

Classification Level: Top secret ( ) Secret ( ) Internal ( ) Public (√)

## RKNN-Toolkit User Guide

(Technology Department, Graphic Computing Platform Center)

Mark:	Version	V1.7.5
[ ] Editing	Author	Rao Hong
[√] Released	Completed Date	2023-07-31
	Reviewer	Vincent
	Reviewed Date	2023-07-31

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd

(Copyright Reserved)

## Revision History

Version	Modifier	Date	Modify description	Reviewer
V0.1	Yang Huacong	2018-08-25	Initial version	Vincent
V0.9.1	Rao Hong	2018-09-29	Added user guide for RKNN-Toolkit, including main features, system dependencies, installation steps, usage scenarios, and detailed descriptions of each API interface.	Vincent
V0.9.2	Randall	2018-10-12	Update version.	Vincent
V0.9.3	Yang Huacong	2018-10-24	Add instructions of connection to development board hardware	Vincent
V0.9.4	Yang Huacong	2018-11-03	Add instructions of docker image	Vincent
V0.9.5	Rao Hong	2018-11-19	Update API description: add pre_compile parameter description to build interface; improve the description of dataset parameter; improve the description of reorder_channel parameter to config interface.	Vincent
V0.9.6	Rao Hong	2018-11-24	1. Add the instructions of get_perf_detail_on_hardware and get_run_duration interfaces 2. Update the instructions of RKNN initialization interface	Vincent
V0.9.7	Rao Hong	2018-12-29	Update API description: rewrite the eval_perf interface instructions, remove the deprecated get_perf_detail_on_hardware and get_run_duration interfaces; Update the description of RKNN initialization interface; add the description of init_runtime interface.	Vincent

Version	Modifier	Date	Modify description	Reviewer
V0.9.7.1	Rao Hong	2019-01-11	Update API description: improve the description of init_runtime.	Vincent
V0.9.8	Rao Hong	2019-01-30	Update API description: add verbose and verbose_file parameters description to RKNN initialization interface.	Vincent
V0.9.9	Rao Hong	2019-03-06	Add API description for get_sdk_version and eval_memory interfaces.	Vincent
V1.0.0	Rao Hong	2019-05-06	<ol style="list-style-type: none"> <li>1. Add the instructions for using the hybrid quantization function and the corresponding interface.</li> <li>2. Update API description: add async_mode parameter description to init_runtime interface; add inputs_pass_through parameter description to inference interface.</li> </ol>	Vincent
V1.1.0	Rao Hong	2019-06-28	<ol style="list-style-type: none"> <li>1. Add instructions for ARM64 Linux / Windows / MacOS versions, including dependency information and usage, etc.</li> <li>2. Add API description for list_devices.</li> </ol>	Vincent
V1.2.0	Rao Hong	2019-08-21	<ol style="list-style-type: none"> <li>1. Add the descriptions for model pre-compilation, model segmentation, custom operator functions and related interfaces.</li> <li>2. Add function support list for each platform.</li> <li>3. Update API description: add rknn_batch_size parameter description to build interface</li> </ol>	Vincent
V1.2.1	Rao Hong	2019-09-26	<ol style="list-style-type: none"> <li>1. Update API description: improve parameter descriptions for config; add load_model_in_npu parameter description to load_rknn interface.</li> </ol>	Vincent

Version	Modifier	Date	Modify description	Reviewer
V1.3.0	Rao Hong	2019-12-23	<ol style="list-style-type: none"> <li>1. Add the description of PyTorch/MXNet model conversion, accuracy analysis function and corresponding interface.</li> <li>2. Add visualization function description and corresponding usage.</li> <li>3. Update API description: add optimization_level parameter description to config interface.</li> <li>4. Improve the description of the hybrid quantizaion.</li> </ol>	Vincent
V1.3.2	Rao Hong	2020-04-03	<ol style="list-style-type: none"> <li>1. Add support for RV1109 and RV1126.</li> <li>2. Update API description: add target_platform parameter description to config interface; description of the eval_perf interface to remove deprecated parameters such as inputs/data_type/data_format.</li> </ol>	Vincent
V1.4.0	Rao Hong	2020-08-13	<ol style="list-style-type: none"> <li>1. Improve the description of quantization accuracy analysis function.</li> <li>2. Update API description: add mean_values and std_values parameter description to config interface; description of the config interface to remove deprecated channel_mean_value parameter; description of the load_tensorflow interface to remove deprecated mean_values/std_values parameter.</li> </ol>	Vincent

Version	Modifier	Date	Modify description	Reviewer
V1.6.0	Rao Hong Chifred	2020-12-31	<ol style="list-style-type: none"> <li>1. Update usage of docker image.</li> <li>2. Add for Keras model conversion, model encryption, device query command function and corresponding interface; improve the description of the model segmentation function.</li> <li>3. Update API description: add convert_engine parameter description to load_pytorch interface.</li> </ol>	Vincent
V1.6.1	Rao Hong	2021-05-21	<ol style="list-style-type: none"> <li>1. Update API description: add quantized_algorithm and mmse_epoch parameter description to config interface; add the usage of the environment variable RKNN_DRAW_DATA_DISTRIBUTION to the build interface.</li> </ol>	Vincent
V1.7.0	Rao Hong	2021-08-08	<ol style="list-style-type: none"> <li>1. Improve the description of hybrid quantization, quantized-parameter-optimization algorithm MMSE, quantization accuracy analysis.</li> <li>2. Update API description: add inputs/outputs/input_size_list parameter description to load_onnx interface; add description of ONNX quantization model support to load_onnx interface.</li> </ol>	Vincent
V1.7.1	Rao Hong	2021-11-20	<ol style="list-style-type: none"> <li>1. Refactor the document directory structure.</li> <li>2. Add the corresponding deep learning framework and supported versions; add reference system configuration; add Rockchip NPU device connection instructions; rewrite the RKNN Toolkit usage; add model evaluation and deployment instructions.</li> </ol>	Vincent

Version	Modifier	Date	Modify description	Reviewer
V1.7.3	Rao Hong	2022-08-07	<ol style="list-style-type: none"> <li>1. Update the operating system that the tool is compatible with, remove the related instructions for Ubuntu 16.04 Python3.5, and add the related instructions for Ubuntu 20.04 Python3.8.</li> <li>2. The tool installation is explained by taking Ubuntu 18.04 / Python3.6 as an example.</li> <li>3. Update the usage instructions of config, accuracy_analysis and other interfaces.</li> </ol>	Vincent
V1.7.5	Rao Hong	2023-07-31	<ol style="list-style-type: none"> <li>1. Remove simulator-related content; remove offline pre-compilation related content.</li> <li>2. Add an explanation of the quantized parameter export interface.</li> <li>3. Enhancing the content of accuracy analysis and performance evaluation sections.</li> <li>4. Enhancing the usage instructions for config, build, init_runtime, eval_perf, export_rknn_precompile and related interfaces.</li> </ol>	Vincent

# Table of Contents

<b>1</b>	<b>Overview .....</b>	<b>10</b>
1.1	Main function description.....	10
1.2	Applicable chip model.....	14
1.3	Applicable Operating System .....	14
1.4	Applicable Deep Learning Framework.....	14
<b>2</b>	<b>Development environment setup.....</b>	<b>18</b>
2.1	Reference system configuration.....	18
2.2	Requirements/Dependencies.....	18
2.3	Installation .....	20
2.3.1	<i>Install by the Docker Image.....</i>	<i>20</i>
2.3.2	<i>Install by pip command .....</i>	<i>21</i>
2.3.3	<i>Installation FAQs.....</i>	<i>22</i>
2.4	Connect to Rockchip NPU device .....	22
2.4.1	<i>Connect to the device through the USB-OTG port .....</i>	<i>23</i>
2.4.2	<i>Connect the device through the DEBUG port .....</i>	<i>24</i>
2.4.3	<i>Confirm that the device is already connected.....</i>	<i>26</i>
2.4.4	<i>Connection FAQs.....</i>	<i>27</i>
<b>3</b>	<b>Usage .....</b>	<b>28</b>
3.1	Main steps.....	28
3.2	Model conversion .....	28
3.2.1	<i>Model conversion process .....</i>	<i>28</i>
3.2.2	<i>Model quantization.....</i>	<i>29</i>
3.2.3	<i>Model convert example.....</i>	<i>32</i>
3.2.4	<i>Model conversion FAQs.....</i>	<i>34</i>

3.3	Model evaluation .....	34
3.4	Model deployment.....	35
3.4.1	Prepare project files .....	36
3.4.2	Application example .....	36
3.4.3	Application execution.....	39
3.4.4	Model pre-compile.....	40
3.4.5	Model encryption.....	40
3.4.6	Multi-model scheduling .....	41
3.4.7	Deployment FAQs .....	43
<b>4</b>	<b>Accuracy evaluation.....</b>	<b>44</b>
4.1	evaluation method.....	44
4.2	Troubleshooting for accuracy issues .....	46
4.2.1	Troubleshooting for float model .....	47
4.2.2	Troubleshooting for quantized model accuracy.....	48
4.3	Accuracy analysis .....	50
4.3.1	Function description.....	50
4.3.2	Usage steps.....	50
4.3.3	Output file.....	51
4.4	Optimized Quantization.....	53
4.5	Hybrid quantization .....	53
4.5.1	Instructions of hybrid quantization.....	54
4.5.2	Hybrid quantization profile .....	54
4.5.3	Usage flow of hybrid quantization.....	56
4.6	Export model quantization parameters .....	58
4.6.1	Usage flow.....	58
4.6.2	Quantization parameter file description.....	59



4.7	FAQs .....	61
<b>5</b>	<b>Performance evaluation.....</b>	<b>62</b>
5.1	Evaluation method.....	62
5.2	Evaluation example .....	63
5.3	Description of evaluation results .....	65
5.3.1	<i>Description of the performance evaluation results of the development board .....</i>	<i>65</i>
5.4	Common performance optimization methods.....	68
<b>6</b>	<b>Memory evaluation .....</b>	<b>69</b>
6.1	Evaluation method.....	69
6.2	Evaluation example .....	69
6.3	Description of evaluation results .....	70
<b>7</b>	<b>RKNN-Toolkit API description.....</b>	<b>72</b>
7.1	RKNN object initialization and release .....	72
7.2	RKNN model configuration .....	73
7.3	Loading non-RKNN model .....	77
7.3.1	<i>Loading Caffe model .....</i>	<i>78</i>
7.3.2	<i>Loading Darknet model.....</i>	<i>78</i>
7.3.3	<i>Loading Keras model .....</i>	<i>79</i>
7.3.4	<i>Loading MXNet model.....</i>	<i>79</i>
7.3.5	<i>Loading ONNX model .....</i>	<i>80</i>
7.3.6	<i>Loading PyTorch model.....</i>	<i>81</i>
7.3.7	<i>Loading TensorFlow model.....</i>	<i>82</i>
7.3.8	<i>Loading TensorFlow Lite model.....</i>	<i>83</i>
7.4	Building RKNN model .....	84
7.5	Export RKNN model .....	85

7.6	Loading RKNN model.....	86
7.7	Initialize the runtime environment.....	87
7.8	Inference with RKNN model.....	90
7.9	Evaluate model performance .....	92
7.10	Evaluating memory usage.....	93
7.11	Hybrid Quantization .....	94
7.11.1	hybrid_quantization_step1 .....	94
7.11.2	hybrid_quantization_step2 .....	95
7.12	Accuracy analysis .....	96
7.13	Register Custom OP .....	99
7.14	Export a pre-compiled model(online pre-compilation) .....	99
7.15	Export a segmentation model .....	101
7.16	Export encrypted RKNN model .....	102
7.17	Export quantization parameters .....	103
7.18	Get SDK version.....	104
7.19	List Devices .....	105
7.20	Query RKNN model runnable platform .....	106
<b>8</b>	<b>Appendix.....</b>	<b>107</b>
8.1	Reference documents.....	107
8.2	Issue feedback.....	107

# 1 Overview

## 1.1 Main function description

RKNN-Toolkit is a development kit that provides users with model conversion, inference and performance evaluation on PC and Rockchip NPU platforms. Users can easily complete the following functions through the Python interface provided by the tool:

- 1) Model conversion: Support the conversion of Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet、PyTorch、MXNet or Keras model to RKNN model.
  - Since version 1.6.0, support for Keras and H5 model exported by TensorFlow 2.0.
- 2) Quantization: Support to convert float model to quantization model, currently the available quantized methods including asymmetric quantization (asymmetric\_quantized-u8) and dynamic fixed point quantization (dynamic\_fixed\_point-i8 and dynamic\_fixed\_point-i16). Starting from version 1.0.0, RKNN-Toolkit began to support hybrid quantization. For more detail about hybrid quantization, please refer to Section 3.3.
  - Since version 1.6.1, RKNN-Toolkit provides quantized-parameter-optimization algorithm MMSE and KL divergence.
  - Since version 1.7.0, RKNN-Toolkit support loading quantized ONNX models.
  - Since version 1.7.1, RKNN-Toolkit support loading quantized PyTorch models.
- 3) Model inference: Support distributing the RKNN model to specified NPU device for inference. Since version 1.7.5, **support for simulators has been removed**. The RKNN models can only be deployed and inferred on Rockchip NPU devices.
- 4) Performance evaluation: Support distributing the RKNN model to specified NPU device for performance evaluation. Since version 1.7.5, **support for simulators has been removed**. Connecting to the Rockchip NPU device is required for performance evaluation.
- 5) Memory evaluation: Evaluate system and NPU memory consumption at runtime of the model. It is allowed to distribute the RKNN model to the specified NPU device to inference, and then call

- the relevant interface to obtain memory usage information.
- 6) Model pre-compilation: With pre-compilation method, model loading-time can be decreased on NPU devices, and model size can also be reduced for some models. However, the pre-compiled RKNN model can only be run on a hardware platform with an NPU.
    - Since version 1.4.0, the RKNN Toolkit has introduced online pre-compilation functionality. For more details, please refer to the Chapter 7.14.
    - Since version 1.7.5, support for offline pre-compilation has been removed.
  - 7) Model segmentation: This function is used in a scenario where multiple models run simultaneously. One single model can be divided into multiple segments to be executed on the NPU, avoiding the NPU-occupying model inference takes too much time and stop other models to run. This feature is supported since version 1.2.0. Currently, only RK1806/RK1808/RV1109/RV1126 support this feature and the NPU driver version must be greater than 0.9.8.
  - 8) Custom OP: If the model contains an OP that is not supported by RKNN-Toolkit, it will fail during the model conversion phase. At this time, you can use the custom layer feature to define an unsupported OP so that the model can be converted and run successfully. Please refer to the *Rockchip\_Developer\_Guide\_RKNN\_Toolkit\_Custom\_OP\_CN* document for the use and development of custom OP. Currently this function only support TensorFlow model.
  - 9) Quantization-error analysis: This function will compute the Euclidean or cosine distance of the inference float results for each layer, compared to its quantized results. This can be used to analyze where quantization-error occurs, and provide ideas for improving the accuracy of quantized models. This feature is supported since version 1.3.0.
    - Since version 1.4.0, new feature called individual-quantization-accuracy analysis was added. Float results of last layer will be recorded and pushed to the quantized next layer. This can avoid cumulative error and more accurately reflect the influence of quantization on each layer.

- 10) Visualization: This function presents various functions of RKNN-Toolkit with graphical interface, simplifying the user's operation steps. Users can complete model conversion and inference by filling out forms and clicking function buttons, and no need to write scripts manually. Please refer to the *Rockchip\_User\_Guide\_RKNN\_Toolkit\_Visualization\_EN* document for the use of visualization.
- Version 1.4.0 improves the support for multi-inputs models and supports new NPU devices such as RK1806/RV1109/RV1126 as target.
  - Since version 1.6.0, keras model is support in visualization function.
- 11) Model optimization level adjustment: RKNN-Toolkit optimizes the model during model conversion. The default optimization selection may have some impact on model accuracy. By setting the optimization level, you can turn off some or all optimization options to analyze the impact of RKNN-Toolkit model optimization options on accuracy. For specific usage of optimization level, please refer to the description of optimization\_level option in config interface.
- 12) Model encryption: Use the specified encryption method to encrypt the RKNN model. This feature is supported since version 1.6.0. While the encryption of the RKNN model is done in the NPU driver, the encryption model can be loaded just like a normal RKNN model, and the NPU driver will automatically decrypt it.
- 13) Quantization parameter export: allows for exporting the quantization parameters of each layer in an RKNN model. This feature is supported since version 1.7.5. For pre-compiled RKNN models, it is limited to obtaining the quantization parameters of the input and output layers. For encrypted models, it is not possible to retrieve the quantization parameters.

