输入/输出/权重/中间结果内存由外部分配

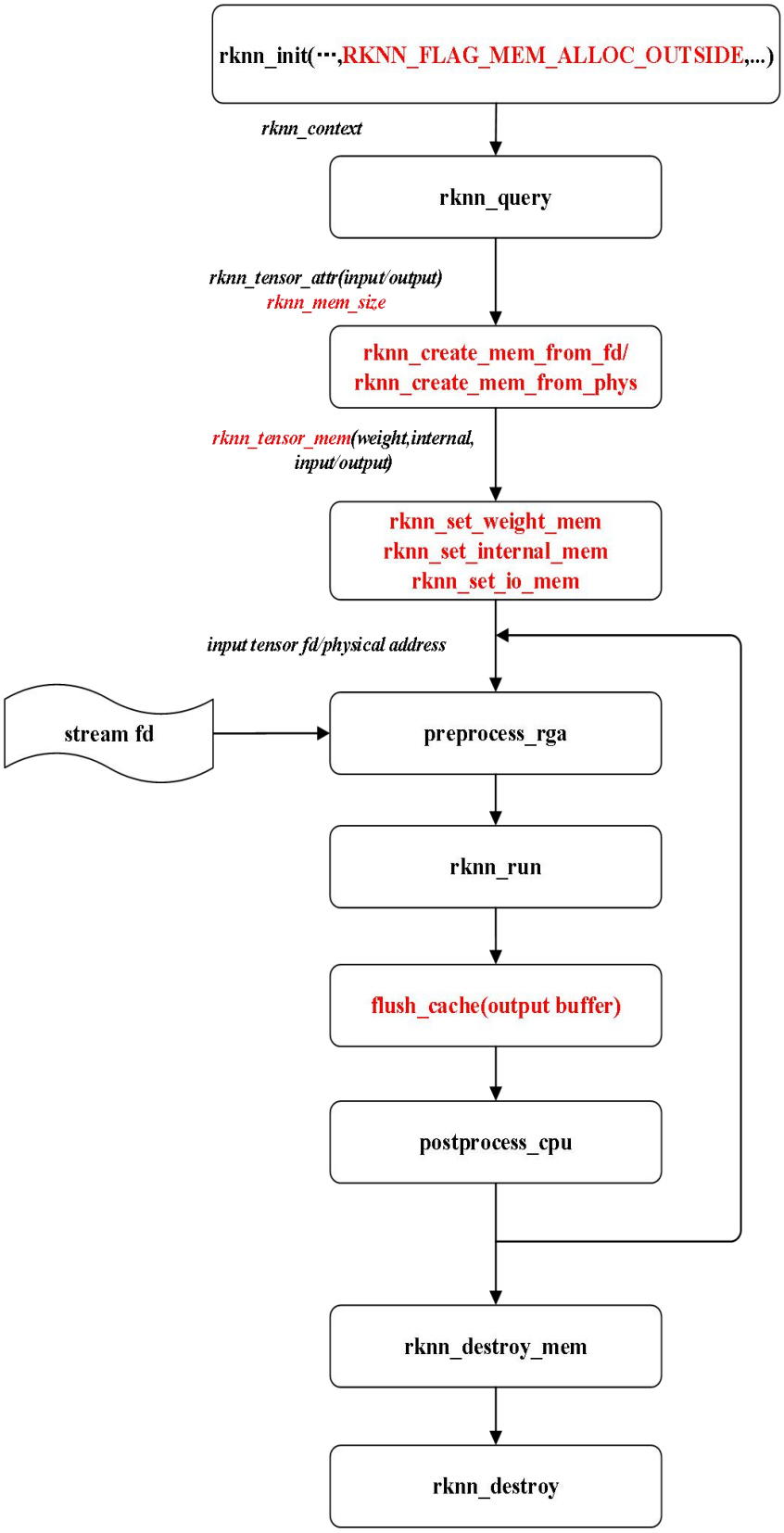

图 3-4 零拷贝 API 接口调用流程（输入/输出/权重/中间结果外部分配）

3.4.1.1 API 内部处理流程

在推理 RKNN 模型时，原始数据要经过输入处理、NPU 运行模型、输出处理三大流程。目前根据不同模型输入格式和量化方式，通用 API 接口内部会存在以下两种处理流程。

