密级状态：绝密(　　)　　秘密(　　)　　内部(　　)　　公开(√)

RKNN-Toolkit2 API 参考手册

（图形计算平台中心）

文件状态： [　] 正在修改 [√] 正式发布	当前版本：	v2.0.0-beta0
	作　　者：	HPC 团队
	完成日期：	2024-3-22
	审　　核：	熊伟
	完成日期：	2024-3-22

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
v1.6.0	HPC 团队	2023-11-15	初始版本	熊伟
v2.0.0-beta0	HPC 团队	2024-3-22	1. 增加 RK3576 相关描述 2. 增加 2.2 章节的 sparse_infer 推理接口说明 3. 更新 2.7 章节的 core_mask 参数说明 4. 增加 2.9 章节的 eval_perf 的 fix_freq 参数说明 5. 更新 2.16 章节的自定义算子用法说明	熊伟

目 录

1 概述

RKNN-Toolkit2 是为用户提供在 PC 平台上进行模型转换、推理和性能评估的开发套件。

1.1 适用芯片

RKNN-Toolkit2 当前版本所支持芯片的型号如下：

- RK3566 系列
- RK3568 系列
- RK3588 系列
- RV1103
- RV1106
- RK3562
- RK3576

注：后文用 RK3566 / RK3568 / RK3588 分别统称 RK3566 系列 / RK3568 系列 / RK3588 系列。

1.2 系统依赖说明

使用 RKNN-Toolkit2 时需要满足以下运行环境要求：

操作系统版本	Ubuntu18.04（x64）	Ubuntu20.04（x64）	Ubuntu22.04（x64）
Python 版本	3.6 / 3.7	3.8 / 3.9	3.10 / 3.11

注：

1. 具体 python 库依赖详见 doc/requirements*.txt
2. 本文档主要以 Ubuntu 20.04 / Python3.8 为例进行说明

1.3 适用的深度学习框架

RKNN-Toolkit2 支持的深度学习框架包括 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet

和 Pytorch。

它和各深度学习框架的版本对应关系如下：

RKNN-Toolkit2	Caffe	TensorFlow	TF Lite	ONNX	Darknet	Pytorch
1.4.0 1.4.2 1.5.0 1.5.2	1.0	1.12.0~2.8.0	Schema version = 3	1.7.0~1.10.0	Commit ID: 810d7f7	1.6.0~1.10.1
1.6.0	1.0	1.12.0~2.14.0	Schema version = 3	1.7.0~1.14.0	Commit ID: 810d7f7	1.6.0~1.13.1

注：

1. 依照语义版本，用某一版本 TensorFlow 导出的任何图或检查点，都可以通过相同主要版本中更高（次要或补丁）版本的 TensorFlow 来进行加载和评估，所以理论上，1.14.0 之前版本的 TensorFlow 导出的 pb 文件，RKNN-Toolkit2 1.4.0 及之后的版本都是支持的。关于 TensorFlow 版本兼容性的更多信息，可以参考官方资料：https://www.tensorflow.org/guide/versions?hl=zh-cn
2. 因为 TFLite 不同版本的 schema 之间是互不兼容的，所以构建 TFLite 模型时使用与 RKNN-Toolkit2 不同版本的 schema 可能导致加载失败。
3. RKNN-Toolkit2 使用的 caffe protocol 是基于 berkeley 官方修改的 protocol：https://github.com/BVLC/caffe/tree/master/src/caffe/proto ，commit 值为 828dd10，RKNN-Toolkit2 在此基础上新增了一些 OP。
4. ONNX release version 和 opset version、IR version 之间的关系参考 onnxruntime 官网说明：https://github.com/microsoft/onnxruntime/blob/v1.10.0/docs/Versioning.md
5. Darknet 官方 Github 链接：https://github.com/pjreddie/darknet。RKNN-Toolkit2 现在的转换规则是基于 master 分支的最新提交（commit 值：810d7f7）制定的。
6. 加载 Pytorch 模型（torchscript 模型）时，推荐使用相同版本的 Pytorch 导出模型并转为 RKNN 模型，前后版本不一致时有可能会导致转 RKNN 模型失败。