Note: Some features are limited by the operating system or chip platform and cannot be used on some operating systems or platforms. The feature support list of each operating system (platform) is as follows:

Table 1-1 Support list of each platform function

	Ubuntu 18.04/20.04	Windows 7/10	Debian 9.8 / 10 (ARM 64)	MacOS Mojave / Catalina
Model conversion	yes	yes	yes	yes
Quantization	yes	yes	Partial support (do not support MMSE)	yes
Model inference	yes	yes	yes	yes
Performance evaluation	yes	yes	yes	yes
Memory evaluation	yes	yes	yes	yes
Model pre-compilation	yes	yes	yes	yes
Model segmentation	yes	yes	yes	yes
Custom OP	yes	no	no	no
Multiple inputs	yes	yes	yes	yes
Batch inference	yes	yes	yes	yes
List devices	yes	yes	yes	yes
Query SDK version	yes	yes	yes	yes
Quantitative error analysis	yes	yes	yes	yes
Visualization	yes	yes	no	yes
Model optimization level	yes	yes	yes	yes
Model encryption	yes	yes	yes	yes

## 1.2 Applicable chip model

RKNN-Toolkit supports the following chips with NPU that Rockchip has released:

- RK1806
- RK1808
- RK3399Pro
- RV1109
- RV1126

**Note: RK3566 and RK3568 need to use RKNN-Toolkit2. Please refer to the following project for more information:**

<https://github.com/rockchip-linux/rknn-toolkit2>

## 1.3 Applicable Operating System

RKNN-Toolkit is a cross-platform development kit. The supported operating systems are as follows:

- Ubuntu: 18.04 (x64) or later
- Windows: 7 (x64) or later
- MacOS: 10.13.5 (x64) or later
- Debian: 9.8 (aarch64) or later

## 1.4 Applicable Deep Learning Framework

The deep learning frameworks supported by RKNN Toolkit include TensorFlow, TensorFlow Lite, Caffe, ONNX, Darknet and Keras.

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

Table 1-2 The first part of the support of each deep learning framework

RKNN Toolkit	Caffe	Darknet	Keras	MXNet
1.0.0	1.0	Commit ID: 810d7f7	Not Support	Not Support
1.1.0	1.0	Commit ID: 810d7f7	Not Support	Not Support
1.2.0	1.0	Commit ID: 810d7f7	Not Support	Not Support
1.2.1	1.0	Commit ID: 810d7f7	Not Support	Not Support
1.3.0	1.0	Commit ID: 810d7f7	Not Support	>=1.4.0, <=1.5.1
1.3.2	1.0	Commit ID: 810d7f7	Not Support	>=1.4.0, <=1.5.1
1.4.0	1.0	Commit ID: 810d7f7	Not Support	>=1.4.0, <=1.5.1
1.6.0	1.0	Commit ID: 810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.6.1	1.0	Commit ID: 810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.0	1.0	Commit ID: 810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.1	1.0	Commit ID: 810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.3	1.0	Commit ID: 810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.5	1.0	Commit ID: 810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1



Table 1-3 The second part of the support of each deep learning framework

RKNN Toolkit	ONNX	Pytorch	TensorFlow	TF Lite
1.0.0	1.3.0	Not Support	>=1.10.0, <=1.13.2	Schema version = 3
1.1.0	1.3.0	Not Support	>=1.10.0, <=1.13.2	Schema version = 3
1.2.0	1.4.1	Not Support	>=1.10.0, <=1.13.2	Schema version = 3
1.2.1	1.4.1	Not Support	>=1.10.0, <=1.13.2	Schema version = 3
1.3.0	1.4.1	>=1.0.0, <=1.2.0	>=1.10.0, <=1.13.2	Schema version = 3
1.3.2	1.4.1	>=1.0.0, <=1.2.0	>=1.10.0, <=1.13.2	Schema version = 3
1.4.0	1.4.1	>=1.0.0, <=1.2.0	>=1.10.0, <=1.13.2	Schema version = 3
1.6.0	1.6.0	>=1.0.0, <=1.6.0	>=1.10.0, <=2.0.0	Schema version = 3
1.6.1	1.6.0	>=1.0.0, <=1.6.0	>=1.10.0, <=2.0.0	Schema version = 3
1.7.0	1.6.0	>=1.0.0, <=1.6.0	>=1.10.0, <=2.0.0	Schema version = 3
1.7.1	1.6.0	>=1.5.1, <=1.9.0	>=1.10.0, <=2.0.0	Schema version = 3
1.7.3	1.6.0	>=1.5.1, <=1.10.0	>=1.10.0, <=2.2.0	Schema version = 3
1.7.5	1.7.0	>=1.5.1, <=1.10.0	>=1.10.0, <=2.2.0	Schema version = 3

Note:

1. According to the protobuf version, any graph or checkpoint built with a certain version of TensorFlow can be loaded and evaluated by a higher (minor or patch) version of TensorFlow in the same major version. Theoretically, the pb files generated by TensorFlow with versions before 1.14 are supported by RKNN Toolkit 1.0.0 and later versions. 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 Toolkit relies may cause loading failure. The tflite schema currently used by RKNN Toolkit is based on the

following submission on the TensorFlow Lite's official GitHub master branch:

0c4f5dfea4ceb3d7c0b46fc04828420a344f7598. The specific schema link is as follows:

<https://github.com/tensorflow/tensorflow/commits/master/tensorflow/lite/schema/schema.fbs>

- There are two caffe protocols used by RKNN Toolkit: one is the protocol based on the official modification of berkeley, the other one is the protocol containing the LSTM layer support.

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 21d0608.

RKNN Toolkit adds some OPs on this basis.

As for the protocol containing the LSTM layer support, please refers to the following link:

<https://github.com/xmfbt/warpctc-caffe/tree/master/src/caffe/proto>, and the commit number is

bd6181b. These two protocols are specified by the proto parameter in the load\_caffe interface.

- For the relationship between ONNX release versions and opset versions and IR versions, please refer to the official website:

<https://github.com/microsoft/onnxruntime/blob/v1.7.0/docs/Versioning.md>

Table1-4 Correspondence between release version, opset version and IR version

ONNX release version	ONNX opset version	Supported ONNX IR version
1.6.0	11	6
1.7.0	12	7

- The official Github link of Darknet: <https://github.com/pjreddie/darknet>, RKNN Toolkit's current conversion rules are based on the latest submission of the master branch (commit number: 810d7f7).
- 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.
- Currently, RKNN Toolkit mainly supports the versions of Keras with TensorFlow as the backend. The tested Keras versions are those that come with TensorFlow.

## 2 Development environment setup

This chapter introduces the preparation of the development environment before using RKNN Toolkit.

### 2.1 Reference system configuration

The system configuration recommended for RKNN Toolkit is as follows:

Table 2-1 RKNN Toolkit recommended system configuration

Hardware/operating system	Recommand
CPU	Intel Core i3 or above or equivalent Xeon / AMD processor
RAM	16G or above
operating system	Linux: Ubuntu 16.04 64bit or Ubuntu 18.04 64bit MacOS: 10.13.5 Windows: 7 or 10

**Note:** It is recommended to use the PC with the recommended configuration to complete the model conversion stage. Model conversion can also be completed when using specified firmware on RK3399Pro, RK1808 computing card and other devices, but because of the limited CPU and memory resources of these devices, it is not recommended to use them for model conversion.

### 2.2 Requirements/Dependencies

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

Table 2-2 run time environments

Operation System	Ubuntu18.04 (x64) or later Windows 7 (x64) or later Mac OS X 10.13.5 (x64) or later Debian 9.8 (aarch64) or later
Python	3.5 / 3.6 / 3.7 / 3.8
Python Requirements	'numpy == 1.19.5' 'scipy == 1.4.1' 'networkx == 1.11' 'flatbuffers == 1.10' 'protobuf == 3.13.0' 'ruamel.yaml == 0.15.81' 'psutil == 5.6.2' 'ply == 3.11' 'sklearn == 0.0' 'matplotlib == 3.3.4' 'tqdm == 4.63.0' 'flask == 2.0.2' 'Jinja2 == 3.0.0' 'requests == 2.22.0' 'onnxoptimizer >= 0.1.0' 'tensorflow == 1.14.0' 'onnx == 1.10.0' 'onnxruntime == 1.9.0' 'torch == 1.10.0' 'mxnet == 1.5.0'

Note:

1. Windows only support Python 3.6 currently; MacOS support Python 3.6 and Python 3.7; Arm64 platform support Python 3.5(Debian 9.8) and python 3.7(Debian 10).
2. Linux x86 platform removes support for python3.5 and adds support for Python3.8 since RKNN Toolkit Version 1.7.3.
3. Since some dependencies cannot be installed or used normally in the Python3.8 environment, in the Linux x86\_64 / Python3.8 environment, upgrade the TensorFlow version to 2.2.0.
4. When using pip to install RKNN Toolkit, tensorflow/torch/mxnet will not be installed by default. Please install the corresponding version according to actual needs. The version in the

above table is the recommended version.

5. The application on ARM64 platform does not need to require flask, sklearn, requests.
6. Jinja2 is only used when creating custom op.

## 2.3 Installation

There are two ways to install RKNN-Toolkit: one is through the Python-package-installation and management-tool pip, the other is running docker image with full RKNN-Toolkit environment. The specific steps of the two installation ways are described below.

### 2.3.1 Install by the Docker Image

In docker folder, there is a Docker image that has been packaged for all development requirements. After loading the image, user can directly use RKNN-toolkit. Using steps are as follows:

#### 1. Install Docker

Please install Docker according to the official manual:

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

#### 2. Load Docker image

Execute the following command to load Docker image:

```
docker load --input rknn-toolkit-1.7.x-docker.tar.gz
```

After loading successfully, execute “docker images” command and the image of rknn-toolkit appears as follows:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
rknn-toolkit	1.7.x	xxxxxxxxxx	1 hours ago	x.xxGB

#### 3. Run image

Execute the following command to run the docker image. Then it will enter the bash environment.

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb rknn-toolkit:1.7.x /bin/bash
```

Mapping the folder to the Docker environment can be achieved through additional parameters “-

v <host src folder>:<image dst folder>” , for example:

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb -v /home/rk/test:/test rknn-  
toolkit:1.7.x /bin/bash
```

#### 4. Run demo

```
cd /example/tflite/mobilenet_v1  
python test.py
```

**Note: The Docker image is based on Ubuntu 20.04 and Python 3.8 since RKNN-Toolkit version**

**1.7.5. This image can only be used on Linux x86 platform.**

### 2.3.2 Install by pip command

1. Create virtualenv environment. If there are multiple versions of the Python environment in the system, it is recommended to use virtualenv to manage the Python environment.

```
sudo apt install virtualenv  
sudo apt-get install libpython3-dev  
sudo apt install python3-tk  
  
virtualenv -p /usr/bin/python3 venv  
source venv/bin/activate
```

2. Install dependent libraries: TensorFlow and opencv-python

```
# Install tensorflow gpu  
pip install tensorflow-gpu==1.11.0  
# Install tensorflow cpu. Only one version of tensorflow can be installed.  
pip install tensorflow==1.11.0  
# Install PyTorch and torchvision  
pip install torch==1.10.0+cpu -f https://download.pytorch.org/whl/torch_stable.html  
pip install torchvision==0.11.0+cpu -f https://download.pytorch.org/whl/torch_stable.html  
# Install mxnet  
pip3 install mxnet==1.5.0
```

3. Install RKNN-Toolkit

```
pip install package/rknn_toolkit-1.7.x-cp38-cp38-linux_x86_64.whl
```

Please select corresponding installation package (located at the *packages/* directory) according to different python versions and processor architectures:

- **Python3.5 for arm\_x64:** rknn\_toolkit-1.7.x-cp35-cp35m-linux\_aarch64.whl
- **Python3.6 for x86\_64:** rknn\_toolkit-1.7.x-cp36-cp36m-linux\_x86\_64.whl
- **Python3.7 for arm\_x64:** rknn\_toolkit-1.7.x-cp37-cp37m-linux\_aarch64.whl
- **Python3.6 for Windows x86\_64:** rknn\_toolkit-1.7.x-cp36-cp36m-win\_amd64.whl
- **Python3.6 for Mac OS X:** rknn\_toolkit-1.7.x-cp36-cp36m-macosx\_10\_15\_x86\_64.whl
- **Python3.7 for Mac OS X:** rknn\_toolkit-1.7.x-cp37-cp37m-macosx\_10\_15\_x86\_64.whl
- **Python3.7 for arm\_x64:** rknn\_toolkit-1.7.x-cp37-cp37m-linux\_aarch64.whl
- **Python3.8 for x86\_64:** rknn\_toolkit-1.7.x-cp38-cp38-linux\_x86\_64.whl

### 2.3.3 Installation FAQs

For common problems of RKNN Toolkit installation, please refer to the document *Rockchip\_Trouble\_Shooting\_RKNN\_Toolkit\_EN.pdf*.

## 2.4 Connect to Rockchip NPU device

During model evaluation or deployment, Rockchip NPU equipment needs to be connected. This section will give the connection method of RK3399Pro, RK1808, RV1109, RV1126 development board, and how to confirm whether the device is successfully connected.

#### **Note:**

1.If you are using the Toybrick development board, please refer to the boot and serial port debugging content in the following link:

<https://t.rock-chips.com/wiki.php?filename=%E7%BD%91%E7%AB%99%E5%AF%BC%E8%88%AA/%E9%A6%96%E9%A1%B5>

### 2.4.1 Connect to the device through the USB-OTG port

The connection between RKNN Toolkit and Rockchip NPU development board is created base on the USB-OTG interface. In the stage of model evaluation or model deployment, it is necessary to connect the PC and Rockchip NPU device for the related function.

The position of the RK3399Pro USB-OTG interface is shown in the red circle as shown in the figure below, and it is connected to the PC through the USB Type-C cable:

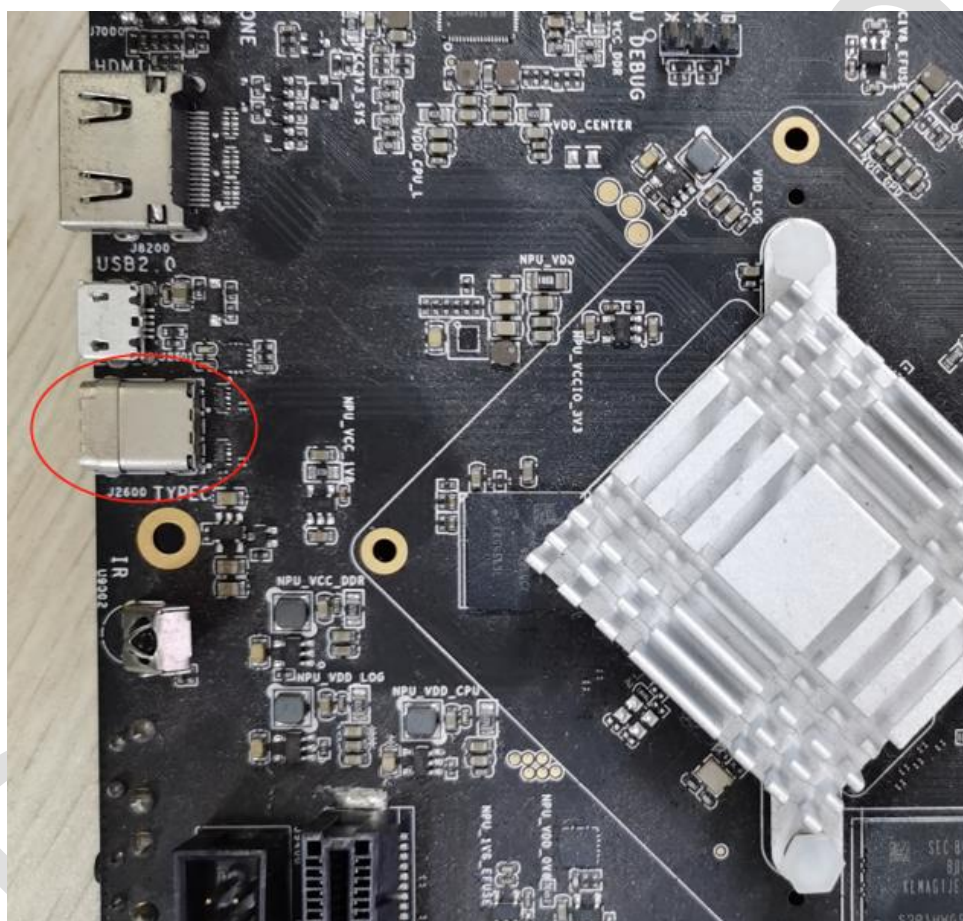


Figure 2-1 RK3399Pro USB-OTG port

The location of the RK1808 USB-OTG interface is shown in the figure below, and it is connected to the PC via a Micro-USB cable:



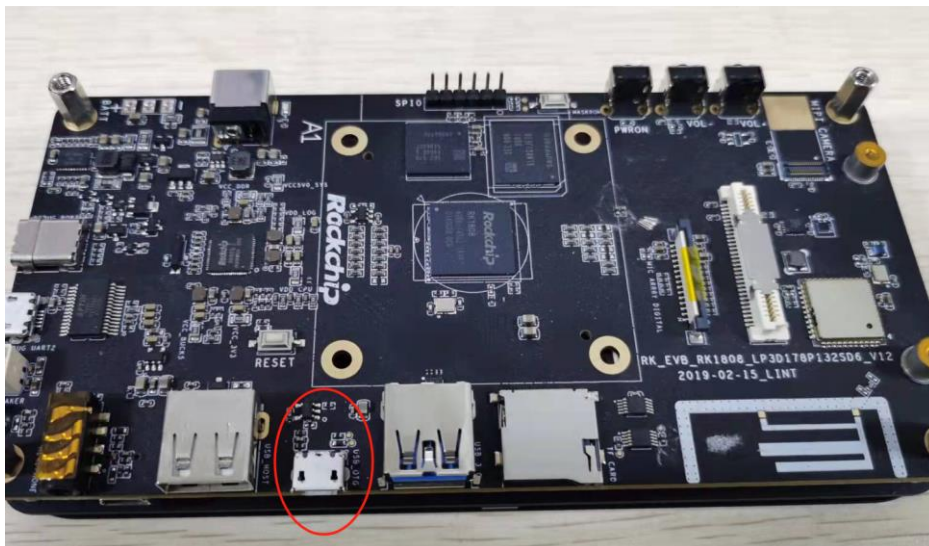


Figure 2-2 RK1808 USB-OTG port

The position of the USB-OTG interface of RV1109/RV1126 is shown in the red circle as shown in the figure below, and it is connected to the PC through the Micro-USB cable:

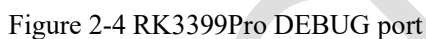


Figure 2-3 RV1109/RV1126 USB-OTG port

## 2.4.2 Connect the device through the DEBUG port

In the model evaluation or model deployment step, if you encounter an error on the development board, connection to the NPU device through the DEBUG port can help to grab the error log.

