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

RKNN-Toolkit User Guide

(Technology Department, Graphic Computing Platform Center)

Mark:	Version	V1.6.1
[ ] Editing	Author	Rao Hong
[√] Released	Completed Date	2021-05-21
	Reviewer	Vincent
	Reviewed Date	2021-05-21

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd

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	Optimize the way of performance evaluation	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	1. Add an npy file as a usage specification for the quantized rectified data 2. The instructions of pre-compile parameter in build interface 3. Improve the instructions of reorder_channel parameter in the 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

Version	Modifier	Date	Modify description	Reviewer
V0.9.7	Rao Hong	2018-12-29	1. Interface optimization: delete the instructions of get_run_duration, get_perf_detail_on_hardware 2. Rewrite the instructions of eval_ perf interface 3. Rewrite the instructions of RKNN() interface 4. Add instructions of the init_runtime interface	Vincent
V0.9.7.1	Rao Hong	2019-01-11	1. Solve the bug that the program may hang after multiple calls to inference 2. Interface adjustment: init_runtime does not need to specify host, the tool will automatically determine	Vincent
V0.9.8	Rao Hong	2019-01-30	1. New feature： if set verbose parameter to True when init RKNN object, users can fetch detailed log information.	Vincent
V0.9.9	Rao Hong	2019-03-06	1. New feature: add eval_memory interface to check memory usage when model running. 2. Optimize inference interface; Optimize error message. 3. Add description for API interface: get_sdk_version.	Vincent

Version	Modifier	Date	Modify description	Reviewer
V1.0.0	Rao Hong	2019-05-06	1. Add async mode for init_runtime interface. 2. Add input passthrough mode for inference interface. 3. New feature: hybrid quantization. 4. Optimize initialize time of pre-compiled model. Pre-compiled model generated by RKNN-Toolkit-v1.0.0 can not run on device installed old driver (NPU driver version < 0.9.6), and pre-compiled model generated by old RKNN-Toolkit (version < 1.0.0) can not run on device installed new NPU driver (NPU drvier version == 0.9.6). 5. Adjust the shape of the inference results: Before version 1.0.0, if the output of the original model is arranged in "NHWC" (such as TensorFlow models), the tool will convert the result to "NCHW"; starting from version 1.0.0, this conversion will not be done, but keep consistent with the original model.	Vincent
V1.1.0	Rao Hong	2019-06-28	1. Support TB-RK1808S0 AI Compute Stick. 2. New interface: list_devices, used to query devices connected to PC or RK3399Pro Linux development board. 3. Support run on ARM64 platform with python 3.5. 4. Support run on Windows / Mac OS X.	Vincent

Version	Modifier	Date	Modify description	Reviewer
V1.2.0	Rao Hong	2019-08-21	1. Add support for model with multiple inputs. 2. New feature: batch inference. 3. New feature: model segmentation. 4. New feature: custom op.	Vincent
V1.2.1	Rao Hong	2019-09-26	1. New feature: load_rknn interface supports direct loading of RKNN in NPU. 2. Adjust the default value of batch_size and epochs in config interface. 3. Bug fix.	Vincent
V1.3.0	Rao Hong	2019-12-23	1. Solve the problem of creating RKNN object for too long. 2. New feature: support loading PyTorch model. 3. New feature: support loading mxnet model. 4. New feature: added support for 4-channel input. 5. New feature: error analysis caused by quantization. 6. New feature: visualization. 7. New feature: model optimization level. 8. Optimize hybrid quantization.	Vincent

Version	Modifier	Date	Modify description	Reviewer
V1.3.2	Rao Hong	2020-04-03	1. Support for new chips: RV1109 / RV1126 2. Improve eval_perf function, no longer need to fill in inputs. 3. TensorFlow: Add support for reducemax; improve support for dilated convolution. TFLite: Add support for dilated convolution. Caffe: Add support for CRNN. ONNX: Add support for Gather and Cast; improve support for avg_pool. PyTorch: Add support for upsample_nearest2d, contiguous, softmax, permute, leaky_relu, prelue, log, deconv and sub; improve support for Reshape, Constant. MXNet: Add support for Crop, UpSampling, SoftmaxActivation, _minus_scalar, log. RKNN: Add support for reshape, concat, split. 4. Fix some known bugs.	Vincent

Version	Modifier	Date	Modify description	Reviewer
V1.4.0	Rao Hong	2020-08-13	1. New features: add layer-by-layer quantitative analysis sub-function; Input preprocessing supports multiple std_values; support to export pre-compiled models from the development board. 2. Function optimization: optimize the channel_mean_value parameter and change it to mean_values/std_values; remove mean_values and std_values in the load_tensorflow interface; the visualizaion improves support for multiple inputs, add support for RK1806/RV1109/RV1126; the accuracy analysis function adds non-normalized cosine distance and Euclidean distance. 3. TensorFlow: add support for dense. TFLite: add support for split_v; imporove support for pad. ONNX: add support forprelu, deconvolution, avg_pool / clip. PyTorch: add support for pixel_shuffle, unsqueeze, sum,select, hardtanh, elu, slice, squeeze, exp,relu6, threshold_, matmul, exp, pad; improve support for adaptive_avg_pool2d, upsample_bilinear, relu6. MXNet: improve support for fc. Darknet: add support for mish; improve support for route. 4. Fix some known bugs.	Vincent

Version	Modifier	Date	Modify description	Reviewer
V1.6.0	Rao Hong Chifred	2020-12-31	1. New features: Support Keras and support h5 model exported by TensorFlow 2.0; support PyTorch 1.6.0; support ONNX 1.6.0; add model encryption function; support list devices with command line; offline pre-compile supports multi-inputs model. 2. Function optimization: optimize accuracy analysis function; optimize the pre-process of input, improve model inference performance; optimize performance evaluation function, when printing details, streamline ops of each layer;the model segmentation feature is limited to RK1806 / RK1808 / RV1109 / RV1126 chip range; the operate system of docker image update to Ubuntu 18.04, Python version updates to 3.6. 3. Improve support ops for each deep learning framework. 4. Fix known bugs.	Vincent

Version	Modifier	Date	Modify description	Reviewer
V1.6.1	Rao Hong	2021-05-21	1. New features: Add quantization parameter optimization method MMSE; support exporting the distribution histogram of each layer of weight, bias and output data for analyzing accuracy problems. 2. Function optimization: improve the prompt message when the mean_values, std_values are set improperly. 3. Improve support ops for each deep learning framework. 4. Fix known bugs.	Vincent

Table of Contents

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 to convert Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet、PyTorch、 MXNet or Keras model to RKNN model, support RKNN model import/export, which can be used on Rockchip NPU platform later. Support for multiple input models since version 1.2.0. Support for PyTorch and MXNet since version 1.3.0. Support for Keras and H5 model exported by TensorFlow 2.0 since version 1.6.0.

2） Quantization: support to convert float model to quantization model, currently support quantized methods including asymmetric quantization (asymmetric_quantized-u8) and dynamic fixed point quantization (dynamic_fixed_point-8 and dynamic_fixed_point-16). Starting with version 1.0.0, RKNN-Toolkit began to support hybrid quantization. For a detailed description of hybrid quantization, please refer to Section 3.3. Since version 1.6.1, RKNN-Toolkit provides quantized parameter optimization algorithm MMSE.

3） Model inference: Able to simulate Rockchip NPU to run RKNN model on PC and get the inference result. This tool can also distribute the RKNN model to the specified NPU device to run, and get the inference results.

4） Performance evaluation: Able to simulate Rockchip NPU to run RKNN model on PC, and evaluate model performance (including total time and time-consuming information of each layer). This tool can also distribute the RKNN model to the specified NPU device to run, and evaluate the model performance in the actual device.

5） Memory evaluation: Evaluate system and NPU memory consumption at runtime of the model. When using this function, he RKNN model must be distributed to the NPU device to run, and then call the relevant interface to obtain memory information. This feature is supported since version 0.9.9.

