RKNN3-Toolkit API Reference

ID: RK-YH-YF-412

Release Version: V1.0.0

Release Date: 2026-01-10

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

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 RKNN3-Toolkit API Reference.

RKNN3-Toolkit 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
V0.2.0	HPC Team	2025-8-6	Initial version	Vincent
V0.3.0b0	NN Team	2025-9-12	Update the interface descriptions for load_rknn/init_runtime/inference/llm.inference/list_devices	Vincent
V0.4.0b0	NN Team	2025-10-18	Update rknn.config new param "rknn_build_parallelism", "llm_head_dtype" and "llm_head_quantized_method" description	Vincent
V0.5.0	NN Team	2025-12-01	Support for accuracy analysis on connected target boards; updated init_runtime/inference API documentation; added documentation for the kvcache_controller/query API.	Vincent
V1.0.0	NN Team	2026-01-10	Release version.	Vincent

Contents

1 Requirement

1.1 Applicable chip model

- RK1820
- RK1828

1.2 Requirements Dependencies

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

Operating system version	Ubuntu22.04 (x64)	Ubuntu24.04 (x64)
Python version	3.10	3.12

Note:

1. For more detail about python library dependencies, see requirements*.txt
2. This document mainly uses Ubuntu 22.04 / Python3.10 as an example.

1.3 Supported Deep Learning Framework

The deep learning frameworks supported by RKNN3-Toolkit include Caffe, TensorFlow, TensorFlow Lite, ONNX, DarkNet and PyTorch.

The corresponding relationship between RKNN3-Toolkit and the version of each deep learning framework is as follows:

RKNN3-Toolkit	Caffe	TensorFlow	TF Lite	ONNX	DarkNet	PyTorch
0.2.0	1.0	1.12.0~ 2.14.0	Schema version=3	1.17.0~ 1.18.0	Commit ID: 810d7f7	2.7.0~ 2.8.0
0.3.0b0	1.0	1.12.0~ 2.14.0	Schema version=3	1.17.0~ 1.18.0	Commit ID: 810d7f7	2.7.0~ 2.8.0
V0.4.0b0	1.0	1.12.0~ 2.14.0	Schema version=3	1.17.0~ 1.18.0	Commit ID: 810d7f7	2.7.0~ 2.8.0
V0.5.0	1.0	1.12.0~ 2.14.0	Schema version=3	1.17.0~ 1.18.0	Commit ID: 810d7f7	2.7.0~ 2.8.0
V1.0.0	1.0	1.12.0~ 2.14.0	Schema version=3	1.17.0~ 1.18.0	Commit ID: 810d7f7	2.7.0~ 2.8.0

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 RKNN3-Toolkit relies may cause loading failure.
3. The Caffe protocols used by RKNN3-Toolkit 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. RKNN3-Toolkit 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. RKNN3-Toolkit’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.

2 RKNN3-Toolkit 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 RKNN3-Toolkit, 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 RKNN3-Toolkit 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 RKNN3-Toolkit or the run function of the C-API.**
	quantized_dtype: Quantization type, the quantization types currently supported are "w8a8" and "w4a16". The default value is "w8a8". **- w8a8:** The weight is 8bit asymmetric quantitative accuracy, and the activation value is 8bit asymmetric quantitative accuracy. **- w4a16:** The weight is 4bit asymmetric quantitative accuracy, the activation value is 16bit floating point accuracy.
	quantized_algorithm: The quantization algorithm used when calcaulating the quantization parameters of each layer. Currently support: **normal**, **mmse**, **kl_divergence**, **gdq** and **grq**. 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, which can effectively improve the weight accuracy of the w4a16. the recommended quantization data is more than 200 pieces. GPU acceleration is required due to high computational demands. The **grq** quantization algorithm is only valid at w4a16, which can effectively improve the weight accuracy of the w4a16. the recommended quantization data is more than 200 pieces. GPU acceleration is required due to high computational demands. However, GRQ is faster and more memory-efficient than GDQ, so it should be tried first.