The location of the RK3399Pro NPU DEBUG interface is shown in the figure below. It is connected to the PC through the USB-serial conversion board. The PC needs to install Minicom or Putty etc. software:



The position of the RV1109/RV1126 DEBUG interface is shown in the red circle in the figure below, and it is connected to the PC through the Micro-USB cable:

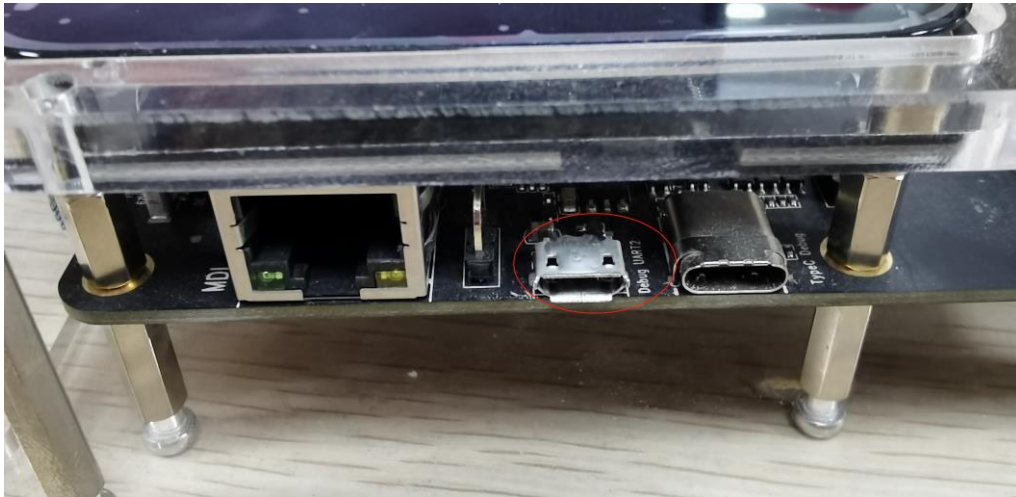


Figure 2-6 RV1109/RV1126 DEBUG port

Note: The DEBUG port and the USB-OTG port are two different ports. When troubleshooting, it is usually necessary to connect both ports to the PC.

### 2.4.3 Confirm that the device is already connected

If the device is connected correctly, execute the "`python -m rknn.bin.list_devices`" command on the PC's terminal to list the IDs of the connected devices. The reference output is as follows:

```
*****
all device(s) with adb mode:
cf8f01ae745b49ce
all device(s) with ntb mode:
1126
*****
```

It's shown that the PC is currently connected to two development boards, the device id of the first development board is cf8f01ae745b49ce, and the device id of the second development board is 1126.

**Note:**

1.If you are using an x86\_64 Linux system, you may need to update the USB device permissions, for the first time to connect to a Rockchip NPU devices, otherwise the system may not have read and write permissions on the devices. The update way is to execute the update\_rk1808\_usb\_rule.sh script in the platform-tools/update\_rk\_usb\_rule/linux/ directory, and restart the PC system on after execution.

2.If you are using Rockchip NPU device on Windows, you need to turn on the NTB communication mode first (RK1808 and Toybrick development boards are turned on by default). Open NTB method: Connect the development board to the PC through the DEBUG port; use putty and other software to enter the development board system through the serial mode; modify the file /etc/init.d/.usb\_config, add a line in this file: usb\_ntb\_en, and then restart the development board. The original usb\_adb\_en should be kept..

#### 2.4.4 Connection FAQs

For common problems of device connection, please refer to the document:

*Rockchip\_Trouble\_Shooting\_RKNN\_Toolkit\_EN.pdf.*

## 3 Usage

### 3.1 Main steps

The main usege steps of RKNN-Toolkit can be divided into three stages:

1. Model conversion: Convert the model trained and exported from the opensource DL framework (Caffe, Darknet, etc.) into an RKNN model that RKNPU can excute.
2. Model evaluation: Through inference, eval\_perf, eval\_memory and other interfaces to evaluate the accuracy, performance, and memory usage of the RKNN model.
3. Model deployment: After confirming that the RKNN model meets the deployment requirements through the model evaluation, the RKNN model can be deployed through the RKNN Toolkit Python API or the RKNN C API. For more detail about using RKNN Toolkit Python API for model deployment, please refer to Chapter 3.4; for more detail about using RKNN C API for model deployment, please refer to RKNN C API related documents:

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

### 3.2 Model conversion

In model conversion stage, RKNN Toolkit convert the models, exported by various opensource DL frameworks, and generate RKNN models that Rockchip NPU can excute. Please refer to section 1.4 for the support of RKNN Toolkit for each DL framework.

**Note:** Because the model conversion requires more CPU (or GPU), memory and other resources, it is recommended to perform this operation on a PC that meets the system configuration requirements of chapter 2.1, but not do it on development borad.

#### 3.2.1 Model conversion process

The main workflow of model conversion is shown in the figure below:



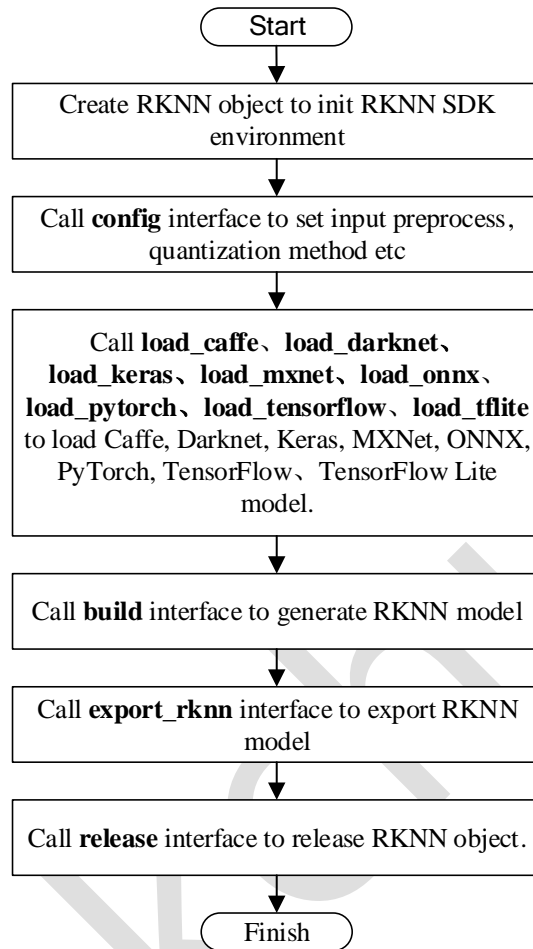


Figure 3-1 RKNN model conversion process

After successfully performing the above steps, the RKNN model will be saved in the specified directory. The subsequent evaluation and deployment will be based on the RKNN model.

**Note:** Please refer to Chapter 7 for the detailed description of each interface used in the above process.

## 3.2.2 Model quantization

### 3.2.2.1 Model quantization overview

When use this method, user loads the well-trained float point model, and RKNN Toolkit will do the quantization according to the dataset provided by user. Dataset should try to cover as many input type of model as possible. To make example simple, generally put only one picture. Suggest to put more.

RKNN supports two kinds for model quantization:

- RKNN Toolkit quantize the float model according to the calibration dataset and generate the

quantized RKNN model.

- Support quantize type: int16, int8, uint8
- Quantize method: Post Training Static Quantization
- Support quantize schema: per-tensor(or per-layer), **per-channel is not supported**
- Loading quantized model, which exported from the opensource DL framework, RKNN Toolkit can use the already exists quantization information to generate the quantized RKNN model.
  - Support framework(The main supported versions are in brackets, it's recommended to export the quantized model by that version):  
Pytoch(v1.9.0)、Onnx(Onnxruntime v1.5.1)、Tensorflow、Tflite
  - Quantized method: Post Training Static Quantization, Quantization Aware Training(QAT)

### 3.2.2.2 Model quantization detail

- **Post training static quantization**

Currently RKNN Toolkit supports three kinds of quantization methods:

- asymmetric\_quantized-u8 (default)

This is the quantization method supported by tensorflow, which is also recommended by Google.

According to the description in the article of Quantizing deep convolutional networks for efficient inference: A whitepaper, the accuracy loss of this quantization method is the smallest for most networks.

Its calculation formula is as follows:

$$\begin{aligned} quant &= round\left(\frac{float\_num}{scale}\right) + zero\_point \\ quant &= cast\_to\_bw \end{aligned}$$

Where 'quant' represents the quantized number; 'float\_num' represents float; data type of 'scale' is float32; data type of 'zero-points' is int32, it represents the corresponding quantized value when the real number is 0. Finally saturate 'quant' to [range\_min, range\_max].

$$\begin{aligned} range\_max &= 255 \\ range\_min &= 0 \end{aligned}$$

Currently only supports the inverse quantization of u8, the calculation formula is as follows:

$$\text{float\_num} = \text{scale}(\text{quant} - \text{zero\_point})$$

#### ■ dynamic\_fixed\_point-8

For some models, the quantization accuracy of dynamic\_fixed\_point-8 is higher than asymmetric\_quantized-u8.

Its calculation formula is as follows:

$$\begin{aligned}\text{quant} &= \text{round}(\text{float\_num} * 2^{\text{fl}}) \\ \text{quant} &= \text{cast\_to\_bw}\end{aligned}$$

Where ‘quant’ represents the quantized number; ‘float\_num’ represents float; ‘fl’ is the number of digits shifted to the left. Finally saturate ‘quant’ to [range\_min, range\_max].

$$\begin{aligned}\text{range\_max} &= 2^{bw-1} - 1 \\ \text{range\_min} &= -(2^{bw-1} - 1)\end{aligned}$$

If ‘bw’ equals 8, the range is [-127, 127].

#### ■ dynamic\_fixed\_point-16

The quantization formula of dynamic\_fixed\_point-16 is the same as dynamic\_fixed\_point-8, except bw=16. For RK3399pro/RK1808, there is 300Gops int16 computing unit inside NPU, for some quantized to 8 bit network with relatively high accuracy loss, you can consider to use this quantization method.

#### ● Quantization aware training(QAT)

A model with quantized params/weights can be obtained through the Quantization-aware training. RKNN Toolkit currently supports TensorFlow and PyTorch frameworks for QAT models. Please refer to the following link for the technical details of QAT:

Tensorflow:

[https://www.tensorflow.org/model\\_optimization/guide/quantization/training](https://www.tensorflow.org/model_optimization/guide/quantization/training)

PyTorch:



<https://pytorch.org/blog/introduction-to-quantization-on-pytorch/>

This method requires to use the original framework to train (or fine tune) to obtain a quantized model, and then use RKNN Toolkit to import the quantized model (you need to set `do_quantization=False` in the build setting during model conversion). At this time, RKNN Toolkit will use the quantized parameters of the model, so theoretically there will be almost no loss of accuracy comparing quantization via RKNN Toolkit.

**Note:** RKNN Toolkit also supports post-training quantization models of ONNX, PyTorch, TensorFlow, and TensorFlow Lite. The model conversion method is the same as the quantization-aware training model. `Do_quantization` should be set to `False` when calling the build interface. Update to RKNN Toolkit v1.7.0 or later for loading quantized onnx model. Update to RKNN Toolkit v1.7.1 or later for loading quantized PyTorch model.

### 3.2.3 Model convert example

Taking PyTorch framework for example, a case of model conversion is as follows:

```
import torchvision.models as models
import torch
from rknn.api import RKNN

PT_PATH = './resnet18.pt'

def export_pytorch_model():
    net = models.resnet18(pretrained=True)
    net.eval()
    trace_model = torch.jit.trace(net, torch.Tensor(1,3,224,224))
    trace_model.save(PT_PATH)

if __name__ == '__main__':

    # export PT model
    export_pytorch_model()

    # create RKNN module
    rknn = RKNN()

    # Setting preprocess parameters
    rknn.config(mean_values=[[123.675, 116.28, 103.53]], std_values=[[58.395, 58.395,
58.395]], reorder_channel='0 1 2')

    # load pytorch model
    ret = rknn.load_pytorch(model=PT_PATH, input_size_list=[[3, 224, 224]])
    if ret != 0:
        print('Load Pytorch model failed!')
        exit(ret)

    # build quantized RKNN model
    ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
    if ret != 0:
        print('Build model failed!')
        exit(ret)

    # export RKNN model to specific path
    ret = rknn.export_rknn('./resnet18.rknn')
    if ret != 0:
        print('Export resnet18.rknn failed!')
        exit(ret)

    rknn.release()
```

For other frameworks, please refer to the examples in the SDK/examples directory.

For the quantized models conversion, please refer to the examples in the SDK/examples/common\_function\_demos/ directory.

### 3.2.4 Model conversion FAQs

For common problems of model conversion, please refer to the document:

*Rockchip\_Trouble\_Shooting\_RKNN\_Toolkit\_EN.pdf*

## 3.3 Model evaluation

After the RKNN model is obtained through model conversion, the Python interface provided by RKNN Toolkit can be used to evaluate the accuracy, performance, and memory usage of RKNN model on the Rockchip NPU development board.

**Note:**

1. Before using the Rockchip NPU development board for evaluation, please refer to chapter 2.4 to correctly connect the PC with the development board, and ensure that the RKNN Toolkit can recognize the device.
2. If it is evaluated on the RK3399Pro development board, because it has own NPU, there is no need to connect the Rockchip NPU device again. The RKNN Toolkit Python interface only supports running on the RK3399Pro with Debian firmware. If the firmware of the development board is an Android system, please use the PC to connect to RK3399Pro for evaluation.