1）int8 量化模型且输入通道数是 1 或 3 或 4

如图 3-5 所示，原始数据的处理流程经过优化。假设输入是 3 通道的模型，用户必须保证 R、G、B 三个通道的颜色顺序与训练模型时一致，归一化、量化和模型推理都会在 NPU 上运行，NPU 输出的数据布局格式和反量化过程在 CPU 上运行。

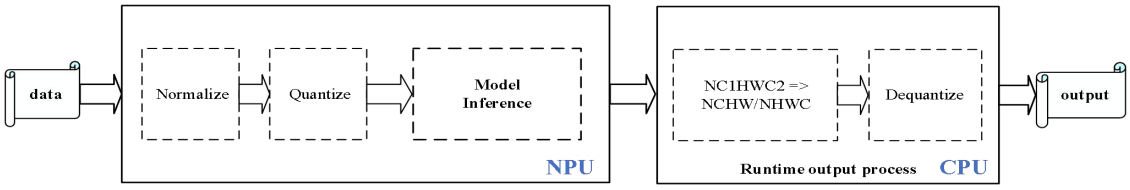

图 3-5 优化的数据处理流程

2）输入通道数是 2 或大于等于 4 的量化模型或非量化模型

对数据处理的流程如图 3-6 所示。对于数据的归一化、量化、数据布局格式转换、反量化均在 CPU 上运行，模型本身的推理在 NPU 上运行。此场景下，对于输入数据流程的处理效率会低于图 3-5 中优化的输入数据处理流程。

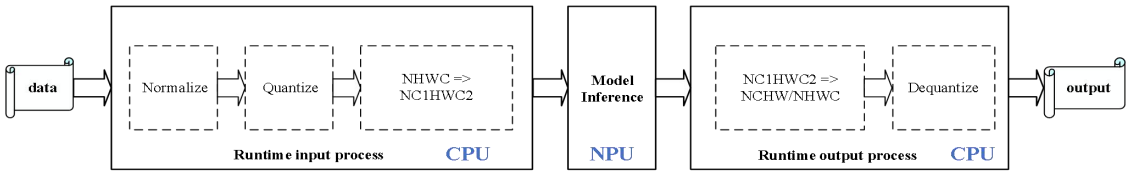

图 3-6 普通的数据处理流程

对于零拷贝的 API，API 内部流程只存在一种运行情形，如图 3-5 所示。零拷贝场景的条件如下：

1. 输入通道数是 1 或 3 或 4。

2. 输入的宽度是 8 像素对齐。

3. int8 非对称量化模型。

3.4.1.2 量化和反量化

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，RK3566/RK3568 只支持非对称量化，不支持动态定点量化，数据类型和量化方式组合包括：

- int8（非对称量化）
- int16（非对称量化，暂未实现）
- float16

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

1）float32 转 int8（非对称量化）

假设输入 tensor 的非对称量化参数是 S_q，ZP，数据 D 量化过程表示为下式：

$$D_q = round(clamp(D / S_q + ZP, -128, 127))$$

上式中，clamp 表示将数值限制在某个范围。round 表示做舍入处理。

2）float32 转 int16（非对称量化）

假设输入 tensor 的非对称量化参数是 S_q、ZP，数据 D 量化过程表示为下式：

$$D_q = round(clamp(D / S_q + ZP, -32768, 32767))$$

反量化流程是量化的逆过程，可以根据上述量化公式反推出反量化公式，这里不做赘述。

3.4.2 API 详细说明

3.4.2.1 rknn_init

rknn_init 初始化函数将创建 rknn_context 对象、加载 RKNN 模型以及根据 flag 和 rknn_init_extend 结构体执行特定的初始化行为。