2 API 详细说明

2.1 RKNN 初始化及释放

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：

```
# 在屏幕打印详细的日志信息
rknn = RKNN(verbose=True)

…

rknn.release()
```

2.2 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	设置模型转换参数。
	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。 默认值为 None，表示所有的 mean 值为 0。
	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。 默认值为 None，表示所有的 std 值为 1。
	quant_img_RGB2BGR：表示在加载量化图像时是否需要先做 RGB2BGR 的操作。如果有多个输入，则用列表包含起来，如[True, True, False]。默认值为 False。 该配置一般用在 Caffe 的模型上，Caffe 模型训练时大多会先对数据集图像进行 RGB2BGR 转换，此时需将该配置设为 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），并不会保存在最终的 RKNN 模型中，因此如果模型的输入为 BGR，则在调用 toolkit2 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（asymmetric_quantized-16 目前版本暂不支持）。默认值为 asymmetric_quantized-8。
	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，mmse 及 kl_divergence。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。 **kl_divergence** 量化算法所用时间会比 normal 多一些，但比 mmse 会少很多，在某些场景下（feature 分布不均匀时）可以得到较好的改善效果，推荐量化数据量一般为 20-100 张左右。
	quantized_method：目前支持 layer 或者 channel。默认值为 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。
	float_dtype: 用于指定非量化情况下的浮点的数据类型，目前支持的数据类型有 float16。默认值为 float16。
	optimization_level：模型优化等级。默认值为 3。 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”、“rk3588”、“rv1103”、“rv1106”、“rk3562”和“rk3576”。该参数对大小写不敏感。默认值为 None。
	custom_string：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，方便部署时根据不同的 RKNN 模型做特殊的处理。默认值为 None。
	remove_weight：去除 conv 等权重以生成一个 RKNN 的从模型，该从模型可以与带完整权重的 RKNN 模型共享权重以减少内存消耗。默认值为 False。
	compress_weight：压缩模型权重，可以减小 RKNN 模型的大小。默认值为 False。
	single_core_mode：是否仅生成单核模型，可以减小 RKNN 模型的大小和内存消耗。默认值为 False。目前仅对 RK3588 / RK3576 生效。默认值为 False。
	model_pruning：对模型进行无损剪枝。对于权重稀疏的模型，可以减小转换后 RKNN 模型的大小和计算量。默认值为 False。
	op_target：用于指定 OP 的具体执行目标（如 NPU/CPU/GPU 等），格式为 {'op0_output_name':'cpu', 'op1_output_name':'npu', ...}。默认值为 None。 其中，'op0_output_name'和'op1_output_name'为对应 OP 的输出 tensor 名，可以通过精度分析（accuracy_analysis）功能的返回结果中获取。'cpu'和'npu'则表示该 tensor 对应的 OP 的执行目标是 CPU 或 NPU，目前可选的选项有：'cpu' / 'npu' / 'gpu' / 'auto'，其

	中，'auto'是自动选择执行目标。
	dynamic_input：用于根据用户指定的多组输入 shape，来模拟动态输入的功能。格式为[[input0_shapeA, input1_shapeA, ...], [input0_shapeB, input1_shapeB, ...], ...]。 默认值为 None，实验性功能。 假设原始模型只有一个输入，shape 为[1,3,224,224]，或者原始模型的输入 shape 本身就是动态的，如 shape 为[1,3,height,width]或[1,3,-1,-1]，但部署的时候，需要该模型支持 3 种不同的输入 shape，如[1,3,224,224], [1,3,192,192]和[1,3,160,160]，此时可以设置 dynamic_input=[[[1,3,224,224]], [[1,3,192,192]], [[1,3,160,160]]]，转换成 RKNN 模型后进行推理时，需传入对应 shape 的输入数据。 注： 1. **需要原始模型本身支持动态输入才可开启此功能，否则会报错**。 2. **如果原始模型输入 shape 本身就是动态的，则只有动态的轴可以设置不同的值。**
	quantize_weight: 在 build 接口的 do_quantization 为 False 情况下，通过对一些权重进行量化以减小 rknn 模型的大小。 默认值为 False。
	remove_reshape: 删除模型的输入和输出中可能存在的 Reshape 的 OP，以提高模型运行时性能（因为目前很多平台的 Reshape 是跑在 cpu 上，相对较慢）。 默认为 False。 **注： 开启后可能会修改模型的输入或输出节点的 shape，需要留意观察转换过程中的 warning 打印，并在部署时也需要考虑输入和输出 shape 变化的影响。**
	sparse_infer: 在已经稀疏化过的模型上进行稀疏化推理，以提高推理性能。 目前仅对 RK3576 生效。默认为 False。
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')
```

2.3 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。

2.3.1 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）路径。
	blobs：caffe 模型的二进制数据文件（.caffemodel 后缀文件）路径。
	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；默认值为 None，表示按 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',
                      blobs='./mobilenet_v2.caffemodel')
```

