RKNN-Toolkit2 API Reference

ID: RK-YH-YF-412

Release Version: V2.3.2

Release Date: 2025-4-1

Security Level: □Top-Secret □Secret □Internal ■Public

DISCLAIMER

THIS DOCUMENT IS PROVIDED “AS IS”. ROCKCHIP ELECTRONICS CO., LTD.(“ROCKCHIP”)DOES NOT PROVIDE ANY WARRANTY OF ANY KIND, EXPRESSED, IMPLIED OR OTHERWISE, WITH RESPECT TO THE ACCURACY, RELIABILITY, COMPLETENESS,MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE OR NON-INFRINGEMENT OF ANY REPRESENTATION, INFORMATION AND CONTENT IN THIS DOCUMENT. THIS DOCUMENT IS FOR REFERENCE ONLY. THIS DOCUMENT MAY BE UPDATED OR CHANGED WITHOUT ANY NOTICE AT ANY TIME DUE TO THE UPGRADES OF THE PRODUCT OR ANY OTHER REASONS.

Trademark Statement

"Rockchip", "瑞芯微", "瑞芯" shall be Rockchip’s registered trademarks and owned by Rockchip. All the other trademarks or registered trademarks mentioned in this document shall be owned by their respective owners.

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This is a RKNN-Toolkit2 API Reference.

RKNN-Toolkit2 is a development kit that provides users with model conversion, inference and performance evaluation on PC platforms.

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Revision History

Version	Author	Date	Change Description	Reviewer
V1.6.0	HPC Team	2023-11-15	Initial version	Vincent
V2.0.0 -beta0	HPC Team	2024-03-22	1. Add RK3576-related description 2. Add description of the sparse_infer inference interface in Chapter 2.2 3. Update description of the core_mask parameter in Chapter 2.7 4. Add description of the fix_freq parameter of eval_perf in Chapter 2.9 5. Update usage instructions of the custom operator in Chapter 2.16	Vincent
V2.1.0	HPC Team	2024-07-29	1. Update description of the quantized_dtype parameter in Chapter 2.2 2. Add description of the enable_flash_attention parameter in Chapter 2.2 3. Add description of the fallback_prior_device parameter in Chapter 2.7 4. Remove description of the cpp_gen_cfg parameter in Chapter 2.5 5. Add codegen instructions in Chapter 2.17	Vincent
V2.2.0	HPC Team	2024-09-04	1. Add RV1106B-related description 2. Add Python3.12 description in Chapter 1.2	Vincent
V2.3.0	HPC Team	2024-11-04	1. Add ARM64 description 2. Update version of Deep Learning Framework in Chapter 1.3 3. Update description of the quantized_dtype parameter in Chapter 2.2	Vincent
V2.3.2	HPC Team	2025-4-1	1. Add description of automatic hybrid quantization 2. Add description of group quantization parameter 3. Add description of GDQ quantization algorithm 4. Add RV1126B-related description	Vincent

Contents

1 Requirement

1.1 Applicable chip model

- RV1103
- RV1103B
- RV1106
- RV1106B
- RV1126B
- RK2118
- RK3562
- RK3566 series
- RK3568 series
- RK3576 series
- RK3588 series

Note: RK3566 / RK3568 / RK3576 / RK3588 are collectively referred to as RK3566 series / RK3568 series / RK3576 series / RK3588 series in the following text.

1.2 Requirements Dependencies

It is recommended to meet the following requirements in the operating system environment:

Operating system version	Ubuntu18.04 (x64)	Ubuntu20.04 (x64)	Ubuntu22.04 (x64)	Ubuntu24.04 (x64)
Python version	3.6 / 3.7	3.8 / 3.8	3.10 / 3.11	3.12

For ARM64:

Operating system version	Debian10 (x64)	Debian11 (x64)	Debian12 (x64)
Python version	3.6 / 3.7	3.8 / 3.8	3.10 / 3.11 / 3.12

Note:

1. For more detail about python library dependencies, see doc/requirements*.txt
2. This document mainly uses Ubuntu 20.04 / Python3.8 as an example.

1.3 Supported Deep Learning Framework

The deep learning frameworks supported by RKNN-Toolkit2 include Caffe, TensorFlow, TensorFlow Lite, ONNX, DarkNet and PyTorch.