API	rknn_init
功能	初始化 rknn
参数	rknn_context *context：rknn_context 指针。函数调用之后，context 将会被赋值。
	void *model：RKNN 模型的二进制数据或者 RKNN 模型路径。
	uint32_t size：当 model 是二进制数据，表示模型大小，当 model 是路径，则设置为 0。
	uint32_t flag：特定的初始化标志。
	rknn_init_extend：特定初始化时的扩展信息。当前没有使用，传入 NULL 即可。
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, NULL);
```

3.4.2.2 rknn_destroy

rknn_destroy 函数将释放传入的 rknn_context 及其相关资源。

API	rknn_destroy
功能	销毁 rknn_context 对象及其相关资源。
参数	rknn_context context：要销毁的 rknn_context 对象。
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_destroy (ctx);
```

3.4.2.3 rknn_query

rknn_query 函数能够查询获取到模型输入输出信息、逐层运行时间、模型推理的总时间、SDK 版本、内存占用信息、用户自定义字符串等信息。

API	rknn_query
功能	查询模型与 SDK 的相关信息。
参数	rknn_context context：rknn_context 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：

查询命令	返回结果结构体	功能
RKNN_QUERY_IN_OUT_NUM	rknn_input_output_num	查询输入输出 Tensor 个数
RKNN_QUERY_INPUT_ATTR	rknn_tensor_attr	使用通用 API 接口时,查询输入 Tensor 属性
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	使用通用 API 接口时,查询输出 Tensor 属性
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	查询网络各层运行时间，**需要调用 rknn_init 接口时，设置 RKNN_FLAG_COLLECT_PERF_MASK 标志才能生效**
RKNN_QUERY_PERF_RUN	rknn_perf_run	查询推理模型（不包含设置输入/输出）的耗时，单位是微秒
RKNN_QUERY_SDK_VERSION	rknn_sdk_version	查询 SDK 版本
RKNN_QUERY_MEM_SIZE	rknn_mem_size	查询分配给权重和网络中间 tensor 的内存大小

RKNN_QUERY_CUSTOM_STRING	rknn_custom_string	查询 RKNN 模型里面的用户自定义字符串信息
RKNN_QUERY_NATIVE_INPUT_ATTR	rknn_tensor_attr	使用零拷贝 API 接口时,查询原生输入 Tensor 属性，它是 NPU 直接读取的模型输入属性
RKNN_QUERY_NATIVE_OUTPUT_ATTR	rknn_tensor_attr	使用零拷贝 API 接口时,查询原生输出 Tensor 属性，它是 NPU 直接输出的模型输出属性

接下来的将依次详解各个查询命令如何使用。

1） 查询 SDK 版本

传入 RKNN_QUERY_SDK_VERSION 命令可以查询 RKNN SDK 的版本信息。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("sdk api version: %s\n", version.api_version);
printf("driver version: %s\n", version.drv_version);
```

2） 查询输入输出 Tensor 个数

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_IN_OUT_NUM 命令可以查询模型输入输出 Tensor 的个数。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） 查询输入 Tensor 属性(用于通用 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_INPUT_ATTR 命令可以查询模型输入 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));
}
```

4） 查询输出 Tensor 属性(用于通用 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_OUTPUT_ATTR 命令可以查询模型输出 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));
}
```

5） 查询模型推理的逐层耗时

在 rknn_run 接口调用完毕后，rknn_query 接口传入 RKNN_QUERY_PERF_DETAIL 可以查询网络推理时逐层的耗时，单位是微秒。使用该命令的前提是，在 rknn_init 接口的 flag 参数需要包含 RKNN_FLAG_COLLECT_PERF_MASK 标志。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size,
                    RKNN_FLAG_COLLECT_PERF_MASK, NULL);
...
ret = rknn_run(ctx,NULL);
...
rknn_perf_detail perf_detail;
ret = rknn_query(ctx, RKNN_QUERY_PERF_DETAIL, &perf_detail,
                    sizeof(perf_detail));
```

