

密级状态：绝密() 秘密() 内部() 公开(√)

Rockchip User Guide RKNN API

(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：	1.7.5
	作 者：	HPC
	完成日期：	2023-07-31
	审 核：	熊伟
	完成日期：	2023-07-31

瑞芯微电子股份有限公司

Rockchips Semiconductor Co., Ltd

(版本所有,翻版必究)

更新记录

版本	修改人	修改日期	修改说明	核定人
v0.9.6	杨华聪	2018-12-20	初始版本	熊伟
V0.9.7	杨华聪	2019-01-16	1) 添加 RKNN SDK 开发流程说明 2) 添加 RKNN Python API 说明	熊伟
V0.9.8	杨华聪	2019-06-05	1) 添加 RKNN API 库的说明 2) 更新 rknn example 和 rknn-toolkit 路径 3) 修正文档中错误	卓鸿添
V1.3.3	卓鸿添	2020-06-02	1) 支持 RV1126/RV1109 2) 删除 python 支持说明，关于 python 支持见 RKNN Toolkit Lite 相关文档。	卓鸿添
V1.4.0	卓鸿添	2020-09-10	1) 增加错误码	卓鸿添
v1.6.0	洪启飞	2021-1-15	1) 增加 Matmul 高级 API 说明 2) 增加一组设置输入和一组获取输出的接口 3) 增加做量化和反量化的算法说明 4) 增加 NPU 驱动说明 5) 增加 FAQ	卓鸿添
v1.6.1	洪启飞	2021-4-26	1) 更新 Matmul 高级 API 说明	卓鸿添
v1.7.0	洪启飞	2021-8-5	1) 更新版本号	卓鸿添
v1.7.1	洪启飞	2021-12-2	1) rknn_inputs_set 接口单通道输入性能优化 2) 已知 Bug 修复	卓鸿添
v1.7.3	洪启飞	2022-8-13	1) 更新版本号	卓鸿添

版本	修改人	修改日期	修改说明	核定人
V1.7.5	HPC 组	2023-07-31	<p>1) 移除旧零拷贝接口说明；增加 rknn_create_mem / rknn_set_io_mem / rknn_destroy_mem 等新零拷贝接口的说明；</p> <p>2) 更新用户常见问题；</p> <p>3) Matmul 算子支持带转置功能,增加 rknn_matmul_load2 接口</p>	熊伟

目 录

1	主要功能说明.....	6
2	硬件平台.....	6
3	使用说明.....	6
3.1	RKNN SDK 开发流程	6
3.2	RKNN C API	7
3.2.1	RKNN API 库.....	7
3.2.2	EXAMPLE 使用说明.....	7
3.2.3	API 流程说明	8
3.2.3.1	API 内部处理流程.....	13
3.2.3.2	量化和反量化.....	13
3.2.3.3	零拷贝.....	14
3.2.4	API 详细说明	16
3.2.4.1	<i>rknn_init</i>	16
3.2.4.2	<i>rknn_destroy</i>	16
3.2.4.3	<i>rknn_query</i>	17
3.2.4.4	<i>rknn_inputs_set</i>	19
3.2.4.5	<i>rknn_inputs_map(deprecated)</i>	20
3.2.4.6	<i>rknn_inputs_sync(deprecated)</i>	20
3.2.4.7	<i>rknn_inputs_unmap(deprecated)</i>	20
3.2.4.8	<i>rknn_run</i>	20
3.2.4.9	<i>rknn_outputs_get</i>	21
3.2.4.10	<i>rknn_outputs_release</i>	21
3.2.4.11	<i>rknn_outputs_map(deprecated)</i>	22
3.2.4.12	<i>rknn_outputs_sync(deprecated)</i>	22

3.2.4.13	<i>rknn_outputs_unmap(deprecated)</i>	22
3.2.4.14	<i>rknn_create_mem</i>	22
3.2.4.15	<i>rknn_set_io_mem</i>	23
3.2.4.16	<i>rknn_destroy_mem</i>	24
3.2.5	RKNN 数据结构定义	24
3.2.5.1	<i>rknn_input_output_num</i>	24
3.2.5.2	<i>rknn_tensor_attr</i>	25
3.2.5.3	<i>rknn_input</i>	26
3.2.5.4	<i>rknn_tensor_mem</i>	26
3.2.5.5	<i>rknn_output</i>	27
3.2.5.6	<i>rknn_perf_detail</i>	27
3.2.5.7	<i>rknn_sdk_version</i>	28
3.2.6	RKNN 返回值错误码	28
4	高级 API 使用说明	30
4.1	MATMUL 算子库	30
4.1.1	简介	30
4.1.2	数据结构定义	30
4.1.3	详细 API 说明	31
4.1.3.1	<i>rknn_matmul_load</i>	31
4.1.3.2	<i>rknn_matmul_load2</i>	32
4.1.3.3	<i>rknn_matmul_run</i>	33
4.1.3.4	<i>rknn_matmul_unload</i>	33
4.1.4	实现限制	34
4.1.4.1	维度限制	34
4.1.4.2	输入数据类型限制	34
4.1.4.3	其他限制	34

4.1.5	基准测试	34
5	NPU 驱动说明.....	36
5.1.1	NPU 驱动目录说明	36
5.1.2	NPU full driver 与 mini driver 的区别.....	36
6	FAQ.....	37
6.1.1	输入输出数据格式问题	37
6.1.2	输入输出接口使用问题	38
6.1.3	API 调用流程问题	38
6.1.3.3	项目中同时要跑 2 个模型，因此创建了两个 rknn_context，分别获得 output1 和 output2，释放时只需要进行一次 rknn_outputs_release 吗？	39
6.1.4	性能问题	39
6.1.4.4	Python API 和 C API 调用，在速度方面差距大吗？	40
6.1.5	Demo 问题.....	40
6.1.5.1	为什么有些例子只支持 bmp 格式的图片？要如何才能支持其他格式的图片？	40
6.1.5.2	交叉编译工具链获取方法.....	40
6.1.6	零拷贝相关问题	41
6.1.6.1	使用零拷贝接口，会不会影响精度？	41
6.1.6.2	使用零拷贝接口时，内存分配失败要怎么办？	41
6.1.7	驱动问题	41
6.1.7.1	galcore_puma.ko 和 galcore_puma_tb.ko 是有什么区别吗.....	41