The typical evaluation process is shown in the figure below:

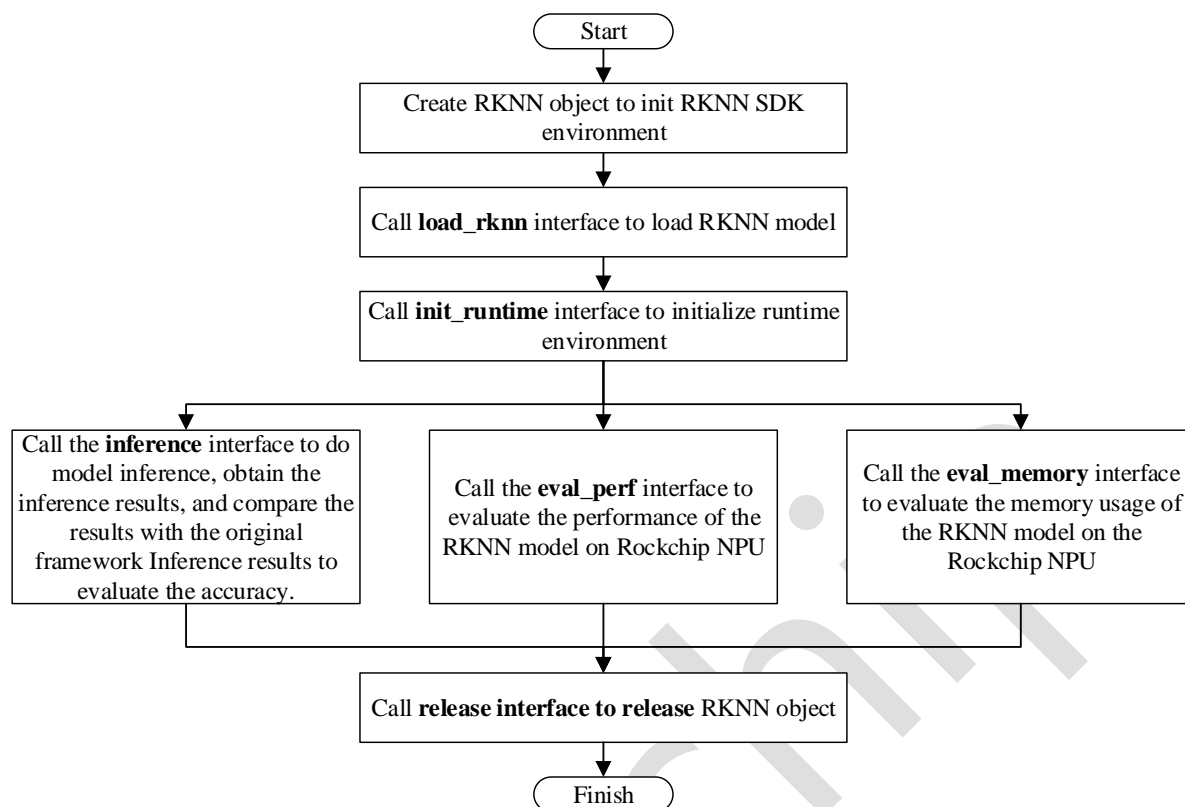


Figure 3-2 RKNN model evaluation process

Model evaluation is mainly divided into three parts: accuracy evaluation, performance evaluation and memory evaluation.

- **Accuracy evaluation:** Evaluate the accuracy of the inference results of the RKNN model.
- **Performance evaluation:** Evaluate the time-consuming inference of the RKNN model on the development board.
- **Memory evaluation:** Evaluate the memory usage on Rockchip NPU during RKNN model inference.

Accuracy evaluation related content will be expanded in detail in Chapter 4, performance evaluation related content will be expanded in detail in Chapter 5, and memory evaluation related content will be expanded in detail in Chapter 6.

### 3.4 Model deployment

This section mainly introduces how to use the Python interface provided by RKNN Toolkit for

deploying the RKNN model, already evaluated and meets the accuracy requirement in the previous step, on a Rockchip NPU device.

### 3.4.1 Prepare project files

The simplest application only needs to include input data, RKNN model and application script.

Here is an ImageNet image classifier based on the resnet18 model as an example. The project contains the following files:

```
ai_demo/
├── imagenet_classes.txt
├── imgs
│   └── space_shuttle_224.jpg
├── resnet18_classifier.py
└── resnet18.rknn
```

The file description is as follows:

- imagenet\_classes.txt: A file that records image categories.
- imgs directory: Store pictures to be classified, here is only one picture space\_shuttle\_224.jpg, you can store more pictures if needed.
- resnet18\_classifier.py: The main program for image classification. The detailed description of the program will be given in the next section.
- resnet18.rknn: The RKNN model converted according to Chapter 3.2 (Note: The target device of this model is RK1808, which can only be deployed on RK1808 or RK3399Pro. If you want to deploy on RV1109 or RV1126, you need to modify the target\_platform parameter of the build interface, please refer to Chapter 8.2 for specific instructions).

### 3.4.2 Application example

The running process of the simplest application is shown in the figure below:

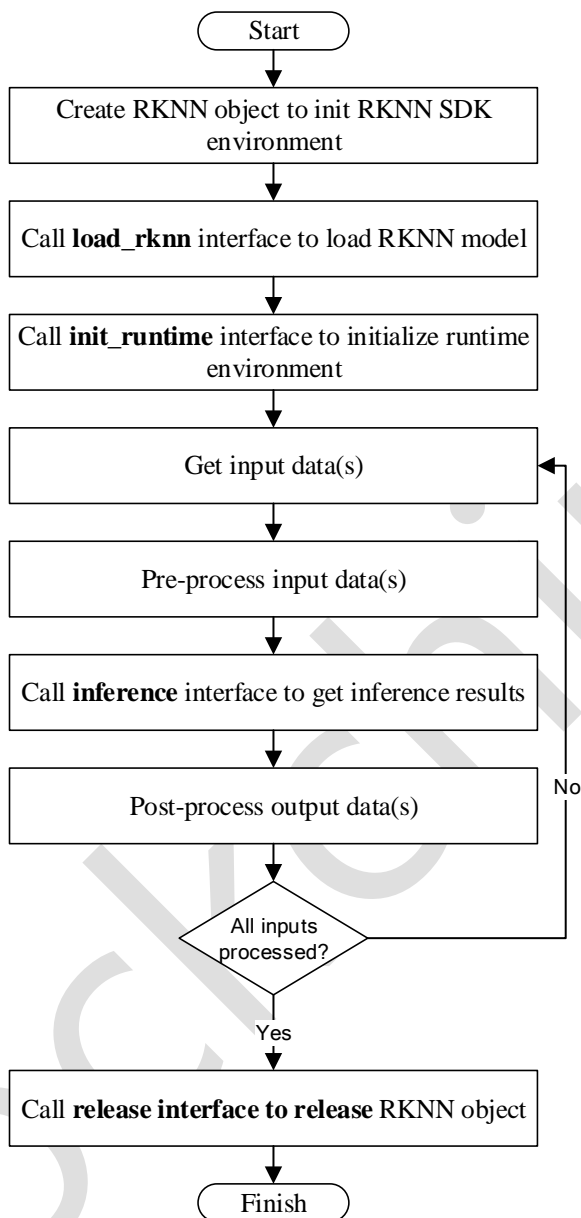


Figure 3-3 Simple application flowchart

In this process, the core steps are to load the RKNN model, initialize the runtime environment, prepare the input data, model inference and post-processing of the results.

Here is an example of image classification to show how to deploy the classification model resnet18.rknn to the RK1808 development board.

```
import os
import cv2
import torch
import numpy as np
from rknn.api import RKNN

RKNN_PATH = './resnet18.rknn'
IMGS_DIR = './imgs'

def classification_post_process(output, categories):
    output = output.reshape(-1)
    probabilites = np.exp(output)/sum(np.exp(output))
    reverse_sort_index = np.argsort(output)[::-1]
    print('-----TOP 1-----')
    for i in range(1):
        print(categories[reverse_sort_index[i]], ':', probabilites[reverse_sort_index[i]])

if __name__ == '__main__':
    # Create RKNN object
    rknn = RKNN()

    # Load RKNN model
    ret = rknn.load_rknn(path=RKNN_PATH)
    if ret != 0:
        print('Load Pytorch model failed!')
        exit(ret)

    # Init_runtime, set the target as RK1808
    # If there is only one device connected, device_id can be left blank
    ret = rknn.init_runtime(target='rk1808', device_id='1808')
    if ret != 0:
        print('Init runtime environment failed')
        exit(ret)

    # load imagenet_classes.txt to obtain class info
    with open("imagenet_classes.txt", "r") as f:
        categories = [s.strip() for s in f.readlines()]

    for img_file in os.listdir(IMGS_DIR):
        img_path = os.path.join(IMGS_DIR, img_file)

        # load image, resize to target shape, convert BGR2RGB.
        img = cv2.imread(img_path)
        img = cv2.resize(img, (224, 224), interpolation=cv2.INTER_CUBIC)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Call the inference interface
        outputs = rknn.inference(inputs=[img])
```

```
# Call the post-processing interface to get the category, print category and probability
of the picture
classification_post_process(outputs[0], categories)

# After all the images are processed, released RKNN object.
rknn.release()
```

For the deployment of the target detection model, please refer to SDK/examples/caffe/vgg-ssd, SDK/examples/darknet/yolov3, SDK/examples/onnx/yolov5, etc.

For the deployment of the image segmentation model, please refer to SDK/examples/mxnet/fcn\_resnet101, etc.

For the deployment of other classification models, please refer to SDK/examples/caffe/mobilenet\_v2, SDK/examples/keras/xception, SDK/examples/mxnet/resnext50, etc.

**Note:**

1. If the Python interface is used in the final product deployment, it is recommended to use RKNN Toolkit Lite, which only keep the inference function of RKNN Toolkit, which can greatly reduce the space occupied by the SDK. For a detailed introduction of RKNN Toolkit Lite, please refer to the document: *Rockchip\_User\_Guide\_RKNN\_Toolkit\_Lite\_EN.pdf*.

2. If you use RKNN C API when deploying the final product, please refer to the RKNN C API documentation and examples:

- RK1808 / RV1109 / RV1126:

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

- RK3399Pro:

[https://github.com/airockchip/RK3399Pro\\_npu/tree/main/rknn-api](https://github.com/airockchip/RK3399Pro_npu/tree/main/rknn-api)

### 3.4.3 Application excuttion

As in the example in chapter 3.4.2, you can directly execute the "python resnet18\_classifier.py" command in the python environment where RKNN Toolkit is installed.

**Note:** Please connect the Rockchip NPU development board first according to the instructions in



chapter 2.4.

### 3.4.4 Model pre-compile

When Rockchip NPU loads the RKNN model, it constructs a run-time graph based on the network structure, and then converts it into a command to run on the NPU. For model with complex network structure, this step will consume a significant amount of time. In order to solve this problem, RKNN Toolkit provides a model pre-compilation function, which reduces the model initialization time to less than 1 second.

The RKNN Toolkit utilizes the **export\_rknn\_precompile\_model** interface to accomplish online precompilation functionality. When using this precompilation method, it is necessary to connect a Rockchip NPU development board.

For using online pre-compilation, you can refer to the following examples:

**SDK/examples/common\_function\_demos/export\_rknn\_precompile\_model/**

### 3.4.5 Model encryption

In order to avoid the leakage of information such as model structure and weight, RKNN Toolkit provides model encryption function.

RKNN Toolkit provides 3 encryption levels. 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.

The model encryption process is shown in the following figure:

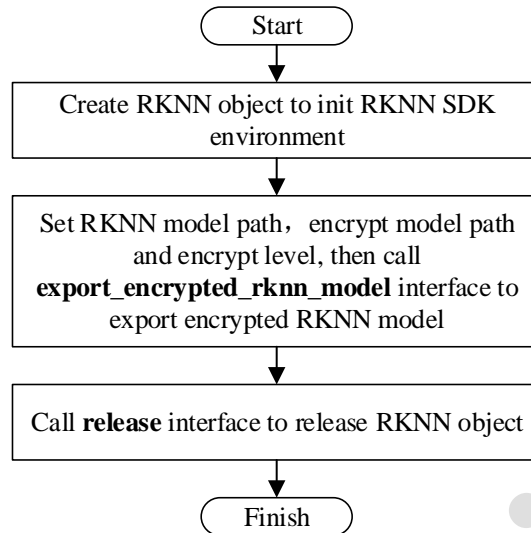


Figure 3-4-5-1 RKNN model encrypt process

The deployment process of the encryption model is the same as normal, no additional operations such as decryption are required.

### 3.4.6 Multi-model scheduling

In the scenario of multiple RKNN models, each RKNN model needs to use limited NPU resources to complete inference tasks, which leads to the problem of NPU resource preemption. NPUs such as RK1808 and RV1109 do not have the task preemption function. Once a model starts inference, it can only release NPU resources after the inference of this model ends, and other models will wait for the resource. This makes it easy for the NPU resource to be occupied by the large model for a long time, and the small model with a higher priority has no chance to run. In order to solve this problem, RKNN Toolkit provides the function of model segmentation, and provides a simple and feasible scheduling method from the software level.

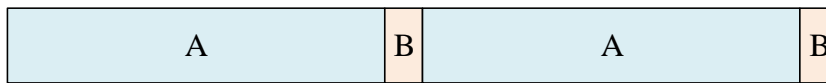
#### 3.4.6.1 Introduction to model segmentation function

RKNN-Toolkit can divide the ordinary RKNN model into multiple segments through the `export_rknn_sync_model` interface. Each segment (the model without the model segmentation function is recognized as single-segment) has an equal chance of preempting the NPU. After the execution of a segment

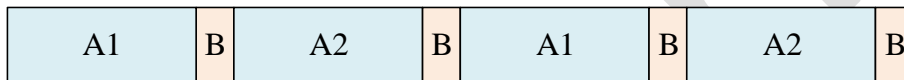
is completed, the NPU will be actively surrendered (if the model has another segment) , The segment will be added to the end of the command queue again). At this time, if there are segments of other models waiting to be executed, the segments of other models will be executed in the order of the command queue.

Examples are as follows:

The current application uses two models, model A takes 200ms for inference, and model B takes 20ms for inference. When initializing, initialize A first, and then initialize B. If model segmentation is not used, the inference queue is as follows:



Model B infers once and wait 200ms for the next one. But if we divide model A into 2 segments, each segment is 100ms, the reasoning queue is as follows:



At this time, Model B infers once and only needs to wait for 100ms. In actual use, you can segment the model according to your needs.

**Note:** Turning on the model segmentation function will reduce the efficiency of single-model reasoning, so in a single-model scenario, it is recommended to turn off this function (do not use the model segmentation function).

#### 3.4.6.2 Model segmentation using process

The model segmentation process is shown in the figure below:

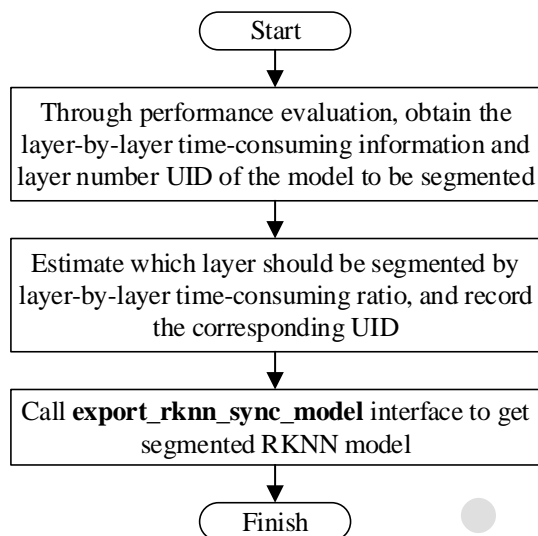


Figure 3-4 RKNN model segmentation process

**Note:**

1. Obtaining each layer performance evaluation of the model may take a long time, so it is recommended to use proportional conversion when calculating. For example, the model has three layers, and the performance evaluation takes 10ms for the A layer, 20ms for the B layer, and 10ms for the C layer. But the actual model inference time is only 24ms. According to the scale, the actual time may be 7ms for the A layer, 14ms for the B layer, and 7ms for the C layer.

2. There are two IDs for performance evaluation, Layer ID and Uid. Record the Uid and use in model segment function.

**3.4.7 Deployment FAQs**

Please refer to document: *Rockchip\_Trouble\_Shooting\_RKNN\_Toolkit\_EN.pdf*

## 4 Accuracy evaluation

This chapter will explain the methods of RKNN model accuracy evaluation in detail, common problems and common accuracy troubleshooting methods.

### 4.1 evaluation method

The simplest evaluation method is to compare the inference result of the RKNN model with the inference result of the original framework, and use Euclidean distance or cosine distance to evaluate the similarity of the two results. You can also use post-processing to verify the accuracy of the results, or directly perform accuracy tests on the data set.

