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

RK3399Pro RKNN API User Guide

(Technology Department, Graphic Computing Platform Center)

Mark: [ ] Editing [√] Released	Version	V1.7.5
	Author	HPC
	Completed Date	2023-07-12
	Reviewer	Vincent
	Reviewed Date	2023-07-12

瑞芯微电子股份有限公司

Fuzhou Rockchip Electronics Co., Ltd.

Revision History

Version	Modifier	Date	Modify description	Reviewer
V0.9.1	Kevin Du	2018-11-27	Initial release.	Randall
V0.9.2	Kevin Du	2018-12-19	Mainly modify the API definition of input and output.	Randall
V0.9.3	Kevin Du	2019-01-24	Add API migration instruction from v0.9.1 to v0.9.2.	Randall
V0.9.4	Kevin Du	2019-03-11	Fix the issue that channel_mean is not in effect.	Randall
V0.9.6	Kevin Du	2019-05-14	Add rknn_init2 Function.	Randall
V0.9.7	Kevin Du	2019-06-13	Add x86 linux support.	Randall
V0.9.8	Kevin Du	2019-06-26	1. Update Linux X86 Demo section. 2. Add support of rknn_batch_size > 1. 3. Add query function of the devices ID list.	Randall
V0.9.9	Kevin Du	2019-07-16	1. Add support of multi-input 2. Fix inference error when input channel > 3 3. Modify the name of document	Randall
V1.2.0	Kevin Du	2019-09-17	Unified Version to V1.2.0	Randall
V1.3.0	Kevin Du	2019-11-27	1. Add multi-channels-mean support 2. Unified Version to V1.3.0	Randall
V1.3.2	Kevin Du	2020-04-02	1. Fix issue that API version missing version number. 2. update multi-context support. 3. fix the data convert overflow when dst format is int8/uint8/int16 4. add AVX support for fp16 convert. (x86) 5. speed up load model in multi-thread. 6. update Version to V1.3.2	Randall
V1.3.3	Kevin Du	2020-05-14	1. Add PX30 host support 2. update Version to V1.3.3	Randall
V1.4.0	Kevin Du	2020-09-10	1. add multi-scale support. 2. optimize input data preprocess. 3. update Version to V1.4.0	Randall

Version	Modifier	Date	Modify description	Reviewer
V1.6.0	Kevin Du	2021-1-22	1. add some error code in rknn_api.h 2. update Version to V1.6.0	Vincent
V1.6.1	HPC	2021-3-15	1. fix input data preprocess bugs 2. update Version to V1.6.1	Vincent
V1.7.0	HPC	2021-8-5	1. update Version to V1.7.0	Vincent
V1.7.3	HPC	2022-08-20	1. Added an overview chapter to introduce RKNN API functions, applicable hardware platforms and system dependencies; 2. Update the examples usage instructions; 3. Refine the RKNN API usage process; 4. Added NPU firmware update instructions.	Vincent
V1.7.5	HPC	2023-07-12	1. Add FAQ.	Vincent

Table of Contents

1 Overview

1.1 Main function description

The RKNN SDK provides a programming interface for the RK3399Pro platform with NPU to help users deploy the RKNN model exported by the RKNN Toolkit and accelerate the implementation of AI applications.

1.2 Applicable hardware platform

This document is applicable to the following hardware platforms:

- RK3399Pro

1.3 System dependency description

1.3.1 Linux

RKNN API SDK is suitable for systems like Buildroot, Debian9, Debian10 Linux.

1.3.2 Android

RKNN API SDK is suitable for Android8.1 and above systems.

2 Introduction to RKNN SDK project

2.1 RKNN API Library

The libraries and header files provided by the RKNN SDK are located in the <sdk>/rknn_api/librknn_api/ directory. Developers can refer to the corresponding header files and dynamic link libraries in their own applications to use the NPU through the interface provided by the SDK.

It should be noted that the RKNN C API interface used by the RK3399Pro platform and the RK1808 platform is compatible, and the applications developed by the two can be easily ported. But in the process of use, pay attention to distinguish the librknn_api.so of the two platforms. If you use librknn_api.so from different platforms incorrectly, the application may not work properly. Developers can distinguish the platforms corresponding to librknn_api.so by the following methods:

```
# When the librknn_api.so of the RK1808 platform filters the version keyword,
# it displays its own version number
$ strings librknn_api.so |grep version
librknn_api version 1.7.1 (2a83bcc build: 2021-12-02 09:46:02)

# When the librknn_api.so of the RK3399Pro platform filters the version
# keyword, it displays the version of transfer.
$ strings librknn_api.so |grep version
rknn_get_sdk_version
Transfer version 2.1.0 (b5861e7@2020-11-23T11:50:51)
.gnu.version
.gnu.version_r
# The RK3399Pro platform can query the API version number through the rknn_query
# interface, or through the rknn_api.h header file
```

Note: The librknn_api.so used by the RK1808 platform can be obtained from the following project: https://github.com/rockchip-linux/rknpu

2.2 Example usage instructions

The SDK provides C Demos related to image classification (MobileNet), target detection(MobileNet SSD and YoloV5), batch inference and input pass through on Linux/Android platforms,

located in the <sdk>/rknn_api/examples/c_demos directory. At the same time, the SDK also provi des the rk_ssd_demo app for the Android platform, which is located in the <sdk>/rknn_api/exam ples/android_apps/rk_ssd_demo/ directory.