1 主要功能说明

RKNN SDK 为 RK1808 等带有 NPU 的平台提供编程接口，能够帮助用户部署使用 RKNN-Toolkit 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：

1) RK1808、RK1806

2) RV1126、RV1109

注意：本文档不适用 RK3399Pro

下面的说明以 RK1808 为例，也同时适用上述其他平台。

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-Toolkit 工具将用户的模型转换为 RKNN 模型，用户可以在 <https://github.com/rockchip-linux/rknn-toolkit> 获取工具的完整安装包及使用文档。

成功转换生成 RKNN 模型之后，用户可以先通过 RKNN-Toolkit 连接 RK1808 等开发板进行联机调试，确保模型的精度性能符合要求。

得到 RKNN 模型文件之后，用户可以选择使用 C 或 Python 接口在 RK1808 等平台开发应用，后续章节将说明如何在 RK1808 等平台上基于 RKNN SDK 进行开发。

3.2 RKNN C API

3.2.1 RKNN API 库

RKNN SDK 所提供的库和头文件位于<sdk>/rknpu/rknn/rknn_api/librknn_api 目录下，开发者可以在自己应用中引用即可开发应用。

需要注意的是，RK1808 和 RK3399Pro 平台的 RKNN API 是兼容的，两者开发的应用程序可以很方便地移植。但是使用过程中需要注意要区分两个平台的 librknn_api.so，如果开发者使用 RK3399Pro 的 librknn_api.so 将无法在 RK1808 平台上运行。开发者可以使用以下方法查看 librknn_api.so 的版本：

```
# 适用于 RK1808 平台的 librknn_api.so
$ strings librknn_api.so |grep version
librknn_api version 1.7.5 (71009c1 build: 2023-07-31 10:30:14)
```

3.2.2 EXAMPLE 使用说明

SDK 提供了 Linux 平台的 MobileNet 图像分类、SSD_Inception_v2 目标检测、YoloV5 目标检测 Demo、零拷贝 Demo、多输入 Demo、批量输入推理 Demo 等。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>/rknpu/rknn/rknn_api/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
# 修改 build.sh 中的 GCC_COMPILER, 指向目标平台的编译器
./build.sh
```

2) 部署到 RK1808 设备