Take the simplest calculation of cosine distance to evaluate the accuracy of the inference result as an example, the code is as follows:

```
import cv2
import torch
import numpy as np
from rknn.api import RKNN

PT_PATH = './resnet18.pt'
RKNN_PATH = './resnet18.rknn'
MEANS = [123.675, 116.28, 103.53]
STDS = [58.395, 58.395, 58.395]

def inference_with_rknn(target, device_id, inputs):
    # Create RKNN object
    rknn = RKNN()

    # Load RKNN model
    ret = rknn.load_rknn(path=RKNN_PATH)
    if ret != 0:
        print('Load Pytorch model failed!')
        exit(ret)

    # Init runtime
    ret = rknn.init_runtime(target=target, device_id=device_id)
    if ret != 0:
        print('Init runtime environment failed')
        exit(ret)

    # call inference interface to get result
    outputs = rknn.inference(inputs=inputs)

    # release RKNN object
    rknn.release()

    return outputs

def inference_with_torch(img):
    # Preprocess for input image
    img = (img - MEANS) / STDS
    # Loading image via OpenCV, the image channel order is HWC, need
    # to convert to NCHW for PyTorch inference.
    img = img.reshape((1, 224, 224, 3))
    img = img.transpose((0, 3, 1, 2))
    img = img.astype(np.float32)
    torch_inputs = [torch.from_numpy(img)]

    # Load PyTorch model
    net = torch.load(PT_PATH)

    # Call forward for PyTorch inference
```

```
outputs = net.forward(*torch_inputs)
return outputs

def compute_cos_dis(x, y):
    cos_dist= (x* y)/(np.linalg.norm(x)*(np.linalg.norm(y)))
    return cos_dist.sum()

if __name__ == '__main__':

    # Load test image
    img = cv2.imread('./space_shuttle_224.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Use RKNN Toolkit inference interface to get result with the RK1808
    # develop board.
    rknn_outs = inference_with_rknn(target='rk1808', device_id='1808', inputs=[img])

    # Get result with PyTorch model
    torch_outs = inference_with_torch(img=img)

    # compute and print the cos_distance between result.
    cos_dis = compute_cos_dis(rknn_outs[0],
    torch_outs[0].cpu().detach().numpy())
    print("Cosine distance of RKNN output and Torch output: {}".format(cos_dis))
```

## 4.2 Troubleshooting for accuracy issues

If the accuracy evaluation result is not satisfactory and the accuracy of the RKNN model does not reach the expected effect, please refer to the following steps to troubleshoot:

- First, ensure that the accuracy of the floating-point RKNN model is similar to the original frame test result;
  - When using the build interface to build an RKNN model, set the value of the do\_quantization parameter to False;
  - Correctly set the mean\_values, std\_values, reorder\_channel and other parameters in the config interface to ensure that these parameters are consistent with the model training;
  - If the model input is a picture, make sure that the channel order of the test picture is RGB (the image read by OpenCV has default channel order for BGR). Regardless of the order of the image channels used during training, the RKNN model will be used for RGB input when testing;
  - If the model input is 4-dimensional, make sure that the data arrangement format of the test picture

is consistent with the `data_format` parameter in the inference interface. OpenCV reads the image format is HWC, for a single image, the arrangement order is as the same as NHWC.

- When the model is quantized, it is recommended to include more representative data (the number is between 100 and 500) for the input data in the calibration data set, and do not include data independent with the testing scenario. When the calibration data set has a lot of data, it may fail due to insufficient memory when using the build interface to build the RKNN model. At this time, it is recommended to set the `batch_size` parameter in the config interface to a smaller value, such as 8, 16, etc.
- If you use a data set for evaluation, try to use a larger data set for evaluation. The classification network compares the accuracy of top1, top5, and the mAP, Recall, etc. of the detection network comparison data set.

After checking the above factors, if the inference result of the RKNN model is still incorrect, the following analysis can be done to find the operator that caused the error. Considering time-consuming and other factors, it is recommended to use a single input data for further analysis.

#### 4.2.1 Troubleshooting for float model

By setting, when performing model inference on the Rockchip NPU, the results of each layer can be saved. Comparing the results of the middle layer with the original framework, you can locate which operator has a problem. If the operator with an error through the inference result, which can be replaced with another operator, please try to replace it, and then retrain the model and convert it to RKNN for evaluation (train a few epochs for verification, and then complete training after the checking pass); if surrogate operator cannot be found, please collect the operator, parameters, input and output information or provide the minimum recurrence model containing the operator and feed back to the Rockchip NPU team for further analysis.

The specific method to get results of the middle layer is as follows:

1. If the RKNN Toolkit Python interface is used to evaluate, the development board and PC need be connected through the serial port. Find a folder on the development board to store the results of the middle



layer and store it in the folder. Set the environment variable `NN_LAYER_DUMP`, and need to restart the `rknn_server` process, the specific command is `"export NN_LAYER_DUMP=1 && restart_rknn.sh"`;

2. If the RKNN C API is used to evaluate, just set the `NN_LAYER_DUMP` environment variable on the development board. The specific command is `"export NN_LAYER_DUMP=1"`. The results of the middle layer will be saved in the directory where the inference program is executed.

**Note: If the development board is RK3399Pro, you need to perform the above operations on the built-in NPU.**

After obtaining the results of the middle layer, compare these results with the original framework. About how to obtaining the middle layer results of the original framework, please refer to the relevant information of each framework.

When comparing data, if you find that the results of the first layer are very different (the cosine distance is less than 0.98), this is usually caused by the inconsistency of the input preprocessing between RKNN and the original framework. Please first check whether the `mean_values`, `std_values`, `reorder_channel` and other parameters in the config are correctly set, or whether the data passed to the inference interface during inference is correct.

In other cases, please feedback the specific operator or minimum recurrence model to the Rockchip NPU team for further analysis.

#### 4.2.2 Troubleshooting for quantized model accuracy

- First, check whether the inference result of the float model is correct, following the previous section to achieve it; if correct, proceed to the next step.
- When quantizing, if there is an outlier warning message, you can pay attention to whether the corresponding network layer has precision loss in the precision analysis. At present, only the weight of the convolution is detected. The example and explanation of the outlier warning message are as follows:

We found outlier value, this may affect quantization accuracy

const name	abs_mean	abs_std	outlier value
convolution_at_input.12_1_1:weight	2.44	2.47	-17.494
convolution_at_input.9_130_130:weight	0.18	0.20	-11.216
convolution_at_input.17_143_143:weight	0.12	0.19	13.361, 13.317
convolution_at_input.1_156_156:weight	0.09	0.14	-10.215

- Const name indicates the name of the corresponding convolutional layer.
- Abs mean refers to the mean value of the absolute value of weight.
- Abs std refers to the standard deviation of the absolute value of weight.
- Outlier value refers to the specific value, which is usually several times the standard deviation and the mean value, resulting in uneven distribution of the entire weight and ultimately poor quantification accuracy. Please note that only the most significant outliers are printed here, which does not mean that the remaining values are evenly distributed.
- Using *accuracy\_analysis* interface for accuracy analysis. You can also refer to the method in the previous section to manually export the inference results of each layer, comparing with the original frame inference results to find the layers that caused the loss of accuracy.
- Analyze the layers with reduced accuracy.

After quantization, the accuracy may decrease in the following three situations:

- RKNN Toolkit has problems with the matching of this layer of operators. Or the driver has problems on the implementation of this operator (the inference result of this layer is usually very different from the result of the float model, and the cosine distance is less than 0.5). For this case, using hybrid quantization can always get a better result.
- The data distribution of this layer (such as weight of convolution) is not friendly to quantization (for example, the data distribution range is relatively wide, and the accuracy loss is serious after low-precision expression). For this case, using MMSE or KL divergence quantization method may help. Otherwise considering using hybrid quantization with the higher-precision method for the quantization-unfriendly operator. After all, trying to do the quantization with the original framework, such as TensorFlow or PyTorch for quantization-

aware-training, and then convert the quantized model to RKNN model.

- Some graph optimizations done by RKNN Toolkit or NPU may cause accuracy to decrease, such as replacing add or average pool with convolution. In this case, you can try to lower the optimization level (like setting the optimization\_level to 2 or 1 in the config interface).

If the accuracy of the model still cannot meet the requirements after trying the above methods, please feedback the relevant model, accuracy analysis results and accuracy test methods to the Rockchip NPU team for further analysis.

**Note:**

1. For detail in using the accuracy analysis function, please refer to chapter 4.3
2. For detail of the MMSE quantization parameter optimization algorithm, please refer to chapter 4.4.
3. For detail in using hybrid quantization interfaces, please refer to chapter 4.5.

## 4.3 Accuracy analysis

### 4.3.1 Function description

The accuracy analysis function of RKNN Toolkit can save the intermediate results of each layer of floating-point model and quantitative model inference, and use Euclidean distance and cosine distance to evaluate their similarity. By observing the changes in the similarity of each layer, you can initially locate which operators have incorrect calculation results or are unfriendly to quantization.

### 4.3.2 Usage steps

The usage steps of the quantized model accuracy analysis function are as follows:

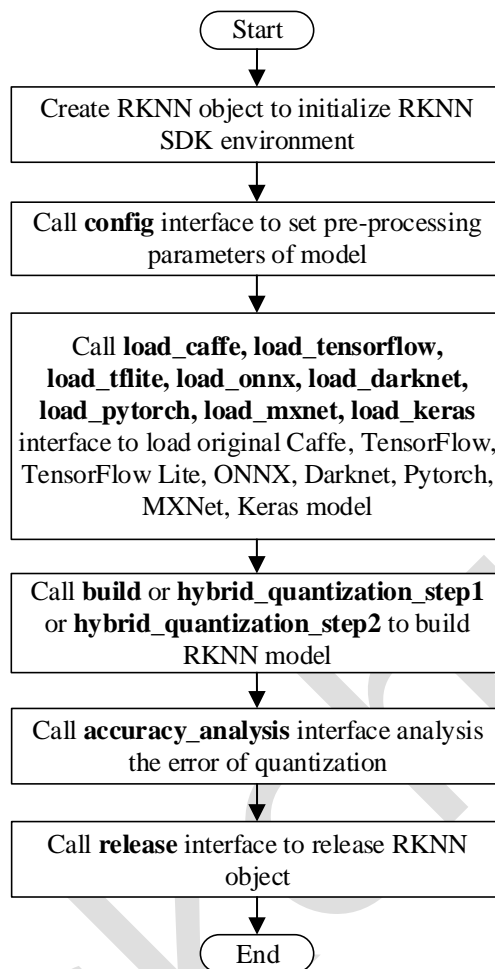


Figure 4-1 Quantitative accuracy analysis use process

**Note:**

1. The quantitative method to be analyzed needs to be specified in config.
2. The inputs specified in accuracy\_analysis can only contain one row of data.
3. The calling order of the above interfaces cannot be changed.

### 4.3.3 Output file

The directory structure of the accuracy analysis output is as follows:

```

├── entire_qnt
├── entire_qnt_error_analysis.txt
├── fp32
├── individual_qnt
├── individual_qnt_error_analysis.txt

```

The meaning of each file/directory is as follows:

- Directory `entire_qnt`: Save the results of each layer when the entire quantitative model is fully run (The data has been converted to float32).
- File `entire_qnt_error_analysis.txt`: Record the cosine distance/Euclidean distance between each layer result and the floating-point model during the complete calculation of the quantized model, and the normalized cosine distance/Euclidean distance. The smaller the cosine distance or the larger the Euclidean distance, the greater the decrease in accuracy after quantization.
- Directory `fp32`: Save the results of each layer when the entire floating-point model is completely run down, and correspond to the original model according to the order of the `order.txt` records in the directory. If the result of the floating-point model itself is not correct, please compare the results of each layer in the catalog with the results of each layer in the original framework inference to determine which layer is the problem. Then feedback to the Rockchip NPU team.
- Directory `individual_qnt`: Split the quantitative model into layers and run layer by layer. The input of each layer during inference is the result of the previous layer's inference in the floating point model. This can avoid accumulated errors.
- File `individual_qnt_error_analysis.txt`: Record the cosine distance/Euclidean distance between the result of each layer and the floating-point model when the quantized model is run layer by layer, and the normalized cosine distance/Euclidean distance. The smaller the cosine distance or the larger the Euclidean distance, the greater the decrease in accuracy after quantization.

If the target is set, the following content will appear in the directory:

```
|— individual_qnt_error_analysis_on_npu.txt
|— qnt_npu_dump
```

- File `individual_qnt_error_analysis_on_npu.txt`: Record the cosine distance/Euclidean distance between the result of each layer and the floating-point model when the quantized model runs layer by layer on the hardware device, and the normalized cosine distance/Euclidean distance. The smaller the cosine distance or the larger the Euclidean distance, the greater the decrease in

accuracy after quantization.

- Directory `qnt_npu_dump`: Split the quantized model into layers and put them to run on the NPU device one by one. The input used is the result of the previous layer of the floating-point model. This directory saves the result of the quantized model when it is actually run on the NPU layer by layer (The data has been converted to float32).

## 4.4 Optimized Quantization

RKNN Toolkit provides two methods of quantization parameter optimization: MMSE and KL divergence, which search for the optimal combination of quantitative parameters according to their respective algorithms to improve accuracy.

If you want to use the MMSE quantization parameter optimization method, specify the value of the `quantized_algorithm` parameter as "mmse" in the config interface, and if you want to use the KL dispersion method, specify the value of the parameter as "kl\_divergence".

## 4.5 Hybrid quantization

The quantization function provided by RKNN-Toolkit can ensure the accuracy of the model as much as possible on the basis of improving the speed of model inference. However, there are still some special models that have more accuracy drops after quantization. In order to achieve a better balance between performance and accuracy, RKNN-Toolkit has provided hybrid quantization function starting from version 1.0.0. Users can specify whether each layer should be quantized, and the quantization parameters can also be modified.

Note:

1. The `examples/common_function_demos` directory provides a hybrid quantization example named `hybrid_quantization`. Users can refer to this example for hybrid quantification practice.

### 4.5.1 Instructions of hybrid quantization

Currently, RKNN-Toolkit has three kind of ways to use hybrid quantization:

1. Convert quantized layer to non-quantized layer (for example: using float32). Due to the low non-quantitative computing power on the NPU, the inference speed will be reduced to a certain extent
2. Convert non-quantized layer to quantized layer.
3. Modify quantization parameters of pointed quantized layer.

### 4.5.2 Hybrid quantization profile

When using the hybrid quantization feature, the first step is to generate a hybrid quantization profile, which is briefly described in this section.

When the hybrid quantization interface `hybrid_quantization_step1` is called, a YAML configuration file named “`{model_name}.quantization.cfg`” is generated in the current directory. The configuration file format is as follows:

```
%YAML 1.2

# add layer name and corresponding quantized_dtype to customized_quantize_layers, e.g
conv2_3: float32
customized_quantize_layers: {}
quantize_parameters:
  '@attach_concat_1/out0_0:out0':
    dtype: asymmetric_affine
    method: layer
    max_value:
      - 10.097497940063477
    min_value:
      - -52.340476989746094
    zero_point:
      - 214
    scale:
      - 0.24485479295253754
    qtype: u8

.....

  '@FeatureExtractor/MobilenetV2/Conv/Conv2D_230:bias':
    dtype: asymmetric_affine
    method: layer
    max_value:
      - 0.00026041566161438823
    min_value:
      - 0.00026041566161438823
    zero_point: 0
    scale:
      - 0.00026041566161438823
    qtype: i32
```

First line is the version of yaml. Second line is separator. Third line is comment. Followed by the main content of the configuration file.

The first line of the body of the configuration file is a dictionary of customized quantize layers, add the layer names and their corresponding quantized type(choose from **asymmetric\_affine-u8**, **dynamic\_fixed\_point-i8**, **dynamic\_fixed\_point-i16**, **float32**) to be changed to customized quantize layers. Since version 1.6.0, the first step of hybrid quantization will give the layers that may improve the accuracy according to certain rules, and specify the quantization method as **dynamic\_fixed\_point-i16**. These layers are for reference only. If you only want to modify the following quantization parameters without using other quantization methods, you need to add {} after this line.

Next is a list of model layers, each layer is a dictionary. The key of each dictionary is composed of



@{layer\_name}\_{layer\_id}:[weight/bias/out{port}], where layer\_name is the name of this layer and layer\_id is an identification of this layer. RKNN-Toolkit usually quantize weight/bias/out when do quantization, and use multiple out0, out1, etc. for multiple outputs. The value of the dictionary is the quantization parameter. If the layer is not be quantized, there is only “dtype” item, and the value of “dtype” is None.

### 4.5.3 Usage flow of hybrid quantization

When using the hybrid quantization function, the specific steps are carried out in four steps.

Step1, load the original model and generate a quantize configuration file, a model structure file and a model weight bias file. The specific interface call process is as follows:

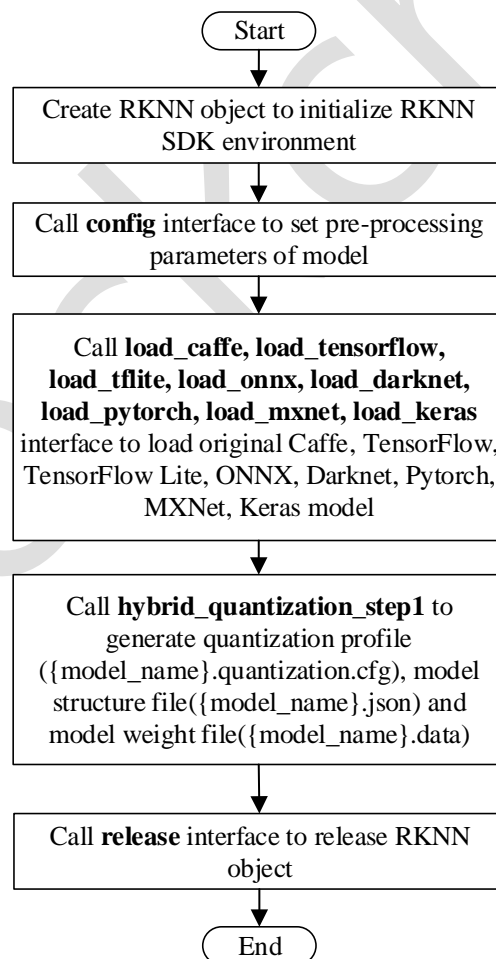


Figure 4-2 call process of hybrid quantization step 1

Step 2, Modify the quantization configuration file generated in the first step.