6） Model pre-compilation: with pre-compilation techniques, model loading time can be reduced, and for some models, model size can also be reduced. However, the pre-compiled RKNN model can only be run on a hardware platform with an NPU, and this feature is currently only supported by the x86_64 Ubuntu platform. RKNN-Toolkit supports the model pre-compilation feature from version 0.9.5, and the pre-compilation method has been upgraded in 1.0.0. The upgraded precompiled model is not compatible with the old driver. Since version 1.4.0, ordinary RKNN models can also be converted into precompiled models through NPU device. For details, please refer to the instructions for export_rknn_precompile_model.

7） Model segmentation: This function is used in a scenario where multiple models run simultaneously. A single model can be divided into multiple segments to be executed on the NPU, thereby adjusting the execution time of multiple models occupying the NPU, and avoiding other models because one model occupies too much execution time. RKNN-Toolkit supports this feature since version 1.2.0. Currently, only RK1806/RK1808/RV1109/RV1126 chips support this feature and the NPU driver version is 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 normally. RKNN-Toolkit supports this feature since version 1.2.0. Please refer to the <Rockchip_Developer_Guide_RKNN_-Toolkit_Custom_OP_CN> document for the use and development of custom OP. This function only supported TensorFlow model.

9） Quantitative error analysis: This function will give the Euclidean or cosine distance of each layer of inference results before and after the model is quantized. This can be used to analyze how quantitative error occurs, and provide ideas for improving the accuracy of quantitative models. This feature is supported since version 1.3.0. Since version 1.4.0, new feature called individual quantization accuracy analysis provided. The tool assigns the input of each layer at runtime as the correct floating point value, and then calculates the quantized error of the layer. This can avoid misjudgments caused by the accumulation of errors layer by layer, and more accurately reflect the influence of quantization on each layer itself.

10）Visualization: This function presents various functions of RKNN-Toolkit in the form of a 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. Add support for Keras model since version 1.6.0.

11）Model optimization level: 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. This feature is supported since version 1.3.0.

12）Model encryption: Use the specified encryption method to encrypt the RKNN model as a whole. This feature is supported since version 1.6.0.

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:

	Ubuntu 16.04/18.04	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	Partial support (support online pre-compilation)	Partial support (support online pre-compilation)	Partial support (support online pre-compilation)
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(D/X)
- RV1109
- RV1126

Note: RK3566 and RK3568 are not supported yet, and need to use another set of tools.

1.3 Applicable Operating System

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

- Ubuntu: 16.04 (x64) or later
- Windows: 7 (x64) or later
- MacOS: 10.13.5 (x64) or later
- Debian: 9.8 (aarch64) or later

2 Requirements/Dependencies

This software development kit supports running on the Ubuntu, Windows, Mac OS X or Debian operating system. It is recommended to meet the following requirements in the operating system environment:

Table 1 Operating system environment

Operating system version	Ubuntu16.04（x64）or later Windows 7 (x64) or later Mac OS X 10.13.5 (x64) or later Debian 9.8 (x64) or later
Python version	3.5/3.6/3.7
Python library dependencies	'numpy == 1.16.3' 'scipy == 1.3.0' 'Pillow == 5.3.0' 'h5py == 2.8.0' 'lmdb == 0.93' 'networkx == 1.11' 'flatbuffers == 1.10', 'protobuf == 3.11.2' 'onnx == 1.6.0' 'onnx-tf == 1.2.1' 'flask == 1.0.2' 'tensorflow == 1.11.0' or 'tensorflow-gpu' 'dill==0.2.8.2' 'ruamel.yaml == 0.15.81' 'psutils == 5.6.2' 'ply == 3.11' 'requests == 2.22.0' 'torch == 1.2.0' or 'torch == 1.5.1' or 'torch==1.6.0' 'mxnet == 1.5.0' 'sklearn == 0.0' 'opencv-python == 4.0.1.23'

Note:

1. Windows only support Python 3.6 currently.
2. MacOS support python 3.6 and python 3.7.
3. Arm64 platform support python 3.5 and python 3.7.

4. Because PyTorch/TensorFlow, etc. gradually stopped supporting Python3.5, the next major version of RKNN-Toolkit will remove the Python3.5 wheel package on the Linux x86 platform, and instead provide Python3.6 and Python3.7 wheel packages.
5. Scipy version on MacOS should be 1.3.0, other platform is >=1.1.0.
6. The application on ARM64 platform does not need to require sklearn and opencv-python.
7. This document mainly uses Ubuntu 16.04 / Python3.5 as an example. For other operating systems, please refer to the corresponding quick start guide: <Rockchip_Quick_Start_RKNN_Toolkit_V1.6.1_EN.pdf>.

3 User Guide

3.1 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.

Note: Please refer to the following link for the installation process of Toybrick devices:

http://t.rock-chips.com/wiki.php?mod=view&id=36

3.1.1 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 libpython3.5-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
pip3 install torch==1.5.1 torchvision==0.4.0
# Install mxnet
pip3 install mxnet==1.5.0
```

3. Install RKNN-Toolkit

```
pip install package/rknn_toolkit-1.6.1-cp35-cp35m-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 x86_64**: rknn_toolkit-1.6.1-cp35-cp35m-linux_x86_64.whl
- **Python3.5 for arm_x64**: rknn_toolkit-1.6.1-cp35-cp35m-linux_aarch64.whl
- **Python3.6 for x86_64**: rknn_toolkit-1.6.1-cp36-cp36m-linux_x86_64.whl
- **Python3.7 for arm_x64**: rknn_toolkit-1.6.1-cp37-cp37m-linux_aarch64.whl
- **Python3.6 for Windows x86_64:** rknn_toolkit-1.6.1-cp36-cp36m-win_amd64.whl
- **Python3.6 for Mac OS X:** rknn_toolkit-1.6.1-cp36-cp36m-macosx_10_15_x86_64.whl
- **Python3.7 for Mac OS X:** rknn_toolkit-1.6.1-cp37-cp37m-macosx_10_15_x86_64.whl
- **Python3.7 for arm_x64:** rknn_toolkit-1.6.1-cp37-cp37m-linux_aarch64.whl

3.1.2 Install by the Docker Image

In docker folder, there is a Docker image that has been packaged for all development requirements, Users only need to load the image and can directly use RKNN-toolkit, detailed 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-1.6.1-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    1.6.1    0d010618b880    1 hours ago    3.47GB
```

3. Run image

Execute the following command to run the docker image. After running, it will enter the bash environment.

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb rknn-toolkit:1.6.1 /bin/bash
```