```
adb push install/rknn_mobilenet_demo /userdata/
```

3) 运行 Demo


```
adb shell
cd /userdata/rknn_mobilenet_demo/
./rknn_mobilenet_demo model/mobilenet_v1_rk180x.rknn
model/dog_224x224.jpg #for RK1808
./rknn_mobilenet_demo model/mobilenet_v1_rv1109_rv1126.rknn
model/dog_224x224.jpg #for RV1109/RV1126
```

3.2.3 API 流程说明

目前在 RK1806/RK1808/RV1109/RV1126 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直接使用预先分配的内存（利用 DRM 框架），减少了内存拷贝的花销。当输入数据只有虚拟地址时，只能使用通用 API 接口；当输入数据有物理地址或 fd 时，两组接口都可以使用。

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，调用 rknn_inputs_set 接口设置模型输入，待推理结束后，调用 rknn_outputs_get 函数获取推理的输出结果，进行后处理。在每次推理前，更新输入数据。通用 API 调用流程如下图所示：

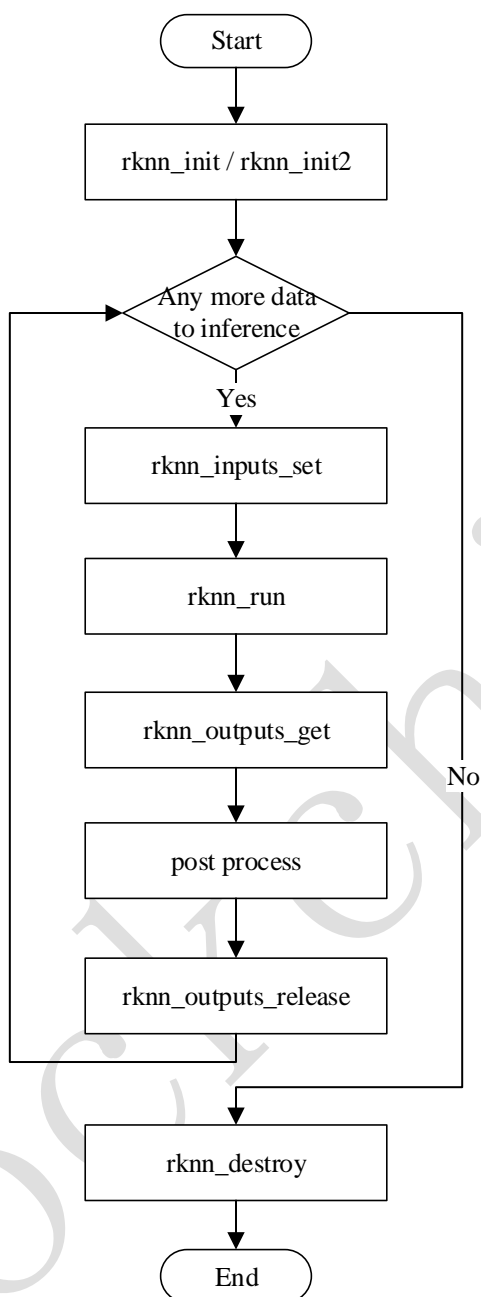


图 3-1 通用 API 接口调用流程

对于零拷贝接口，在设置输入、输出所使用的内存时，调用 `rknn_create_mem` 接口创建内存。该接口将根据所设置内存大小，通过 DRM 框架分配一块物理内存，并将该物理内存对应的物理地址、虚拟地址和文件描述符(fd)保存在 `rknn_tensor_mem` 结构体中。在分配完地址后，通过 `rknn_set_io_mem` 接口为输入、输出内存建立和模型输入、输出节点的关联关系。在做完这两步之后，应用程序就可以通过往输入内存写入数据更新模型的输入数据，模型推理完后会自动将结果更新到输出节点的物理内存中，应用程序直接从输出节点的内存中取数据完成相应的处理即可。在应用节后需要调用 `rknn_destroy_mem` 接口释放前面分配的内存。通过这种方式，在特定场景下，

可以减少模型推理过程中的内存拷贝次数，提高模型整体的运行效率。典型的使用流程如下图所示：

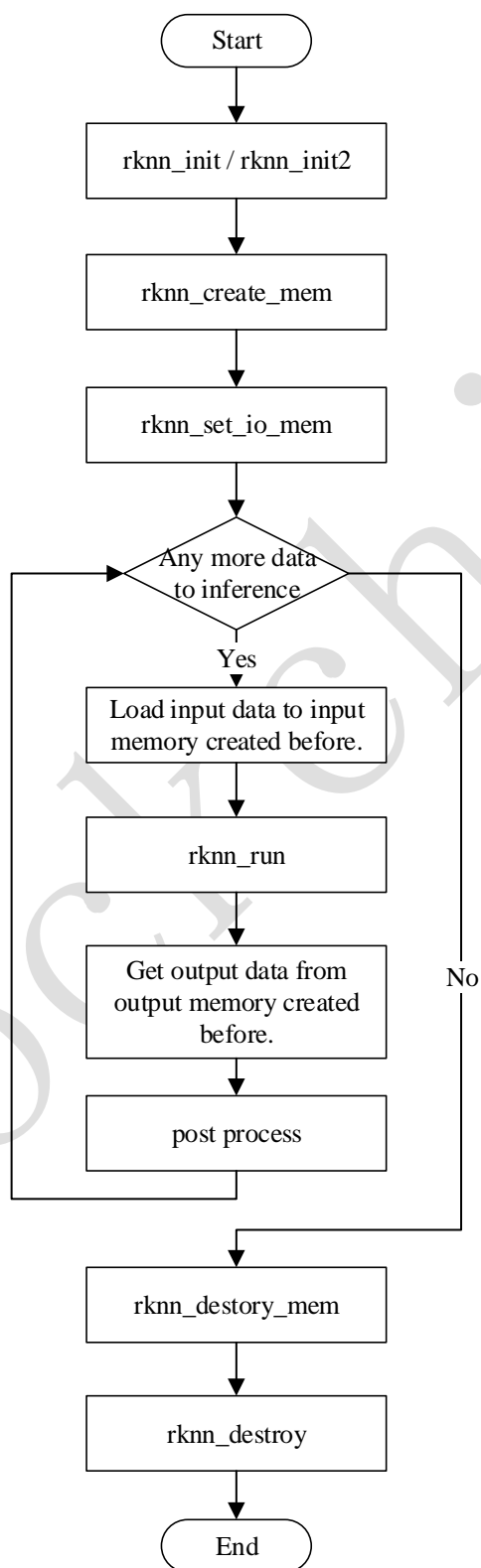


图 3-2 零拷贝接口使用流程

模型的输入和输出是分开的两部分，所以可以混用普通的 set/get 接口和 rknn_set_io_mem 接口。

例如对于模型的输出，使用 `rknn_outputs_get` 接口，但对模型的输入，使用 `rknn_create_mem` 接口创建输入内存，并通过 `rknn_set_io_mem` 接口与模型输入建立关联关系。典型的接口混用流程如下：

Rockchip

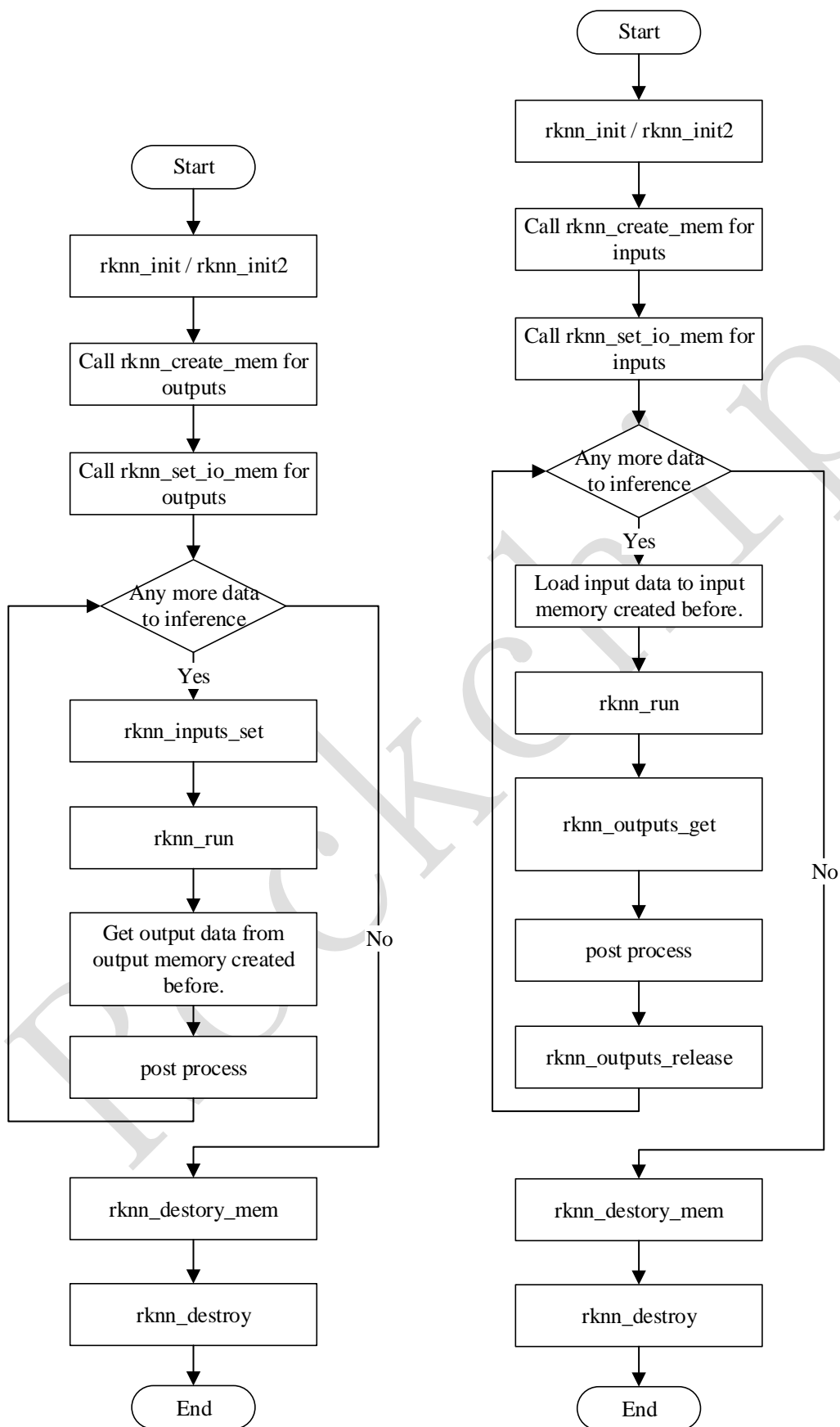


图 3-3 混合使用通用 API 接口和零拷贝接口

3.2.3.1 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。在典型的图片推理场景中，假设输入数据 `data` 是 3 通道的图片且为 NHWC 排布格式，运行时 (Runtime) 对数据处理的流程如图 3-4 所示。在 API 层面上，`rknn_inputs_set` 接口（当 `pass_through=0` 时，详见 [rknn_input](#) 结构体）包含了颜色通道交换、归一化、量化、NHWC 转换成 NCHW 的过程，`rknn_outputs_get` 接口（当 `want_float=1` 时，详见 [rknn_output](#) 结构体）包含了反量化的过程。

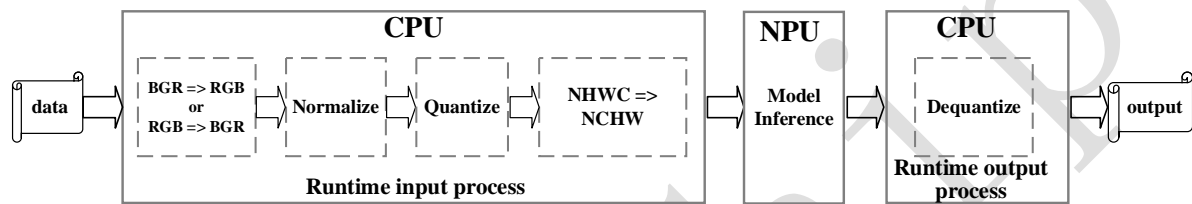


图 3-4 完整的图片数据处理流程

实际上，对于某些 RKNN 模型，输入处理的流程没有全部执行，例如，当输入数据不是 3 通道图像时或者用 `rknn-toolkit` 导出模型的 `config` 函数配置为 `reorder=“0 1 2”` 时，没有颜色通道转换流程。当 RKNN 模型输入 `tensor` 的属性是 NHWC 布局时，没有 NHWC 转换成 NCHW 的流程。`rknn_inputs_set`（当 `pass_through=1` 时）和 `rknn_inputs_map` 不包含任何输入处理流程，`rknn_outputs_get`（当 `want_float=0` 时）和 `rknn_outputs_map` 不包含任何输出处理流程。此时，虽然两组 API 都不包含对应的处理流程，但是 `set/get` 系列接口会比 `map` 系列接口多出数据拷贝过程。

3.2.3.2 量化和反量化

当使用 `rknn_inputs_set` (`pass_through=1`) 和 `rknn_inputs_map` 时，表明在 NPU 推理之前的流程要用户处理。`rknn_outputs_map` 获取输出后，用户也要做反量化得到 32 位浮点结果。