The corresponding relationship between RKNN-Toolkit2 and the version of each deep learning framework is as follows:

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
2.0.0 2.1.0 2.2.0	1.0	1.12.0 ~2.14.0	Schema version=3	1.7.0 ~1.14.1	Commit ID: 810d7f7	1.10.1 ~2.1.0
2.3.0 2.3.2	1.0	1.12.0 ~2.14.0	Schema version=3	1.7.0 ~1.17.0	Commit ID: 810d7f7	1.10.1 ~2.4.0

Note:

1. Due to the compatibility of TensorFlow versions, the pb files exported by versions before TensorFlow 1.12.0 are also supported in theory.. For more information about the compatibility of different TensorFlow versions, please refer to official documentation: https://www.tensorflow.org/guide/versions
2. Since the schemas of different versions of TFLite are incompatible with each other, TFLite model exported from a different schema compared to the schema version RKNN-Toolkit2 relies may cause loading failure.
3. The Caffe protocols used by RKNN-Toolkit2 is the protocol based on the official modification of berkeley. The protocol based on berkeley's official modification comes from: https://github.com/BVLC/caffe/tree/master/src/caffe/proto and the commit ID is 828dd10. RKNN-Toolkit2 adds some OPs on this basis.
4. For the relationship between ONNX release versions and opset versions and IR versions, please refer to the onnxruntime official website: https://github.com/microsoft/onnxruntime/blob/v1.10.0/docs/Versioning.md
5. The official Github link of DarkNet: https://github.com/pjreddie/darknet. RKNN-Toolkit2's current conversion rules are based on the latest submission of the master branch (commit number: 810d7f7).
6. When loading the PyTorch model (torchscript model), it is recommended using the same version of PyTorch to export model and convert model to RKNN model. Inconsistency may result in failure when transferring to the RKNN model.
7. ARM64 only supports the PyTorch and ONNX. Other framework is not supported at the moment.

2 RKNN-Toolkit2 API description

2.1 RKNN initialization and release

The initialization/release function group consists of API interfaces to initialize and release the RKNN object as needed. The **RKNN()** must be called before using all the API interfaces of RKNN-Toolkit2, and call the **release()** method to release the object when task finished.

When the RKNN object is initing, the users can set **verbose** and **verbose_file** parameters, used to print detailed log information of model loading, building and so on. The data type of **verbose** parameter is bool. If the value of this parameter is set to True, the RKNN-Toolkit2 will print detailed log information. The data type of **verbose_file** is string. If the value of this parameter is set to a file path, the detailed log information will be written to this file (**the verbose also need be set to True**).

The sample code is as follows:

```
# Print the detailed log information.
rknn = RKNN(verbose=True)

…

rknn.release()
```

2.2 Model configuration

Before the RKNN model is built, the model needs to be configured first through the **config** interface.

API	**config**
Description	Set model convert parameters.
Parameter	**mean_values:** The mean values of the input. The parameter format is a list. The list contains one or more mean sublists. The multi-input model corresponds to multiple sublists. The length of each sublist is consistent with the number of channels of the input. For example, if the parameter is [[128,128,128]], it means an input subtract 128 from the values of the three channels. The default value is None, means all means is zero.
	std_values: The normalized value of the input. The parameter format is a list. The list contains one or more normalized value sublists. The multi-input model corresponds to multiple sublists. The length of each sublist is consistent with the number of channels of the input. For example, if the parameter is [[128,128,128]], it means the value of the three channels of an input minus the average value and then divide by 128. The default value is None, means all stds is one.

API	config
	quant_img_RGB2BGR: Indicates whether the RGB2BGR operation needs to be done first when loading the quantized image. If there are multiple inputs, the corresponding parameters for each input is split with ‘,’, such as [True, True, False]. The default value is False. This configuration is generally used on the Caffe model. Most of the Caffe model training will perform RGB2BGR conversion on the dataset image firstly. At this time, the configuration needs to be set to True. In addition, this configuration is only valid for the quantized image format of jpg/png/bmp. This configuration is ignored when the npy format is read. Therefore, when the model input is BGR, npy also needs to be in BGR format. **This configuration is only used to read the quantize image in the quantization stage (build interface) or in quantitative accuracy analysis (accuracy_analysis interface), and will not be recorded in the final RKNN model. Therefore, if the input of the model is BGR, you need to ensure that the imported image data is also in BGR format before calling the inference of the toolkit or the run function of the C-API.**
	quantized_dtype: Quantization type, the quantization types currently supported are "w8a8", "w4a16", "w8a16", "w4a8", "w16a16i" and "w16a16i_dfp". The default value is "w8a8". **- w8a8:** The weight is 8bit asymmetric quantitative accuracy, and the activation value is 8bit asymmetric quantitative accuracy. **(RK2118 not supported)** **- w4a16:** The weight is 4bit asymmetric quantitative accuracy, the activation value is 16bit floating point accuracy. **(Only RK3576/RV1126B supported)** **- w8a16:** The weight is 8bit asymmetric quantitative accuracy, the activation value is 16bit floating point accuracy. **(Only RK3562 supported)** **- w4a8:** The weight is 4bit asymmetric quantitative accuracy, the activation value is 8bit asymmetric quantitative accuracy. **(Not supported yet)** **- w16a16i:** The weight is 16bit asymmetric quantitative accuracy, the activation value is 16bit asymmetric quantitative accuracy. **(Only RV1103/RV1106 supported)** **- w16a16i_dfp:** The weight is 16bit dynamic fixed-point quantitative accuracy, and the activation value is 16bit dynamic fixed-point quantitative accuracy. **(Only RV1103/RV1106 supported)**