6） 查询模型推理的总耗时

在 rknn_run 接口调用完毕后，rknn_query 接口传入 RKNN_QUERY_PERF_RUN 可以查询上模型推理（不包含设置输入/输出）的耗时，单位是微秒。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
...
ret = rknn_run(ctx,NULL);
...
rknn_perf_run perf_run;
ret = rknn_query(ctx, RKNN_QUERY_PERF_RUN, &perf_run,
                    sizeof(perf_run));
```

7） 查询模型的内存占用情况

在 rknn_init 接口调用完毕后，当用户需要自行分配网络的内存时，rknn_query 接口传入 RKNN_QUERY_MEM_SIZE 可以查询模型的权重和网络中间 tensor 的内存（不包括输入和输出）占用情况。使用该命令的前提是在 rknn_init 接口的 flag 参数需要包含 RKNN_FLAG_MEM_ALLOC_OUTSIDE 标志。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size,
                    RKNN_FLAG_MEM_ALLOC_OUTSIDE , NULL);
rknn_mem_size mem_size;
ret = rknn_query(ctx, RKNN_QUERY_MEM_SIZE, &mem_size,
                    sizeof(mem_size));
```

8） 查询模型里用户自定义字符串

在 rknn_init 接口调用完毕后，当用户需要查询生成 RKNN 模型时加入的自定义字符串，rknn_query 接口传入 RKNN_QUERY_CUSTOM_STRING 可以获取该字符串。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
rknn_custom_string custom_string;
ret = rknn_query(ctx, RKNN_QUERY_CUSTOM_STRING, &custom_string,
                 sizeof(custom_string));
```

9） 查询原始输入 Tensor 属性(用于零拷贝 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_NATIVE_INPUT_ATTR 命令可以查询模型原生输入 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_NATIVE_INPUT_ATTR,
                     &(input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

10）查询原始输出 Tensor 属性(用于零拷贝 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_NATIVE_OUTPUT_ATTR 命令可以查询模型原生输出 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_NATIVE_OUTPUT_ATTR,
                                    &(output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

3.4.2.4 rknn_inputs_set

通过 rknn_inputs_set 函数可以设置模型的输入数据。该函数能够支持多个输入，其中每个输入是 rknn_input 结构体对象，在传入之前用户需要设置该对象。

API	rknn_inputs_set
功能	设置模型输入数据。
参数	rknn_context context：rknn_contex 对象。
	uint32_t n_inputs：输入数据个数。
	rknn_input inputs[]：输入数据数组，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].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;
inputs[0].pass_through = 0;

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

3.4.2.5 rknn_run

rknn_run 函数将执行一次模型推理，调用之前需要先通过 rknn_inputs_set 函数或者零拷贝的接口设置输入数据。

API	rknn_run
功能	执行一次模型推理。
参数	rknn_context context：rknn_context 对象。
	rknn_run_extend* extend：保留扩展，当前没有使用，传入 NULL 即可。
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);
```

3.4.2.6 rknn_wait

该接口用于非阻塞模式推理，目前暂未实现。

3.4.2.7 rknn_outputs_get

rknn_outputs_get 函数可以获取模型推理的输出数据。该函数能够一次获取多个输出数据。其中每个输出是 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。

对于输出数据的 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 1，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 0 即可，函数执行之后 buf 将指向输出数据。

API	rknn_outputs_get
功能	获取模型推理输出。
参数	rknn_context context：rknn_context 对象。
	uint32_t n_outputs：输出数据个数。
	rknn_output outputs[]：输出数据的数组，其中数组每个元素为 rknn_output 结构体对象，代表模型的一个输出。

	rknn_output_extend* extend：保留扩展，当前没有使用，传入 NULL 即可
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].index = i;
    outputs[i].is_prealloc = 0;
    outputs[i].want_float = 1;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