2.3.2 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）路径。

	inputs：模型的输入节点（tensor 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 shape，所有输入 shape 放在一个列表中。如示例中的 ssd_mobilenet_v1 模型，其输入节点对应的输入 shape 是[[1, 300, 300, 3]]。
	outputs：模型的输出节点（tensor 名），支持多个输出节点。所有输出节点名放在一个列表中。
	input_is_nchw：模型的输入的 layout 是否已经是 NCHW。默认值为 False，表示默认输入 layout 为 NHWC。
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=['Preprocessor/sub'],
                           outputs=['concat', 'concat_1'],
                           input_size_list=[[300, 300, 3]])
```

2.3.3 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）路径。
	input_is_nchw：模型的输入的 layout 是否已经是 NCHW。默认值为 False，即默认输入 layout 为 NHWC。
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')
```

2.3.4 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）路径。
	inputs：模型输入节点（tensor 名），支持多个输入节点，所有输入节点名放在一个列表中。默认值为 None，表示从模型里获取。
	input_size_list：每个输入节点对应的 shape，所有输入 shape 放在一个列表中。如 inputs 有设置，则 input_size_list 也需要被设置。默认值为 None。
	input_initial_val：设置模型输入的初始值，格式为 ndarray 的列表。默认值为 None。主要用于将某些输入固化为常量，对于不需要固化为常量的输入可以设为 None，如 [None, np.array([1])]。
	outputs：模型的输出节点（tensor 名），支持多个输出节点，所有输出节点名放在一个列表中。默认值为 None，表示从模型里获取。
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')
```

2.3.5 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）路径。
	weight：权重文件（.weights 后缀）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')
```

2.3.6 PyTorch 模型加载接口

API	**load_pytorch**
描述	加载 PyTorch 模型。 支持量化感知训练（QAT）模型，但需要将 torch 版本更新至 1.9.0 以上。
参数	model：PyTorch 模型文件（.pt 后缀）路径，而且需要是 torchscript 格式的模型。
	input_size_list：每个输入节点对应的 shape，所有输入 shape 放在一个列表中。
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]])
```

2.4 构建 RKNN 模型

API	**build**
描述	构建 RKNN 模型。
参数	do_quantization：是否对模型进行量化。默认值为 True。
	dataset：用于量化校正的数据集。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 None，表示不进行调整。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')
```

2.5 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。
	cpp_gen_cfg：是否生成 C++部署示例。默认值为 False。 生成文件 - 模型路径同文件夹下，生成 rknn_deploy_demo 文件夹、说明文档。 支持功能 - 验证模型推理时，各 CAPI 接口耗时 - 验证推理结果的余弦精度 - 支持常规 API 接口 - 支持图片/npy 输入 **#请注意，目前 RV1106/RV1103 暂不支持此功能**
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')
```

2.6 加载 RKNN 模型

API	**load_rknn**
描述	加载 RKNN 模型。 加载完 RKNN 模型后，不需要再进行模型配置、模型加载和构建 RKNN 模型的步骤。 并且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。
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')
```