API	config
	quantized_algorithm: The quantization algorithm used when calcaulating the quantization parameters of each layer. Currently support: **normal**, **mmse**, **kl_divergence** and **gdq**. The default value is **normal**. The characteristic of **normal** quantization algorithm is fast. The recommended quantization data is generally about 20-100 pieces. with more data, the accuracy may not be further improved. The **mmse** quantization algorithm is slower due to the violent iteration method, but usually has higher accuracy than normal. The recommended quantization data is generally about 20-50 pieces. Users can also increase or decrease the amount of data appropriately according to the length of the quantization time. The **kl_divergence** quantization algorithm will take more time than normal, but will be much less than mmse. In some scenarios(when the feature distribution is uneven), better improvement effects can be obtained by “kl_divergence”. the recommended quantization data is generally about 20-100 pieces. The **gdq** quantization algorithm is only valid at w4a16 and w8a16, which can effectively improve the weight accuracy of the w4a16 and w8a16. the recommended quantization data is more than 200 pieces.
	quantized_method: Currently support layer, channel or group{SIZE}. The default value is channel. - layer: each weight has only one set of quantization parameters. - channel: each channel of weight has its own set of quantization parameters. usually the channel will be more accurate than the layer. - group{SIZE}: On the basis of 'channel', the weight of each output channel is subdivided into multiple groups according to {SIZE} on the input channel. each group has a set of quantization parameters. Usually, group{SIZE} will be more accurate than channel. {SIZE} is the multiple value of 32 between 32 and 256, such as 'group32' or 'group128'. group{SIZE} is currently only valid when quantized_dtype = w4a16.
	float_dtype: Used to specify the data type of floating in the non-quantized case, the data types currently supported are float16. The default value is float16.
	optimization_level: Model optimization level. The default value is 3. By modifying the model optimization level, you can turn off some or all of the optimization rules used in the model conversion process. The default value of this parameter is 3, and all optimization options are turned on. When the value is 2 or 1, turn off some optimization options that may affect the accuracy of some models. Turn off all optimization options when the value is 0.
	target_platform: Specify which target chip platform the RKNN model is based on. 'rv1103', 'rv1103b', 'rv1106', 'rv1106b', 'rv1126b', 'rk2118', 'rk3562', 'rk3566', 'rk3568', 'rk3576' and 'rk3588' are currently supported. The default value is None.
	custom_string: Add custom string information to RKNN model, then can query the information at runtime. The default value is None.

API	config
	remove_weight: Remove the weights to generate a RKNN slave model that can share weights with the full weighted RKNN model to reduce memory consumption. The default value is False.
	compress_weight: Compress the weights of the model, which can reduce the size of RKNN model. The default value is False.
	single_core_mode: Whether to generate only single-core model, which can reduce the size and memory consumption of the RKNN model. The default value is False. Only valid for RK3588 / RK3576. The default value is False.
	model_pruning: Pruning the model that can reduce the size and calculation of the transformed RKNN model for models with sparse weights. The default value is False.
	op_target: Used to specify the target of each operation (NPU/CPU/GPU etc.), the format is {'op0_output_name':'cpu', 'op1_output_name':'cpu', ...}, or through op_type, the format is {'op_type':'cpu‘, 'op0_output_name':'cpu'....}, also supporting mix use of these two, e.g., {'op_type':'cpu‘, 'op0_output_name':'cpu'....}. The default value is None. 'op0_output_name' and 'op1_output_name' are the output tensor names of the corresponding OP, which can be obtained from the returned results of the accuracy_analysis feature. 'cpu' and 'npu' indicate that the execution target of the OP corresponding to this tensor is CPU or NPU. The currently available options are: 'cpu' / 'npu' / 'gpu' / 'auto', and 'auto' is for automatically selecting the execution target. Through op_type, it is obtained from returned results of the accuracy_analysis feature. This can set up the execution target of all OP with this op_type within this model, e.g., {'Add':'cpu', 'Expand':'cpu',....}.
	dynamic_input: Simulate the function of dynamic input according to multiple sets of input shapes specified by the user. the format is [[input0_shapeA, input1_shapeA, ...], [input0_shapeB, input1_shapeB, ...], ...]. The default value is None, experimental. For example, the input shape of the original model is [1,3,224,224] or [1,3,height,width] or [1,3,-1,-1], but the model for deploy needs to support 3 input shapes: [1,3,224,224], [1,3,192,192] and [1,3,160,160], you can set dynamic_input=[[[1,3,224,224]], [[1,3,192,192]], [[1,3,160,160]]]. When converting to the RKNN model for inference, the input data corresponding to the shape needs to be passed in. Note: **1. This function can only be enabled when the original model itself supports dynamic input, otherwise an error will be reported.** **2. If the original model input shape itself is dynamic, only the dynamic axes can set different values.**
	quantize_weight: When 'do_quantization' of rknn.build is False, reduce the size of the rknn model by quantizing some weights. The default value is False.