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，RK1808/RK3399Pro/RV1109/RV1126 的 NPU 有非对称量化和动态定点量化两种量化方式，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 uint8

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：

$$D_q = \text{round}(\text{clamp}(D / S_q + ZP, 0, 255)) \quad (3-1)$$

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 int8

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

$$D_q = \text{round}(\text{clamp}(D * 2^{f_l}, -128, 127)) \quad (3-2)$$

3) float32 转 int16

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

$$D_q = \text{round}(\text{clamp}(D * 2^{f_l}, -32768, 32767)) \quad (3-3)$$

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。

3.2.3.3 零拷贝

在特定的条件下，可以把输入数据拷贝次数减少到零，即零拷贝。比如，当 RKNN 模型是非对称量化，量化数据类型是 uint8，3 通道的均值是相同的整数同时缩放因子相同的情况下，归一化和量化可以省略。证明如下：

假设输入图像数据是 D_f ，量化参数是 S_q ， ZP 。 M_i 表示第 i 通道的均值， S_i 表示第 i 通道的归一化因子。则第 i 通道归一化后的数据 D_i 如下式子：

$$D_i = (D_f - M_i) / S_i \quad (3-4)$$

数据 D_i 量化过程表示为下式：

$$D_q = \text{clamp}(D_i / S_q + ZP, 0, 255) \quad (3-5)$$

上述两个式子合并后，可以得出

$$D_q = \text{clamp}((D_f - M_i) / (S_i * S_q) + ZP) \quad (3-6)$$

假设量化图片矫正集数据范围包含 0 到 255 的整数值，当 $M_1 = M_2 = M_3$ ， $S_1 = S_2 = S_3$ 时，归一化数值范围表示如下：

$$D_{\min} = (0 - M_i) / S_i = -M_i / S_i \quad (3-7)$$

$$D_{\max} = (255 - M_i) / S_i \quad (3-8)$$

因此，量化参数计算如下：

$$S_q = (D_{\max} - D_{\min}) / 255 = 1 / S_i \quad (3-9)$$

$$ZP = (0 - D_{\min}) / S_q = M_i \quad (3-10)$$

把式（3-9）和式（3-10）代入式（3-6），可以得出 $D_f = D_q$ ，即符合零拷贝的条件下：3 通道的均值是相同的整数同时归一化的缩放因子相同，输入 uint8 数据等于量化后的 uint8 数据。

输入零拷贝能降低 CPU 负载，提高整体的推理速度。针对 RGB 或 BGR 输入数据，实现输入零拷贝的步骤如下：

- 1) 三个通道的均值是相同的整数同时归一化的缩放因子相同。
- 2) 在 rknn-toolkit 的 config 函数中，设置 force_builtin_perm=True，导出 NHWC 输入的 RKNN 模型。
- 3) 调用 rknn_create_mem 接口，为输入 tensor 申请物理内存，并保存内存地址信息。
- 4) 第一次使用时，调用 rknn_set_io_mem 接口为输入内存和输出 tensor 创建关联关系。
- 5) 往内存地址填充输入数据，比如调用 RGA 缩放函数，目标地址使用刚刚创建的物理地址。
- 6) 调用 rknn_run 接口。
- 7) 调用获取输出接口。

3.2.4 API 详细说明

3.2.4.1 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 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。
	uint32_t size: 模型大小。
	uint32_t flag: 特定的初始化标志。目前 RK1808 平台仅支持以下标志： 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 rknn_run 的运行时间将会变长。
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);
```