2.2.1 Quick start for C demos

Taking rknn_mobilenet_demo as an example, the compilation and usage process of the demo is as follows:

1. Compile C demo.

```
cd <sdk>/rknn_api/examples/c_demos/rknn_mobilenet_demo
# Select the compilation script according to the OS of the RK3399Pro
# build_linux.sh for Linux
# build_android.sh for Android
# If using build_linux.sh, please change the GCC_COMPILER in the script
# If using build_android.sh, please change the ANDROID_NDK_PATH

./build_linux.sh
# or
./build_android.sh
```

2. Deploy the demo to RK3399Pro.

```
# for Linux
adb push install/rknn_mobilenet_demo_Linux /userdata

# for Android
adb root
adb remount
adb push install/rknn_mobilenet_demo_Android /usrdata/local
```

3. Run the demo.

```
# for Linux
cd /usrdata/rknn_mobilenet_demo_Linux

# for Android
cd /usrdata/local/rknn_mobilenet_demo_Android

./run_demo.sh
```

2.2.2 Quick start for Android application

The RKNN SDK currently provides a target detection application rk_ssd_demo for camera video streams. The application will display the video stream captured by the camera on the screen, and mark the detected objects with boxes in the video.

The application uses ssd_inception_v2 as the target detection model, and calls the RKNN C API through JNI to use the NPU that comes with RK3399Pro for model inference.

The steps to use the app are as follows:

1. Load the project in the Android Stuio software, and adjust the ndk path, Android Gradle Plugin Version (recommended to use 4.2.2) and Gradle Version (recommended to use 6.7.1) and other information in the project.
2. Compile the release version APK. For the specific compilation method, please refer to the relevant documentation of Android Studio.
3. Install the app on the RK3399Pro development board, and open the permissions (camera, storage, etc.) required by the app in the app permissions.
4. Connect the camera, open the APP.

3 RKNN SDK Instructions

3.1 Calling process

The RKNN API call flow is shown in the following figure:

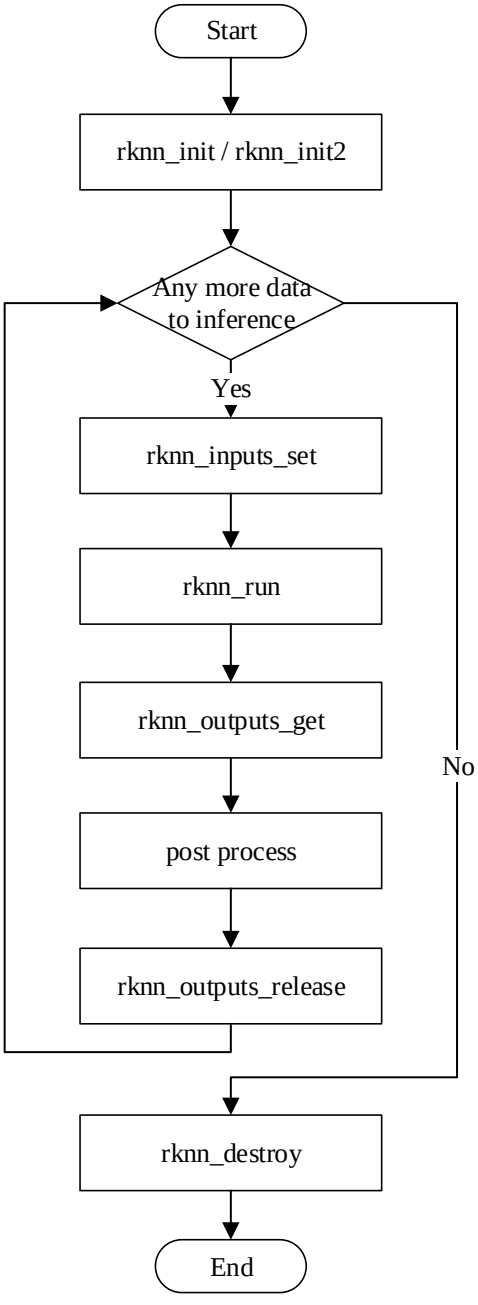

Figure 3-1 RKNN API call flow

The process is detailed as follows:

1. Load RKNN model from file to memory.

2. Call the rknn_init interface to initialize the context and load the RKNN model into the NPU.

The sample code is as follows:

```
rknn_context ctx = 0;
ret = rknn_init(&ctx, model, model_len, RKNN_FLAG_PRIOR_MEDIUM);
if(ret < 0) {
    printf("rknn_init fail! ret=%d\n", ret);
    goto Error;
}
```

Among them, ctx is the context object; model is the pointer of the RKNN model in memory; model_len is the model size; RKNN_FLAG_PRIOR_MEDIUM is the priority flag. (For other flag, please refer to the description in the rknn_init or rknn_init2 interface)

When RK3399Pro is connected to other NPU devices (such as RK1808 computing stick), you need to specify the device ID that runs the RKNN model. In this case, you need to use the rknn_init2 interface and specify the device ID in the extend parameter.

3. The input and output node attributes (such as data types) of the RKNN model may be different from the original model, so it is necessary to obtain the attributes of the input and output nodes through the rknn_query interface. The sample code is as follows:

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
if(ret < 0) {
    printf("rknn_query fail! ret=%d\n",ret);
    goto Error;
}
```

The above interface is used to obtain the number of inputs and outputs, which are stored in io_num.n_input and io_num.n_output.

The sample code to get the properties of the output node is as follows:

```
rknn_tensor_attr output0_attr;
output0_attr.index = 0;
ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &output0_attr, sizeof(output0_attr));
if(ret < 0) {
    printf("rknn_query fail! ret=%d\n",ret);
    goto Error;
}
```

The above interface is used to obtain the attributes of an output node. The index attribute in rknn_tensor_attr must be filled in, and the value of this attribute cannot be greater than or equal to the number of outputs found before. For the detailed definition of this structure, please refer to the description in the data structure rknn_tensor_attr.

The method of obtaining the attributes of the input node is similar to that of the output node

4. According to the input attributes of the RKNN model and the specific format of the input data, call rknn_input_set to set the input of the model. The sample code is as follows:

```
rknn_input inputs[1];
inputs[0].index = input_index;
inputs[0].buf = img.data;
inputs[0].size = img_width * img_height * img_channels;
inputs[0].pass_through = FALSE;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].fmt = RKNN_TENSOR_NHWC;
ret = rknn_inputs_set(ctx, 1, inputs);
if(ret < 0) {
    printf("rknn_input_set fail! ret=%d\n", ret);
    goto Error;
}
```

First, create a rknn_input array (the sample code assumes that the model has only one input), and fill in the attribute values of all members in the array in turn.

index: The index of the input node of the RKNN model.

buf: A pointer to data that the CPU can access, such as image data produced by Camera.

size: The size of the input data buffer.

pass_through: Whether the input data is directly passed to the RKNN model. If the input data attributes (mainly data type, data arrangement, and quantization parameters) passed in by the application are consistent with the model input attributes queried by the rknn_query interface, you can set this variable to TRUE (the type and fmt do not need to be set at this time). In this mode, the rknn_inputs_set interface directly transparently transmits the data Buffer passed in by the application to the input node of the RKNN model. This mode is used to know the input properties of the RKNN model, and the original input data has been preprocessed accordingly.

If the input data attributes passed in by the application are inconsistent with the model input attributes queried by the rknn_query interface, the variable needs to be set to FASLE, and the following type and fmt also need to be set according to the data Buffer passed in by the application. In this mode, the rknn_inputs_set function will automatically perform data type, arrangement format conversion and quantization (or dequantization) processing. Note that currently this mode does not support user input data using dynamic fixed-point quantization (DFP) or asymmetric quantization (AFFINE ASYMMETRIC).

type: The data type of the input data, such as RGB888 data, its type is RKNN_TENS OR_UINT8.

fmt: Arrangement format of the input data, NHWC or NCHW. Generally, the data obtained by Cemara or OpenCV is arranged in the form of RKNN_TENSOR_NHWC.

5. After setting the input data, call the rknn_run interface for model inference. This function will return immediately and will not block. But if three consecutive inference results are not obtained by the application through rknn_outputs_get, the interface will block until rknn_outputs_get is called. The sample code is as follows:

```
ret = rknn_run(ctx, NULL);
if(ret < 0) {
    printf("rknn_run fail! ret=%d\n", ret);
    goto Error;
}
```

6. After executing rknn_run, the application should call the rknn_outputs_get interface to wait for the inference to complete. This function will block until the inference is completed. After the inference is completed, the inference result can be obtained. The sample code is as follows:

```
rknn_output outputs[1];
outputs[0].want_float = TRUE;
outputs[0].is_prealloc = FALSE;
ret = rknn_outputs_get(ctx, 1, outputs, NULL);
if(ret < 0) {
    printf("rknn_outputs_get fail! ret=%d\n", ret);
    goto Error;
}
```

First, create the rknn_output array (the sample code assumes that the model has only one output). The two member variables want_float and is_prealloc in the rknn_output structure must be assigned.

want_float: Since the output of the RKNN model may not be consistent with the output properties of the original model. Usually, the data type of the output node of the RKNN model is UINT8 or FP16. If the user wants to obtain the floating point data of FP32, this attribute can be set to TRUE; if the user wants to obtain the original output data of the RKNN model, then set it to FALSE.

is_prealloc: If the application does not allocate memory for the output data, the pre-allocation flag needs to be set to FALSE. At this time, the remaining member variables in the outputs[0] structure do not need to be assigned, and the runtime component of the NPU will automatically allocate memory for all output nodes. , and fill in the memory address in the buf attribute of outputs[0].

index: the index of the corresponding output node.

buf: The address of the output node data Buffer. If the memory for output data is allocated by the application, the address of the memory should be filled in this property. Otherwise, it does not need to be filled in, and the NPU runtime component will fill in the internally allocated output data Buffer address in this property.

size: The size of the output node data Buffer.

Other attributes of the output node can be obtained by querying rknn_query. It should be noted here that if the memory of the output data is automatically allocated by the NPU runtime

component, you need to call the rknn_outputs_release interface to release the corresponding memory. If the memory is allocated by the application itself, the application needs to release it to avoid memory leaks.

The sample code when the output data memory is freed by the application is as follows:

```
rknn_output outputs[1];
outputs[0].want_float = TRUE;
outputs[0].is_prealloc = TRUE;
outputs[0].index = 0;
outputs[0].buf = output0_buf;
outputs[0].size = output0_attr.n_elems * sizeof(float);
ret = rknn_outputs_get(ctx, 1, outputs, NULL);
if(ret < 0) {
    printf("rknn_outputs_get fail! ret=%d\n", ret);
    goto Error;
}
```

Note that when the application allocates memory, the size of the Buffer needs to be determined according to the attributes of the output node and the value of want_float. When want_float is set to FALSE, the size of the Buffer is the size value of the output attribute obtained by the query, otherwise the size of the Buffer is equal to outputs0_attr.n_elems * sizeof(float).

7. When the output data obtained by using the rknn_outputs_get interface is no longer needed, please call the rknn_outputs_release interface to release the corresponding data, otherwise it will cause memory leaks. The sample code is as follows:

```
rknn_outputs_release(ctx, 1, outputs);
```

It should be noted that, regardless of whether rknn_output[x].is_prealloc passed in by rknn_outputs_get is TRUE or FALSE, this function needs to be called to release the final output.

8. If there is still data to reason about, you can skip back to step 4 for the next reasoning.
9. Before the program exits, you need to call the rknn_destroy interface to unload the RKNN model and destroy the context. The sample code is as follows:

```
rknn_destroy(ctx);
```

For the complete code, please refer to the code in each example/src in SDK/rknn_api/examples/c_demos or android_apps.

3.2 API internal processing flow

When inferring the RKNN model, the original data needs to go through three major processes: input processing, NPU inference, and output processing. In a typical picture inference scenario, assuming that the input data data is a 3-channel picture and the arrangement order is NHWC, the process of data processing at runtime is shown in Figure 3-2-1. At the API level, when the pass_through attribute of the input (see the description of the rknn_input structure for details) is set to FALSE, the rknn_inputs_set interface includes the process of color channel conversion, normalization, quantization, and NHWC to NCHW conversion. When the want_float attribute of the output (see the description of the rknn_output structure for details) is set to TRUE, the rknn_outputs_get interface includes the process of dequantization.

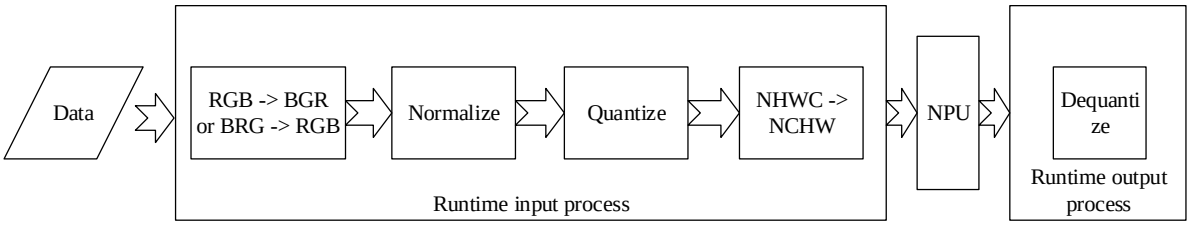

Figure 3-2 Complete picture data processing flow

In fact, for some RKNN models, the flow of input processing is not fully executed. For example, when the input data is not a 3-channel image, or the value of the reorder parameter configured in the config function of the RKNN Toolkit exported model is "0 1 2", the color channel conversion in the input processing flow is not actually performed. When the attribute of the input tensor of the RKNN model is the NHWC layout, there is no NHWC to NCHW process. When the pass_through attribute of the input (see the description of the rknn_input structure for details) is set to TRUE, all input processes in rknn_inputs_set will not be executed, but the input data will be passed directly to the model. When the want_float attribute of the output (see the description of the rknn_output structure for details) is set to

FALSE, the rknn_outputs_get interface will not perform the dequantization operation, but return the actual output data defined by the model.

3.3 Quantization and Dequantization

When the pass_through attribute of the input (see the description of the rknn_input structure for details) is set to 1, it indicates that the input data needs to be processed by the user before NPU inference.

When the want_float attribute of the output (see the description of the rknn_output structure for details) is set to 0, the rknn_outputs_get interface gets the output data type defined in the RKNN model. If it is inconsistent with the data type post-processed by the user, corresponding dequantization needs to be done and so on.

The quantization method, quantization data type, and quantization parameters used in quantization and dequantization can be queried through the rknn_query interface. At present, the NPU of RK3399Pro has two quantization methods: asymmetric and dynamic fixed-point. Each quantization method specifies the corresponding quantized data type. There are a total of four data types and quantification combinations as follows:

- **uint8 (asymmetric quantization)**
- **int8 (dynamic fixed point)**
- **int16 (dynamic fixed point)**
- **float16 (None)**

Usually, the normalized data is stored in 32-bit floating-point numbers, and the 32-bit floating-point numbers are converted to 16-bit floating-point numbers, please refer to the IEEE-754 standard. Assuming that the normalized 32-bit floating-point number is D, the quantization process is described below:

1) float32 to uint8

   Assuming that the asymmetric quantization parameters of the input tensor are Sq, ZP, the data D quantization formula is as follows:

$$D_q = round(clamp(D / S_q + ZP, 0, 255))$$

2) float32 to int8

Assuming that the dynamic fixed-point quantization parameter of the input tensor is fl, the data D quantization formula is as follows:

$$D_q = round(clamp(D * 2^{fl}, -128, 127))$$

3) float32 to int16

Assuming that the dynamic fixed-point quantization parameter of the input tensor is fl, the data D quantization process is expressed as the following formula:

$$D_q = round(clamp(D * 2^{fl}, -32768, 32767))$$

The dequantization process is the inverse process of quantization, and the dequantization formula can be deduced according to the above formula, which will not be repeated here.

3.4 RKNN API Description

The following section is a description of RKNN API.

3.4.1 rknn_init & rknn_init2

API	int rknn_init(rknn_context* context, void* model, uint32_t size, uint32_t flag) int rknn_init2(rknn_context* context, void* model, uint32_t size, uint32_t flag, rknn_init_extend* extend)
Description	Create a context and load the rknn model.
Parameter	rknn_context* context: The pointer of context object. Used to return the created context object.
	void* model: A pointer to the rknn model.
	uint32_t size: The size of the rknn model.
	uint32_t flag: Extended flag: **RKNN_FLAG_PRIOR_HIGH:** Create a high priority context. **RKNN_FLAG_PRIOR_MEDIUM:** Create a medium priority context. **RKNN_FLAG_PRIOR_LOW:** Create a low priority context. **RKNN_FLAG_ASYNC_MASK:** Enable Asynchronous mode. When enable, *rknn_outputs_get* will not block for too long, because it returns the inference result of the previous frame directly (except for the inference result of the first frame), which will significantly improve the inference frame rate in single-thread mode, but the cost is that *rknn_outputs_get* return not the inference results of the current frame. When *rknn_run* and *rknn_outputs_get* are in different threads, there is no need to enable the Asynchronous mode. **RKNN_FLAG_COLLECT_PERF_MASK:** Enable performance collection debug mode. When enable, you can query the running time of each layer of network through the *rknn_query* interface. It should be noted that the total time spent in inferring one frame is longer than *RKNN_FLAG_COLLECT_PERF_MASK* unset, because the execution of each layer needs to be synchronized.
	rknn_init_extend* extend: the pointer of extend information. Used to set or get

	information corresponding to the current *rknn_init*, such as *device_id* (see the rknn_init_extend definition for details). If not used, can be set to NULL.
Return	Error code (see Error Code).

The sample code is as follow:

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0);
```

3.4.2 rknn_destroy

API	int rknn_destroy(rknn_context context)
Description	Unload the rknn model and destroy the context and its associated resource.
Parameter	rknn_context context: The object of context.
Return	Error code (see Error Code).