3.4.2.8 rknn_outputs_release

rknn_outputs_release 函数将释放 rknn_outputs_get 函数得到的输出的相关资源。

API	rknn_outputs_release
功能	释放 rknn_output 对象。
参数	rknn_context context：rknn_context 对象。
	uint32_t n_outputs：输出数据个数。
	rknn_output outputs[]：要销毁的 rknn_output 数组。
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);
```

3.4.2.9 rknn_create_mem_from_mb_blk

目前暂未实现。

3.4.2.10 rknn_create_mem_from_phys

当用户要自己分配内存让 NPU 使用时，通过 rknn_create_mem_from_phys 函数可以创建一

个 rknn_tensor_mem 结构体并得到它的指针，该函数通过传入物理地址、逻辑地址以及大小，外部内存相关的信息会赋值给 rknn_tensor_mem 结构体。

API	rknn_create_mem_from_phys
功能	通过物理地址创建 rknn_tensor_mem 结构体并分配内存
参数	rknn_context context：rknn_context 对象。
	uint64_t phys_addr：内存的物理地址。
	void *virt_addr：内存的虚拟地址。
	uint32_t size：内存的大小。
返回值	rknn_tensor_mem*： tensor 内存信息结构体指针。

示例代码如下：

```
//suppose we have got buffer information as input_phys, input_virt and size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem_from_phys(ctx, input_phys, input_virt, size);
```

3.4.2.11 rknn_create_mem_from_fd

当用户要自己分配内存让 NPU 使用时，rknn_create_mem_from_fd 函数可以创建一个 rknn_tensor_mem 结构体并得到它的指针，该函数通过传入文件描述符 fd、偏移、逻辑地址以及大小，外部内存相关的信息会赋值给 rknn_tensor_mem 结构体。

API	rknn_create_mem_from_fd
功能	通过文件描述符创建 rknn_tensor_mem 结构体
参数	rknn_context context：rknn_context 对象。
	int32_t fd：内存的文件描述符。
	void *virt_addr：内存的虚拟地址，**fd 对应的首地址**。
	uint32_t size：内存的大小。
	int32_t offset：内存相对于文件描述符和虚拟地址的偏移量。
返回值	rknn_tensor_mem*： tensor 内存信息结构体指针。

示例代码如下：

```
//suppose we have got buffer information as input_fd, input_virt and size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem_from_fd(ctx, input_fd, input_virt, size, 0);
```

3.4.2.12 rknn_create_mem

当用户要 NPU 内部分配内存时，rknn_create_mem 函数可以创建一个 rknn_tensor_mem 结构体并得到它的指针，该函数通过传入内存大小，运行时会初始化 rknn_tensor_mem 结构体。

API	rknn_create_mem
功能	运行时内部创建 rknn_tensor_mem 结构体并分配内存
参数	rknn_context context：rknn_context 对象。
	uint32_t size：内存的大小。
返回值	rknn_tensor_mem*： tensor 内存信息结构体指针。

示例代码如下：

```
//suppose we have got buffer size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem(ctx, size);
```

3.4.2.13 rknn_destory_mem

rknn_destory_mem 函数会销毁 rknn_tensor_mem 结构体，用户分配的内存需要自行释放。

API	rknn_destory_mem
功能	销毁 rknn_tensor_mem 结构体
参数	rknn_context context：rknn_context 对象。
	rknn_tensor_mem*：tensor 内存信息结构体指针。
返回值	int 错误码（见 rknn 返回值错误码）。

示例代码如下：

```
rknn_tensor_mem* input_mems [1];
int ret = rknn_destory_mem(ctx, input_mems[0]);
```

3.4.2.14 rknn_set_weight_mem

如果用户自己为网络权重分配内存，初始化相应的 rknn_tensor_mem 结构体后，在调用 rknn_run 前，通过 rknn_set_weight_mem 函数可以让 NPU 使用该内存。

API	rknn_set_weight_mem
功能	设置包含权重内存信息的 rknn_tensor_mem 结构体
参数	rknn_context context：rknn_context 对象。
	rknn_tensor_mem*：权重 tensor 内存信息结构体指针。
返回值	int 错误码（见 rknn 返回值错误码）。

示例代码如下：