3.2.4.2 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：

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

3.2.4.3 rknn_query

rknn_query 函数能够查询获取到模型输入输出、运行时间以及 SDK 版本等信息。

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	查询输入 Tensor 属性
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	查询输出 Tensor 属性
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	查询网络各层运行时间
RKNN_QUERY_SDK_VERSION	rknn_sdk_version	查询 SDK 版本

接下来的将依次详解各个查询命令如何使用。

1) 查询输入输出 Tensor 个数

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);
```

2) 查询输入 Tensor 属性

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));  
}
```

3) 查询输出 Tensor 属性

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));  
}
```

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_run 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。

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);
```

注意，用户不需要释放 rknn_perf_detail 中的 perf_data，SDK 会自动管理该 Buffer 内存。

5) 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);
```

3.2.4.4 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_context 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;

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

3.2.4.5 rknn_inputs_map(deprecated)

该接口已经废弃，请使用 `rknn_create_mem` 和 `rknn_set_io_mem` 接口创建输入内存，并与模型输入节点建立关联关系。如要继续使用该接口，请参考 1.7.3 版本的使用指南文档。

3.2.4.6 rknn_inputs_sync(deprecated)

该接口已经废弃，请使用 `rknn_create_mem` 和 `rknn_set_io_mem` 接口创建输入内存，并与模型输入节点建立关联关系。如要继续使用该接口，请参考 1.7.3 版本的使用指南文档。

3.2.4.7 rknn_inputs_unmap(deprecated)

该接口已经废弃，请使用 `rknn_destroy_mem` 接口删除为输入节点创建的内存。如要继续使用该接口，请参考 1.7.3 版本的使用指南文档。

3.2.4.8 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);
```

3.2.4.9 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：一种是由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].want_float = 1;  
}  
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

3.2.4.10 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);
```

3.2.4.11 rknn_outputs_map(deprecated)

该接口已经废弃，请使用 rknn_create_mem 和 rknn_set_io_mem 接口创建输出内存，并与模型输出节点建立关联关系。如要继续使用接口，请参考 1.7.3 版本的使用指南文档。

3.2.4.12 rknn_outputs_sync(deprecated)

该接口已经废弃，请使用 rknn_create_mem 和 rknn_set_io_mem 接口创建输出内存，并与模型输出节点建立关联关系。如要继续使用接口，请参考 1.7.3 版本的使用指南文档。

3.2.4.13 rknn_outputs_unmap(deprecated)

该接口已经废弃，请使用 rknn_destroy_mem 接口删除为输出节点创建的内存。如要继续使用接口，请参考 1.7.3 版本的使用指南文档。

3.2.4.14 rknn_create_mem

该接口可以分配一块物理内存，用于保存模型输入或输出的数据。

API	rknn_create_mem
功能	创建 rknn_tensor_mem 结构体，并分配指定大小的内存。
参数	rknn_context context: rknn_context 对象。
	uint32_t size: 待分配内存的大小。
返回值	rknn_tensor_mem*: tensor 内存信息结构体指针。

示例代码如下：

```
rknn_tensor_mem* input_mems[1];  
input_mems[0] = rknn_create_mem(ctx, size);
```

3.2.4.15 rknn_set_io_mem

该接口可以将 rknn_create_mem 分配出来的内存和模型的输入、输出节点建立关联关系，让 NPU 可以使用这些内存来获取输入或者保存输出数据。

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

示例代码如下：


```
rknn_tensor_attr output_attrs[1];
rknn_tensor_mem* output_mems[1];

ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &(output_attrs[0]),
sizeof(rknn_tensor_attr));

output_mems[0] = rknn_create_mem(ctx, output_attrs[0].size);

rknn_set_io_mem(ctx, output_mems[0], &output_attrs[0]);
```

3.2.4.16 rknn_destroy_mem

该接口将销毁指定 rknn_tensor_mem 中分配的内存。通常在应用结束前调用，以释放内存资源，避免内存泄漏。

API	rknn_destroy_mem
功能	销毁 rknn_tensor_mem 结构体，释放内存资源。
参数	rknn_context context: rknn_context 对象。
	rknn_tensor_mem* mem: 待销毁的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_destroy_mem(ctx, input_mems[0]);
```

3.2.5 RKNN 数据结构定义

3.2.5.1 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

3.2.5.2 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：

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
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	RKNN_TENSOR_QNT_DFP 量化类型的参数。
zp	uint32_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。

3.2.5.3 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 Buffer 的指针。
size	uint32_t	输入数据 Buffer 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。

3.2.5.4 rknn_tensor_mem

结构体 `rknn_tensor_mem` 表示 tensor 初始化后的存储状态信息，用来作为参数传入给 `rknn_set_io_mem` 以建立输入、输出内存和输入、输出节点的关联关系。结构体的定义如下表所示：