The sample code is as follow:

```
int ret = rknn_destroy (ctx);
```

3.4.3 rknn_query

API	int rknn_query(rknn_context context, rknn_query_cmd cmd, void* info, uint32_t size)
Description	Query the related information of RKNN Model and SDK.
Parameter	rknn_context context: The object of context.
	rknn_query_cmd cmd: The command of query.
	void* info: The structure variable that store the returned result.
	uint32_t size: The size of the structure variable corresponding to ***info***.
Return	Error code (see Error Code).

The supported query commands of current SDK are shown in the following table:

Table 3-1 Query commands supported by RKNN SDK

Command of Query	Returned Structure	Description
RKNN_QUERY_IN_OUT_NUM	rknn_input_output_num	Query the number of input and output tensor.
RKNN_QUERY_INPUT_ATTR	rknn_tensor_attr	Query the attribute of input tensor.
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	Query the attribute of output tensor.
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	Query the running time of each layer of the network. This query requires use the *RKNN_FLAG_COLLECT_PERF_MASK* in *rknn_init*, otherwise no detailed layer performance information can be obtained. In addition, the *rknn_perf_detail.perf_data* returned by the *RKNN_QUERY_PERF_DETAIL* query does not require the user to free actively. **Pay attention that the query can only return the correct query result after the *rknn_outputs_get* function is called**.
RKNN_QUERY_PERF_RUN	rknn_perf_run	Query the hardware execution time of single inference. **Pay attention that the query can only return the correct query result after the *rknn_outputs_get* function is called.**
RKNN_QUERY_SDK_VERSION	rknn_sdk_version	Query the SDK version.

The next section will explain in detail how each query command should be used.

3.4.3.1 Query the number of input/output tensor

The *RKNN_QUERY_IN_OUT_NUM* command can be used to query the number of input/output tensor. The object of *rknn_input_output_num* structure needs to be created first.

The sample code is as follows:

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
printf("model input num: %d, output num: %d\n", io_num.n_input, io_num.n_output);
```

3.4.3.2 Query the attribute of input tensor

The *RKNN_QUERY_INPUT_ATTR* command can be used to query the attribute of input tensor. The object of *rknn_tensor_attr* structure needs to be created first.

The sample code is as follows:

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
    input_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_INPUT_ATTR, &(input_attrs[i]),
                     sizeof(rknn_tensor_attr));
}
```

3.4.3.3 Query the attribute of output tensor

The *RKNN_QUERY_OUTPUT_ATTR* command can be used to query the attribute of output tensor. The object of *rknn_tensor_attr* structure needs to be created first.

The sample code is as follows:

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
    output_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &(output_attrs[i]),
                     sizeof(rknn_tensor_attr));
}
```

3.4.3.4 Query the running time of each layer of the network

If you have set *RKNN_FLAG_COLLECT_PERF_MASK* flag when rknn_init function is called, then you can use *RKNN_QUERY_PERF_DETAIL* to query the running time of each layer of the network after the rknn_outputs_get execution completed.

The object of *rknn_perf_detail* structure needs to be created first.

In addition, the *rknn_perf_detail.perf_data* returned by the *RKNN_QUERY_PERF_DETAIL* query does not require the user to free it.

Pay attention that the query can only return the correct query result after the *rknn_outputs_get* function is called.

The sample code is as follows:

```
rknn_perf_detail perf_detail;
ret = rknn_query(ctx, RKNN_QUERY_PERF_DETAIL, &perf_detail,
sizeof(rknn_perf_detail));
printf("%s", perf_detail.perf_data);
```

3.4.3.5 Query the hardware execution time of single inference.

The *RKNN_QUERY_PERF_RUN* command can be used to query the hardware execution time of single inference. The object of *rknn_perf_run* structure needs to be created first.

Pay attention that the query can only return the correct query result after the *rknn_outputs_get* function is called.

The sample code is as follows:

```
rknn_perf_run perf_run;
ret = rknn_query(ctx, RKNN_QUERY_PERF_RUN, &perf_run,
                 sizeof(rknn_perf_run));
printf("%ld", perf_run.run_duration);
```

3.4.3.6 Query the SDK version

The *RKNN_QUERY_SDK_VERSION* command can be used to query the SDK version. The object of *rknn_sdk_version* structure needs to be created first.

The sample code is as follows:

```
rknn_sdk_version version;
ret = rknn_query(ctx, RKNN_QUERY_SDK_VERSION, &version,
                 sizeof(rknn_sdk_version));
printf("api version: %s\n", version.api_version);
printf("driver version: %s\n", version.drv_version);
```

3.4.4 rknn_inputs_set

API	int rknn_inputs_set(rknn_context context, uint32_t n_inputs, rknn_input inputs[])
Description	Set the buffer pointer and other parameters of inputs. The buffer pointer and parameters of single input need to be stored in *rknn_input*. This function can support multiple inputs.
Parameter	rknn_context context: the object of context.
	uint32_t n_inputs: the number of inputs.
	rknn_input inputs[]: the arrays of inputs information, each element of the array is a *rknn_input* structure object.
Return	Error code (see Error Code).

The sample code is as follows:

```
rknn_input inputs[1];
memset(inputs, 0, sizeof(inputs));
inputs[0].index = 0;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].size = img_width*img_height*img_channels;
inputs[0].pass_through = FALSE;
inputs[0].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;

ret = rknn_inputs_set(ctx, 1, inputs);
```

3.4.5 rknn_run

API	int rknn_run(rknn_context context, rknn_run_extend* extend)
Description	Perform a model inference. The input data need to be set by *rknn_inputs_set* function before calling *rknn_run*. The *rknn_run* will not block normally, but it will block when there are more than 3 inference results not obtained by *rknn_outputs_get*.
Parameter	rknn_context context: the object of context.
	rknn_run_extend* extend: the pointer of extend information. Used to set or get information about the frame corresponding to the current *rknn_run*, such as *frame_id* (see the *rknn_run_extend* definition for details). If not used, can be set to NULL.
Return	Error code (see Error Code).

The sample code is as follows:

```
ret = rknn_run(ctx, NULL);
```

3.4.6 rknn_outputs_get

API	int rknn_outputs_get(rknn_context context, uint32_t n_outputs, rknn_output outputs[], rknn_output_extend* extend)
Description	Waiting for the inference operation to completed and get the output results. This function can obtain multiple output data at one time. Each output corresponds to a rknn_output structure object, you need to create and set each rknn_output object in turn before the function is called. In addition, the function will block until the inference completed (unless there is an exception error). The output results will eventually be stored in the array of *outputs[]*. There are two ways to use the buffer of the output data: 1. Users malloc and free the output buffer themselves. In this mode, the *is_prealloc* of the *rknn_output* object needs to be set to TRUE, and the *rknn_output.buf* also needs to be set by user. 2. The output buffer malloc and free by rknn api. In this mode, the *is_prealloc* of the *rknn_output* object needs to be set to FALSE, and *rknn_output.buf* will point to output data after the function is called.
Parameter	rknn_context context: the object of context.
	uint32_t n_outputs: the number of output arrays. This number must be the same as the number of outputs of rknn model. (the number of outputs of rknn model can be queried by *rknn_query*.)
	rknn_output outputs[]: the arrays of outputs information. Each element of array is a *rknn_output* structure object, representing an output of the model.
	rknn_output_extend* extend: the pointer of extend information. Used to set or get information about the frame corresponding to the current *rknn_outputs_get*, such as *frame_id* (see the *rknn_output_extend* definition for details). If not used, can be set to NULL.
Return	Error code (see Error Code).

The sample code is as follows:

```
rknn_output outputs[io_num.n_output];
memset(outputs, 0, sizeof(outputs));
for (int i = 0; i < io_num.n_output; i++) {
        outputs[i].want_float = TRUE;
        outputs[i].is_prealloc = FALSE;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

3.4.7 rknn_outputs_release

API	int rknn_outputs_release(rknn_context context, uint32_t n_ouputs, rknn_output outputs[])
Description	Release outputs that obtained by *rknn_outputs_get*. When the outputs are no longer used, you need to call the function to release it. (Whether *rknn_output[x].is_prealloc* is TRUE or FALSE, you need to call the function to release the outputs.) After the function is called: when *rknn_output[x].is_prealloc* = FALSE, the *rknn_output[x].buf* obtained by *rknn_outputs_get* is also released automatically; when *rknn_output[x].is_prealloc* = TURE, the *rknn_output[x].buf* requires user to free it.
Parameter	rknn_context context: the object of context.
	uint32_t n_outputs: the number of output arrays. This number must be the same as the number of outputs of rknn model. (the number of outputs of rknn model can be queried by *rknn_query*.)
	rknn_output outputs[]: the arrays of outputs information.
Return	Error code (see Error Code).

The sample code is as follows:

```
ret = rknn_outputs_release(ctx, io_num.n_output, outputs);
```

3.4.8 rknn_find_devices

API	int rknn_find_devices(rknn_devices_id* pdevs)
Description	find the devices information that connected to host.
Parameter	rknn_devices_id* pdevs: the pointer of devices information structure.
Return	Error code (see Error Code).

The sample code is as follows:

```
rknn_devices_id devids;
ret = rknn_find_devices (&devids);
printf("n_devices = %d\n", devids.n_devices);
for(int i=0; i<devids.n_devices; i++) {
    printf("%d:  type=%s, id=%s\n", i, devids.types[i], devids.ids[i]);
}
```

3.5 RKNN Data Structure Definition

3.5.1 rknn_input_output_num

The structure *rknn_input_output_num* represents the number of tensors of input and output. The following table shows the definition of the structure:

Table 3-2 Definition of rknn_input_output_num struture

Field	Data Type	Meaning
n_input	uint32_t	The number of input tensor.
n_output	uint32_t	The number of output tensor.

3.5.2 rknn_tensor_attr

The structure *rknn_tensor_attr* represents the tensor attribute of rknn model, The following table shows the definition of the structure:

Table 3-3 Definition of rknn_tensor_attr

Field	Data Type	Meaning
index	uint32_t	The index of input or output tensor. The index needs to be set before calling the *rknn_query*.
n_dims	uint32_t	The number of tensor dimensions.
dims	uint32_t[]	Each dimension value of tensor.
name	char[]	Name of tensor.
n_elems	uint32_t	The number of tensor elements.
size	uint32_t	The memory size of tensor data.
fmt	rknn_tensor_format	The dimension format of tensor, as follows: ***RKNN_TENSOR_NCHW*** ***RKNN_TENSOR_NHWC***
type	rknn_tensor_type	The data type of tensor, as follows: ***RKNN_TENSOR_FLOAT32*** ***RKNN_TENSOR_FLOAT16*** ***RKNN_TENSOR_INT8*** ***RKNN_TENSOR_UINT8*** ***RKNN_TENSOR_INT16***
qnt_type	rknn_tensor_qnt_type	The quantization type of tensor, ds: ***RKNN_TENSOR_QNT_NONE****:* none quantization. ***RKNN_TENSOR_QNT_DFP****:* Dynamic fixed-point quantization. ***RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC****:*

		Asymmetric affine quantization.
fl	int8_t	Fractional length for *RKNN_TENSOR_QNT_DFP*.
zp	uint32_t	Zero point for *RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC*.
scale	float	Scale for *RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC*.

