RKNN C API Reference

ID: RK-YH-YF-413

Release Version: V2.3.0

Release Date: 2024-11-06

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

Rockchip Electronics Co., Ltd.

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

Website: www.rock-chips.com

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

Customer service Fax: +86-591-83951833

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

Preface

Overview

This article describes Rockchip RKNN C API Reference.

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Revision History

Version	Modifier	Date	Modify Description	Reviewer
v0.6.0	HPC Team	2/Mar/2021	Initial version	Vincent
v0.7.0	HPC Team	22/Apr/2021	Remove description of swapping input data channels	Vincent
v1.0.0	HPC Team	30/Apr/2021	Release version	Vincent
v1.1.0	HPC Team	13/Aug/2021	1.Add rknn_tensor_mem_flags 2.Add query commands for input/output Tensor native attributes 3.Add the memory layout of NC1HWC2	Vincent
v1.2.0b1	NPU Team	07/Jan/2022	1.Add RK3588/RK3588s pla Add rknn_set_core_mask interface 2.Add rknn_dump_context interface 3.Add details for input and output API	Vincent
v1.2.0	HPC Team	14/Jan/2022	1.Add the Glossary 2. Add the NPU SDK guide and building and compiling instruction 3. Add the section of how to do debugging 4. Add the description of C2 in the section NATIVE_LAYOUT	Vincent
v1.3.0	NPU /HPC Team	13/May/2022	1.Fix name from destory to destroy 2.Add RV1106/RV1103 user guide 3.Add details for NATIVE_LAYOUT 4.Add C API hardware platform support description 5.Add NPU version, utilization query and instruction of NPU power manual switch	Vincent
v1.4.0	NPU /HPC Team	31/Aug/2022	1.For RV1106/RV1103 add rknn_create_mem_from_phys/ rknn_create_mem_from_fd/ rknn_set_weight_mem/ rknn_set_internal_mem support 2.Add function of weights shared 3.Add support of utlization of SRAM for RK3588 4.Add the support of single batch multi-cores of NPU for RK3588 5.Add new version of driver supporting the query of frequency, voltage, setting of a delay time for shutdown , etc.	Vincent
v1.4.2	HPC Team	13/Feb/2023	1.Add RK3562 user guide 2.Add RKNN_FLAG_COLLECT_MODEL_INFO_ONLY flag description in rknn_init interface	Vincent
v1.5.0	HPC Team	22/May/2023	1.Add dynamic shape input API instructions and related data structure instructions 2.Add instructions for using Matmul API	Vincent
v1.5.2	HPC Team	22/Aug/2023	1.Add RKNN_FLAG_EXECUTE_FALLBACK_PRIOR_DEVICE_GPU and RKNN_FLAG_INTERNAL_ALLOC_OUTSIDE flag description in rknn_init interface 2.Remove the rknn_set_input_shape interface description of the old dynamic input shape function, and add the rknn_set_input_shapes interface description	Vincent
v1.6.0	HPC Team	28/Nov/2023	1.Add descriptions of RKNN_FLAG_ENABLE_SRAM, RKNN_FLAG_SHARE_SRAM and RKNN_FLAG_DISABLE_PROC_HIGH_PRIORITY flags in the rknn_init interface 2.Add rknn_set_batch_core_num interface description 3.Add rknn_mem_sync interface description	Vincent