API	config
	remove_reshape: Remove possible 'Reshape' in model inputs and outputs to improve model runtime performance. default is False. Note: **Enabling this configuration may modify the shape of the input or output nodes of the model. You need to pay attention to warning printing during the conversion process, and you also need to consider the impact of input and output shape changes when deploying.**
	sparse_infer: Sparse inference on already sparsified models to improve performance. Only valid for RK3576/RV1126B. default is False.
	enable_flash_attention: Whether to enable Flash Attention. default is False. Note: FlashAttention is Based on https://arxiv.org/abs/2307.08691, acceleration and reduced bandwidth usage can be achieved through cache in-loop implementation. However, this may result in an increase in model size. Please choose whether to enable this feature based on the specific scenario and model. For more details, please see the 'exSDPAttention' description in 'RKNN Compiler Support Operator List'.
	auto_hybrid_cos_thresh: The threshold for cosine distance in automatic hybrid quantization during model quantization. The default value is 0.98.
	auto_hybrid_euc_thresh: The threshold for euclidean distance in automatic hybrid quantization during model quantization. The default value is None, meaning it is not enabled.
Return Value	None.

The sample code is as follows:

```
# 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 Loading model

RKNN-Toolkit2 currently supports load non-RKNN models of Caffe, TensorFlow, TensorFlow Lite, ONNX, DarkNet, PyTorch. There are different calling interfaces when loading models, the loading interfaces are described in detail below.

2.3.1 Loading Caffe model

API	load_caffe
Description	Load Caffe model. (Unavailable in ARM64)
Parameter	**model:** The path of Caffe model structure file (suffixed with ".prototxt").
	blobs: The path of Caffe model binary data file (suffixed with ".caffemodel").

API	**load_caffe**
	input_name: When the Caffe model has multiple inputs, you can specify the order of the input layer names through this parameter, such as ['input1','input2','input3'],note that the name needs to be consistent with the model input name； The default value is None, means the sequence is automatically given by the Caffe model file (file suffix with .prototxt).
Return Value	0: Import successfully.
	-1: Import failed.

The sample code is as follows:

```
# Load the mobilenet_v2 Caffe model in the current path
ret = rknn.load_caffe(model='./mobilenet_v2.prototxt',
                      blobs='./mobilenet_v2.caffemodel')
```

2.3.2 Loading TensorFlow model

API	**load_tensorflow**
Description	Load TensorFlow model. (Unavailable in ARM64)
Parameter	**tf_pb:** The path of TensorFlow model file (suffixed with ".pb").
	inputs: The input node (operand name) of model, input with multiple nodes is supported now. All the input node string are placed in a list.
	input_size_list: The shapes of input node, all the input shape are placed in a list. As in the example of ssd_mobilenet_v1 model, the input_size_list parameter should be set to [[1,300,300,3]].
	outputs: The output node (operand name) of model, output with multiple nodes is supported now. All the output nodes are placed in a list.
	input_is_nchw: Whether the input layout of the model is already NCHW. The default value is **False**, means the default input layout is NHWC.
Return value	0: Import successfully.
	-1: Import failed.

The sample code is as follows:

```
# Load ssd_mobilenet_v1_coco_2017_11_17 TF model in the current path
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 Loading TensorFlow Lite model

API	**load_tflite**
Description	Load TensorFlow Lite model. (Unavailable in ARM64)
Parameter	**model:** The path of TensorFlow Lite model file (suffixed with ".tflite").
	input_is_nchw: Whether the input layout of the model is already NCHW. The default value is **False**, that is, the default input layout is NHWC.
Return Value	0: Import successfully.
	-1: Import failed.