Mapping the code 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:1.6.1 /bin/bash
```

4. Run demo

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

Note: The Docker image will base Ubuntu 18.04 and Python 3.6 since RKNN-Toolkit version 1.6.0.

3.2 Usage of RKNN-Toolkit

Currently RKNN-Toolkit can be run on PC (Linux/Windows/MacOS x64), or on RK3399Pro(D/X) development board (Deiban9 or Debian10) or RK1808 computing stick(Debian10).

Next, the use process of RKNN-Toolkit under each use scenario will be given in detail.

Note: for a detailed description of all the interfaces involved in the flow, refer to Section 3.7.

3.2.1 Scenario 1: Inference for Simulation on PC

In this scenario, RKNN-Toolkit runs on the PC, and runs the model through the simulated RK1808/RV1126.

Depending on the type of model, this scenario can be divided into two sub-scenarios: one scenario is that the model is a non-RKNN model, i.e. Caffe, TensorFlow, TensorFlow Lite, ONNX, Darknet, PyTorch, MXNet, Keras model, and the other scenario is that the model is an RKNN model which is a proprietary model of Rockchip with the file suffix “rknn”.

Note: Simulator only supported on x86_64 Linux.

3.2.1.1 Sub-scenario 1: run the non-RKNN model

When running a non-RKNN model, the RKNN-Toolkit usage flow is shown below:

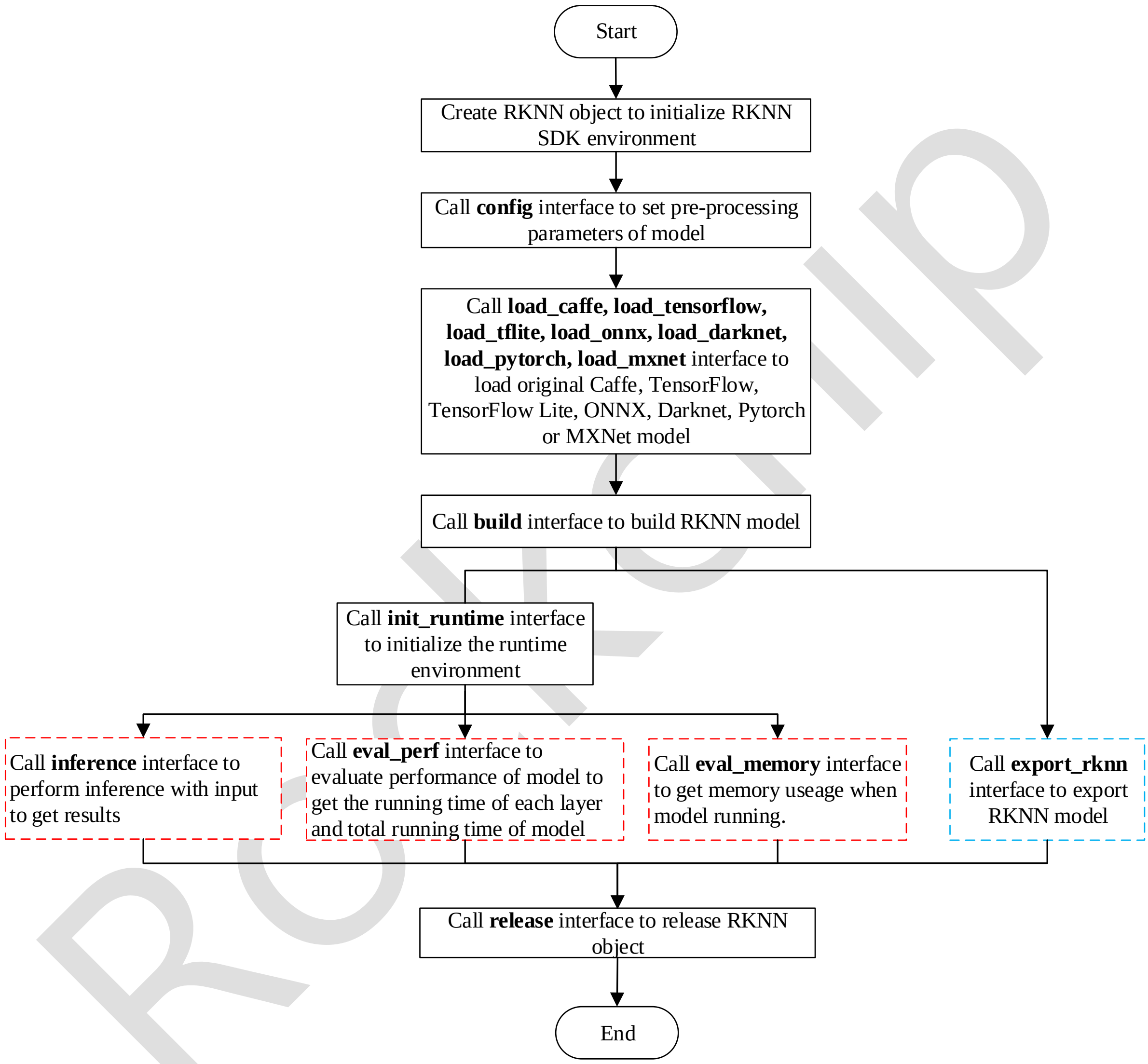

Figure 1 Usage flow of RKNN-Toolkit when running a non-RKNN model on PC

Note:

1. The above steps should be performed in order.

2.The model exporting step marked in the blue box is not necessary. If you exported, you can use load_rknn to load it later on.

3. The order of model inference, performance evaluation and memory evaluation steps marked in red

box is not fixed, it depends on the actual demand.

4. Only when the target hardware platform is Rockchip NPU, the eval_memory interface can be called.

3.2.1.2 Sub-scenario 2: run the RKNN model

When running an RKNN model, users do not need to set model pre-processing parameters, nor do they need to build an RKNN model, the usage flow is shown in the following figure.

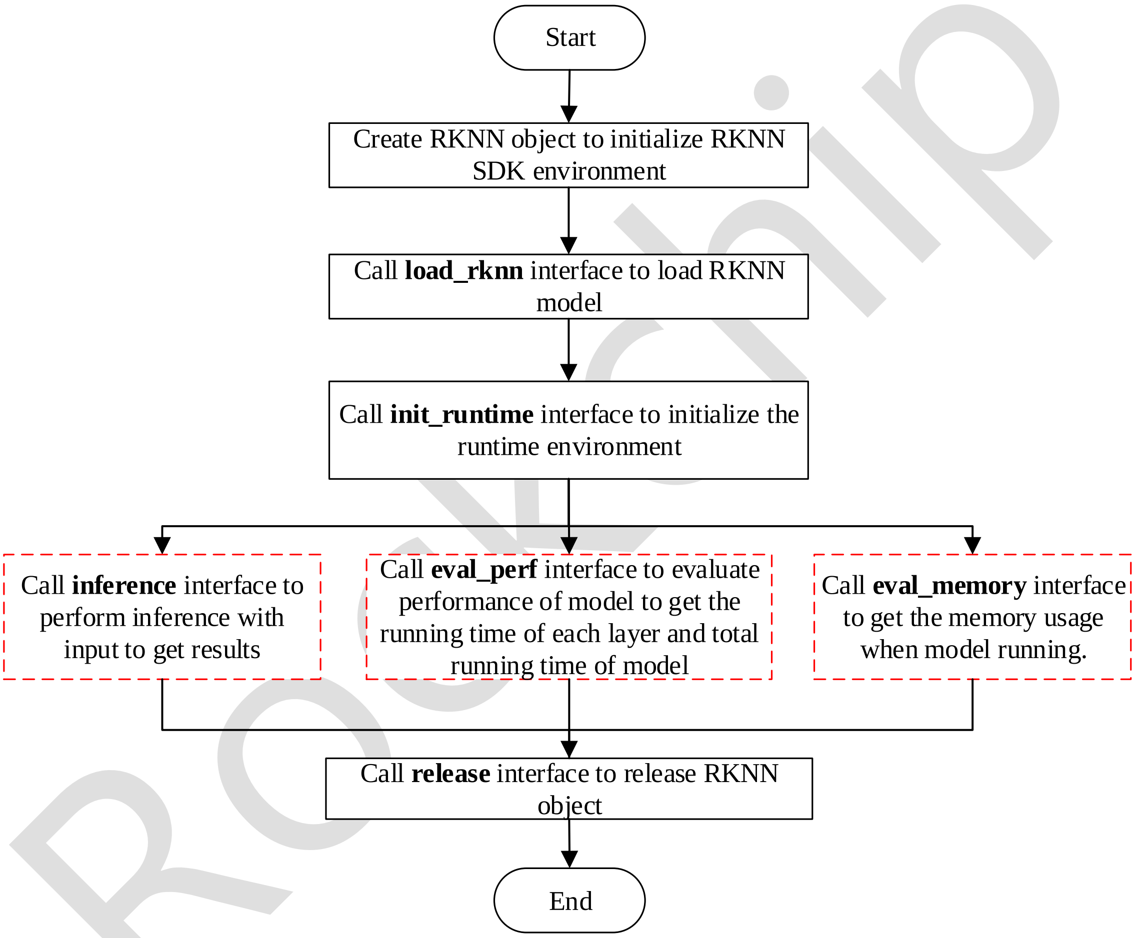

Figure 2 Usage flow of RKNN-Toolkit when running an RKNN model on PC

Note:

1. The above steps should be performed in order.

2. The order of model inference, performance evaluation and memory evaluation steps marked in red box is not fixed, it depends on the actual demand.

3. We can call eval_memory only when the target hardware platform is RK3399Pro, RK1808 or RK3399Pro Linux or TB-RK1808 AI Compute Stick, etc.

3.2.2 Scenario 2: Run on Rockchip NPU connected to the PC.

Rockchip NPU platforms currently supported by RKNN-Toolkit include RK1806, RK1808 (or TB-RK1808), RK3399Pro(D), RV1109 and RV1126.

In this Scenario, RKNN-Toolkit runs on the PC and connects to the NPU device through the PC's USB. RKNN-Toolkit transfers the RKNN model to the NPU device to run, and then obtains the inference results, performance information, etc. from the NPU device

If the model is a non-RKNN model (Caffe, TensorFlow, TensorFlow Lite, ONNX, Darknet, PyTorch, MXNet), the usage flow and precautions of RKNN-Toolkit are the same as the sub-scenario 1 of the scenario 1(see Section 3.2.1.1).

If the model is an RKNN model (file suffix is "rknn"), the usage flow and precautions of RKNN-Toolkit are the same as the sub-scenario 2 of the scenario 1(see Section 3.2.1.2).

In addition, in this scenario, we also need to complete the following two steps:

1. Make sure the USB OTG of development board is connected to PC, and call list_devices interface or using command line "python3 -m rknn.bin.list_devices" will show the devices.

2. "Target" parameter and "device_id" parameter need to be specified when calling "init_runtime" interface to initialize the runtime environment, where "target" indicates the type of hardware, optional values are "rk1808", "rk3399pro", "rv1109" and "rv1126". When multiple devices are connected to PC, "device_id" parameter needs to be specified. It is a string which can be obtained by calling "list_devices" interface or command line "python3 -m rknn.bin.list_devices", for example:

```
all device(s) with adb mode:
[]
all device(s) with ntb mode:
['TB-RK1808S0', '515e9b401c060c0b']
```

Runtime initialization code is as follows:

```
# RK3399Pro
ret = init_runtime(target='rk3399pro', device_id='VGEJY9PW7T')

……
```

```
# RK1808
ret = init_runtime(target='rk1808', device_id='515e9b401c060c0b')

# TB-RK1808 AI Compute Stick
ret = init_runtime(target='rk1808', device_id='TB-RK1808S0')

# RV1109
ret = init_runtime(target='rv1109', device_id=' 60a32d0000bb0709')

# RV1126
ret = init_runtime(target='rv1126', device_id=' c3d9b8674f4b94f6')
```

Note:

1. **Currently, RK1808, RV1109, RV1126 support ADB or NTB. When we use multiple devices on PC or RK3399Pro Linux Development Board, all devices should use same mode, both are ADB or both are NTB.**
2. **When using an NTB device for the first time on Linux, a non-root user needs to obtain the read and write permissions of the USB device. This can be done by executing the SDK/platform-tools/update_rk_usb_rule/linux/update_rk1808_usb_rule.sh script. For details, please refer to the README.txt in the directory.**

3.2.3 Scenario 3: Inference on RK3399Pro Linux development board

In this scenario, RKNN-Toolkit is installed in RK3399Pro Linux system directly. The built or imported RKNN model runs directly on RK3399Pro to obtain the actual inference results or performance information of the model.

For RK3399Pro Linux development board, the usage flow of RKNN-Toolkit depends on the type of model. If the model is a non-RKNN model, the usage flow is the same as that in the sub-scenario 1 of scenario 1(see Section 3.2.1.1), otherwise, please refer to the usage flow in the sub-scenario 2 of scenario1(see Section 3.2.1.2).

3.3 Hybrid Quantization

RKNN-Toolkit supports hybrid quantization from version 1.0.0.

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.

3.3.1 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.

3.3.2 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: {}
```

```yaml
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:
        min_value:
        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. 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.

3.3.3 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:

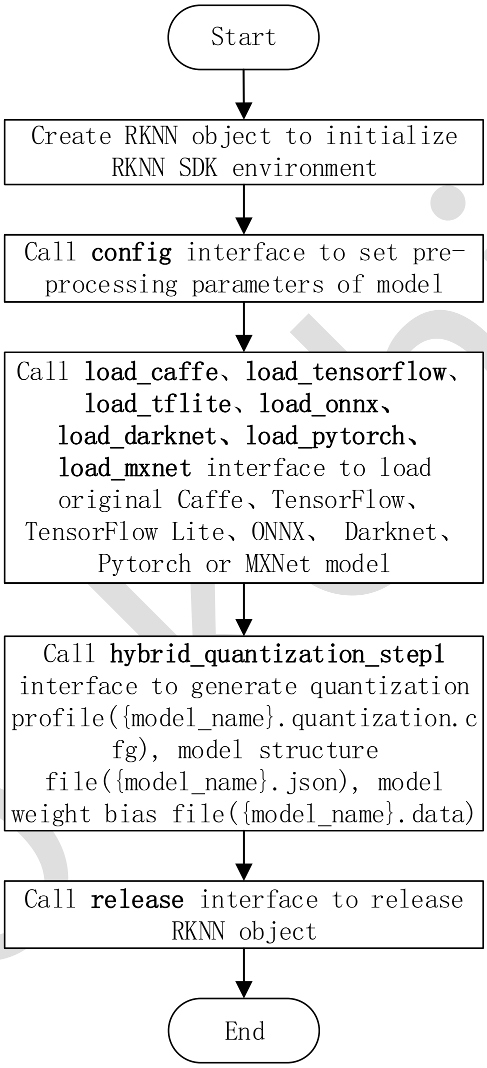

Figure 3-3-3-1 call process of hybrid quantization step 1

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:

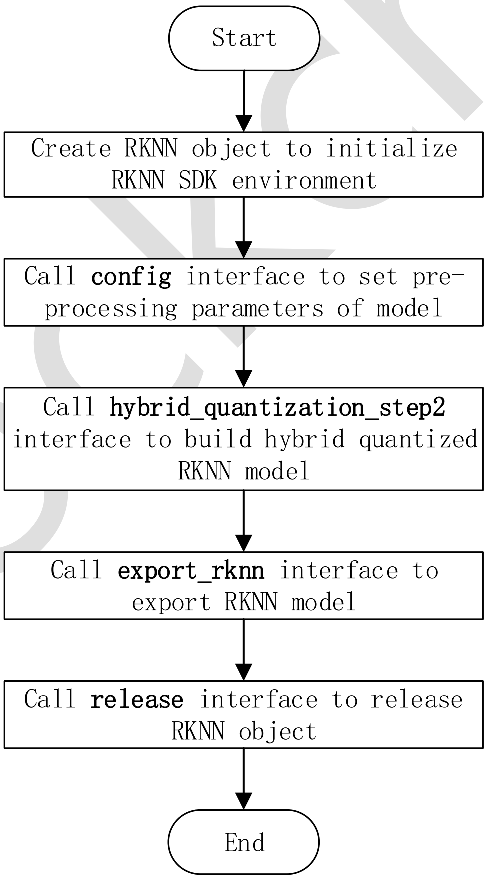

Figure 3-3-3-2 call process of hybrid quantization step 3

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

3.4 Model Segmentation

RKNN-Toolkit supports model segmentation from version 1.2.0. This feature is used in a scenario where multiple models run simultaneously. A single model can be divided into multiple segments to be executed on the NPU, thereby adjusting the execution time of multiple models occupying the NPU, avoiding that one model occupies too much execution time, while other model was not implemented in time.

The chance of each segment preempting the NPU is equal. After a segment execution is completed, it will take the initiative to give up the NPU, if the model has the next segment, it 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, segmentation of other models will be performed in the order of the command queue. Note: The model that does not have model segmentation enabled is by default a segment.

The ordinary RKNN model can be divided into multiple segments by calling the export_rknn_sync_model interface. For the detailed usage of this interface, please refer to section 3.7.15.

If you are in a single model running scenario, you need to turn it off, just do not use a segmentation RKNN model. Because turning on model segmentation reduces the efficiency of single model execution, however, the multi-model running scene does not reduce the efficiency of model execution. Therefore, it is only recommended to use this feature in scenarios where multiple models are running at the same time.

3.5 Accuracy Analysis

3.5.1 Function Description

RKNN-Toolkit introduced the quantitative accuracy analysis function from version 1.3.0 to analyze the difference between the results of each layer of the quantitative model and the floating-point model, and provide improvement ideas for improving model accuracy.

The quantitative accuracy analysis function of version 1.3.0, its working idea is to use the floating-point model on the tool side to make a complete inference, and then the quantized model to make a complete inference. During the inference process, save the results of each layer to the specified directory. Then calculate the Euclidean distance (normalized) and cosine distance (normalized) of each layer to determine the error of each layer.

In the 1.4.0 version of RKNN-Toolkit, two major updates have been made to this function: one is to introduce layer-by-layer error analysis on the tool side; the other is to introduce layer-by-layer error analysis on the device side. The layer-by-layer error analysis on the tool side is based on the original full-model error analysis. The entire model is split into layer by layer, and then each layer takes the result of the previous layer of the floating-point model as input. This can prevent the layer behind the model from having a large deviation in the input itself, resulting in a large difference in the results of this layer. So it is impossible to judge whether it is an error caused by quantization. The previous comparison of the complete model and the layer-by-layer model are all done on the tool side. The second improvement is to use the results of the NPU for comparison, so that you can more intuitively reflect the error of the model when the NPU is running.

Between 1.3.0 and 1.4.0, the error analysis of the complete model did not take into account that the Conv/Relu and other layers were merged together after quantization, resulting in an incorrect comparison when comparing results. Version 1.6.0 has optimized this type of comparison.

3.5.2 Usage

The use process of the quantitative accuracy analysis function is as follows:

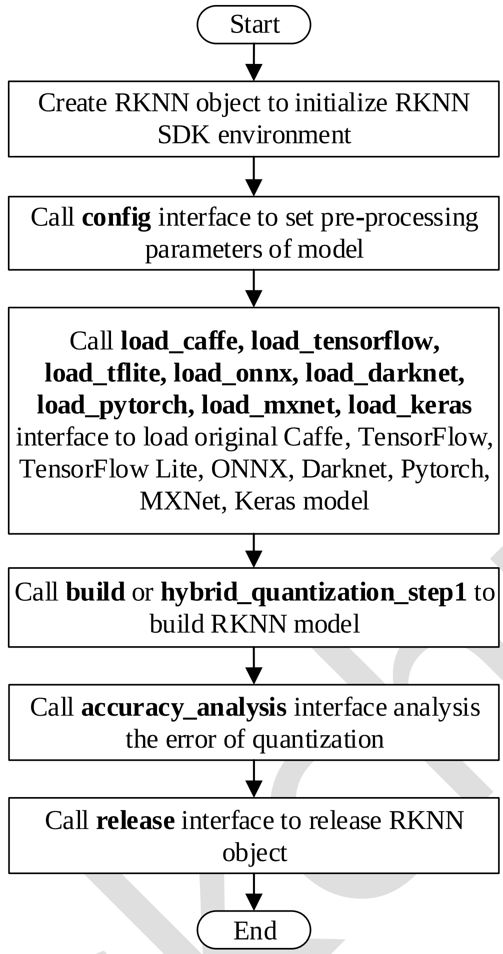

Figure 3-5-2-1 The usage of accuracy_analysis

Note:

1. The quantitative method to be analyzed needs to be specified in config;
2. The dataset.txt in build or hybrid_quantization_step1 can contain multiple pieces of data; but the inputs specified in accuracy_analysis can only contain one piece of data;
3. The calling order of the above interfaces cannot be changed.

3.5.3 Output description

If the target is not set, the output directory structure 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).

3.6 Example

The following is the sample code for loading TensorFlow Lite model (see the *example/tflite/mobilenet_v1* directory for details), if it is executed on PC, the RKNN model will run on the simulator.

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

def show_outputs(outputs):
    output = outputs[0][0]
    output_sorted = sorted(output, reverse=True)
    top5_str = 'mobilenet_v1\n-----TOP 5-----\n'
    for i in range(5):
        value = output_sorted[i]
        index = np.where(output == value)
        for j in range(len(index)):
            if (i + j) >= 5:
                break
            if value > 0:
                topi = '{}: {}\n'.format(index[j], value)
            else:
                topi = '-1: 0.0\n'
            top5_str += topi
    print(top5_str)

def show_perfs(perfs):
    perfs = 'perfs: {}\n'.format(outputs)
    print(perfs)

if __name__ == '__main__':

    # Create RKNN object
    rknn = RKNN()

    # pre-process config
    print('--> config model')
    rknn.config(channel_mean_value='103.94 116.78 123.68 58.82', reorder_channel='0 1 2')
    print('done')
```

```
# Load tensorflow model
print('--> Loading model')
ret = rknn.load_tflite(model='./mobilenet_v1.tflite')
if ret != 0:
    print('Load mobilenet_v1 failed!')
    exit(ret)
print('done')

# Build model
print('--> Building model')
ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
if ret != 0:
    print('Build mobilenet_v1 failed!')
    exit(ret)
print('done')

# Export rknn model
print('--> Export RKNN model')
ret = rknn.export_rknn('./mobilenet_v1.rknn')
if ret != 0:
    print('Export mobilenet_v1.rknn failed!')
    exit(ret)
print('done')

# Set inputs
img = cv2.imread('./dog_224x224.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

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

# Inference
print('--> Running model')
outputs = rknn.inference(inputs=[img])
show_outputs(outputs)
print('done')

# perf
print('--> Begin evaluate model performance')
perf_results = rknn.eval_perf(inputs=[img])
print('done')

rknn.release()
```