3.5.3 rknn_input

The structure *rknn_input* represents an input data of model, used as a parameter to the *rknn_inputs_set* function. The following table shows the definition of the structure:

Table 3-4 Definition of rknn_input

Field	Data Type	Meaning
index	uint32_t	The index of input tensor.
buf	void*	The buffer point of input data.
size	uint32_t	The memory size of input data buffer.
pass_through	uint8_t	The pass-through mode of input. **TRUE:** The input data is passed directly to the input node of rknn model without any conversion, therefore the following ***type*** and ***fmt*** do not need to be set. **FALSE:** The input data will convert to the same data type and format as the input node of the rknn mode according to the following ***type*** and ***fmt***, therefore the following ***type*** and ***fmt*** need to be set.
type	rknn_tensor_type	The data type of input tensor, as follow: ***RKNN_TENSOR_FLOAT32*** ***RKNN_TENSOR_FLOAT16*** ***RKNN_TENSOR_INT8*** ***RKNN_TENSOR_UINT8*** ***RKNN_TENSOR_INT16***
fmt	rknn_tensor_format	The dimension format of input tensor, as follow: ***RKNN_TENSOR_NCHW*** ***RKNN_TENSOR_NHWC***

3.5.4 rknn_output

The structure *rknn_output* represents an output data of the model, used as a parameter to the

rknn_outputs_get function. The following table shows the definition of the structure:

Table 3-5 Definition of rknn_output

Field	Data Type	Meaning
want_float	uint8_t	Identifies whether the output data needs to be converted to float32 type.
is_prealloc	uint8_t	Identifies whether the buffer that holds the output data is pre-allocated.
index	uint32_t	The index of output tensor.
buf	void*	The buffer pointer of output.
size	uint32_t	The memory size of output data buffer.

When the *is_prealloc* is FALSE, the ***index/buf/size*** of *rknn_output* will be set after *rknn_outputs_get* is called, therefore the three members do not need to be pre-set.

When the *is_prealloc* is TRUE, the ***index/buf/size*** of *rknn_output* need to be set before calling *rknn_outputs_get*, otherwise the *rknn_outputs_get* function will fail with an error.

3.5.5 rknn_perf_detail

The structure *rknn_perf_detail* represents the performance details of rknn model. The following table shows the definition of the structure:

Table 3-6 Definition of rknn_perf_detail

Field	Data Type	Meaning
perf_data	char*	Contains the running time of each layer of the network, can be printed directly for viewing.
data_len	uint64_t	The string length of ***perf_data***.

3.5.6 rknn_perf_run

The structure *rknn_perf_run* represents the execution time of a single inference of rknn model.

The following table shows the definition of the structure:

Table 3-7 Definition of rknn_perf_run

Field	Data Type	Meaning
run_duration	int64_t	The hardware execution time (us) of a single inference of rknn model.

3.5.7 rknn_init_extend

The structure *rknn_init_extend* represents the extended information of rknn_init, used as parameter to rknn_init function.

The following table shows the definition of the structure:

Table 3-8 Definition of rknn_init_extend

Field	Data Type	Meaning
device_id	char*	Used to select the connected device. Such as "0123456789ABCDEF", the device id can be query by "adb devices". If only one device connected, can set nullptr.

3.5.8 rknn_run_extend

The structure *rknn_run_extend* represents the extended information of rknn_run, used as parameter to rknn_run function.

The following table shows the definition of the structure:

Table 3-9 Definition of rknn_run_extend

Field	Data Type	Meaning
frame_id	uint64_t	Used to get the frame id after the *rknn_run* function is called. The *frame_id* corresponds to *rknn_output_extend.frame_id* one by one, In the case where *rknn_run* and *rknn_outputs_get* are in different threads, it can be used to determine the correspondence of frame.

3.5.9 rknn_output_extend

The structure *rknn_output_extend* represents the extend information of *rknn_outputs_get*, used as parameter to *rknn_outputs_get* function. The following table shows the definition of the structure:

Table 3-10 Definition of rknn_output_extend

Field	Data Type	Meaning
frame_id	uint64_t	Used to get the frame id after the *rknn_outputs_get* function is called. The *frame_id* corresponds to *rknn_run_extend.frame_id* one by one, In the case where *rknn_run* and *rknn_outputs_get* are in different threads, it can be used to determine the correspondence of frame.

3.5.10 rknn_sdk_version

The structure *rknn_sdk_version* represents the version information of RKNN SDK. The following table shows the definition of the structure:

Table 3-11 Definition of rknn_sdk_version

Field	Data Type	Meaning
api_version	char[]	The version of RKNN API.
drv_version	char[]	The driver version on which RKNN API is based.

3.5.11 rknn_devices_id

The structure *rknn_devices_id* represents the information of device ID list. The following table shows the definition of the structure:

Table 3-12 Definition of rknn_device_id

Field	Data Type	Meaning
n_devices	uint32_t	The number of devices
types	char[][]	The array of device type.
ids	char[][]	The array of device ID.