The sample code is as follows:

```
# Load the mobilenet_v1 TF-Lite model in the current path
ret = rknn.load_tflite(model='./mobilenet_v1.tflite')
```

2.3.4 Loading ONNX model

API	**load_onnx**
Description	Load ONNX model.
Parameter	**model:** The path of ONNX model file (suffixed with ".onnx").
	inputs: The input node (operand name) of model, input with multiple nodes is supported now. All the input node string are placed in a list. The default value is None, means get from model.
	input_size_list: The shapes of input node, all the input shape are placed in a list. If inputs set, the input_size_list should be set also. defualt is None.
	input_initial_val: Set the initial value of the model input, the format is ndarray list. The default value is None. Mainly used to fix some input as constant, For the input that does not need to be fix as a constant, it can be set to None, such as [None, np.array([1])].
	outputs: The output node (operand name) of model, output with multiple nodes is supported now. All the output nodes are placed in a list. The default value is None, means get from model.
Return Value	0: Import successfully.
	-1: Import failed.

The sample code is as follows:

```
# Load the arcface onnx model in the current path
ret = rknn.load_onnx(model='./arcface.onnx')
```

2.3.5 Loading DarkNet model

API	**load_darknet**
Description	Load DarkNet model. (Unavailable in ARM64)
Parameter	**model:** The path of DarkNet model structure file (suffixed with ".cfg").
	weight: The path of weight file (suffixed with ".weight").
Return Value	0: Import successfully.
	-1: Import failed.

The sample code is as follows:

```
# Load the yolov3-tiny DarkNet model in the current path
ret = rknn.load_darknet(model='./yolov3-tiny.cfg',
                    weight= './yolov3.weights')
```

2.3.6 Loading PyTorch model

API	**load_pytorch**
Description	Load PyTorch model. Support the Quantization Aware Training (QAT) model, but need update torch version to '1.9.0' or higher.
Parameter	**model:** The path of PyTorch model structure file (suffixed with ".pt"), and need a model in the torchscript format.
	input_size_list: The shapes of input node, all the input shape are placed in a list.
Return Value	0: Import successfully.
	-1: Import failed.

The sample code is as follows:

```
# Load the PyTorch model resnet18 in the current path
ret = rknn.load_pytorch(model='./resnet18.pt',
                    input_size_list=[[1,3,224,224]])
```

2.4 Building RKNN model

API	build
Description	Build corresponding RKNN model according to imported model.
Parameter	**do_quantization:** Whether to quantize the model. The default value is True.
	dataset: A input dataset for rectifying quantization parameters. Currently supports text file format, the user can place the path of picture(jpg or png) or npy file which is used for rectification. A file path for each line. Such as: a.jpg b.jpg or a.npy b.npy If there are multiple inputs, the corresponding files are divided by space. Such as: a.jpg a2.jpg b.jpg b2.jpg or a.npy a2.npy b.npy b2.npy Note: It is generally recommended to select the quantization image which is consistent with the prediction scene.
	rknn_batch_size: Use to adjust batch size of input. default is None. If greater than 1, NPU can inference multiple frames of input image or input data in one inference. For example, original input of MobileNet is [1, 224, 224, 3], output shape is [1, 1001]. When rknn_batch_size is set to 4, the input shape of MobileNet becomes [4, 224, 224, 3], output shape becomes [4, 1001]. Note: 1. rknn_batch_size can improve performance (increase core utilization) only on NPU multi-core platforms, so the value of rknn_batch_size is recommended to match the number of NPU cores. 2. After the rknn_batch_size is modified, the shape of input and output will be modified. So the inputs of inference should be set to correct size. It`s also needed to process the returned outputs on post processing.
	auto_hybrid: Whether to enable automatic hybrid quantization to adjust accuracy or overflow. The default value is False, which means no adjustment is performed. When the model is being quantized, enabling auto_hybrid will convert operations with cosine distance and euclidean distance below the specified thresholds to FP16 computation (currently only supported for 'w8a8' quantization). When the model is not quantized, enabling auto_hybrid will convert operations that exceed the FP16 value range to INT16 computation. Note: 1. The thresholds for cosine distance and euclidean distance can be configured through using auto_hybrid_cos_thresh and auto_hybrid_euc_thresh in the config interface.

API	build
Return value	0: Build successfully.
	-1: Build failed.

The sample code is as follows:

```
# Build and quantize RKNN model
ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
```

2.5 Export RKNN model

The RKNN model built by ‘build’ interface can be saved as a file, it can used to model deployment.

API	export_rknn
Description	Save RKNN model in the specified file (suffixed with ".rknn").
Parameter	**export_path:** The path of generated RKNN model file.
Return Value	0: Export successfully.
	-1: Export failed.