Version	Modifier	Date	Modify Description	Reviewer
v2.0.0-beta0	HPC Team	18/Mar/2024	1.Add rknn_create_mem2 interface description 2. Modify the rknn_matmul_info and rknn_matmul_type structures 3. Add rknn_quant_params and rknn_matmul_shape structures 4. Added rknn_matmul_set_quant_params, rknn_matmul_get_quant_params, rknn_matmul_create_dyn_shape, rknn_matmul_set_dynamic_shape interfaces	Vincent
v2.1.0	HPC Team	02/Aug/2024	1. Add RV1103B and RK2118 user guide 2. Add descriptions of RKNN_FLAG_DISABLE_FLUSH_INPUT_MEM_CACHE, RKNN_FLAG_DISABLE_FLUSH_OUTPUT_MEM_CACHE, RKNN_FLAG_MODEL_BUFFER_ZERO_COPY flags in the rknn_init interface 3. Add RKNN_MEM_FLAG_ALLOC_NO_CONTEXT flag description 4. Add RKNN_QUERY_DEVICE_MEM_INFO query command description 5. Modify rknn_init_extend, rknn_matmul_info, rknn_matmul_type and rknn_quant_params data structures 6. Modify rknn_B_normal_layout_to_native_layout interface 7. Add RKNN_FLOAT16_MM_INT4_TO_FLOAT32, RKNN_FLOAT16_MM_INT4_TO_FLOAT16, RKNN_INT8_MM_INT4_TO_INT32 data type support 8. Add rknn_matmul_layout data structure， add B_layout=RKNN_MM_LAYOUT_TP_NORM support 9. Add rknn_matmul_quant_type data structure, support Matmul API's B-matrix grouping symmetric quantization 10. Scrap rknn_matmul_create_dyn_shape and use rknn_matmul_create_dynamic_shape interfaces instead	Vincent
v2.2.0	HPC Team	04/Sep/2024	Update version number	Vincent
v2.3.0	HPC Team	06/Nov/2024	Update version number	Vincent

Contents

1. Overview

RKNN C API is the C language interfaces for RKNPU Runtime (runtime library). By using the RKNN C API, developers can use the computing power of the NPU to perform efficient RKNN model inference or matrix multiplication calculation tasks. This article explains each function, data structure and return code definition of RKNN C API.

2. Supported Hardware Platforms

This document applies to the following hardware platforms:

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

3. RKNPU compilation instructions

When developers compile applications, they must include the header file where the interface function is located, and link to the corresponding RKNPU runtime library according to the hardware platform and system type used. The following describes the RKNN C API header files and runtime library files.

3.1 RKNN C API header files

According to different functional characteristics, the interface of RKNN C API can be divided into three parts. The corresponding relationships between the functions, data structure definitions and header files of each part are as follows:

1. "rknn_api.h" defines the basic interface and data structure for deploying the RKNN model.
2. "rknn_matmul_api.h" defines the matrix multiplication interface and data structure.
3. "rknn_custom_op.h" defines custom operator interface and data structure.

3.2 RKNPU Runtime Library For Linux

1. For the RK3562 , RK3566 series, RK3568 series, RK3576 series and RK3588 series hardware platforms, the RKNPU runtime library file is librknnrt.so in the <sdk_path>/rknpu2/runtime directory, where <sdk_path> is the path to the Rockchip NPU software development kit.
2. For RV1106/RV1103/RV1106B/RV1103B hardware platforms, the RKNPU runtime library file is librknnmrt.so in the <sdk_path>/rknpu2/runtime directory.

3.3 RKNPU Runtime Library For Android

There are two ways to call the RKNN API on the Android platform:

1. The application can link librknnrt.so directly.
2. Application link to librknn_api_android.so implemented by HIDL on Android platform.

Android devices that need to pass the CTS/VTS test need to use the RKNN API implemented based on the Android platform HIDL (**librknn_api_android.so does not include matrix multiplication and custom operator functions**). If the device does not need to pass the CTS/VTS test, it is recommended to use librknnrt.so directly (including matrix multiplication and custom operator functions). The link to each interface call process is shorter and can provide better performance.