成员变量	数据类型	含义
logical_addr	void*	该输入的虚拟地址。
physical_addr	uint64_t	该输入的物理地址。
fd	int32_t	该输入的 fd。
size	uint32_t	该输入 tensor 占用的内存大小。
handle	uint32_t	该输入的 handle。
priv_data	void*	保留的数据。
reserved_flag	uint64_t	保留的标志位。

3.2.5.5 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	标识存放输出数据的 Buffer 是否是预分配。
index	uint32_t	该输出的索引位置。
buf	void*	输出数据 Buffer 的指针。
size	uint32_t	输出数据 Buffer 所占内存大小。

3.2.5.6 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。

3.2.5.7 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。

3.2.6 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_INCOMPATIBLE_PRE_COMPILE_MODEL(-11)	RKNN 模型使用 pre_compile 模式，但是和当前驱动不兼容
RKNN_ERR_INCOMPATIBLE_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，一般是将 RK1808 的平台的 RKNN 模型放到了 RV1109/RV1126 上。
RKNN_ERR_NON_PRE_COMPILED_MODEL_ON_MINI_DRIVER(-14)	RKNN 模型不是 pre_compile 模式，在 mini-driver 上无法执行

4 高级 API 使用说明

4.1 Matmul 算子库

4.1.1 简介

高级 API 旨在利用 NPU 高算力特性，执行特定的数学运算，提供简洁的接口调用，达到计算加速的效果。其中，Matmul 算子库是一个定点数矩阵乘法的加速库。该操作定义如下：

$$C = A^T * B$$

这里：

A, B 和 C 是 2 维矩阵

A 是一个 $K * M$ 的矩阵，

B 是一个 $K * N$ 的矩阵

C 是一个 $M * N$ 的矩阵

4.1.2 数据结构定义

`rknn_matmul_handle_t` 表示用于执行 Matmul 算子操作的句柄，它包含了运行时环境的上下文和输入 buffer 的信息。结构体的定义如下表所示：

成员变量	数据类型	含义
A	void*	运算时第一个矩阵 buffer 的指针。
B	void*	运算时第二个矩阵 buffer 的指针。
M	int32_t	A 矩阵的低维度元素个数。
K	int32_t	A 和 B 矩阵的高维度元素个数。
N	int32_t	B 矩阵的低维度元素个数。
in_dtype	rknn_tensor_type	输入数据的类型。
rknn_ctx	rknn_context	运行时的上下文对象。

transposeA	int8_t	A 矩阵转置的标志，0 表示 A 矩阵不做转置，1 表示 A 矩阵做转置后再做矩阵乘法。
transposeB	int8_t	B 矩阵转置的标志，0 表示 B 矩阵不做转置，1 表示 B 矩阵做转置后再做矩阵乘法。

4.1.3 详细 API 说明

4.1.3.1 rknn_matmul_load

rknn_matmul_load 函数将加载用户创建的输入 buffer，返回 rknn_matmul_handle_t 类型对象。

Matmul 算子 API 不负责管理输入 buffer 的生命周期，用户要确保输入 buffer 在 Matmul 算子 API 调用内有效。

API	rknn_matmul_load
功能	初始化和设置输入 buffer 指针。
参数	void *a: 用户创建的第一个矩阵 buffer 的指针，只支持输入是 8-bit 无符号整型或 8-bit 有符号整型的一维数组指针。
	void *b: 用户创建的第二个矩阵 buffer 的指针，只支持输入是 8-bit 无符号整型或 8-bit 有符号整型的一维数组指针。
	int32_t M: A 矩阵的低维度元素个数。
	int32_t K: A 和 B 矩阵的高维度元素个数。
	int32_t N: B 矩阵的低维度元素个数。
	rknn_tensor_type dtype: 用户指定的输入数据类型，只支持 RKNN_TENSOR_INT8 或 RKNN_TENSOR_UINT8 类型
返回值	rknn_matmul_handle_t 对象。

示例代码如下：


```
rknn_tensor_type dtype = RKNN_TENSOR_INT8;  
int8_t x[256*1] = {0};  
int8_t y[256*4096] = {0};  
rknn_matmul_handle_t handle= rknn_matmul_load(x,y,1,256,4096,dtype);
```

4.1.3.2 rknn_matmul_load2

rknn_matmul_load2 函数在 rknn_matmul_load 基础上增加对 B 矩阵做转置后再矩阵乘法的功能。

API	rknn_matmul_load2
功能	初始化和设置输入 buffer 指针。
参数	void *a: 用户创建的第一个矩阵 buffer 的指针, 只支持输入是 8-bit 无符号整型或 8-bit 有符号整型的一维数组指针。
	void *b: 用户创建的第二个矩阵 buffer 的指针, 只支持输入是 8-bit 无符号整型或 8-bit 有符号整型的一维数组指针。
	int32_t M: A 矩阵的低维度元素个数。
	int32_t K: A 和 B 矩阵的高维度元素个数。
	int32_t N: B 矩阵的低维度元素个数。
	rknn_tensor_type dtype: 用户指定的输入数据类型, 只支持 RKNN_TENSOR_INT8 或 RKNN_TENSOR_UINT8 类型
	transposeA: A 矩阵转置的标志, 0 表示 A 矩阵数据排布是[K,M], 目前仅支持 transposeA=0, M=1。
返回值	transposeB: B 矩阵转置标志, 0 表示 B 矩阵数据排布是[K,N], 1 表示 B 矩阵数据排布是[N,K]。
	rknn_matmul_handle_t 对象。

示例代码如下:

```
rknn_tensor_type dtype = RKNN_TENSOR_INT8;
int8_t x[256*1] = {0};
int8_t y[256*4096] = {0};
rknn_matmul_handle_t handle=
    rknn_matmul_load2(x,y,1,256,4096,dtype,0,1);
```

4.1.3.3 rknn_matmul_run

在 rknn_matmul_load/rknn_matmul_load2 被调用后和执行 rknn_matmul_run 前，输入 buffer 的数据由外部更新，不用重新调用 rknn_matmul_load/rknn_matmul_load2。

API	rknn_matmul_run
功能	执行 Matmul 操作。
参数	rknn_matmul_handle_t matmul_handle: 由 rknn_matmul_load 接口返回的句柄。
	float *c: 用户创建的矩阵浮点 buffer 的指针，用于获取输出。
返回值	int 错误码（见 rknn 返回值错误码 ）。