3.5.12 Error Code

The return error code of RKNN API. The following table shows the definition:

Table 3-13 RKNN error code

Error Code	Meaning
RKNN_SUCC	Execution is successful.
RKNN_ERR_FAIL	Execution is failed.
RKNN_ERR_TIMEOUT	Execution timeout.
RKNN_ERR_DEVICE_UNAVAILABLE	The NPU device is unavailable.
RKNN_ERR_MALLOC_FAIL	Memory allocation is failed.
RKNN_ERR_PARAM_INVALID	The parameter is invalid.
RKNN_ERR_MODEL_INVALID	The RKNN model is invalid.
RKNN_ERR_CTX_INVALID	The rknn_context is invalid.
RKNN_ERR_INPUT_INVALID	The object of rknn_input is invalid.
RKNN_ERR_OUTPUT_INVALID	The object of rknn_output is invalid.
RKNN_ERR_DEVICE_UNMATCH	The device version does not match.
RKNN_ERR_INCOMPATILE_PRE_COMPILE_MODEL	PreCompile mode is not compatible with current driver.
RKNN_ERR_INCOMPATILE_OPTIMIZATION_LEVEL_VERSION	Optimization level in model is not compatible with current driver.
RKNN_ERR_TARGET_PLATFORM_UNMATCH	Target platform in model is not compatible with current platform.

4 NPU firmware description

4.1 NPU Firmware Directory Description

The NPU driver of RK3399Pro is packaged in the boot.img file of the NPU. When RK3399Pro updates the NPU driver, just replace the corresponding boot.img and other files.

Different RK3399Pro development boards may communicate with NPU in different ways (PCIE and USB 3.0), and use different NPU firmware such as boot.img.

The main directories include:

```
npu_firmware/
├──── npu_fw
└──── npu_fw_pcie
```

- npu_fw_pcie: NPU firmware for PCIE interface, including boot.img, MiniLoaderAll.bin, trust.img, uboot.img, etc.
- npu_fw: NPU firmware for USB interface, including boot.img, MiniLoaderAll.bin, trust.img, uboot.img, etc.

Note: You can simply judge which interface mode the NPU of the current development board is using by short-circuiting the NPU CONNECTION pin. If the NPU connection pins are shorted between 1 and 2, then USB3.0 is used. If 2 and 3 are shorted, PCIE is used. NPU CONNECTION location as shown below:

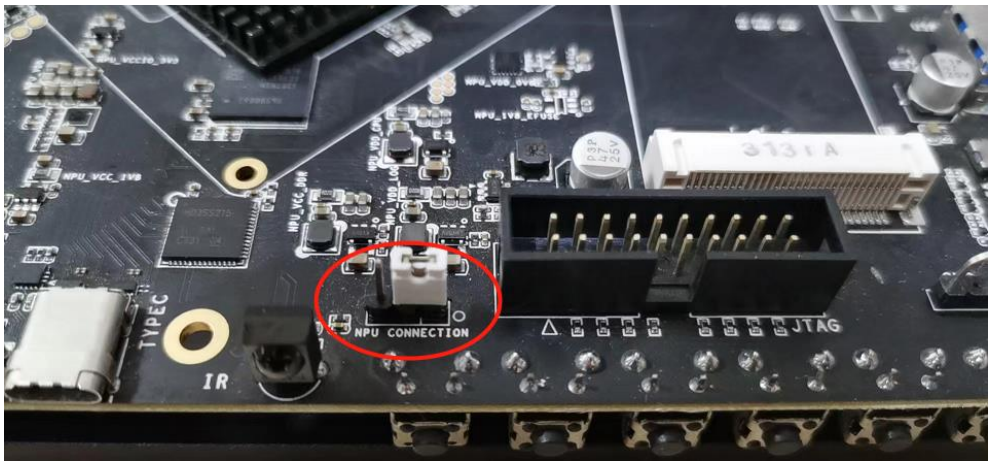

Figure 4-1 Schematic diagram of the location of NPU CONNECTION

If there is no NPU CONNECTION pin, it is generally connected by USB3.0.

4.2 Update NPU driver

Updating the NPU driver on RK3399Pro is achieved by updating the NPU-related root.img and other files. The specific update method is as follows:

- Update PCIE interface NPU driver:

```
# For Android OS, please get root permission first
# commands below is not necessary for Linux OS.
adb shell root
adb shell remount
# update NPU firmware
adb push npu_firmware/npu_pcie_fw/* /vendor/etc/npu_fw/
adb shell reboot
```

- Update USB interface NPU driver:

```
# For Android OS, please get root permission first
# commands below is not necessary for Linux OS.
adb shell root
adb shell remount
# update NPU firmware
adb push npu_firmware/npu_fw/* /vendor/etc/npu_fw/
adb shell reboot
```

Note: The path of npu_fw may be different for different RK3399Pro firmware. It is recommended to confirm the location of this folder before updating boot.img and other files.

5 FAQ

5.1 Communication related issues

5.1.1 After the model runs for a period of time, the inference hangs and cannot continue.

There may be two reasons for this phenomenon: npu_transfer_proxy does not match the version of librknn_api / rknn_server used; the underlying communication (USB / PCIE) is abnormal.

For the first case, please unify the versions of npu_transfer_proxy / librknn_api / rknn_server and try again after using the latest version. Note: please update rknn_server by updating the npu firmware.

For the second case, please collect RK3399Pro CPU-side system kernel logs and NPU-side kernel logs, and feed them back to Rockchip engineers for further analysis.

5.2 Questions about demo

5.2.1 How to obtain the cross-compilation tools

Please get the cross-compilation tool used for compiling the RK3399Pro Linux demo from the link below:

https://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/aarch64-linux-gnu/

Choose: gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu.tar.xz

6 Appendix

6.1 Reference documents

RKNN Toolkit manual:

Rockchip_User_Guide_RKNN_Toolkit_EN.pdf

RKNN Toolkit Lite manual:

Rockchip_User_Guide_RKNN_Toolkit_Lite_EN.pdf

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

If you want to use the RKNN C API for RK1808, please refer to the following projects:

https://github.com/rockchip-linux/rknpu

6.2 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 contact them to report the issue.