The code for the RKNN API implemented using Android HIDL is located in the vendor/rockchip/hardware/interfaces/neuralnetworks directory of the RK3562/RK3566/RK3568/RK3576/RK3588 Android system SDK. After the Android system is compiled, a series of library files related to RKNPU will be generated (for application development, you only need to link and use librknn_api_android.so), as shown below:

```
/vendor/lib/librknn_api_android.so
/vendor/lib/librknnhal_bridge.rockchip.so
/vendor/lib64/librknn_api_android.so
/vendor/lib64/librknnhal_bridge.rockchip.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0-adapter-helper.so
/vendor/lib64/hw/rockchip.hardware.neuralnetworks@1.0-impl.so
/vendor/bin/hw/rockchip.hardware.neuralnetworks@1.0-service
```

Alternatively, user can use the following command to recompile and generate the above library separately.

```
mmm vendor/rockchip/hardware/interfaces/neuralnetworks/ -j8
```

3.4 RKNPU Runtime Library For RK2118

For the RK2118 hardware platform, the RKNPU runtime library file is located in the system SDK of the RK2118, please refer to the section "Prepare RK2118 SDK" in **"Rockchip_RK2118_Quick_Start_RKNN_SDK_EN.pdf"**.

4. RKNN C API Description

4.1 C API support of each hardware platform

Due to the different hardware characteristics of different chip platforms, the RKNN C API interface and interface parameter support are also different. The RKNN C API interface support of each hardware platform is shown in Table 4-1:

Table 4-1 RKNN C API interface support for each hardware platform

	RKNN C API	RK3562/RK3566/ RK3568	RK3576/RK3588	RV1106/RV1103/ RV1103B/RV1106B	RK2118
1	rknn_init	√	√	√	√
2	rknn_set_core_mask	×	√	×	×
3	rknn_dup_context	√	√	×	×
4	rknn_destroy	√	√	√	√
5	rknn_query	√	√	√	√
6	rknn_inputs_set	√	√	×	×
7	rknn_run	√	√	√	√
8	rknn_wait	×	×	×	×
9	rknn_outputs_get	√	√	×	×
10	rknn_outputs_release	√	√	×	×
11	rknn_create_mem_from_mb_blk	×	×	×	×
12	rknn_create_mem_from_phys	√	√	√	√
13	rknn_create_mem_from_fd	√	√	√	×
14	rknn_create_mem	√	√	√	√
15	rknn_destroy_mem	√	√	√	√
16	rknn_set_weight_mem	√	√	√	√
17	rknn_set_internal_mem	√	√	√	√
18	rknn_set_io_mem	√	√	√	√
19	rknn_set_input_shapes	√	√	×	×
20	rknn_mem_sync	√	√	√	√
21	rknn_matmul_create	√	√	×	×
22	rknn_matmul_set_io_mem	√	√	×	×
23	rknn_matmul_set_core_mask	×	√	×	×
24	rknn_matmul_run	√	√	×	×
25	rknn_matmul_destroy	√	√	×	×
26	rknn_register_custom_ops	√	√	×	×
27	rknn_custom_op_get_op_attr	√	√	×	×
28	rknn_set_batch_core_num	×	√	×	×
29	rknn_matmul_set_quant_params	√	√	×	×
30	rknn_matmul_get_quant_params	√	√	×	×

	RKNN C API	RK3562/RK3566/ RK3568	RK3576/RK3588	RV1106/RV1103/ RV1103B/RV1106B	RK2118
31	rknn_matmul_create_dynamic_shape	√	√	×	×
32	rknn_matmul_set_dynamic_shape	√	√	×	×
33	rknn_B_normal_layout_to_native_layout	√	√	×	×
34	rknn_create_mem2	√	√	√	√

The query parameters supported by each hardware platform using the rknn_query function are shown in Table 4-2:

Table 4-2 Query parameters supported by the rknn_query function on each hardware platform