Where dataset.txt is a text file containing the path of the test image. For example, if a picture of dog_224x224.jpg in the *example/tflite/mobilenet_v1* directory, then the corresponding content in dataset.txt is as follows:

```
dog_224x224.jpg
```

When performing model inference, the result of this demo is as follows:

```
-----TOP 5-----
[156]: 0.85107421875
[155]: 0.09173583984375
[205]: 0.01358795166015625
[284]: 0.006465911865234375
[194]: 0.002239227294921875
```

When evaluating model performance, the result of this demo is as follows (since it is executed on PC, the result is for reference only).

```
================================================
                    Performance
================================================
Layer ID    Name                                Time(us)
0           tensor.transpose_3                  72
44          convolution.relu.pooling.layer2_2   363
59          convolution.relu.pooling.layer2_2   201
45          convolution.relu.pooling.layer2_2   185
60          convolution.relu.pooling.layer2_2   243
46          convolution.relu.pooling.layer2_2   98
61          convolution.relu.pooling.layer2_2   149
47          convolution.relu.pooling.layer2_2   104
62          convolution.relu.pooling.layer2_2   120
48          convolution.relu.pooling.layer2_2   72
63          convolution.relu.pooling.layer2_2   101
49          convolution.relu.pooling.layer2_2   92
64          convolution.relu.pooling.layer2_2   99
50          convolution.relu.pooling.layer2_2   110
65          convolution.relu.pooling.layer2_2   107
51          convolution.relu.pooling.layer2_2   212
66          convolution.relu.pooling.layer2_2   107
52          convolution.relu.pooling.layer2_2   212
67          convolution.relu.pooling.layer2_2   107
53          convolution.relu.pooling.layer2_2   212
68          convolution.relu.pooling.layer2_2   107
54          convolution.relu.pooling.layer2_2   212
69          convolution.relu.pooling.layer2_2   107
55          convolution.relu.pooling.layer2_2   212
```

```
70                convolution.relu.pooling.layer2_2              107
56                convolution.relu.pooling.layer2_2              174
71                convolution.relu.pooling.layer2_2              220
57                convolution.relu.pooling.layer2_2              353
28                pooling.layer2_1                                36
58                fullyconnected.relu.layer_3                    110
30                softmaxlayer2.layer_1                           90
Total Time(us): 4694
FPS(800MHz): 213.04
==============================================
```

3.7 RKNN-Toolkit API description

3.7.1 RKNN object initialization and release

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

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

The sample code is as follows:

```
# 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()
```