The sample code is as follows:

```
# save the built RKNN model as a mobilenet_v1.rknn file in the current path
ret = rknn.export_rknn(export_path='./mobilenet_v1.rknn')
```

2.6 Loading RKNN model

API	load_rknn
Description	Load RKNN model. After loading the RKNN model, there is no need to perform the steps of model configuration, loading model and building RKNN model. And the loaded model is limited to connecting to the NPU hardware for inference or performance data acquisition. It can not be used for simulator or accuracy analysis.
Parameter	**path:** The path of RKNN model file.
Return Value	0: Load successfully.
	-1: Load failed.

The sample code is as follows:

```
# Load the mobilenet_v1 RKNN model in the current path
ret = rknn.load_rknn(path='./mobilenet_v1.rknn')
```

2.7 Initialize the runtime environment

Before inference or performance evaluation, the runtime environment must be initialized. This interface determines the type of runtime (hardware platform or software simulator).

API	init_runtime
Description	Initialize the runtime environment.
Parameter	**target:** Target hardware platform, now supports 'rv1103', 'rv1103b', 'rv1106', 'rv1106b', 'rv1126b', 'rk3562', 'rk3566', 'rk3568', 'rk3576' and 'rk3588'. The default value is "None", means model runs on simulator. Note: When target is set to None, the build or hybrid_quantization interface needs to be called first.
	device_id: Device identity number, if multiple devices are connected to PC, this parameter needs to be specified which can be obtained by calling "**list_devices**" interface. The default value is None.
	perf_debug: Debug mode option for performance evaluation. In debug mode, the running time of each layer can be obtained, otherwise, only the total running time of model can be given. The default value is False.
	eval_mem: Whether enter memory evaluation mode. If set True, the eval_memory interface can be called later to fetch memory usage of model running. The default value is False.
	async_mode: Whether to use asynchronous mode. The default value is False. When calling the inference interface, it involves setting the input picture, model running, and fetching the inference result. If the asynchronous mode is enabled, setting the input of the current frame will be performed simultaneously with the inference of the previous frame, so in addition to the first frame, each subsequent frame can hide the setting input time, thereby improving performance. In asynchronous mode, the inference result returned each time is the previous frame. **(Not Supported yet)**

API	init_runtime
	core_mask: Sets the NPU cores at runtime. The supported platform is RK3588 / RK3576, and the supported configurations are as follows: RKNN.NPU_CORE_AUTO: Indicates the automatic scheduling model, which automatically runs on the currently idle NPU core. RKNN.NPU_CORE_0: Indicates running on the NPU0 core. RKNN.NPU_CORE_1: Indicates running on the NPU1 core. RKNN.NPU_CORE_2: Indicates running on the NPU2 core. RKNN.NPU_CORE_0_1: Indicates running on NPU0 and NPU1 cores at the same time. RKNN.NPU_CORE_0_1_2: Indicates running on NPU0, NPU1, NPU2 cores at the same time. RKNN.NPU_CORE_ALL: Indicates running on the number of NPU cores depending on the platform. The default value is "RKNN.NPU_CORE_AUTO". Note: RK3576 has only 2 cores, so NPU_CORE_2 and NPU_CORE_0_1_2 cannot be set.
	fallback_prior_device: set fallback prior device when OP is not supported by NPU. currently support: 'gpu' or 'cpu', 'gpu' is only valid for platform which has gpu hardware. The default value is 'cpu'.
Return Value	0: Initialize the runtime environment successfully.
	-1: Initialize the runtime environment failed.

The sample code is as follows:

```
# Initialize the runtime environment
ret = rknn.init_runtime(target='rk3566')
```

2.8 Inference with RKNN model

This interface kicks off the RKNN model inference and get the result of inference.

API	inference
Description	Use the model to perform inference with specified input and get the inference result. If the target is set to Rockchip NPU when initializing the runtime environment, the inference of model is performed on the specified hardware platform. If the target is not set, the inference of model is performed on the simulator.
Parameter	**inputs:** Inputs list to be inferred, The object type is ndarray.
	data_format: The layout list of input data. "nchw" or "nhwc" , only valid for 4-dims input. The default value is None, means all inputs layout is "nhwc".

API	inference
	inputs_pass_through: The pass_through flag. The default value is None, means all input is not pass through. In non-pass_through mode, the tool will reduce the mean, divide the variance, etc. before the input is passed to the NPU driver; in pass_through mode, these operations will not be performed. The value of this parameter is an list. For example, to pass input0 and not input1, the value of this parameter is [1, 0].
Return Value	results: The result of inference, the object type is ndarray list.