	rknn_query params	RK3562/RK3566/ RK3568	RK3576/ RK3588	RV1106/RV1103/ RV1103B/RV1106B/RK2118
1	RKNN_QUERY_IN_OUT_NUM	√	√	√
2	RKNN_QUERY_INPUT_ATTR	√	√	√
3	RKNN_QUERY_OUTPUT_ATTR	√	√	√
4	RKNN_QUERY_PERF_DETAIL	√	√	×
5	RKNN_QUERY_PERF_RUN	√	√	×
6	RKNN_QUERY_SDK_VERSION	√	√	√
7	RKNN_QUERY_MEM_SIZE	√	√	√
8	RKNN_QUERY_CUSTOM_STRING	√	√	√
9	RKNN_QUERY_NATIVE_INPUT_ATTR	√	√	√
10	RKNN_QUERY_NATIVE_OUTPUT_ATTR	√	√	√
11	RKNN_QUERY_NATIVE_NC1HWC2_INPUT_ATTR	√	√	√
12	RKNN_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR	√	√	√
13	RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR	√	√	√
14	RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR	√	√	√
15	RKNN_QUERY_INPUT_DYNAMIC_RANGE	√	√	×
16	RKNN_QUERY_DEVICE_MEM_INFO	×	×	√
17	RKNN_QUERY_CURRENT_INPUT_ATTR	√	√	×
18	RKNN_QUERY_CURRENT_OUTPUT_ATTR	√	√	×
19	RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR	√	√	×
20	RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR	√	√	×

4.2 Definition of Basic Data Structcture

rknn_sdk_version

The structure rknn_sdk_version is used to indicate the version information of the RKNN SDK. The following table shows the definition:

Field	Type	Meaning
api_version	char[]	SDK API version information.
drv_version	char[]	SDK driver version information.

rknn_input_output_num

The structure rknn_input_output_num represents the number of input and output Tensor， The following table shows the definition:

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

rknn_input_range

The structure rknn_input_range represents an input support shape list information. It contains the input index, the number of supported shapes, data layout format, name and shape list. The definition of the specific structure is shown in the following table:

Field	Type	Meaning
index	uint32_t	The index position of the input.
shape_number	uint32_t	The number of input shapes supported by the RKNN model.
fmt	rknn_tensor_format	The data layout format corresponding to the shape.
name	char[]	The name of the input.
dyn_range	uint32_t[][]	The input shape list, which is a two-dimensional array containing multiple shape arrays, and the shape is stored first.
n_dims	uint32_t	The number of valid dimensions for each shape array.

rknn_tensor_attr

The structure rknn_tensor_attr represents the attribute of the model's Tensor. The following table shows the definition:

Field	Type	Meaning
index	uint32_t	Indicates the index position of the input and output Tensor.
n_dims	uint32_t	The number of Tensor dimensions.
dims	uint32_t[]	Values for each dimension.
name	char[]	Tensor name.
n_elems	uint32_t	The number of Tensor data elements.
size	uint32_t	The memory size of Tensor data.
fmt	rknn_tensor_format	The format of Tensor dimension, has the following format: **RKNN_TENSOR_NCHW**, **RKNN_TENSOR_NHWC**, **RKNN_TENSOR_NC1HWC2**
type	rknn_tensor_type	Tensor data type, has the following data types: **RKNN_TENSOR_FLOAT32**, **RKNN_TENSOR_FLOAT16**, **RKNN_TENSOR_INT8**, **RKNN_TENSOR_UINT8**, **RKNN_TENSOR_INT16**, **RKNN_TENSOR_UINT16**, **RKNN_TENSOR_INT32**, **RKNN_TENSOR_INT64**, **RKNN_TENSOR_BOOL**
qnt_type	rknn_tensor_qnt_type	Tensor Quantization Type, has the following types of quantization: **RKNN_TENSOR_QNT_NONE:** Not quantized; **RKNN_TENSOR_QNT_DFP**: Dynamic fixed point quantization; **RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC**: Asymmetric quantification.
fl	int8_t	RKNN_TENSOR_QNT_DFP: quantization parameter.
scale	float	RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC: quantization parameter.
w_stride	uint32_t	The number of pixels that actually store one line of image data, which is equal to the number of valid data pixels in one row + the number of invalid pixels that are filled in for the hardware to quickly jump to the next line (unit: per pixel) .
size_with_stride	uint32_t	The actual byte size of image data (including the byte size of filled invalid pixels).
pass_through	uint8_t	0: means unconverted data. 1: means converted data, Note: Conversion includes normalization and quantization.