```
rknn_tensor_mem* weight_mems [1];
int ret = rknn_set_weight_mem(ctx, weight_mems[0]);
```

3.4.2.15 rknn_set_internal_mem

如果用户自己为网络中间 tensor 分配内存，初始化相应的 rknn_tensor_mem 结构体后，在调用 rknn_run 前，通过 rknn_set_internal_mem 函数可以让 NPU 使用该内存。

API	rknn_set_internal_mem
功能	设置包含中间 tensor 内存信息的 rknn_tensor_mem 结构体
参数	rknn_context context：rknn_context 对象。
	rknn_tensor_mem*：模型中间 tensor 内存信息结构体指针。
返回值	int 错误码（见 rknn 返回值错误码）。

示例代码如下：

```
rknn_tensor_mem* internal_tensor_mems [1];
int ret = rknn_set_internal_mem(ctx, internal_tensor_mems[0]);
```

3.4.2.16 rknn_set_io_mem

如果用户自己为网络输入/输出 tensor 分配内存，初始化相应的 rknn_tensor_mem 结构体后，在调用 rknn_run 前，通过 rknn_set_io_mem 函数可以让 NPU 使用该内存。

API	rknn_set_io_mem
功能	设置包含模型输入/输出内存信息的 rknn_tensor_mem 结构体
参数	rknn_context context：rknn_context 对象。
	rknn_tensor_mem*：输入/输出 tensor 内存信息结构体指针。
返回值	int 错误码（见 rknn 返回值错误码）。

示例代码如下：