The sample code is as follows:

For classification model, such as mobilenet_v1, the code is as follows (refer to *example/tfilte/mobilenet_v1* for the complete code):

```
# Preform inference for a picture with a model and get a top-5 result
outputs = rknn.inference(inputs=[img])
show_outputs(outputs)
```

The result of top-5 is as follows:

```
-----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 Evaluate model performance

API	eval_perf
Description	Evaluate model performance. Model must run on RV1103 / RV1103B / RV1106 / RV1106B / RV1126B / RK3562 / RK3566 / RK3568 / RK3576 / RK3588 which connected to PC.If setting perf_debug to False when initializing runtime environment via the interface of "**init_runtime**", the performance information is obtained from hardware, which only contains the total running time of model. If the perf_debug is set to True, the running time of each layer will also be captured in detail.
Parameter	**is_print:** Whether to print performance information. The default value is True.
	fix_freq: Whether to fix hardware frequency. The default value is True.
Return Value	perf_result: Performance information (strings).

The sample code is as follows:

```
# Evaluate model performance
perf_detail = rknn.eval_perf()
```

2.10 Evaluating memory usage

API	**eval_memory**
Description	Fetch memory usage when model is running on hardware platform. Model must run on RV1103 / RV1103B / RV1106 / RV1106B / RV1126B / RK3562 / RK3566 / RK3568 / RK3576 / RK3588 which connected to PC.
Parameter	**is_print:** Whether to print memory evaluation results in the canonical format. The default value is True.

Return Value	memory_detail: Detail information of memory usage. Data format is dictionary.
	Data shows as below:
	{
	    'weight_memory': 3698688,
	    'internal_memory': 1756160,
	    'other_memory': 484352,
	    'total_memory': 5939200,
	}
	- The 'weight_memory' represents the memory footprint of the weights in the model.
	- The 'internal_memory' represents the memory usage of the internal tensor in the model.
	- The 'other_memory' represents the memory usage of other tensor in the model.
	- The 'total_memory' represents the memory footprint of the model, that is, the sum of the weight, internal tensor and other memory.

The sample code is as follows:

```
# eval memory usage
memory_detail = rknn.eval_memory()
```

For examples/caffe/mobilenet_v2 in examples directory, the memory usage when model running on RK3588 is printed as follows:

```
========================================================
            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 Get SDK version

API	get_sdk_version
Description	Get API version and driver version of referenced SDK. Note: Before we use this interface, we must load model and initialize runtime first. And this API can only used on RV1103 / RV1103B / RV1106 / RV1106B / RV1126B / RK3562 / RK3566 / RK3568 / RK3576 / RK3588.
Parameter	None.
Return Value	sdk_version: API and driver version. Data type is string.

The sample code is as follows:

```
# Get SDK version
sdk_version = rknn.get_sdk_version()
print(sdk_version)
```

The SDK version looks like below:

```
==============================================
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 Hybrid Quantization

2.12.1 hybrid_quantization_step1

When using the hybrid quantization function, the main interface called in the first phase is hybrid_quantization_step1, which is used to generate the temporary model file (<model_name>.model), the data file (<model_name>.data), and the quantization configuration file (<model_name>.quantization. cfg). Interface details are as follows:

API	hybrid_quantization_step1
Description	Corresponding temporary model files, data files, and quantization profiles are generated according to the loaded original model.
Parameter	**dataset:** See 2.4 Building RKNN model.
	rknn_batch_size: See 2.4 Building RKNN model.
	proposal: Generate hybrid quantization config suggestions. The default value is False.
	proposal_dataset_size: The size of dataset used for proposal. The default value is 1. Because the proposal function is time-consuming, so the default size is 1.

API	**hybrid_quantization_step1**
	custom_hybrid: Select the hybrid quantization subgraph according to multiple sets of input names and output names specified by the user. The format is [[input0_name, output0_name], [input1_name, output1_name], …]. The default value is None. Note: Input names and output names should be chosen based on the temporary model file(<model_name>.model).
Return Value	0: success.
	-1: failure.

The sample code is as follows:

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

2.12.2 hybrid_quantization_step2

When using the hybrid quantization function, the primary interface for generating a hybrid quantized RKNN model phase call is hybrid_quantization_step2. The interface details are as follows:

API	**hybrid_quantization_step2**
Description	The temporary model file, the data file, the quantization profile, and the correction data set are received as inputs, and the hybrid quantized RKNN model is generated.
Parameter	**model_input:** The temporary model file (<model_name>.model) path generated in the hybrid_quantization_step1.
	data_input: The model data file (<model_name>.data) path generated in the hybrid_quantization_step1.
	model_quantization_cfg: Path to the modified model quantization configuration file (<model_name>.quantization.cfg) generated by hybrid_quantization_step1.
Return Value	0: success.
	-1: failure.

The sample code is as follows:

```
# 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 Quantitative accuracy analysis

The function of this interface is inference with quantized model and generate outputs of each layers for quantitative accuracy analysis.