API	config
	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.
	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. 'rk1820' are currently supported. The default value is None.
	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.
	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. 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.**
	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.**

API	config
	core_num: The number of NPU cores required by the model. For the RK1820, the NPU core number ranges from 1 to 8. Default is 0 (auto mode). Note: 1. Transformer-based models can be configured to use the desired number of NPU cores, while other models (such as CNN models) are recommended to use 1 core. 2. In auto mode, models loaded via **load_llm** will use 8 NPU cores, while all other cases will use 1 NPU core.
	rknn_build_parallelism: The number of parallel threads for compiling the rknn model. The default value is -1, which indicates automatic configuration. No more than 32 threads will be used during automatic configuration.
	distribute_strategy: The model splitting strategy. Available options: 'less_subgraph', 'best_perf', 'less_mem'. Default: 'less_mem'.
	input_attrs: Model deployment input properties. Default value is {}. The format is: {input_name: {attr_name: attr_value, ...}, ...}, e.g.: {'input_name': {'dtype': 'float32', 'layout': 'NCHW', 'shape': [1, 3, 224, 224]}, ...} attr_name: dtype, layout, shape, stride, etc.
	output_attrs: Model deployment output properties. Default value is {}. The format is: {output_name: {attr_name: attr_value, ...}, ...}, e.g.: {'output_name': {'dtype': 'float32', 'layout': 'NCHW', 'shape': [1, 1000]}, ...} attr_name: dtype, layout, shape, stride, etc.
	kvcache_store_method: Specifies the Kvcache storage method. "GroupQuant" specifies quantizing the Kvcache to save memory. "Normal" specifies maintaining the fp16 format for Kvcache storage. The default setting is "Normal."
	kvcache_dtype: Specifies the data type used for Kvcache storage and calculations. In "GroupQuant" mode, it supports "**Int4_to_F16**", which means that the data is stored as int4 and the calculation process uses fp16; In "Normal" mode, it supports "**Float16**", which means that fp16 is used for both storage and calculation. The default is "Float16"
	kvcache_group_size: Specifies the number of groups stored in Kvcache. When Kvcache_dtype is Int4_to_F16, kvcache_group_size must be configured to 16.
	kvcache_residual_depth: Specifies the size of the Kvcache cache. When the cache is full, data is stored in the Kvcache storage area according to the type specified by kvcache_dtype. When kvcache_dtype is "Float16", kvcache_residual_depth must be 32; when kvcache_dtype is "Int4_to_F16", kvcache_residual_depth must be 64. The default value is 32.
	max_ctx_len: Specifies the maximum length of the Kvcache, which affects the inference results of long text models and the Kvcache memory usage. During inference, if the Kvcache length exceeds the maximum length, the Kvcache will be overwritten. The overwriting behavior follows the first-in-first-out principle. Currently, due to memory planning issues, the Kvcache length can only be specified during the model conversion phase. The default value is 1024.