```
rknn_tensor_mem* io_tensor_mems [1];
int ret = rknn_set_io_mem(ctx, io_tensor_mems[0]);
```

3.4.3 RKNN 数据结构定义

3.4.3.1 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：

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

3.4.3.2 rknn_input_output_num

结构体 rknn_input_output_num 表示输入输出 Tensor 个数，其结构体成员变量如下表所示：

成员变量	数据类型	含义
n_input	uint32_t	输入 Tensor 个数
n_output	uint32_t	输出 Tensor 个数

3.4.3.3 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：

成员变量	数据类型	含义
index	uint32_t	表示输入输出 Tensor 的索引位置。
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** **RKNN_TENSOR_NC1HWC2**
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** **RKNN_TENSOR_UINT16** **RKNN_TENSOR_INT32** **RKNN_TENSOR_INT64**

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	RKNN_TENSOR_QNT_DFP 量化类型的参数。
zp	int32_t	RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC 量化类型的参数。
scale	float	RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC 量化类型的参数。
stride	uint32_t	实际存储一行图像数据的像素数目，等于一行的有效数据像素数目 + 为硬件快速跨越到下一行而补齐的一些无效像素数目，单位是像素。
size_with_stride	uint32_t	实际存储图像数据所占的存储空间的大小（包括了补齐的无效像素的存储空间大小）。
pass_through	uint8_t	0 表示未转换的数据，1 表示转换后的数据，转换包括归一化和量化。

3.4.3.4 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：

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。

3.4.3.5 rknn_perf_run

结构体 rknn_perf_run 表示模型的总体性能，结构体的定义如下表所示：

成员变量	数据类型	含义
run_duration	int64_t	网络总体运行（不包含设置输入/输出）时间，单位是微秒。

3.4.3.6 rknn_mem_size

结构体 rknn_mem_size 表示初始化模型时的内存分配情况，结构体的定义如下表所示：

成员变量	数据类型	含义
total_weight_size	uint32_t	网络的权重占用的内存大小。
total_internal_size	uint32_t	网络的中间 tensor 占用的内存大小。

3.4.3.7 rknn_tensor_mem

结构体 rknn_tensor_mem 表示 tensor 的内存信息。结构体的定义如下表所示：

成员变量	数据类型	含义
virt_addr	void*	该 tensor 的逻辑地址。
phys_addr	uint64_t	该 tensor 的物理地址。
fd	int32_t	该 tensor 的文件描述符。
offset	int32_t	相较于文件描述符和逻辑地址的偏移量。
size	uint32_t	该 tensor 占用的内存大小。
flags	uint32_t	rknn_tensor_mem 的标志位，有以下标志： **RKNN_TENSOR_MEMORY_FALGS_ALLOC_INSIDE:** 表明 rknn_tensor_mem 结构体由运行时创建。 **用户不用关注该标志**。
priv_data	void*	内存的私有数据。

3.4.3.8 rknn_input

结构体 rknn_input 表示模型的一个数据输入，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：

成员变量	数据类型	含义
index	uint32_t	该输入的索引位置。
buf	void*	输入数据的指针。
size	uint32_t	输入数据所占内存大小。
pass_through	uint8_t	设置为 1 时会将 buf 存放的输入数据直接设置给模型的输入节点，不做任何预处理。
type	rknn_tensor_type	输入数据的类型。
fmt	rknn_tensor_format	输入数据的格式。

3.4.3.9 rknn_output

结构体 rknn_output 表示模型的一个数据输出，用来作为参数传入给 rknn_outputs_get 函数，在函数执行后，结构体对象将会被赋值。结构体的定义如下表所示：

成员变量	数据类型	含义
want_float	uint8_t	标识是否需要将输出数据转为 float 类型输出。
is_prealloc	uint8_t	标识存放输出数据是否是预分配。
index	uint32_t	该输出的索引位置。
buf	void*	输出数据的指针。
size	uint32_t	输出数据所占内存大小。

3.4.3.10 rknn_init_extend

结构体 rknn_init_extend 表示初始化模型时的扩展信息，**目前暂不支持使用**。结构体的定义如下表所示：

成员变量	数据类型	含义
core_id	int32_t	指定 NPU 的核心。
reserved	uint8_t[128]	预留数据位。

3.4.3.11 rknn_run_extend

结构体 rknn_run_extend 表示模型推理时的扩展信息，**目前暂不支持使用**。结构体的定义如下表所示：

成员变量	数据类型	含义
frame_id	uint64_t	表示当前推理的帧序号。
non_block	int32_t	0 表示阻塞模式，1 表示非阻塞模式，非阻塞即 rknn_run 调用直接返回。
timeout_ms	int32_t	推理超时的上限，单位毫秒。

3.4.3.12 rknn_output_extend

结构体 rknn_output_extend 表示获取输出的扩展信息，**目前暂不支持使用**。结构体的定义如下表所示：

成员变量	数据类型	含义
frame_id	int32_t	输出结果的帧序号。

3.4.3.13 rknn_custom_string

结构体 rknn_custom_string 表示转换 RKNN 模型时，用户设置的自定义字符串，结构体的定义如下表所示：

成员变量	数据类型	含义
string	char[]	用户自定义字符串。

3.4.4 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：

错误码	错误详情
RKNN_SUCC（0）	执行成功
RKNN_ERR_FAIL（-1）	执行出错
RKNN_ERR_TIMEOUT（-2）	执行超时
RKNN_ERR_DEVICE_UNAVAILABLE（-3）	NPU 设备不可用
RKNN_ERR_MALLOC_FAIL（-4）	内存分配失败
RKNN_ERR_PARAM_INVALID（-5）	传入参数错误
RKNN_ERR_MODEL_INVALID（-6）	传入的 RKNN 模型无效
RKNN_ERR_CTX_INVALID（-7）	传入的 rknn_context 无效
RKNN_ERR_INPUT_INVALID（-8）	传入的 rknn_input 对象无效
RKNN_ERR_OUTPUT_INVALID（-9）	传入的 rknn_output 对象无效
RKNN_ERR_DEVICE_UNMATCH（-10）	版本不匹配
RKNN_ERR_INCOMPATILE_OPTIMIZATION_LEVEL_VERSION（-12）	RKNN 模型设置了优化等级的选项，但是和当前驱动不兼容
RKNN_ERR_TARGET_PLATFORM_UNMATCH（-13）	RKNN 模型和当前平台不兼容