3.7.2 RKNN model configuration

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

API	**config**
Description	Set model parameters
Parameter	batch_size: 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.
	mean_values: 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.
	std_values: 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.
	epochs: 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.
	reorder_channel: A permutation of the dimensions of input image **(for three-channel input only, other channel formats can be ignored)**. 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. If there are multiple inputs, the corresponding parameters for each input is split with ‘#’,

	such as ’0 1 2#0 1 2’. **Note**: each value of reorder_channel must not be set to the same value.
	need_horizontal_merge: Indicates whether to merge horizontal, the default value is False. If the model is inception v1/v3/v4, it is recommended to enable this option, it can improve the performance of inference.
	quantized_dtype: Quantization type, the quantization types currently supported are asymmetric_quantized-u8,dynamic_fixed_point-8,dynamic_fixed_point-16. The default value is asymmetric_quantized-u8.
	quantized_algorithm: Quantization parameter optimization algorithm. Currently supported algorithms are “normal” and “mmse”, 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.
	mmse_epoch: The number of iterations of the mmse quantization algorithm, the default value is 3. Generally, the more iterations, the higher the accuracy.
	optimization_level: 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. The default value of this parameter is 3, and all optimization options are turned on. When the value is 2 or 1, turn off some optimization options that may affect the accuracy of some models. Turn off all optimization options when the value is 0.
	target_platform: Specify which target chip platform the RKNN model is based on. 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.
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'])
```

3.7.3 Loading non-RKNN model

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

3.7.3.1 Loading Caffe model

API	**load_caffe**
Description	Load Caffe model
Parameter	model: The path of Caffe model structure file (suffixed with ".prototxt").
	proto: Caffe model format (valid value is 'caffe' or 'lstm_caffe'). Plaese use 'lstm_caffe' when the model is RNN model.
	blobs: 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 Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

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

3.7.3.2 Loading TensorFlow model

API	**load_tensorflow**
Description	Load TensorFlow model
Parameter	tf_pb: The path of TensorFlow model file (suffixed with ".pb").
	inputs: The input node of model, input with multiple nodes is supported now. All the input node string are placed in a list.
	input_size_list: 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].
	outputs: The output node of model, output with multiple nodes is supported now. All the output nodes are placed in a list.
	predef_file: In order to support some controlling logic, a predefined file in npz format needs to be provided. This predefined fie 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 value	0: Import successfully
	-1: Import failed

The sample code is as follows:

```
# Load ssd_mobilenet_v1_coco_2017_11_17 TF model in the current path
```

```
ret = rknn.load_tensorflow(
        tf_pb='./ssd_mobilenet_v1_coco_2017_11_17.pb',
        inputs=['FeatureExtractor/MobilenetV1/MobilenetV1/Conv2d_0
                /BatchNorm/batchnorm/mul_1'],
        outputs=['concat', 'concat_1'],
        input_size_list=[[300, 300, 3]])
```

3.7.3.3 Loading TensorFlow Lite model

API	**load_tflite**
Description	Load TensorFlow Lite model. Note: RKNN-Toolkit uses the tflite schema commits as in link: https://github.com/tensorflow/tensorflow/commits/master/tensorflow/lite/schema/schema.fbs commit hash: 0c4f5dfea4ceb3d7c0b46fc04828420a344f7598 Because the tflite schema may not compatible with each other, tflite models in older or newer schema may not be imported successfully.
Parameter	model: The path of TensorFlow Lite model file (suffixed with ".tflite").
Return Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

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

3.7.3.4 Loading ONNX model

API	**load_onnx**
Description	Load ONNX model

Parameter	model: The path of ONNX model file (suffixed with ".onnx")
Return Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

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

3.7.3.5 Loading Darknet model

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

The sample code is as follows:

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

3.7.3.6 Loading PyTorch model

API	**load_pytorch**
Description	Load PyTorch model
Parameter	model： The path of PyTorch model structure file (suffixed with ".pt"), and need a model in the torchscript format. Required.
	input_size_list： 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.
	convert_engine: RKNN-Toolkit add this parameter since version 1.6.0. This parameter is used to choose pytorch converter engine. RKNN-Toolkit support two kinds of convert engine: torch1.2 and torch. The “torch1.2” follows the old conversion engine and can only convert PyTorch models between torch 1.1 and 1.2. And the “torch” engine can support PyTorch 1.6.0, and the torch version is required to be higher than 1.5.0, which is also the conversion engine used by RKNN-Toolkit 1.6.0 by default. Optional, default value is “torch”.
Return Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

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

3.7.3.7 Loading MXNet model

API	**load_mxnet**
Description	Load MXNet model
Parameter	symbol：Network structure file of MXNet model, suffixed with “json”. Required.
	params：Network parameters file of MXNet model, suffixed with “params”. Required.
	input_size_list：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 Value	0: Import successfully
	-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]])
```

3.7.3.8 Loading Keras model

API	**load_keras**
Description	Load Keras model
Parameter	model: The path of Keras model file (suffixed with ".h5")
Return Value	0: Import successfully
	-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')
```

3.7.4 Building RKNN model

API	**build**
Description	Build corresponding RKNN model according to the loaded model structure and weight data.
Parameter	do_quantization: Whether to quantize the model, optional values are True and False.
	dataset: 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: a.jpg b.jpg or a.npy b.npy

	If there are multiple inputs, the corresponding files are divided by space. Such as: a.jpg a2.jpg b.jpg b2.jpg or a.npy a2.npy b.npy b2.npy
	pre_compile: If this option is set to True, it may reduce the size of the model file, increase the speed of the first startup of the model on the device. However, if this option is enabled, the built model can be only run on the hardware platform, and the inference or performance evaluation cannot be performed on simulator. If the hardware is updated, the corresponding model need to be rebuilt. **Note:** **1. The pre compile is not supported on RK3399Pro Linux development board or Windows PC or Mac OS X PC.** **2. Pre-compiled model generated by RKNN-Toolkit-v1.0.0 or later can not run on device installed old driver (NPU driver version < 0.9.6), and pre-compiled model generated by old RKNN-Toolkit (version < 1.0.0) can not run on device installed new NPU driver (NPU drvier version >= 0.9.6). The get_sdk_version interface can be called to fetch driver version.** **3. If there are multiple inputs, this option needs to be set to False.**
	rknn_batch_size：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]. **Note：**

	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 and increase the delay of single frame. **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)** **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.** **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.**
Return value	0: Build successfully
	-1: Build failed

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.

The sample code is as follows:

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

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

The sample code is as follows:

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

3.7.6 Loading RKNN model

API	**load_rknn**
Description	Load RKNN model
Parameter	path: The path of RKNN model file.
	load_model_in_npu: Whether to load RKNN model in NPU directly. The path parameter should fill in the path of the model in NPU. It can be set to True only when RKNN-Toolkit run on RK3399Pro Linux or NPU device(RK3399Pro, RK1808 or TB-RK1808 AI Compute Stick) is connected. Default value is False.
Return Value	0: Load successfully
	-1: Load failed

The sample code is as follows:

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

3.7.7 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.

API	**init_runtime**
Description	Initialize the runtime environment. Set the device information (hardware platform, device ID). Determine whether to enable debug mode to obtain more detailed performance information for performance evaluation.
Parameter	target: 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 PC, the default device is simulator, and if RKNN-Toolkit is used in RK3399Pro Linux development board, the default device is RK3399Pro. The “rk1808” includes TB-RK1808 AI Compute Stick.
	device_id: Device identity number, if multiple devices are connected to PC, this parameter needs to be specified which can be obtained by calling “***list_devices***” interface. The default value is “None “. Note: Mac OS X platform does not supple multiple devices.
	perf_debug: Debug mode option for performance evaluation. In debug mode, the running time of each layer can be obtained, otherwise, only the total running time of model can be given. The default value is False.
	eval_mem: Whether enter memory evaluation mode. If set True, the eval_memory interface can be called later to fetch memory usage of model running. The default value is False.
	async_mode: Whether to use asynchronous mode. 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 Value	0: Initialize the runtime environment successfully
	-1: Initialize the runtime environment failed

The sample code is as follows:

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

3.7.8 Inference with RKNN model

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

API	**inference**
Description	Use the model to perform inference with specified input and get the inference result. Detailed scenarios are as follows: 1. If RKNN-Toolkit is running on PC and the target is set to Rockchip NPU when initializing the runtime environment, the inference of model is performed on the specified hardware platform. 2. If RKNN-Toolkit is running on PC and the target is not set when initializing the runtime environment, the inference of model is performed on the simulator. The simulator can simulate RK1808 or RV1126. Which chip to simulate depends on the target_platform parameter value of the RKNN model 3. If RKNN-Toolkit is running on RK3399Pro Linux development board, the inference of model is performed on the actual hardware.
Parameter	inputs: Inputs to be inferred, such as images processed by cv2. The object type is ndarray list.
	data_type: The numerical type of input data. Optional values are ’float32’, ‘float16’, ‘int8’, ‘uint8’, ‘ing16’. The default value is ’uint8’.

	data_format: The shape format of input data. Optional values are “nchw”, “nhwc”. The default value is ‘nhwc’.
	inputs_pass_through: 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	results: The result of inference, the object type is ndarray list。

The sample code is as follows:

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

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

The result of top-5 is as follows:

```
-----TOP 5-----
[156]: 0.85107421875
[155]: 0.09173583984375
[205]: 0.01358795166015625
[284]: 0.006465911865234375
[194]: 0.002239227294921875
```

For object detection model, such as ssd_mobilenet_v1, the code is as follows (refer to *example/tensorflow/ssd_mobilenet_v1* for the complete code):

```
# Perform inference for a picture with a model and get the result of object
# detection
……
outputs = rknn.inference(inputs=[image])
……
```

After the inference result is post-processed, the final output is shown in the following picture (the color of the object border is randomly generated, so the border color obtained will be different each time):

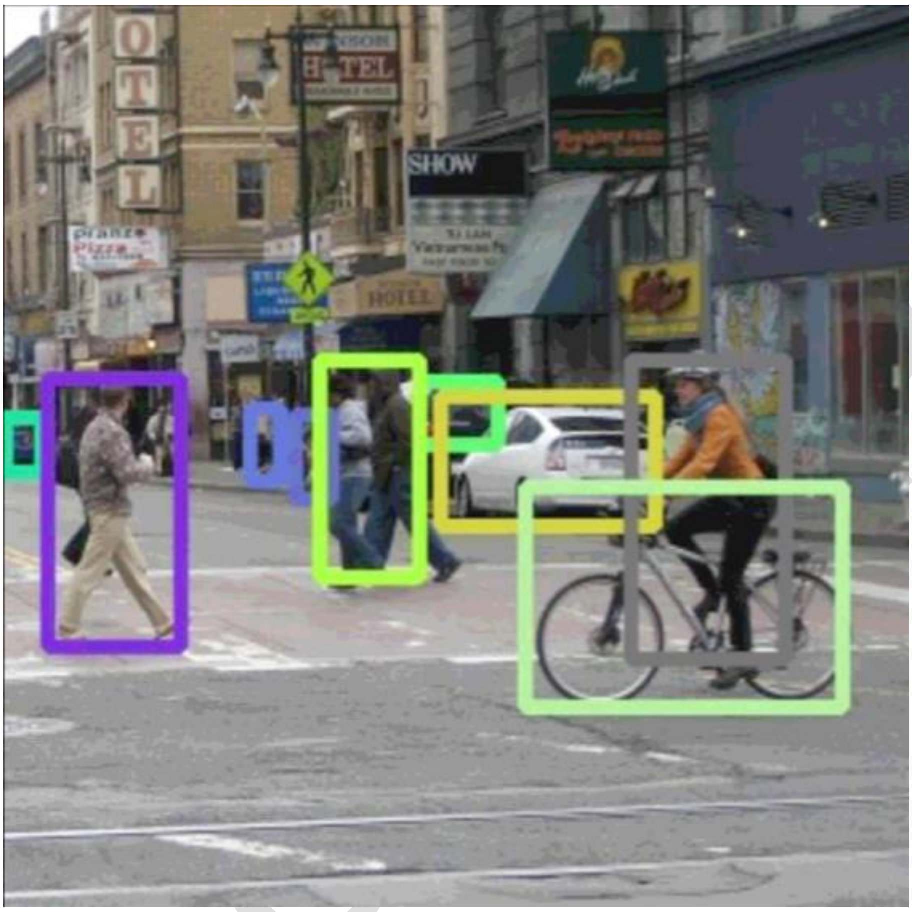

Figure 3 ssd_mobilenet_v1 inference result

3.7.9 Evaluate model performance

API	**eval_perf**
Description	Evaluate model performance. Detailed scenarios are as follows: 1. If running on PC and not setting the target when initializing the runtime environment, the performance information is obtained from simulator, which contains the running time of each layer and the total running time of model. 2. If running on Rockchip NPU device which connected to PC and setting perf_debug to

	False when initializing runtime environment, the performance information is obtained from Rockchip NPU, which only contains the total running time of model. And if the perf_debug is set to True, the running time of each layer will also be captured in detail. 3. If running on RK3399Pro Linux development board and setting perf_debug to False when initializing runtime environment, the performance information is obtained from RK3399Pro, which only contains the total running time of model. And if the perf_debug is set to True, the running time of each layer will also be captured in detail.
Return Value	perf_result: Performance information. The object type is dictionary. If running on device (RK3399Pro or RK1808) and set perf_debug to False when initializing the runtime environment, the dictionary gives only one field ‘total_time’, example is as follows: `{ ‘total_time’: 1000 }` In other scenarios, the obtained dictionary has one more filed called ‘layers’ which is also a dictionary type. The ‘layers’ takes the ID of each layer as the key, and its value is one dictionary which contains 'name' (name of layer), 'operation' (operator, which is only available when running on the hardware platform), 'time'(time-consuming of this layer). Example is as follows: `{ 'total_time', 4568, 'layers', { '0': { 'name': 'convolution.relu.pooling.layer2_2', 'operation': 'CONVOLUTION', 'time', 362 } '1': { 'name': 'convolution.relu.pooling.layer2_2', 'operation': 'CONVOLUTION', 'time', 158 } } }`

The sample code is as follows:

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

For tensorflow/ssd_mobilenet_v1 in example directory, the performance evaluation results are printed as follows(The following is the result obtained on the PC simulator. The details obtained when connecting the hardware device are slightly different from the result.):

```
==============================================
                        Performance
==============================================
Layer ID     Name                                  Time(us)
0            tensor.transpose_3                    125
71           convolution.relu.pooling.layer2_3     324
105          convolution.relu.pooling.layer2_2     331
72           convolution.relu.pooling.layer2_2     438
106          convolution.relu.pooling.layer2_2     436
73           convolution.relu.pooling.layer2_2     223
107          convolution.relu.pooling.layer2_2     374
74           convolution.relu.pooling.layer2_2     327
108          convolution.relu.pooling.layer2_3     533
75           convolution.relu.pooling.layer2_2     167
109          convolution.relu.pooling.layer2_2     250
76           convolution.relu.pooling.layer2_2     293
110          convolution.relu.pooling.layer2_2     249
77           convolution.relu.pooling.layer2_2     164
111          convolution.relu.pooling.layer2_2     256
78           convolution.relu.pooling.layer2_2     319
112          convolution.relu.pooling.layer2_2     256
79           convolution.relu.pooling.layer2_2     319
113          convolution.relu.pooling.layer2_2     256
80           convolution.relu.pooling.layer2_2     319
114          convolution.relu.pooling.layer2_2     256
81           convolution.relu.pooling.layer2_2     319
115          convolution.relu.pooling.layer2_2     256
82           convolution.relu.pooling.layer2_2     319
83           convolution.relu.pooling.layer2_2     173
27           tensor.transpose_3                    48
84           convolution.relu.pooling.layer2_2     45
28           tensor.transpose_3                    6
116          convolution.relu.pooling.layer2_3     299
85           convolution.relu.pooling.layer2_2     233
117          convolution.relu.pooling.layer2_2     314
86           convolution.relu.pooling.layer2_2     479
```