- If some quantization layers is changed to a non-quantization layer, find the layer that is not to be quantized, and add these layers name and float32 to customized\_quantize\_layers, such as “<layername>: float32”. Other quantization methods can also be used. For example, the original asymmetric\_affine-u8 is used, but it can also be changed to dynamic\_fixed\_point-i8 or dynamic\_fixed\_point-i16. But a model can only have two quantitative methods at most at the same time. The layer name is best enclosed in double quotes to avoid parsing failure due to special characters.
- If some layers are changed from non-quantization to quantization, add these layers named and corresponding quantize type to customized\_quantize\_layers, such as “<layername>: asymmetric\_affine-u8”.
- If the quantization parameter is to be modified, directly modify the quantization parameter of the specified layer.

**Note: The quantization config file will give some suggestions for hybrid quantization since version 1.6.0. This suggestions are for reference only.**

Step 3, generate hybrid quantized RKNN model. The specific interface call flow is as follows:

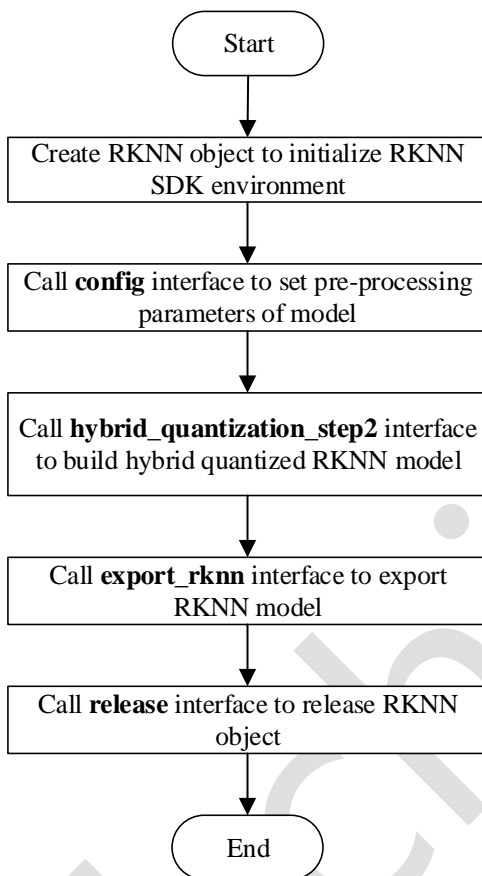


Figure 4-3 call process of hybrid quantization step 3

Step 4, use the RKNN model generated in the previous step to inference.

## 4.6 Export model quantization parameters

RKNN Toolkit has provided the function of exporting model quantization parameters since version 1.7.5. Users can obtain quantitative information of all nodes of the model through this function.

### 4.6.1 Usage flow

The process of exporting the quantization parameters of the RKNN model is shown in the figure below:

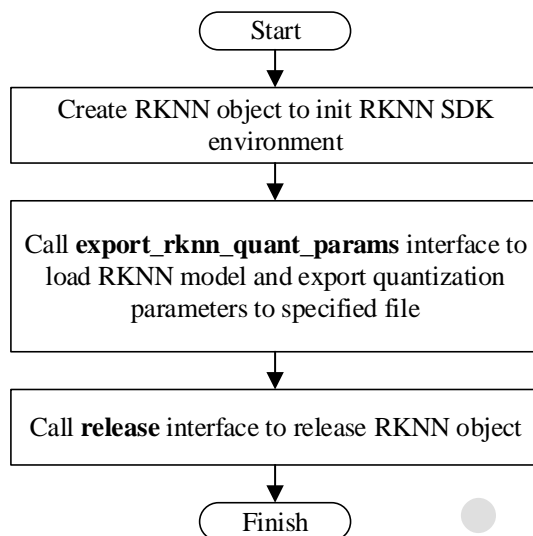


Figure 4-4 RKNN model quantization parameter export process

Note:

1. The `only_io` parameter can specify whether to export only the quantization parameters of the model input and output nodes.
2. If the model is pre-compiled, only the quantization parameters of input and output nodes can be exported; if it is an encrypted model, the quantization parameters can not be obtained, and an error code will be returned when calling this interface.
3. The quantization parameters will be saved in `{model_nmae}_quant_param.yml` file.

#### 4.6.2 Quantization parameter file description

The quantization parameters are stored in the yml file in form of a dictionary.

The specific format is as follows:

```
%YAML 1.2
model_name: MobilenetV1A
quant_params:
  input_0:
    qtype: asymmetric_affine
    dtype: u8
    scale: 0.007843137718737125
    zero_point: 127
  MobilenetV1/Predictions/Reshape_1_58_59:
    qtype: none
    dtype: fp16
  MobilenetV1/MobilenetV1/Conv2d_0/Relu6_1_acuity_mark_perm_60:out0:
    op: VSI_NN_OP_PERMUTE
    qtype: asymmetric_affine
    dtype: u8
    scale: 0.007843137718737125
    zero_point: 127
  MobilenetV1/MobilenetV1/Conv2d_0/Relu6_1:out0:
    op: VSI_NN_OP_CONV2D
    qtype: asymmetric_affine
    dtype: u8
    scale: 0.0235294122248888
    zero_point: 0
  MobilenetV1/MobilenetV1/Conv2d_0/Relu6_1:in1:
    op: VSI_NN_OP_CONV2D
    qtype: asymmetric_affine
    dtype: u8
    scale: 0.019131293520331383
    zero_point: 148
```

There are two keywords **model\_name** and **quant\_params** in each quantization parameter file. The **model\_name** stores the name of the model, and the **quant\_params** stores the quantization parameters of each layer. Taking convolution as an example, “MobilenetV1/MobilenetV1/Conv2d\_0/Relu6\_1:out0” in the above table indicates the quantization parameter of the convolution layer output tensor, “MobilenetV1/MobilenetV1/Conv2d\_0/Relu6\_1:in1” indicates the quantization parameter of the convolution layer weight, “MobilenetV1/MobilenetV1 /Conv2d\_0/Relu6\_1:in2” represents the quantization parameter of the convolutional layer bias.

The specific content of quantization parameters is related to the selected quantization method.

The quantization parameter of an unquantized tensor consists of three parts: op, qtype and dtype. The

op records the operator name, the value of qtype is none, and the value of dtype is fp16. For nodes quantized using asymmetric\_affine-u8, the quantization parameter has two more fields, scale and zero\_point, where the qtype value is asymmetric\_affine, and the dtype is u8 or i32 (bias is generally expressed in i32). For a tensor quantized with dynamic\_fixed\_point-i16, the quantization parameter has an additional fl field, where the dtype value is i16 or i64 (bias is generally i64), the qtype value is dynamic\_fixed\_point, and fl represents the displacement in the quantization formula. The tensor quantized with dynamic\_fixed\_point-i8 has the same quantization parameters as dynamic\_fixed\_point-i16, except that its dtype is generally i8 or i32 (bias is generally expressed in i32).

Note: fp16 is short for float16, u8 is short for uint8, i8 is short for int8, i16 is short for int16, i32 is short for int32, and i64 is short for int64.

## 4.7 FAQs

For accuracy evaluation FAQs, please refer to the documentation:

*Rockchip\_Trouble\_Shooting\_RKNN\_Toolkit\_EN.pdf*

## 5 Performance evaluation

This section will explain in detail for performance evaluation methods, optimization methods, common problems about RKNN model.

### 5.1 Evaluation method

RKNN Toolkit provides model performance evaluation interface `eval_perf`. This interface can evaluate the inference-time for the model run on Rockchip NPU. The evaluation record inference-time for both full-model and each layer .

The main interfaces and their config parameters involved in the performance evaluation step are as follows:

- `init_runtime`: Initialize the runtime environment. The **target** and **perf\_debug** parameters determine the target device during performance evaluation and whether to evaluate the time-consuming of each layer.
- `eval_perf`: Performance evaluation interface. Calling this interface will return the time used for each inference of the RKNN model on the specified device according to the parameters set by the `init_runtime` interface. If the **perf\_debug** parameter is set to True, the inference-time information of each layer will also be returned. The **loop\_cnt** parameter of this interface can specify the number of loop inferences.

Note:

1. If evaluated on the simulator, no matter whether **perf\_debug** is set to True or not, the inference-time information of each layer will be given. The evaluation result with the simulator may differ from that of the actual EVB board, and it is recommended to perform the evaluation with the connection of EVB board before model deployment.
2. If the **perf\_debug** parameter sets True, in order to collect the inference-time information of each layer, the NPU driver will repeatedly recall some code in each layer, resulting in a total inference-

time higher than the actual case, which is normal and needn't worry about it.

3. For more detail about `init_runtime` and `eval_perf` interfaces, please refer to chapters 7.7 and 7.9.

## 5.2 Evaluation example

Please refer to the following code to use the RKNN Toolkit interface to complete the performance evaluation:

Rockchip



```
import cv2
import torch
import numpy as np
from rknn.api import RKNN

RKNN_PATH = './resnet18.rknn'

def eval_perf_with_simulator():
    # create RKNN object
    rknn = RKNN()

    # load RKNN model resnet_18 from current directory
    ret = rknn.load_rknn(path=RKNN_PATH)
    if ret != 0:
        print('Load Pytorch model failed!')
        exit(ret)

    # init RKNN runtime
    ret = rknn.init_runtime()
    if ret != 0:
        print('Init runtime environment failed')
        exit(ret)

    # call eval_perf interface to do performance evaluation
    rknn.eval_perf()

    # release RKNN object
    rknn.release()

def eval_perf_with_rk1808():
    # create RKNN object
    rknn = RKNN()

    # load RKNN model resnet_18 from current directory
    ret = rknn.load_rknn(path=RKNN_PATH)
    if ret != 0:
        print('Load Pytorch model failed!')
        exit(ret)

    # init RKNN runtime
    ## The default perf_debug is False.
    ## If you want to evaluate the time-consuming of each layer, set the value of this
    ## parameter to True
    ret = rknn.init_runtime(target='rk1808', device_id='1808', perf_debug=True)
    if ret != 0:
        print('Init runtime environment failed')
        exit(ret)

    # call eval_perf interface to do performance evaluation
```

```
# loop_cnt specifies the number of loops and returns the average time
rknn.eval_perf(loop_cnt=100)

# release RKNN object
rknn.release()

if __name__ == '__main__':

    # Use the simulator for performance evaluation (only available on x86_64 Linux)
    # eval_perf_with_simulator()

    # Use RK1808 for performance evaluation
    eval_perf_with_rk1808()
```

Note: For models with higher fps, it is recommended to set loop\_cnt to a larger value to obtain stable performance data.

## 5.3 Description of evaluation results

The simulator evaluation results are slightly different from the evaluation results on the development board. The following sections describe their evaluation results respectively.

### 5.3.1 Description of the performance evaluation results of the development board

When perf\_debug is set to False, only the average inference time (unit is us) and the corresponding frame rate are printed.

For example:

```
=====
Average inference Time(us): 5438.98
FPS: 183.86
=====
```

When **perf\_debug** sets True, the performance evaluation result consists of the following:

- inference-time layer by layer: includes layer ID, layer name, operator name, UID and the inference-time;
- Total time: the inference-time of the whole model.
- FPS: The frame rate calculated according to the total time.
- Operator performance statistics table: Sort all types of operators in descending order of time

consumption, and show top5.

When `perf_debug` sets `True`, the performance evaluation results of the development board are as follows:

Rockchip

Layer ID	Name	Operator	Uid	Time(us)
4	conv1_1_3	RESHUFFLE	1	4943
		CONVOLUTION		
0	pool1_3_3	POOLING	3	1649
5	fire2/squeeze1x1_4_2	CONVOLUTION	4	5131
6	fire2/expand1x1_6_2	CONVOLUTION	6	3940
7	fire2/expand3x3_8_2	CONVOLUTION	8	4636
8	fire3/squeeze1x1_11_2	CONVOLUTION	11	3430
9	fire3/expand1x1_13_2	CONVOLUTION	13	2436
10	fire3/expand3x3_15_2	CONVOLUTION	15	3025
1	pool3_18_3	POOLING	18	1737
11	fire4/squeeze1x1_19_2	CONVOLUTION	19	92
12	fire4/expand1x1_21_2	CONVOLUTION	21	377
13	fire4/expand3x3_23_2	CONVOLUTION	23	197
14	fire5/squeeze1x1_26_2	CONVOLUTION	26	399
15	fire5/expand1x1_28_2	CONVOLUTION	28	220
16	fire5/expand3x3_30_2	CONVOLUTION	30	400
2	pool5_33_3	POOLING	33	517
17	fire6/squeeze1x1_34_2	CONVOLUTION	34	196
18	fire6/expand1x1_36_2	CONVOLUTION	36	97
19	fire6/expand3x3_38_2	CONVOLUTION	38	127
20	fire7/squeeze1x1_41_2	CONVOLUTION	41	94
21	fire7/expand1x1_43_2	CONVOLUTION	43	77
22	fire7/expand3x3_45_2	CONVOLUTION	45	125
23	fire8/squeeze1x1_48_2	CONVOLUTION	48	93
24	fire8/expand1x1_50_2	CONVOLUTION	50	95
25	fire8/expand3x3_52_2	CONVOLUTION	52	444
26	fire9/squeeze1x1_55_2	CONVOLUTION	55	212
27	fire9/expand1x1_57_2	CONVOLUTION	57	99
28	fire9/expand3x3_59_2	CONVOLUTION	59	172
29	conv10_63_2	CONVOLUTION	63	575
30	pool10_65_2	DEPTH_WISE_CONV	65	508
3	Softmax2Layer_1	SOFTMAX		657

Total Time(us): 36700

FPS: 27.25

Operator time-consuming statistics			
OP	Call Number	Total Time(us)	Time Ratio(%)
CONVOLUTION	25	26689	72.72
RESHUFFLE	1	4943	13.47
POOLING	3	3903	10.63
SOFTMAX	1	657	1.79
DEPTH_WISE_CONV	1	508	1.38

## 5.4 Common performance optimization methods

1. If the model initialization cost too much time, please refer to chapter 3.4.4 for model pre-compilation.
2. If RKNN C API used for deployment, and setting input or obtaining inference results takes too much time, please refer to the document *Rockchip\_User\_Guide\_RKNN\_API\_EN.pdf* for optimization.
3. Refer to the *Rockchip\_Trouble\_Shooting\_RKNN\_Toolkit\_EN.pdf* document for the optimizing advices of convolutional neural networks. Try to optimize the network structure and improve the performance of the model on Rockchip NPU.

## 6 Memory evaluation

This section explains in detail how to use the RKNN Toolkit interface to obtain the memory usage of the RKNN model at runtime.

### 6.1 Evaluation method

RKNN Toolkit provides the RKNN model memory evaluation interface `eval_memory`. This interface can evaluate the memory usage of the RKNN model during inference on Rockchip NPU.

Note: Cannot use simulator for memory evaluation.

### 6.2 Evaluation example

Please refer to the following code to use the RKNN Toolkit interface to complete the memory evaluation.

```
from rknn.api import RKNN

RKNN_PATH = './resnet18.rknn'

def eval_mem_with_rk1808():
    # Create RKNN object
    rknn = RKNN()

    # load RKNN model resnet_18 from current directory
    ret = rknn.load_rknn(path=RKNN_PATH)
    if ret != 0:
        print('Load Pytorch model failed!')
        exit(ret)

    # init RKNN runtime
    ## Set eval_mem to True to enter the memory evaluation mode
    ret = rknn.init_runtime(target='rk1808', device_id='1808', eval_mem=True)
    if ret != 0:
        print('Init runtime environment failed')
        exit(ret)

    # Call the eval_memory interface to count the memory usage of the model when it is
    inferencing
    rknn.eval_memory()

    # Release RKNN object
    rknn.release()

if __name__ == '__main__':

    # Do memory evaluation on RK1808 EVB board.
    eval_mem_with_rk1808()
```

## 6.3 Description of evaluation results

The memory evaluation interface returns the memory usage of the RKNN model inference, encapsulated in the dictionary `memory_detail`. The `memory_detail` consists of the following:

```
{
  'system_memory', {
    'maximum_allocation': 128000000,
    'total_allocation': 152000000
  },
  'npu_memory', {
    'maximum_allocation': 30000000,
    'total_allocation': 40000000
  },
  'total_memory', {
    'maximum_allocation': 158000000,
    'total_allocation': 192000000
  }
}
```

- system\_memory: System memory allocated by non-NPU drivers, including memory allocated in the system for models and input data.
- npu\_memory: Indicates the memory allocated by the NPU driver during model inference.
- total\_memory: the sum of system\_memory and npu\_memory.
- maximum\_allocation: the peak memory usage, the unit is Byte. Indicates the maximum allocation value of memory from the beginning to the end of the model inference.
- total\_allocation: represents the sum of all memory allocated during the operation of the RKNN model.