API	**accuracy_analysis**
Description	Inference with quantized model and generate snapshot, that is dump tensor data of each layers. It will dump a snapshot of both data types include fp32 & quant for calculate quantitative error. Note: **1. This interface can only be called after build or hybrid_quantization_step2.** **2. If target is None and the original model is quantized model (QAT model), the call will fail.** **3. The quantization method used by this interface is consistent with the setting in config.**
Parameter	**inputs:** the path list of image (jpg/png/bmp/npy).
	output_dir: output directory, all snapshot data will stored here. The default value is './snapshot'. If the target is not set, the following content will be output under 'output_dir': - Directory simulator: Save the results of each layer on simulator when the entire quantitative model is fully run (The output has been converted to float32). - Directory golden: Save the results of each layer on simulator when the entire floating-point model is completely run down. - error_analysis.txt: Record the the cosine distance (entire_error and single_error) between each layer result on simulator and the floating-point model on simulator during the complete calculation of the quantized model. The different of entire_error/single_error is the input of each layer is come from the quantization model or floating-point model. See the error_analysis.txt file for more details. If the target is set, more content will output under 'output_dir': - Directory runtime: Save the results of each layer when the entire quantitative model is fully run in NPU (The output has been converted to float32). - error_analysis.txt: Record the the cosine distance (entire_error) between each layer result on simulator and each layer on NPU during the complete calculation of the quantized model additionally. See the error_analysis.txt file for more details.
	target: Target hardware platform, now supports 'rv1103', 'rv1103b', 'rv1106', 'rv1106b', 'rv1126b', 'rk3562', 'rk3566', 'rk3568', 'rk3576' and 'rk3588'. The default value is "None". If target is set, the output of each layer of NPU will be obtained, and analyze it's accuracy.
	device_id: Device identity number, if multiple devices are connected to PC, this parameter needs to be specified which can be obtained by calling "list_devices" interface. The default value is "None ".
Return Value	0: success.
	-1: failure.

The sample code is as follows:

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

2.14 List Devices

API	list_devices
Description	List connected RV1103 / RV1103B / RV1106 / RV1106B / RV1126B / RK3562 / RK3566 / RK3568 / RK3576 / RK3588. Note: There are currently two device connection modes: ADB and NTB. Make sure their modes are the same when connecting multiple devices.
Parameter	None.
Return Value	Return adb_devices list and ntb_devices list. If there are no devices connected to PC, it will return two empty list.

The sample code is as follows:

```
rknn.list_devices()
```

The devices list looks like below:

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

2.15 Export encrypted RKNN model

The function of this interface is to encrypt the ordinary RKNN model and obtain the encrypted model.

API	export_encrypted_rknn_model
Description	The common RKNN model is encrypted according to the encryption level specified by the user. Note: RV1103/RV1103B/RV1106/RV1106B/RK2118 is not supported yet.
Parameter	**input_model:** The path of the RKNN model to be encrypted.
	output_model: Save path of encrypted model. The default value is None, means the {original_model_name}.crypt.rknn will be the save path of encrypted model.
	crypt_level: Crypt level, currently, support level 1, 2 or 3. The default value is 1. The higher the level, the higher the security and the more time-consuming decryption; on the contrary, the lower the security, the faster the decryption.

API	**export_encrypted_rknn_model**
Return Value	0: Success.
	-1: Failure.

The sample code is as follows:

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

2.16 Register custom operator

The function of this interface is to register a custom operator.

API	**reg_custom_op**
Description	Register custom operator, only supported for ONNX model.
Parameter	**custom_op:** Custom operator class. For the user needs to customize a new OP, which is not within the ONNX OP specification. The op_type of this new OP is recommended to start with 'cst', and the ‘shape_infer’ and ‘compute’ functions need to be implemented by user. Note: The custom_op operator class is only used for model conversion and generation of RKNN models with custom operators. When deploying on the device side, you also need to refer to Chapter 5.5 of the "RKNN SDK User Guide".
Return Value	0: Success.
	-1: Failure.

The sample code is as follows:

```
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)
```

2.17 Generate C++ deployment example

API	**codegen**
Description	Generate C++ deployment example
Parameter	**output_path:** The output folder directory, the user can configure the directory name
	inputs: Fills in the path list of model input. It is allowed to leave it blank. The valid file format is jpg/png/npy. When npy files are used as input, the dimension information of npy data should be consistent with the dimension information of the model input
	overwrite: When overwrite is set to True, the files in the directory specified by output_path will be overwritten. The default value is False
Return Value	0: Success.
	-1: Failure.

The sample code is as follows:

```
ret = rknn.codegen(output_path='./rknn_app_demo',
            inputs=['./mobilenet_v2/dog_224x224.jpg'],
            overwrite=True)
```