```
87          convolution.relu.pooling.layer2_2          249
35          tensor.transpose_3                         29
88          convolution.relu.pooling.layer2_2          30
36          tensor.transpose_3                         5
89          convolution.relu.pooling.layer2_2          122
90          convolution.relu.pooling.layer2_3          715
91          convolution.relu.pooling.layer2_2          98
41          tensor.transpose_3                         10
92          convolution.relu.pooling.layer2_2          11
42          tensor.transpose_3                         5
93          convolution.relu.pooling.layer2_2          31
94          convolution.relu.pooling.layer2_3          205
95          convolution.relu.pooling.layer2_2          51
47          tensor.transpose_3                         6
96          convolution.relu.pooling.layer2_2          6
48          tensor.transpose_3                         4
97          convolution.relu.pooling.layer2_2          17
98          convolution.relu.pooling.layer2_3          204
99          convolution.relu.pooling.layer2_2          51
53          tensor.transpose_3                         5
100         convolution.relu.pooling.layer2_2          6
54          tensor.transpose_3                         4
101         convolution.relu.pooling.layer2_2          10
102         convolution.relu.pooling.layer2_2          21
103         fullyconnected.relu.layer_3                13
104         fullyconnected.relu.layer_3                8
Total Time(us): 10622
FPS(800MHz): 94.14
=============================================
```

3.7.10 Evaluating memory usage

API	**eval_memory**
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	is_print: Whether to print performance evaluation results in the canonical format. The default value is True.
Return	memory_detail：Detail information of memory usage. Data format is dictionary.

Value	Data shows as below： ``` { 'system_memory', { 'maximum_allocation': 128000000, 'total_allocation': 152000000 }, 'npu_memory', { 'maximum_allocation': 30000000, 'total_allocation': 40000000 }, 'total_memory', { 'maximum_allocation': 158000000, 'total_allocation': 192000000 } } ``` ● The ‘system_memory’ means memory usage of system. ● The ‘npu_memory’ means memory usage inside the NPU. ● The ‘total_memory’ means the sum of system and npu`s memory usage. ● The ‘maximum_allocation’ filed means the maximum memory usage(unit: Byte) from start the model to dump the information. It is the peak memory usage. ● The ‘total_allocation’ means the accumulation memory usage(unit: Byte) of allocate memory from start the model to dump the information.

The sample code is as follows:

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

For tflite/mobilenet_v1 in example directory, the memory usage when model running on RK1808 is printed 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
================================================
```

3.7.11 Hybrid Quantization

3.7.11.1 hybrid_quantization_step1

When using the hybrid quantization function, the main interface called in the first phase is hybrid_quantization_step1, which is used to generate the model structure file ({model_name}.json), the weight file ({model_name}.data), and the quantization configuration file ({model_name}.quantization. Cfg). Interface details are as follows：

API	**hybrid_quantization_step1**
Description	Corresponding model structure files, weight files, and quantization profiles are generated according to the loaded original model.
Parameter	dataset: 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: a.jpg b.jpg or a.npy b.npy
Return Value	0: success
	-1: failure

The sample code is as follows：

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

3.7.11.2 hybrid_quantization_step2

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

API	**hybrid_quantization_step2**
Description	The model structure file, the weight file, the quantization profile, and the correction data set are received as inputs, and the hybrid quantized RKNN model is generated.
Parameter	model_input: The model structure file generated in the first step, which is shaped like "{model_name}.json". The data type is a string. Required parameter.
	data_input: The model weight file generated in the first step, which is shaped like "{model_name}.data". The data type is a string. Required parameter.
	model_quantization_cfg: The modified model quantization profile, whick is shaped like "{model_name}.quantization.cfg". The data type is a string. Required parameter.
	dataset: 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: a.jpg b.jpg or a.npy b.npy
	pre_compile: If this option is set to True, it may reduce the size of the model file, increase

	the speed of the first startup of the model on the device. However, if this option is enabled, the built model can be only run on the hardware platform, and the inference or performance evaluation cannot be performed on simulator. If the hardware is updated, the corresponding model need to be rebuilt. **Note:** **1. The pre compile is not supported on RK3399Pro Linux development board or Windows PC or Mac OS X PC.** **2. Pre-compiled model generated by RKNN-Toolkit-v1.0.0 or later can not run on device installed old driver (NPU driver version < 0.9.6), and pre-compiled model generated by old RKNN-Toolkit (version < 1.0.0) can not run on device installed new NPU driver (NPU drvier version >= 0.9.6). The get_sdk_version interface can be called to fetch driver version.** **3. If there are multiple inputs, this option needs to be set to False.**
Return Value	0: success
	-1: failure

The sample code is as follows：

```
# Call hybrid_quantization_step2 to generate hybrid quantized RKNN model
……
ret = rknn.hybrid_quantization_step2(
          model_input='./ssd_mobilenet_v2.json',
          data_input='./ssd_mobilenet_v2.data',
          model_quantization_cfg='./ssd_mobilenet_v2.quantization.cfg',
          dataset='./dataset.txt')
……
```

3.7.12 Quantitative accuracy analysis

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

API	**accuracy_analysis**

Description	Use floating-point and quantized model inference and take snapshots of the results of each layer. Then, based on the data in the snapshot, compare the gap between each layer of the quantized model and the floating-point model to evaluate the error generated by the quantization. **Note:** 1. **this interface can only be called after build or hybrid_quantization_step1, and the original model should be a non-quantized model, otherwise the call will fail.** 2. **The quantization method used by this interface is consistent with the setting in config.**
Parameter	inputs: dataset file that include input image or data. (same as "dataset" parameter of build, see section "Building RKNN model", but only can include one line in dataset file)
	output_dir: output directory, all snapshot data will stored here. For a detailed description of the contents of this directory, see section 3.5.3.
	calc_qnt_error: whether to calculate quantitative error. (default is True)
	target: 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.
	device_id: If the PC is connected to multiple NPU devices, you need to specify the ID of the specific device.
Return Value	0: success
	-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.

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')

……
```

3.7.13 Register Custom OP

API	**register_op**
Description	Register custom op.
Parameter	op_path: rknnop file path of custom op build output
Return Value	Void

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(…)
```

3.7.14 Export a pre-compiled model(online pre-compilation)

When building an RKNN model, you can specify pre-compilation options(set pre_compile=True) to export the pre-compiled model, which is called offline compilation. Similarly, RKNN-Toolkit also provides an interface for online compilation: export_rknn_precompile_model. Using this interface, you can convert ordinary RKNN models into pre-compiled models.

API	**export_rknn_precompile_model**
Description	Export the pre-compiled model after online compilation. Note: 1. Before using this interface, you must first call the load_rknn interface to load the normal rknn model; 2. 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, not a simulator; and the rknn2precompile parameter must be set to True.
Parameter	export_path: Export model path. Required.
Return Value	0: success
	-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()
```

3.7.15 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	**export_rknn_sync_model**
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. RKNN-Toolkit will insert a 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()
```

3.7.16 Export encrypted RKNN model

The function of this interface is to encrypt the ordinary RKNN model according to a certain algorithm.

Note:

1. The encrypted RKNN model is used in the same way as the ordinary RKNN model.
2. If the encrypted RKNN model is deployed with the Python interface, the target cannot be a simulator; if the C API is used for deployment, only the RKNN model needs to be replaced, and other codes do not need to be modified.

API	**export_encrypted_rknn_model**
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 Value	0: Success -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()
```

3.7.17 Get SDK version

API	**get_sdk_version**
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	sdk_version：API and driver version. Data type is string.

The sample code is as follows：

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

The SDK version looks like below：

```
==============================================
RKNN VERSION:
    API: 1.6.1 (f78b668 build: 2021-05-17 16:33:46)
```

```
        DRV: 1.6.1 (f78b668 build: 2021-04-25 15:46:12)
==============================================
```

3.7.18 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.

3.7.19 Query RKNN model runnable platform

API	**list_support_target_platform**
Description	Query the chip platform that a given RKNN model can run on.
Parameter	rknn_model: 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']
**************************************************
```