## 7 RKNN-Toolkit API description

This chapter explains in detail how to use each interface of RKNN Toolkit.

### 7.1 RKNN object initialization and release

Before using any API interface of RKNN Toolkit, RKNN object must be determine by call RKNN().

After all the task finished, call release() to release the RKNN object.

When the RKNN object is initing, the users can set **verbose** and **verbose\_file** parameters, which allow showing detailed log information of model loading, building and so on. The data type of verbose parameter is bool. If the value sets True, the RKNN-Toolkit will show detailed log information on screen. The data type of verbose\_file is string. If the value sets with a file path, the detailed log information will be written to this file (**verbose** also need set True). If running error occurs and the **verbose\_file** is not set, the error log will be automatically written to the log\_feedback\_to\_the\_rknn\_toolkit\_dev\_team.log file.

When feedback error information to Rockchip NPU team, it is recommended to feedback the complete log.

The sample code is as follows:

```
# Show the detailed log information on screen, and saved to
# mobilenet_build.log
rknn = RKNN(verbose=True, verbose_file='./mobilenet_build.log')
# Only show the detailed log information on screen.
rknn = RKNN(verbose=True)
...
rknn.release()
```

## 7.2 RKNN model configuration

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

Rockchip

API	<b>config</b>
Description	Set model parameters
Parameter	<p><b>batch_size</b>: The size of each batch of data sets. The default value is 100. When quantifying, the amount of data fed in each batch will be determined according to this parameter to correct the quantization results. If the amount of data in the dataset is less than batch_size, this parameter will automatically adjust the amount of data in the dataset. If there is insufficient memory during quantization, it is recommended to set this value to a smaller value, such as 8.</p>
	<p><b>mean_values</b>: The mean values of the input. This parameter and the channel_mean_value parameter can not be set at the same time. 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. If reorder_channel is set to "2 1 0", the channel adjustment will be done first, and then the average value will be subtracted.</p>
	<p><b>std_values</b>: The normalized value of the input. This parameter and the channel_mean_value parameter can not be set at the same time. 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. If reorder_channel is set to "2 1 0", the channel adjustment will be performed first, followed by subtracting the mean value and dividing by the normalized value.</p>
	<p><b>epochs</b>: Number of iterations in quantization. Quantization parameter calibration is performed with specified data at each iteration. Default value is -1, in this situation, the number of iteration is automatically calculated based on the amount of data in the dataset.</p>

	<p><b>reorder_channel:</b> A permutation of the dimensions of input image (<b>for three-channel input only, other channel formats can be ignored</b>). The new tensor dimension i will correspond to the original input dimension reorder_channel[i]. For example, if the original image is RGB format, '2 1 0' indicates that it will be converted to BGR.</p> <p>If there are multiple inputs, the corresponding parameters for each input is split with '#', such as '0 1 2#0 1 2'. Default value of this parameter is None, for Caffe model which input is 3 channel, it means input channel will be reversed, for other framework, the input will not be reversed.</p> <p><b>Note:</b> each value of reorder_channel must not be set to the same value.</p>
	<p><b>need_horizontal_merge:</b> Indicates whether to merge horizontal, the default value is False.</p> <p>If the model is inception v1/v3/v4, it is recommended to enable this option, it can improve the performance of inference.</p>
	<p><b>quantized_dtype:</b> Quantization type, the quantization types currently supported are asymmetric_quantized-u8, dynamic_fixed_point-i8, dynamic_fixed_point-i16. The default value is asymmetric_quantized-u8.</p>
	<p><b>quantized_algorithm:</b> Quantization parameter optimization algorithm. Currently supported algorithms are "normal", "mmse" and "kl_divergence", and the default value is "normal". Among them, the normal algorithm is characterized by faster speed. The mmse algorithm, because of the need to adjust the quantization parameters many times, its speed will be much slower, but it can usually get higher accuracy than the normal algorithm. The "kl_divergence" algorithm will take more time than normal, but will be much less than mmse. In some scenarios, better improvement effects can be obtained by "kl_divergence".</p>
	<p><b>mmse_epoch:</b> The number of iterations of the mmse quantization algorithm, the default value is 3. Generally, the more iterations, the higher the accuracy.</p>
	<p><b>optimization_level:</b> Model optimization level. By modifying the model optimization level, you can turn off some or all of the optimization rules used in the model conversion process.</p>

	<p>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.</p> <p><b>target_platform:</b> Specify which target chip platform the RKNN model is based on. RK1806, RK1808, RK3399Pro, RV1109 and RV1126 are currently supported. The RKNN model generated based on RK1806, RK1808 or RK3399pro can be used on both platforms, and the RKNN model generated based on RV1109 or RV1126 can be used on both platforms. If the model is to be run on RK1806, RK1808 or RK3399Pro, the value of this parameter can be ["rk1806"], ["rk1808"], ["rk3399pro"] or ["rk1806", "rk1808", "rk3399pro"], etc. If the model is to be run on RV1109 or RV1126, the value of this parameter can be ["rv1126"], ["rv1109"] or ["rv1109", "rv1126"], etc. But you cannot fill in ["rk1808", "rv1109"], because these two chips are incompatible, the RKNN model generated based on them cannot be run on another chip platform. If this parameter is not set, the default is ["rk1808"], and the generated RKNN model can be run on RK1806, RK1808 and RK3399Pro platforms.</p> <p><b>quantize_input_node:</b> If sets True, regardless of whether the entire model is quantized in the build phase, the input-node will be forced to be quantized. Generally, model with quantized input-node consumes less time while using rknn_input_set interface on NPU devices. Consider to set True when RKNN-Toolkit does not quantize the input-nodes of model as it supposed to be done(in this case, only support model with image as input), or converting a quantized model, generated by others DL framework(in this case, if first layer is quantize_layer, it will be merged into the input node). The default value is False.</p> <p><b>merge_dequant_layer_and_output_node:</b> Merge the model output-node and the last layer(need to be dequantize-layer) into a quantized output-node, allowing the model to return uint8 or float type inference results on NPU devices. This configuration is only effective when loading quantized model generated by other DL framework. The default is False.</p>
--	--

	<p><b>model_pruning:</b> Enable structured pruning when set True. By eliminating invalid channel dimensions, the calculation and parameter of are reduced. Currently valid on onnx/pytorch/tensorflow models, supporting floating-point and quantized models.</p> <p><b>Experimental feature.</b> The default is False.</p>
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]],
            reorder_channel='0 1 2',
            need_horizontal_merge=True,
            target_platform=['rk1808', 'rk3399pro'])
```

### 7.3 Loading non-RKNN model

RKNN-Toolkit currently supports Caffe, Darknet, Keras, MXNet, ONNX, PyTorch, TensorFlow, TensorFlow Lite. There are different interfaces when loading models, the loading interfaces for these frameworks are described in detail below.

### 7.3.1 Loading Caffe model

API	<b>load_caffe</b>
Description	Load Caffe model
Parameter	<b>model:</b> The path of Caffe model structure file (suffixed with “.prototxt” ).
	<b>proto:</b> Caffe model format (valid value is ‘caffe’ or ‘lstm_caffe’). Please use ‘lstm_caffe’ when the model is RNN model.
	<b>blobs:</b> The path of Caffe model binary data file (suffixed with “.caffemodel”). The value can be None, RKNN-Toolkit will randomly generate parameters such as weights.
Return	0: Import successfully
Value	-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',
                      proto='caffe',
                      blobs='./mobilenet_v2.caffemodel')
```

### 7.3.2 Loading Darknet model

API	<b>load_darknet</b>
Description	Load Darknet model
Parameter	<b>model:</b> The path of Darknet model structure file (suffixed with “.cfg”).
	<b>weight:</b> The path of weight file (suffixed with “.weight”).
Return	0: Import successfully
Value	-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')
```

### 7.3.3 Loading Keras model

API	<b>load_keras</b>
Description	Load Keras model
Parameter	<b>model:</b> The path of Keras model file (suffixed with “.h5”)
Return	0: Import successfully
Value	-1: Import failed

The sample code is as follows:

```
# Load the keras xception model in the current path
ret = rknn.load_keras(model='./xception_v3.h5')
```

### 7.3.4 Loading MXNet model

API	<b>load_mxnet</b>
Description	Load MXNet model
Parameter	<b>symbol:</b> Network structure file of MXNet model, suffixed with “json”. Required.
	<b>params:</b> Network parameters file of MXNet model, suffixed with “params”. Required.
	<b>input_size_list:</b> The size and number of channels of each input node. For example, [[1,224,224],[3,224,224]] means there are two inputs. One of the input shapes is [1, 224, 224], and the other input shape is [3, 224, 224]. Required.
Return	0: Import successfully
Value	-1: Import failed

The sample code is as follows:

```
# Load the mxnet model resnext50 in the current path
ret = rknn.load_mxnet(symbol='resnext50_32x4d-symbol.json',
                      params='resnext50_32x4d-4ecf62e2.params',
                      input_size_list=[[3,224,224]])
```



### 7.3.5 Loading ONNX model

API	<b>load_onnx</b>
Description	Load ONNX model
Parameter	<p><b>model:</b> The path of ONNX model file (suffixed with “.onnx”)</p> <p><b>inputs:</b> Specify model input nodes. The data type is list. For example, in the resnet50v2 model in the demo, the input node is ['data']. Default value of this parameter is None, the toolkit will query input nodes from model. Optional.</p> <p><b>input_size_list:</b> Specify the shapes of model input tensors. For example, in the resnet50v2 model in the demo, the input tensor shape is [[3, 224, 224]]. Optional.</p> <p><b>Note:</b></p> <ol style="list-style-type: none"> <li><b>Do not fill in the batch dimension when filling in the input data shape. If you want to batch inference, please use the rknn_batch_size parameter of the build interface.</b></li> <li><b>If the inputs node is specified, this parameter must be filled in.</b></li> </ol> <p><b>outputs:</b> Specify model output nodes. The data type is list. For example, in the resnet50v2 model in the demo, the output node is ['resnetv24_dense0_fwd']. Default value of this parameter is None, the toolkit will query output nodes from model. Optional.</p>
Return	0: Import successfully
Value	-1: Import failed

Note:

Since RKNN Toolkit version 1.7.0, quantized ONNX model is supported to load. The currently supported quantization model is obtained through the static quantization method of onnxruntime (version 1.5.0 to 1.5.2). For related quantification methods, please refer to the link below:

<https://onnxruntime.ai/docs/how-to/quantization.html>

The sample code is as follows:

```
# Load the resnet50v2 onnx model in the current path
ret = rknn.load_onnx(model = './resnet50v2.onnx',
                    inputs = ['data'],
                    input_size_list = [[3, 224, 224]],
                    outputs=['resnetv24_dense0_fwd'])
```

### 7.3.6 Loading PyTorch model

API	<b>load_pytorch</b>
Description	Load PyTorch model
Parameter	<b>model:</b> The path of PyTorch model structure file (suffixed with “.pt”), and need a model in the torchscript format. Required.
	<b>input_size_list:</b> The size and number of channels of each input node. For example, [[1,224,224],[3,224,224]] means there are two inputs. One of the input shapes is [1, 224, 224], and the other input shape is [3, 224, 224]. Required.
Return	0: Import successfully
Value	-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=[[3,224,224]])
```

### 7.3.7 Loading TensorFlow model

API	<b>load_tensorflow</b>
Description	Load TensorFlow model
Parameter	<b>tf_pb</b> : The path of TensorFlow model file (suffixed with “.pb”).
	<b>inputs</b> : The input node of model, input with multiple nodes is supported now. All the input node string are placed in a list.
	<b>input_size_list</b> : The size and number of channels of the image corresponding to the input node. As in the example of mobilenet_v1 model, the input_size_list parameter should be set to [224,224,3].
	<b>outputs</b> : The output node of model, output with multiple nodes is supported now. All the output nodes are placed in a list.
	<b>predef_file</b> : In order to support some controlling logic, a predefined file in npz format needs to be provided. This predefined file can be generated by the following function call:  np.savez('prd.npz', [placeholder name]=prd_value)。 If there are / in placeholder name, use # to replace.
Return	0: Import successfully
value	-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=['FeatureExtractor/MobilenetV1/MobilenetV1/Conv2d_0
           /BatchNorm/batchnorm/mul_1'],
    outputs=['concat', 'concat_1'],
    input_size_list=[[300, 300, 3]])
```

### 7.3.8 Loading TensorFlow Lite model

API	<b>load_tflite</b>
Description	<p>Load TensorFlow Lite model.</p> <p>Note:</p> <p>RKNN-Toolkit uses the tflite schema commits as in link:</p> <p><a href="https://github.com/tensorflow/tensorflow/commits/master/tensorflow/lite/schema/schema.ubs">https://github.com/tensorflow/tensorflow/commits/master/tensorflow/lite/schema/schema.ubs</a></p> <p>commit hash:</p> <p>0c4f5dfea4ceb3d7c0b46fc04828420a344f7598</p> <p>Because the tflite schema may not compatible with each other, tflite models in older or newer schema may not be imported successfully.</p>
Parameter	<b>model:</b> The path of TensorFlow Lite model file (suffixed with “.tflite”).
Return	0: Import successfully
Value	-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')
```

## 7.4 Building RKNN model

API	<b>build</b>
Description	Build corresponding RKNN model according to the loaded model structure and weight data.
Parameter	<p><b>do_quantization:</b> Whether to quantize the model, optional values are True and False.</p> <p><b>dataset:</b> A input data set 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:</p> <p>a.jpg b.jpg or a.npy b.npy</p> <p>If there are multiple inputs, the corresponding files are divided by space. Such as:</p> <p>a.jpg a2.jpg b.jpg b2.jpg or a.npy a2.npy b.npy b2.npy</p> <p><del><b>pre_compile:</b> This parameter is deprecated and will be removed in next version.</del></p> <p><b>rknn_batch_size:</b> batch size of input, default is 1. 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].</p> <p><b>Note:</b></p> <p><b>1. The adjustment of rknn_batch_size does not improve the performance of the general model on the NPU, but it will significantly increase memory consumption</b></p>

	<p>and increase the delay of single frame.</p> <p>2. The adjustment of rknn_batch_size can reduce the consumption of the ultra-small model on the CPU and improve the average frame rate of the ultra-small model. (Applicable to the model is too small, CPU overhead is greater than the NPU overhead)</p> <p>3. The value of rknn_batch_size is recommended to be less than 32, to avoid the memory usage is too large and the reasoning fails.</p> <p>4. 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.</p>
Return	0: Build successfully
value	-1: Build failed

Note: If the RKNN\_DRAW\_DATA\_DISTRIBUTE environment variable sets 1 before executing the script, the RKNN Toolkit will save the histogram of each layer's weight, bias (if available) and output data in the dump\_data\_distribute folder in the current directory during quantization. When exporting the histogram of data, it is recommended to put only one set of input data in the calibration dataset.

The sample code is as follows:

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

## 7.5 Export RKNN model

The RKNN model built by this tool can be exported and stored as an RKNN model file through this interface for model deployment.

API	<b>export_rknn</b>
Description	Save RKNN model in the specified file (suffixed with “.rknn”).
Parameter	<b>export_path</b> : The path of generated RKNN model file.
Return	0: Export successfully
Value	-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')
```

## 7.6 Loading RKNN model

API	<b>load_rknn</b>
Description	Load RKNN model
Parameter	<b>path</b> : The path of RKNN model file. <b>load_model_in_npu</b> : Whether to load RKNN model in NPU directly. The path parameter should fill in the path of the model in NPU. Default value is False.
Return	0: Load successfully
Value	-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')
```

## 7.7 Initialize the runtime environment

Before model inference or performance evaluation, the runtime environment must be initialized, and the operating platform of the model (specific target hardware platform or software simulator) must be clarified.