API	config
	max_position_embeddings: Specifies the maximum position ID size, which affects the maximum rope encoding position. Note that this parameter cannot be larger than the maximum rope encoding configured during model training, otherwise unexpected behavior may occur. Defaults to 8192.
	subgraph_dtype_target: Sets the inference precision type of a specified segment in the model to the target type. This parameter currently only takes effect when quantization is enabled for the model and currently only supports specifying that the corresponding segment use w16a16 (equivalent to Float16) for inference. The parameter is a two-level list, for example, [[subgraph0_inputs, subgraph0_outputs, dtype], [subgraph1_inputs, subgraph1_outputs, dtype], ...], where subgraph_inputs is a list containing the names of one or more input tensors for the fragment, and subgraph_outputs is a list containing the names of one or more output tensors for the fragment. The tensor names can be obtained from the table generated by the precision analysis function. dtype indicates the inference precision type. Note 1: In particular, when subgraph_inputs and subgraph_outputs are the input and output of a certain op, this is equivalent to specifying the assigned inference precision type for the op. Note 2: Currently, it is not necessary for subgraph_inputs and subgraph_outputs to contain all the inputs and outputs of the corresponding fragment. In this case, some ops in the fragment will use the corresponding type. The judgment logic is that the op input has an inference association with subgraph_inputs, and the op output has an inference association with subgraph_outputs. In this case, the op will be configured to the specified inference precision type.
	llm_config: Configure LLM-related optimization options. The configuration format is a dictionary. See section 2.2.1 for detailed configuration instructions. After configuration, you need to call the `rknn.load_llm` interface to load the model for the changes to take effect.
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]],
            target_platform='rk1820')
```

2.2.1 llm_config

For LLM models, `rknn.config` has added the `llm_config` parameter. The LLM config has four dictionary sub-items: `attention_config`, `llm_head`, `vocab`, and `keep_one_logit`.

attention_config: This is used to configure the "**KV cacahe**" definition and the inference method of the Attention op. It accepts lists, and multiple dictionaries can be included in the list to specify the specific parameters of various Attention structures. The sub-dictionaries of attention_config are defined as follows:

Parameter	**Function**	**Dtype:default value**	**Notice**
mask_name	Specify the mask name. In subsequent processes, the Attention op will determine its configuration by recognizing the mask input. **This parameter is mandatory!**	String: 'attention_mask'	Different types of Attention structures are not allowed to share the same mask input.
kvcache_buffer_len	Specifies the KV Cache length. During inference, if the context length exceeds the set length, it will be overwritten cyclically according to the first-in-first-out principle. A larger configuration value will result in more memory usage for the KV Cache, and theoretically, it will perform better on long text tasks.	Int:1024	Value must be 32-bit aligned
kvcache_dtype	Specifying the data type for context storage and computation affects inference performance and cache memory usage. Regarding memory usage, Float16 > Int8_to_FP16 > Int4_to_FP16	String: 'Float16'	Support Float16, Int8_to_FP16, Int4_to_FP16
attention_type	Specify Attention type	String: 'FullAttention'	Support FullAttention (Currently not support SlidingAttention)
sliding_window_size	Specifies the sliding window length for Sliding Attention. This parameter has no effect when the attention type is FullAttention.	Int: -1	/

Parameter	Function	Dtype:default value	Notice
position_name	Specify the input name for the position id, used to extract position-encoded feature values. For earlier models, position encoding may not exist; in this case, a default value of None is allowed.	String: 'position_ids'	Different types of Attention structures do not allow sharing the same position_ids input.
max_position_embeddings	Specifies the maximum length of the positional encoding. An error will be thrown if the context length exceeds the set length during inference.	Int: 8192	Value must be 32-bit aligned
mrope_type	When the LLM model is a special mrope type, the mrope function can be configured and enabled. Currently, the ['Qwen2.5-VL', 'Qwen3-VL'] type is supported. **Note:** When enabling mrope, position_name must be configured, and this position_name is a one-dimensional input. After enabling the mrope function, the toolkit will automatically generate an input with [seq_len, 3] dimensions.	Strings: None	/
mrope_section	Configure the mrope section information. For Qwen2.5-VL and Qwen3-VL models, the corresponding configuration values can be found in the model's configuration file config.json.	list: None	/

Parameter	Function	Dtype:default value	Notice
mrope_new_id_name	Configure mrope to generate new position names. For the currently supported ['Qwen2.5-VL', 'Qwen3-VL'], a new input with dimension [seq_len, 3] will be generated.	Strings: None	/

---

llm_head: This is used to configure and enable optimization behavior for the LLM model's Head layer. (The Head layer is typically a single Linear layer with a large number of parameters). Multiple LLM Head layers are not currently supported; this will be added in future updates.

Parameter	Function	Dtype:default value	Notice
name	Configure the weight names of the LLM head layer.	String: auto	Currently, only `auto` is supported; specifying a specific value is not yet supported.
device	Configuring computing devices for the LLM head layer	String: rk1820	support rk1820, rk3588, rk3576
quantized_dtype	Configure the quantization type for the LLM head layer. The LLM head layer is quite large; using quantization can effectively reduce memory usage.	String: w6a16	support w16a16, w4a16, w6a16, w8a8. When the platform is rk3576 or rk3588, it will be forcibly changed to w8a8.
quantized_method	Configure the quantization method for the LLM head layer.	String: group32	Supports layer, channel, and group{Size}. When the platform is rk3576 or rk3588, it will be forced to change to channel.

---

vocab: Used to configure and enable the LLM model's optimization behavior for the dictionary layer. (**This feature is not currently implemented.**)

Parameter	Function	Dtype:default value	Notice
index_name	Configure the vocab layer index input name	String: auto	Currently, only the `auto` option is supported; specifying a different option is not yet supported.
tie_embedding	Configure the tie embedding parameter for the vocab layer. When tie_embedding is True, it means that vocab and llm_head share the same array. Enabling this eliminates the need to store vocab separately, but because llm_head has quantization behavior, dequantizing it to float16 embedding will reduce inference performance.	Bool: False	This feature is not currently supported in the current version.
store_on_compute_device	Configure whether the vocab is stored on the computing device. Storing the vocab on the computing device can reduce the transfer time required for each round of LLM inference, but it will significantly increase the memory usage of the computing device.	Bool: False	The current version does not support this.

---

keep_one_logit: This is used to configure and enable single-value sampling (gather op) on a specified axis of the output. For most LLM models, only the last valid token needs to be sampled during output sampling. For example, with an input token sequence length of 100, the 100th output sequence is valid, and all other outputs are discarded. Therefore, this discarding behavior can be moved earlier in the `llm_head`, reducing the computational cost of the linear layer in the `llm_head` and the memory usage required during computation.

Parameter	Function	Dtype:default value	Notice
output_name	Specify output name	String: None	Set None to disable 'keep_onxe_logit' function.

Parameter	Function	Dtype:default value	Notice
idx_name	Generate the names of the gather indices input and add them to the model's input.	String: num_logit_to_keep	/
axis	Specify the axis for which gather is active	Int: None	/

The sample code is as follows:

```
from rknn.api import RKNN, DEFAULT_RKNN_LLM_CONFIG
rknn = RKNN(verbose=True)
my_config = DEFAULT_RKNN_LLM_CONFIG.copy()
rknn.config(target_platform='rk1820', llm_config=my_config)

# default config is
DEFAULT_RKNN_LLM_CONFIG = {
'attention_config': [{                                    # accept multi internal kvcache
as a list of dicts
    'mask_name'              : 'attention_mask',          # required, this is used for
recognize which attn belong to this config
    'kvcache_buffer_len'     : 1024,                      # feed int
    'kvcache_dtype'          : 'Float16',                 # Float16, Int8_to_F16,
Int4_to_F16
    'attention_type'         : 'FullAttention',           # FullAttention,
SlidingAttention
    'sliding_window_size'    : -1,                        # feed int, -1 mean disable.
    'position_name'          : 'position_ids',            # allow None if no position_ids
input, positon id here was used for extract rope.
    'max_position_embeddings': 8192,
    'mrope_type'             : None,                      # 'Qwen2.5-VL', 'Qwen3-VL'
    'mrope_section'          : None,                      # [24,20,20]
    'mrope_new_id_name'      : None,                      # new name to insert position id
input
}],
'llm_head':[{
    'name'                  : 'auto',                     # auto or specify head tensor
name, if multiple heads in model, please specify head tensor name
    'device'                : 'RK1820',                   # RK3588, RK3576,
    'quantized_dtype'       : 'w6a16',                    # w16a16, w4a16, w6a16, w8a8
    'quantized_method'      : 'group32',                  # layer, channel, group{SIZE}
}],
'vocab':[{
    'index_name'            : 'auto',                     # auto or specify embedding
tensor name, if multiple embeddings in model, please specify embedding tensor name
    'tie_embeddings'         : False,                     #! True, False, current not
support
    'store_on_compute_device': False,                     # True, False
}],
'keep_one_logit': [{                                        # accept multi internal kvcache
as list(dict)
    'output_name': None,
    'idx_name'    : 'num_logits_to_keep',
    'axis'        : None,
}],
}
```

2.3 Loading model

RKNN3-Toolkit 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.
Parameter	**model:** The path of Caffe model structure file (suffixed with ".prototxt").
	blobs: The path of Caffe model binary data file (suffixed with ".caffemodel").
	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.
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.

API	load_tensorflow
	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.
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.
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]])
```

2.3.7 Loading LLM model

API	load_llm
Description	Load the LLM model. The LLM model is exported from the rkllm3_model_zoo project. (For example, rkllm3_model_zoo/examples/Qwen2_5/python/export_llm.py)
Parameter	**model**: LLM model path (.onnx).
	config: LLM config file path.
	seq_lens: LLM input sequence lengths. Default: [1, 128].
	llm_head_dtype: The quantization data type for the LLM model head weight. Currently supported types are int4, int6, int8 and float16.
	llm_head_quantized_method: The quantization method for the LLM model head weight. Currently, group{SIZE} is supported, where {SIZE} is a multiple of 32 between 32 and 256 (e.g., group32 or group128).When llm_head_dtype is set to float16, this parameter is ignored.
Return Value	0: Import successfully.
	-1: Import failed.