2.7 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。
参数	target：目标硬件平台，支持“rk3566”、“rk3568”、“rk3588”、“rv1103”、“rv1106”、“rk3562”、“rk3576”。默认值为 None，即在 PC 使用工具时，模型在模拟器上运行。 注：target 设为 None 时，需要先调用 build 或 hybrid_quantization 接口才可让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。默认值为 False。 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。（目前版本该参数暂不支持）
	core_mask：设置运行时的 NPU 核心。支持的平台为 RK3588 / RK3576，支持的配置如下： RKNN.NPU_CORE_AUTO：表示自动调度模型，自动运行在当前空闲的 NPU 核上。 RKNN.NPU_CORE_0：表示运行在 NPU0 核心上。 RKNN.NPU_CORE_1：表示运行在 NPU1 核心上。 RKNN.NPU_CORE_2：表示运行在 NPU2 核心上。 RKNN.NPU_CORE_0_1：表示同时运行在 NPU0、NPU1 核心上。 RKNN.NPU_CORE_0_1_2：表示同时运行在 NPU0、NPU1、NPU2 核心上。 RKNN.NPU_CORE_ALL：表示根据平台自动配置 NPU 核心数量。 默认值为 RKNN.NPU_CORE_AUTO。
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')
```

2.8 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，并返回推理结果。 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：待推理的输入列表，格式为 ndarray。
	data_format：输入数据的 layout 列表，“nchw”或“nhwc”，只对 4 维的输入有效。默认值为 None，表示所有输入的 layout 都为 NHWC。
	inputs_pass_through：输入的透传列表。默认值为 None，表示所有输入都不透传。 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，而是直接将输入传给 NPU。 该参数的值是一个列表，比如要透传 input0，不透传 input1，则该参数的值为[1, 0]。
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] score:0.928223 class:"Shih-Tzu"
[ 155] score:0.063171 class:"Pekinese, Pekingese, Peke"
[ 205] score:0.004299 class:"Lhasa, Lhasa apso"
[ 284] score:0.003096 class:"Persian cat"
```

```
[ 285] score:0.000171 class:"Siamese cat, Siamese"
```

2.9 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 / RK3588 / RV1103 / RV1106 / RK3562 / RK3576 上。如果调用“**init_runtime**”的接口来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。
	is_print：是否打印性能信息，默认值为 True。
	fix_freq：是否固定硬件设备的频率，默认值为 True。
返回值	perf_result：性能信息（字符串）。

举例如下：

```
# 对模型性能进行评估
perf_detail = rknn.eval_perf()
```

2.10 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 / RK3588 / RV1103 / RV1106 / RK3562 / RK3576 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： { 'weight_memory': 3698688, 'internal_memory': 1756160,

	'other_memory': 484352, 'total_memory': 5939200, } ● 'weight_memory' 字段表示运行时模型权重的内存占用。 ● 'internal_memory' 字段表示运行时模型中间 tensor 内存占用。 ● 'other_memory' 字段表示运行时其他的内存占用。 ● 'total_model_allocation' 表示运行时的总内存占用，即权重、中间 tensor 和其他的内存占用之和。

举例如下：

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

如 examples/caffe/mobilenet_v2，它在 RK3588 上运行时内存占用情况如下：

```
==============================================================
                Memory Profile Info Dump
==============================================================
NPU model memory detail(bytes):
    Weight Memory: 3.53 MiB
    Internal Tensor Memory: 1.67 MiB
    Other Memory: 473.00 KiB
    Total Memory: 5.66 MiB

INFO: When evaluating memory usage, we need consider
the size of model, current model size is: 4.09 MiB
==============================================================
```

2.11 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 / RK3588 / RV1103 / RV1106 / RK3562 / RK3576 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()
print(sdk_version)
```

返回的 SDK 信息类似如下：

```
=============================================
RKNN VERSION:
    API: 1.5.2 (8babfea build@2023-08-25T02:31:12)
    DRV: rknn_server: 1.5.2 (8babfea build@2023-08-25T10:30:12)
    DRV: rknnrt: 1.5.3b13 (42cbca6f5@2023-10-27T10:13:21)
=============================================
```

2.12 混合量化

2.12.1 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：见构建 RKNN 模型的 dataset 说明。
	rknn_batch_size：见构建 RKNN 模型的 rknn_batch_size 说明。
	proposal：产生混合量化的配置建议值。 默认值为 False。
	proposal_dataset_size：proposal 使用的 dataset 的张数。默认值为 1。 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。
	custom_hybrid：用于根据用户指定的多组输入和输出名，选取混合量化对应子图。 格式为[[input0_name, output0_name], [input1_name, output1_name], …]。默认值为 None。

	注：输入和输出名应根据生成的临时模型文件({model_name}.model)来选择。
返回值	0：成功。
	-1：失败。

举例如下：