示例代码如下：

```
...
float out_fp32_buf[4096] = {0};
rknn_matmul_run(handle,out_fp32_buf);
```

4.1.3.4 rknn_matmul_unload

rknn_matmul_load 和 rknn_matmul_load2 接口初始化的上下文均使用 rknn_matmul_unload 销毁。

API	rknn_matmul_unload
功能	销毁 Matmul 算子运行时上下文。
参数	rknn_matmul_handle_t matmul_handle: 由 rknn_matmul_load 接口返回的句柄。
返回值	int 错误码（见 rknn 返回值错误码 ）。

示例代码如下：

```
rknn_matmul_unload(handle);
```

4.1.4 实现限制

Matmul 算子库是基于 NPU 的硬件架构实现，为了达到精度和速度的平衡，有如下一些限制。

4.1.4.1 维度限制

按照上述操作描述，该库实现了 $M=1$ 的矩阵乘法。具体而言，Matmul 算子输入 A 必须是 $K \times 1$ 形状的 buffer，即用户必须创建一块包含 K 个 8-bit 无符号整型或者 8-bit 有符号整型元素的数据。当算子库运行在 Mini driver 上，K 的值只能设置为 128 或 256 或 512，N 固定为 4096，而在 Full driver 上运行，没有此限制，但建议 K 的值为 128, 256, 512, 1024, 2048，N 取 2 的偶数次幂，建议 N 不大于 4096。

4.1.4.2 输入数据类型限制

只支持 8-bit 无符号整型和 8-bit 有符号整型两种输入。

4.1.4.3 其他限制

目前，只支持在 Full driver 上运行 rknn_matmul_load2 接口，transposeA 必须是 0。

4.1.5 基准测试

输入的两个矩阵使用随机数情况下，在 RV1109-EVB 板子上实测结果如表-1 所示。速度是 rknn_matmul_run 接口调用循环 100 次后的平均时间，平均相对误差是 NPU 和 CPU 上执行相同算法的结果的误差值，具体公式是：

$$\sum_k (abs(R_1 - R_2) / R_2) / N$$

其中，

R_1 是 Matmul 算子库输出向量，包含 N 个元素。

R_2 是 CPU 输出向量，包含 N 个元素。

表-1 Matmul 算子库速度/精度结果（RV1109, int8, transposeA=0, transposeB=0）

K	N	速度（ms）	平均相对误差
128	1024	1.0	0.00034
256	1024	1.6	0.00032
512	2024	3.0	-0.00015
1024	1024	5.4	0.00047
128	4096	3.0	0.00051
256	4096	5.6	0.00024
512	4096	10.7	0.00024
1024	4096	20.9	0.00051

注意，速度可能因为 NPU 驱动版本不同而有些许差异。误差值则根据每次测试的随机数不同也可能有些许差异。

5 NPU 驱动说明

5.1.1 NPU 驱动目录说明

NPU 的驱动在\$SDK/external/rknpu/drivers/目录下或者

<https://github.com/rockchip-linux/rknpu/tree/master/drivers>

其中的编译、安装规则参考\$SDK/buildroot/package/rockchip/rknpu/rknpu.mk

drivers/

- |— common
- |— linux-aarch64 (for RK1808 npu full driver)
- |— linux-aarch64-mini (for RK1808 npu mini driver)
- |— linux-armhf (for RK1806 npu full driver)
- |— linux-armhf-mini (for RK1806 npu mini driver)
- |— linux-armhf-puma (for RV1126/RV1109 npu full driver)
- |— linux-armhf-puma-mini (for RV1126/RV1109 npu mini driver)
- |— npu_ko (NPU kernel driver)

5.1.2 NPU full driver 与 mini driver 的区别

主要包含以下几点：

- 1) Mini driver 只支持预编译的 rknn 模型，如果跑非预编译模型，会出现 RKNN_ERR_MODEL_INVALID 的错误，从 1.6.0 开始，会返回 RKNN_ERR_NON_PRE_COMPILED_MODEL_ON_MINI_DRIVER 的错误；
- 2) Full driver 支持 RKNN Toolkit 的联机调试功能，mini driver 不支持；
- 3) Mini driver 库大小比 full driver 小很多，以 RV1109/RV1126 1.6.0 驱动为例，full driver 大小为 87MB，mini driver 大小为 7.1MB，可以有效的节省 flash 大小。
- 4) Mini driver 库运行时占用的内存比 full driver 小。

6 FAQ

6.1.1 输入输出数据格式问题

6.1.1.1 三通道图片数据输入，采用 RGB 还是 BGR 排布？

建议用户输入数据统一使用 RGB 排布。在导出 RKNN 模型时，config 函数的 reorder_channel 参数，有以下两种可能：

- 1) 如果原始模型使用 BGR 图片训练，reorder_channel='2 1 0'。
- 2) 如果原始模型使用 RGB 图片训练，reorder_channel='0 1 2'。

6.1.1.2 rknn_input 结构体该设置的 RKNN_TENSOR_NHWC 还是 RKNN_TENSOR_NCHW？ 两种设置耗时为何不同？

rknn_input 结构体根据用户自己的数据格式而定，C API 内部会自动转换成 NPU 需要的格式。耗时不同的原因是不同输入格式计算量不同，优化方式也不同。

6.1.1.3 没量化的 RKNN 模型，输出 rknn_tensor_attr 里面 size 和 rknn_outputs_get 接口返回的 rknn_output 的 size 为何不同？

没量化 RKNN 模型，NPU 内部输出数据类型是 float16，大小是元素数量*2 字节。当用户设置 want_float=1 时，想要的是 float32 数据，float16 会转换成 float32，大小是元素数量*4 字节。

6.1.1.4 rknn_output.index 是用户输入还是驱动返回？

驱动返回。

6.1.1.5 rknn_tensor_attr 中的 dims 数组为何会出现 0？

0 表示该维度无效。rknn_tensor_attr 中的 n_dims 表示 dims 数组的有效维度数量。

6.1.1.6 rknn_tensor_attr 中的 dims 数组顺序与 rknn_toolkit 的获取的 numpy 的顺序相反?