Rockchip



API	<b>init_runtime</b>
Description	Initialize the runtime environment. Set the device information. Determine whether to enable debug mode to obtain more detailed performance information for performance evaluation.
Parameter	<b>target</b> : Target hardware platform, now supports “rk3399pro”, “rk1806”, “rk1808”, “rv1109”, “rv1126”. The default value is “None”, which indicates model runs on default hardware platform and system. Specifically, if RKNN-Toolkit is used in RK3399Pro Linux development board, the default device is RK3399Pro. The “rk1808” includes TB-RK1808 AI Compute Stick. Since from version 1.7.5, the emulator is not supported, please fill target with correct NPU device.
	<b>device_id</b> : Device identity number, if multiple devices are connected to PC, this parameter needs to be specified which can be obtained by calling “ <i>list_devices</i> ” interface. The default value is “None”.
	<b>perf_debug</b> : 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.
	<b>eval_mem</b> : 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.
	<b>async_mode</b> : Whether to use asynchronous mode. 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. The default value for this parameter is False.
Return	0: Initialize the runtime environment successfully
Value	-1: Initialize the runtime environment failed

The sample code is as follows:

```
# Initialize the runtime environment
ret = rknn.init_runtime(target='rk1808', device_id='012345789AB')
if ret != 0:
    print('Init runtime environment failed')
    exit(ret)
```

Rockchip

## 7.8 Inference with RKNN model

Before model inference, an RKNN model must be built or loaded first.

API	<b>inference</b>
Description	Use the model to perform inference with specified input and get the inference result.  Note: Since version 1.7.5, inference with simulator is not supported.
Parameter	<b>inputs</b> : Inputs to be inferred, such as images processed by cv2. The object type is ndarray list.
	<b>data_type</b> : The numerical type of input data. Optional values are 'float32', 'float16', 'int8', 'uint8', 'int16'. The default value is 'uint8'.
	<b>data_format</b> : The shape format of input data. Optional values are "nchw", "nhwc". The default value is 'nhwc'.
	<b>inputs_pass_through</b> : Pass the input transparently to the NPU driver. In non-transparent mode, the tool will reduce the mean, divide the variance, etc. before the input is passed to the NPU driver; in transparent mode, these operations will not be performed. The value of this parameter is an array. For example, to pass input0 and not input1, the value of this parameter is [1, 0]. The default value is None, which means that all input is not transparent.
Return Value	<b>results</b> : The result of inference, the object type is ndarray list.

The sample code is as follows (refer to *example/tfite/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]: 0.85107421875
[155]: 0.09173583984375
[205]: 0.01358795166015625
[284]: 0.006465911865234375
[194]: 0.002239227294921875
```

For more examples, please refer to other demos in SDK/examples/.

## 7.9 Evaluate model performance

API	<b>eval_perf</b>
Description	<p>Evaluate model performance.</p> <p>Detailed scenarios are as follows:</p> <p>Note: Since version 1.7.5, evaluate performance with simulator is not supported.</p>
Parameter	<p><b>loop_cnt</b>: Specify the number of inferences of the RKNN model to calculate the average inference time. This parameter only takes effect when perf_debug of init_runtime is False and evaluating performance on hardware.</p> <p><b>export_perf_statistics</b>: Whether to export the operator performance statistics table, the default is False, no export. This parameter only takes effect when perf_debug is set to True.</p>
Return Value	<b>perf_result</b> : Performance information. For details, please refer to chapter 5.3.

The sample code is as follows:

```
# Evaluate model performance
.....
rknn.eval_perf(inputs=[image], is_print=True)
.....
```

## 7.10 Evaluating memory usage

API	<b>eval_memory</b>
Description	Fetch memory usage when model is running on hardware platform.  Model must run on Rockchip NPU devices.  Note: When users use this API, the driver version must on 0.9.4 or later. Users can get driver version via get_sdk_version interface.
Parameter	<b>is_print</b> : Whether to print performance evaluation results in the canonical format. The default value is True.
Return Value	<b>memory_detail</b> : Memory usage during model inference. For details, please refer to chapter 6.3.

The sample code is as follows:

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

For tflite/mobilenet\_v1 in example directory, the memory usage shows as follows:

```
=====
Memory Profile Info Dump
=====
System memory:
    maximum allocation : 22.65 MiB
    total allocation   : 72.06 MiB
NPU memory:
    maximum allocation : 33.26 MiB
    total allocation   : 34.57 MiB

Total memory:
    maximum allocation : 55.92 MiB
    total allocation   : 106.63 MiB

INFO: When evaluating memory usage, we need consider
the size of model, current model size is: 4.10 MiB
=====
```

## 7.11 Hybrid Quantization

### 7.11.1 hybrid\_quantization\_step1

API	<b>hybrid_quantization_step1</b>
Description	Use the input data in the dataset to calculate the quantization parameters of each node, and export the quantization config file, RKNN model structure file and RKNN model weight file.
Parameter	<b>dataset</b> : A input data set for rectifying quantization parameters. For details, refer to the build interface.
Return	0: success
Value	-1: failure

The sample code is as follows:

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

## 7.11.2 hybrid\_quantization\_step2

API	<b>hybrid_quantization_step2</b>
Description	According to the user configuration, recalculate and adjust the quantization parameters of each node, then export the hybrid quantization RKNN model.
Parameter	<b>model_input</b> : The model structure file generated in the first step, which is shaped like "{model_name}.json". The data type is a string. Required parameter.
	<b>data_input</b> : The model weight file generated in the first step, which is shaped like "{model_name}.data". The data type is a string. Required parameter.
	<b>model_quantization_cfg</b> : The modified model quantization profile, which is shaped like "{model_name}.quantization.cfg". The data type is a string. Required parameter.
	<b>dataset</b> : A input data set for rectifying quantization parameters. For details, refer to the build interface.
	<b>pre_compile</b> : This parameter is deprecated and will be removed in next version.
Return	0: success
Value	-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.json',
    data_input='./ssd_mobilenet_v2.data',
    model_quantization_cfg='./ssd_mobilenet_v2.quantization.cfg',
    dataset='./dataset.txt')
.....
```



## 7.12 Accuracy analysis

This interface compares the inference results of each layer of the floating-point model and the quantized model, and is used to analyze the problem of the accuracy decline after the quantization of the model.

Rockchip

API	<b>accuracy_analysis</b>
Description	<p>This interface will compare the output results of the floating-point model and the quantized model layer by layer, and show the cosine distance and Euclidean distance. It is used to analyze the reasons for the decline in the accuracy of the quantitative model.</p> <p><b>Note:</b></p> <ol style="list-style-type: none"> <li>1. <b>this interface can be called after build or hybrid_quantization_step1 or hybrid_quantization_step2, and the original model should be a non-quantized model, otherwise the call will fail.</b></li> <li>2. <b>The quantization method used by this interface is consistent with the setting in config.</b></li> </ol>
Parameter	<p><b>inputs:</b> The dataset file that include input image or data. (same as “dataset” parameter of build, see section “Building RKNN model”, but only can include one set of input)</p> <p><b>output_dir:</b> The output directory, all snapshot data will stored here. For a detailed description of the contents of this directory, see section 4.3.3.</p> <p><b>calc_qnt_error:</b> Whether to calculate quantitative error. (default is True)</p> <p><b>target:</b> Specify target device. If target is set, in the individual quantization error analysis, the toolkit will connect to the NPU to obtain the real results of each layer. Then compared with the float result. It can more accurately reflect the actual runtime error.</p> <p><b>device_id:</b> Used to specify the ID of the specific device.</p> <p><b>dump_file_type:</b> The accuracy analysis process will dump the results of each layer of the model. This parameter specifies the type of the output file. Valid values are ‘tensor’ and ‘numpy’, the default value is ‘tensor’. If specify the data type as 'numpy', the time-consuming of this interface can be reduced.</p>
Return	0: success
Value	-1: failure

Note: If the RKNN\_DRAW\_DATA\_DISTRIBUTE environment variable is set to 1 before executing

the script, the RKNN Toolkit will save the histogram of each layer's weight, bias (if any) and output data in the dump\_data\_distribute folder in the current directory during quantization. When exporting the histogram of data, it is recommended to put only one set of input data in the calibration dataset.

The sample code is as follows:

```
.....

print('--> config model')
rknn.config(channel_mean_value='0 0 0 1', reorder_channel='0 1 2')
print('done')

print('--> Loading model')
ret = rknn.load_onnx(model='./mobilenetv2-1.0.onnx')
if ret != 0:
    print('Load model failed! Ret = {}'.format(ret))
    exit(ret)
print('done')

# Build model
print('--> Building model')
ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
if ret != 0:
    print('Build rknn failed!')
    exit(ret)
print('done')

print('--> Accuracy analysis')
rknn.accuracy_analysis(inputs='./dataset.txt', target='rk1808')
print('done')

.....
```

## 7.13 Register Custom OP

API	<b>register_op</b>
Description	Register custom op.
Parameter	<b>op_path</b> : rknnop file path of custom op build output
Return	Void
Value	

The sample code is as follows. Note that this interface need be called before model converted. Please refer to the "Rockchip\_Developer\_Guide\_RKNN\_Toolkit\_Custom\_OP\_CN" document for the use and development of custom operators.

```
rknn.register_op('./resize_area/ResizeArea.rknnop')

rknn.load_tensorflow(...)
```

## 7.14 Export a pre-compiled model(online pre-compilation)

RKNN-Toolkit provides an online pre-compilation interface, which can convert ordinary RKNN models into pre-compiled model.

API	<b>export_rknn_precompile_model</b>
Description	Export the pre-compiled model after online compilation. Note: <ol style="list-style-type: none"> <li>Before using this interface, please call the load_rknn interface first;</li> <li>Before using this interface, the init_runtime interface must be called to initialize the model running environment. The target must be an RK NPU device, and the rknn2precompile parameter must be set to True.</li> </ol>
Parameter	<b>export_path</b> : Export model path. Required.
Return	0: success
Value	-1: failure

The sample code is as follows:

```
from rknn.api import RKNN

if __name__ == '__main__':
    # Create RKNN object
    rknn = RKNN()

    # Load rknn model
    ret = rknn.load_rknn('./test.rknn')
    if ret != 0:
        print('Load RKNN model failed.')
        exit(ret)

    # init runtime
    ret = rknn.init_runtime(target='rk1808', rknn2precompile=True)
    if ret != 0:
        print('Init runtime failed.')
        exit(ret)

    # Note: the rknn2precompile must be set True when call init_runtime
    ret = rknn.export_rknn_precompile_model('./test_pre_compile.rknn')
    if ret != 0:
        print('export pre-compile model failed.')
        exit(ret)

    rknn.release()
```

## 7.15 Export a segmentation model

The function of this interface is to convert the ordinary RKNN model into a segment model, and the position of the segment is specified by the user.

API	<b>export_rknn_sync_model</b>
Description	Insert a sync layer after the user-specified layer to segment the model and export the segmented model.
Parameter	input_model: the model which need segment. Data type is string, required.
	sync_uids: uids of the layer which need insert sync layer.  Note:  1. Uid can be obtained through the eval_perf interface, and perf_debug should be set to True when call init_runtime interface. When we want to obtain uids, we need connect a RK1806 or RK1808 or TB-RK1808 AI Compute Stick or RV1109 or RV1126.  2. The value of sync_uids cannot be filled in at will. It must be obtained by eval_perf interface, Otherwise unpredictable consequences may occur.
	output_model: export rknn model path.
Return	0: success
Value	-1: failure

The sample code is as follows:

```
from rknn.api import RKNN

if __name__ == '__main__':
    rknn = RKNN()
    ret = rknn.export_rknn_sync_model(
        input_model='./ssd_inception_v2.rknn',
        sync_uids=[206, 186, 152, 101, 96, 67, 18, 17],
        output_model='./ssd_inception_v2_sync.rknn')
    if ret != 0:
        print('export sync model failed.')
        exit(ret)
    rknn.release()
```

## 7.16 Export encrypted RKNN model

API	<b>export_encrypted_rknn_model</b>
Description	The common RKNN model is encrypted according to the encryption level specified by the user.
Parameter	input_model: The path of the RKNN model to be encrypted. String. Required.
	output_model: Save path of encrypted model. String. Optional, if None, the {original_model_name}.crypt.rknn will be save path of encrypted model.
	crypt_level: Crypt level. 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. Integer. Optional, default value is 1. Currently, support level 1, 2 or 3.
Return	0: Success
Value	-1: Failure.

The sample code is as follows:

```
from rknn.api import RKNN

if __name__ == '__main__':
    rknn = RKNN()
    ret = rknn.export_encrypted_rknn_model('test.rknn')
    if ret != 0:
        print('Encrypt RKNN model failed.')
        exit(ret)
    rknn.release()
```

## 7.17 Export quantization parameters

API	<code>export_rknn_quant_params</code>
Description	<p>Export RKNN model quantization parameters.</p> <p>Note:</p> <p><b>For pre-compiled models, only the quantization parameters of input and output nodes can be exported; for encrypted models, quantization parameters cannot be exported, and calling this interface will return an error.</b></p>
Parameter	<p><b>model</b>: The RKNN model path. Required.</p> <p><b>only_io</b>: Whether to export only input and output nodes. If set to True, only the quantization parameters of the input and output nodes are exported, otherwise the quantization parameter information of each layer is exported. Optional parameter, default value is True.</p>
Return Value	0: export success; -1: export failure.



## 7.18 Get SDK version

API	<b>get_sdk_version</b>
Description	Get API version and driver version of referenced SDK.  Note: Before use this interface, users must load model and initialize runtime first. And this interface can only be used when the target is Rockchip NPU or RKNN-Toolkit running on RK3399Pro Linux development board.
Parameter	None
Return Value	<b>sdk_version:</b> API and driver version. Data type is string.

The sample code is as follows:

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

The SDK version looks like below:

```
=====
RKNN VERSION:
  API: 1.7.1 (566a9b6 build: 2021-10-28 14:53:41)
  DRV: 1.7.1 (566a9b6 build: 2021-11-12 20:24:57)
=====
```

## 7.19 List Devices

API	list_devices
Description	List connected RK3399Pro/RK1808/TB-RK1808S0 AI Compute Stick/RV1109/RV1126.  Note:  There are currently two device connection modes: ADB and NTB. RK1808 and RV1109/RV1126 support both ADB and NTB, RK3399Pro only support ADB, TB-RK1808 AI Compute Stick only support 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.  For example, there are two TB-RK1808 AI Compute Sticks connected to PC, it's return looks like below:  adb_devices = []  ntb_devices = ['TB-RK1808S0', '515e9b401c060c0b']

The sample code is as follows:

```
from rknn.api import RKNN

if __name__ == '__main__':
    rknn = RKNN()
    rknn.list_devices()
    rknn.release()
```

The devices list looks like below:

```
*****
all device(s) with adb mode:
['515e9b401c060c0b', 'XGOR2N4EZR']
*****
```

Note: When using multiple devices, you need to ensure that their connection modes are consistent, otherwise it will cause conflicts and cause device communication to fail.

## 7.20 Query RKNN model runnable platform

API	<b>list_support_target_platform</b>
Description	Query the chip platform that a given RKNN model can run on.
Parameter	<b>rknn_model</b> : RKNN model path. If the model path is not specified, the chip platforms currently supported by the RKNN-Toolkit are printed by category
Return Value	support_target_platform (dict):

The sample code is as follows:

```
rknn.list_support_target_platform(rknn_model='mobilenet_v1.rknn')
```

The runnable chip platforms look like below:

```
*****
Target platforms filled in RKNN model:      []
Target platforms supported by this RKNN model: ['RK1806', 'RK1808', 'RK3399PRO']
*****
```

## 8 Appendix

### 8.1 Reference documents

OP support list:

*RKNN\_OP\_Support.md*

RKNN Toolkit manual:

*Rockchip\_User\_Guide\_RKNN\_Toolkit\_EN.pdf*

Quick start manual:

*Rockchip\_Quick\_Start\_RKNN\_SDK\_EN.pdf*

Trouble shooting manual:

*Rockchip\_Trouble\_Shooting\_RKNN\_Toolkit\_EN.pdf*

Custom OP define manual:

*Rockchip\_Developer\_Guide\_RKNN\_Toolkit\_Custom\_OP\_EN.pdf*

Visualization function manual:

*Rockchip\_User\_Guide\_RKNN\_Toolkit\_Visualization\_EN.pdf*

RKNN Toolkit Lite manual:

*Rockchip\_User\_Guide\_RKNN\_Toolkit\_Lite\_EN.pdf*

The above documents can be found in the SDK/doc directory. You can also visit the following link to view: <https://github.com/rockchip-linux/rknn-toolkit/tree/master/doc>

### 8.2 Issue feedback

All the issue can be feedback via the follow ways:

- QQ group: 1025468710
- Github link: <https://github.com/rockchip-linux/rknn-toolkit/issues>
- Rockchip Redmine: <https://redmine.rock-chips.com/>

**Note: Redmine account can only be registered by an authorized sales. If your development board is from the third-party manufacturer, please find the corresponding manufacturer to give feedback first.**