```
# 调用 hybrid_quantization_step1 产生量化配置文件
ret = rknn.hybrid_quantization_step1(dataset='./dataset.txt')
```

2.12.2 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：hybrid_quantization_step1 生成的临时模型文件（{model_name}.model）路径。
	data_input：hybrid_quantization_step1 生成的数据文件（{model_name}.data）路径。
	model_quantization_cfg：hybrid_quantization_step1 生成并经过修改后的模型量化配置文件（{model_name}.quantization.cfg）路径。
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')
```

2.13 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，并进行量化精度分析。

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。** 2. **如未指定 target，且原始模型应该为已量化的模型（QAT 模型），则会调用失败。** 3. **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。默认值为'./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 simulator 上量化模型逐层运行时每一层的结果与 golden 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（single_error cosine），更详细的信息请查看 error_analysis.txt 文件。 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）等信息，更详细的信息请查看 error_analysis.txt 文件。
	target：目标硬件平台，支持“rk3566”、“rk3568”、“rk3588”、“rv1103”、“rv1106”、“rk3562”、“rk3576”，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：

```
# Accuracy analysis
ret = rknn.accuracy_analysis(inputs=['./dog_224x224.jpg'])
```

2.14 获取设备列表

API	**list_devices**
描述	列出已连接的 RK3566 / RK3568 / RK3588 / RV1103 / RV1106 / RK3562 / RK3576。 注：目前设备连接模式有两种：ADB 和 NTB。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：

```
rknn.list_devices()
```

返回的设备列表信息如下：

```
**************************
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。

2.15 导出加密模型

该接口的功能是将普通的 RKNN 模型进行加密，得到加密后的模型。

API	**export_encrypted_rknn_model**
描述	根据用户指定的加密等级对普通的 RKNN 模型进行加密。
参数	input_model: 待加密的 RKNN 模型路径。
	output_model: 模型加密后的保存路径。默认值为 None，表示使用 {original_model_name}.crypt.rknn 作为加密后的模型名字。
	crypt_level: 加密等级，有 1，2 和 3 三个等级。默认值为 1。 等级越高，安全性越高，解密越耗时；反之，安全性越低，解密越快。数据类型为整型，
返回值	0：成功。
	-1：失败。

举例如下：

```
ret = rknn.export_encrypted_rknn_model('test.rknn')
```

2.16 注册自定义算子

该接口的功能是注册一个自定义算子。

API	**reg_custom_op**
描述	注册用户提供的自定义算子类。目前只支持 ONNX 模型。
参数	custom_op: 用户自定义的算子类。用于用户需要自定义一个不存在于 ONNX 算子规范内的新算子。该算子的 op_type 推荐以“cst”字符开头，并且其算子类的 shape_infer 和 compute 函数需要用户自己实现。

	注：custom_op 算子类仅用于模型转换并生成带有自定义算子的 RKNN 模型，在设备端进行部署时还需要参考《RKNN 用户指南》的 5.5 章节。
返回值	0：成功。
	-1：失败。

举例如下：

```
import numpy as np
from rknn.api.custom_op import get_node_attr
class cstSoftmax:
    op_type = 'cstSoftmax'
    def shape_infer(self, node, in_shapes, in_dtypes):
        out_shapes = in_shapes.copy()
        out_dtypes = in_dtypes.copy()
        return out_shapes, out_dtypes
    def compute(self, node, inputs):
        x = inputs[0]
        axis = get_node_attr(node, 'axis')
        x_max = np.max(x, axis=axis, keepdims=True)
        tmp = np.exp(x - x_max)
        s = np.sum(tmp, axis=axis, keepdims=True)
        outputs = [tmp / s]
        return outputs

ret = rknn.reg_custom_op(cstSoftmax)
```