是。C API 中的数组排布跟 Python 相反，比如 rknn-toolkit 的 run () 接口获得 numpy 输出形状是[1,255,20,20],C API 中 dims 数组是{20,20,255,1}。

6.1.2 输入输出接口使用问题

6.1.2.1 pass_through 用法以及使用 rknn_set_io_mem 接口时，如何预处理数据?

请参考 https://github.com/rockchip-linux/rknpu/tree/master/rknn/rknn_api/examples 下的 rknn_pass_through_demo 示例。

6.1.2.2 使用 rknn_inputs_map 或者 rknn_outputs_map 获取的物理地址为什么会无效? 如何获取有效的物理地址?

输入/输出无法分配到物理连续的内存，可能的原因有：

- 1) 输入/输出的占用空间过大，超过了总的物理连续的内存大小（默认是 4MB）。
- 2) 系统中没有足够的物理连续的内存可用。
- 3) 导出 RKNN 模型时，config 函数添加如下参数：output_optimize=1。

用户可以尝试重启系统，或者让 NPU 驱动挂载时配置更大的连续地址空间，配置方法参考 [rknn_inputs_map](#) 接口说明。

注：该组接口已经废弃，请使用新增加的 rknn_create_mem 为输入、输出创建内存，使用 rknn_set_io_mem 接口为输入、输出内存建立与 RKNN 模型的关联关系。

6.1.3 API 调用流程问题

6.1.3.1 rknn_init 成功后，模型文件占用内存是否可以释放?

可以。

6.1.3.2 rknn_output.is_prealloc=1 时, rknn_outputs_release 是否需要调用?

需要。

6.1.3.3 项目中同时要跑 2 个模型, 因此创建了两个 rknn_context, 分别获得 output1 和 output2, 释放时只需要进行一次 rknn_outputs_release 吗?

要调用两次 rknn_outputs_release 接口, 分别释放 output1 和 output2 的内存。调用时要注意使用不同的 context。

6.1.4 性能问题

6.1.4.1 rknn_init 耗时过长?

使用预编译模型。使用方法参考 <https://github.com/rockchip-linux/rknn-toolkit/tree/master/doc> 下 User Guide 文档的相关章节。

6.1.4.2 rknn_inputs_set 接口耗时过长?

可能原因是数据量大或格式转换耗时长。如果是格式转换耗时长, 用户可以尝试 pass_through 用法自己做转换。转换方式请参考

https://github.com/rockchip-linux/rknnpu/tree/master/rknn/rknn_api/examples 下的

rknn_pass_through_demo 示例。或者尝试在导出 RKNN 模型时, config 函数添加如下参数:

output_optimize=1。

6.1.4.3 rknn_outputs_get 接口耗时过长?

可能原因是数据量大或转换格式耗时长。如果是转换格式耗时长, 用户可以尝试设置 want_float=0, 再自己做转换。或者尝试在导出 RKNN 模型时, config 函数添加如下参数:

output_optimize=1。

6.1.4.4 Python API 和 C API 调用，在速度方面差距大吗？

Python API 本质上是对 C API 的封装，有额外的开销，速度会慢些。而且不支持零拷贝等特性，在性能优化时可能会受到限制，如果对性能要求比较高，不建议用 Python API。

6.1.5 Demo 问题

6.1.5.1 为什么有些例子只支持 bmp 格式的图片？要如何才能支持其他格式的图片？

在 rknn_yolov5_demo 等示例中，为了将检测结果在图像中绘制出来，使用 CImg 库中的方法，而这个库只支持 bmp 格式的图片。

如果要支持其他格式的图片，可以使用 OpenCV 等图形库，代码也需要做相应改动。如果不需要绘制结果图，也可以将 CImg 相关的代码移除，读取图片所用的 stb_image 库支持 JPEG/PNG 等其他图片格式。

6.1.5.2 交叉编译工具链获取方法

对于 RV1109 / RV1126 平台，请通过如下链接下载交叉编译工具：

https://developer.arm.com/-/media/Files/downloads/gnu-a/8.3-2019.03/binrel/gcc-arm-8.3-2019.03-x86_64-arm-linux-gnueabi.tar.xz?revision=e09a1c45-0ed3-4a8c-b06b-db3978fd8d56&rev=e09a1c450ed34a8eb06bdb3978fd8d56&hash=9C4F2E8255CB4D87EABF5769A2E65733

对于 RK1808 平台，请通过如下链接下载交叉编译工具：

https://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/aarch64-linux-gnu/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu.tar.xz

对于 RK1806 平台，请通过如下链接下载交叉编译工具：

https://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/arm-linux-gnueabi/gcc-linaro-6.3.1-2017.05-x86_64_arm-linux-gnueabi.tar.xz

6.1.6 零拷贝相关问题

6.1.6.1 使用零拷贝接口，会不会影响精度？

理论上不会。出现精度问题，通常是输入数据没有正确处理导致的。

6.1.6.2 使用零拷贝接口时，内存分配失败要怎么办？

如果模型输入、输出所需要的内存比较大，超过 4MB，那在调用旧的零拷贝接口 `rknn_inputs_map` 时可能出现 `physical_addr` 为非法地址的问题。

出现这种情况是因为旧零拷贝接口使用的是系统预留的 DMA 内存，非常小。

遇到这种问题，建议使用新的零拷贝接口。1.7.3 版本之后，零拷贝示例都将使用新的零拷贝接口。

6.1.7 驱动问题

6.1.7.1 `galcore_puma.ko` 和 `galcore_puma_tb.ko` 驱动的差异？

`ko` 文件与系统的 Linux 内核关联，不同的开发板内核会有所差异。`galcore_puma_tb.ko` 是匹配瑞芯微 RV1109/RV1126 Toybrick 板子内核的驱动（`tb` 是 Toybrick 缩写），而 `galcore_puma.ko` 匹配瑞芯微 RV1109/RV1126 EVB 开发板内核的驱动。如果是其他厂商的板子，请联系相应厂商获取 `ko`。