The sample code is as follows:

```
# Load Qwen2.5-0.5B locally
ret = rknn.load_llm(model='./Qwen2.5-0.5B.onnx',
                    config='./Qwen2.5-0.5B.pkl')
```

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.
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") and model weight save in the specified file (suffixed with ".weight"). If save_ctx=True, saves context (.rknn.ctx).
Parameter	**export_path:** The path of generated RKNN model file.
	save_ctx: Export context info (for load_rknn's load_ctx). Default: False.
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 and model weight as a
mobilenet_v1.weight 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. If load_ctx=False, 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. If load_ctx=True, the loaded model can connecting to the NPU hardware for inference or performance data acquisition. also can be used for simulator or accuracy analysis.
Parameter	**model_path:** The path of RKNN model file.
	weight_path: The path of weight file of RKNN model.
	load_ctx: Whether to load context information. Default: False.
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(model_path='./mobilenet_v1.rknn', weight_path='./mobilenet_v1.weight')
```

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, default is "None", means model runs on simulator, when the platform is specified, it runs on the target.
	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 device ID must be either "rk3576" or "rk3588". The default value is None.
	core_mask: The core of the model's target inference. If target is seted, this parameter needs to be specified. It supports mask values from 0x1 to 0xff. The default value is -1.
Return Value	0: Initialize the runtime environment successfully.
	-1: Initialize the runtime environment failed.

Note: **Currently, device can be connected to the PC via ADB mode. The device's SoC must be RK3576 or RK3588, and it must be interconnected with the target RK1820 through PCIe or USB.**

The sample code is as follows:

```
# Initialize the runtime environment
ret = rknn.init_runtime(target=None)
```

2.8 Inference with RKNN model and accuracy analysis

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.
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".
	accuracy_analysis: Whether to enable per-layer accuracy analysis. The default value is False. If the target parameter is set in init_runtime, on-device accuracy analysis will be performed. When performing link precision analysis, set `profile_mode=True` in the `config`.
	output_dir: output directory, all snapshot data will stored here. The default value is './snapshot'. If the target is not set in init_runtime, 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 in init_runtime, 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.
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 *examples/tflite/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 Query Function

API	**query**
Description	Queries model-related information or status.
Parameters	**cmd**： The type of query command, provided as a string.
Return Value	results： The result of the query

2.9.1 Query Command Types

COMMAND	**DESCRIPTION**
QUERY_ROPE_CACHE	Queries the sin/cos cache used for the model's RoPE (Rotary Position Embedding) input.

The sample code is as follows:

```
# Query the sin/cos cache used for the model's RoPE input
rope_cache = rknn.query('QUERY_ROPE_CACHE')
```

2.10 KVCache Controller

2.10.1 Obtain KVCache Control Inputs

API	**kvcache_controller.generate_kvcache_control_tensors**
Description	Obtains the inputs for controlling the KVCache and the ID corresponding to the model's dynamic shape.
Parameters	**input_seq_len**: The number of tokens in the model input sequence.
Parameters	**system_prompt_len**: The number of tokens in the system prompt. Default value is 0.
Return Value	Returns a list of KVCache control inputs and the dynamic shape ID. If multi-turn inference is required, the returned list length will be greater than 1.

The sample code is as follows:

```
attention_inputs, dynamic_idx =
rknn.kvcache_controller.generate_kvcache_control_tensors(input_seq_len)
```

2.10.2 Clear KVCache Status

API	**kvcache_controller.clear_kvcache_status**
Description	Clears the KVCache status.
Parameters	None.
Return Value	None.

The sample code is as follows:

```
rknn.kvcache_controller.clear_kvcache_status()
```

2.11 Evaluate model performance

API	eval_perf
Description	Evaluate model performance. The model must be running on a device connected to the PC. In `config`, set `profile_mode=True`.
Parameter	**log_level:** Level of printed performance information, default 0. 0: print summary table only; 1: print detailed per-Op table and summary table; 2: print per-Op details including time, cycles and bandwidth, plus summary table.
	dynamic_idx: Index for dynamic input, only effective for dynamic shapes. Default 0.
Return Value	0: Eval successfully.
	-1: Eval failed.

The sample code is as follows:

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

2.12 Evaluating memory usage

API	eval_memory
Description	Fetch memory usage when model is running on hardware platform. The model must be running on a device connected to the PC.
Parameter	None
Return Value	0: Eval successfully.
	-1: Eval failed.

The sample code is as follows:

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

2.13 List Devices

API	**list_devices**
Description	List connected RK3576 / RK3588 / RK1820. Note: There are currently ADB device connection modes.
Parameter	None.
Return Value	Return adb_devices list 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
*************************
```