Field	Type	Meaning
h_stride	uint32_t	This is only used in the context of multi-batch input and can be set by users. The purpose of this is to allow NPU to read out beginning of memory address for every batch correctly. It is equivalent of original model input height + the number of invalid pixels that are filled in for the hardware to quickly jump to the next line . If its value is 0, it is the same as model input height (unit: per pixel) .

rknn_perf_detail

The structure rknn_perf_detail represents the performance details of the model. The definition of the structure is shown in the following table: (**Note: RV1106/RV1103/RV1106B/RV1103B/RK2118 unsupported**)

Field	Type	Meaning
perf_data	char*	The performance details contain the running time of each layer of the network stored in string type.
data_len	uint64_t	The data length of string type of perf_data.

rknn_perf_run

The structure rknn_perf_run represents the total inference time of the model. The definition of the structure is shown in the following table: (**Note: RV1106/RV1103/RV1106B/RV1103B/RK2118 unsupported**)

Field	Type	Meaning
run_duration	int64_t	The total inference time of the network (not including setting input/output) in microseconds.

rknn_mem_size

The structure rknn_mem_size represents the memory allocation when the model is initialized. The definition of the structure is shown in the following table:

Field	Type	Meaning
total_weight_size	uint32_t	The memory size allocated for weights of the network.
total_internal_size	uint32_t	The memory size allocated for internal tensor of the network.
total_dma_allocated_size	uint64_t	All dma memory size allocated for the network.
total_sram_size	uint32_t	Only for RK3588, memory size of SRAM reserved for NPU (Referring to the <<RK3588_NPU_SRAM_usage.md>> for more details) .
free_sram_size	uint32_t	Only for RK3588, the current available SRAM (Referring to the <<RK3588_NPU_SRAM_usage.md>> for more details) .
reserved[12]	uint32_t	Reserve.

rknn_tensor_mem

The structure rknn_tensor_mem represents the tensor memory information. The definition of the structure is shown in the following table:

Field	Type	Meaning
virt_addr	void*	The virtual address of tensor.
phys_addr	uint64_t	The physical address of tensor.
fd	int32_t	The file descriptor of tensor.
offset	int32_t	The offset of fd and virtual address.
size	uint32_t	The actual size of tensor.
flags	uint32_t	rknn_tensor_mem has the following type of flags: **RKNN_TENSOR_MEMORY_FALGS_ALLOC_INSIDE:** indicates that the rknn_tensor_mem structure is created during runtime; **RKNN_TENSOR_MEMORY_FLAGS_FROM_FD:** It represents the rknn_tensor_mem created by fd; **RKNN_TENSOR_MEMORY_FLAGS_FROM_PHYS:** It represents rknn_tensor_mem created by physical address; The user does not need to pay attention to the flags.
priv_data	void*	private data.

rknn_input

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

Field	Type	Meaning
index	uint32_t	The index position of this input.
buf	void*	The pointer of the input data buffer.
size	uint32_t	The memory size of the input data buffer.
pass_through	uint8_t	When set to 1, buf will be directly set to the input node of the model without any pre-processing.
type	rknn_tensor_type	The type of input data.
fmt	rknn_tensor_format	The format of input data.

rknn_output

The structure rknn_output represents a data output of the model, used as a parameter to the rknn_outputs_get function. This structure will be assigned with data after calling rknn_outputs_get. The following table shows the definition of each member of rknn_output:

Field	Type	Meaning
want_float	uint8_t	Indicates whether the output data needs to be converted to float type, this field is set by the user.
is_prealloc	uint8_t	Indicates whether the buffer that stores the output data is pre-allocated, this field is set by the user.
index	uint32_t	The index position of this output, this field is set by the user, this field is returned by the interface.
buf	void*	The pointer pointing to the output data buffer, this field is returned by the interface.
size	uint32_t	Output data buffer size in byte, this field is returned by the interface.

rknn_init_extend

The structure rknn_init_extend represents the extended information when the model is initialized. **It is not available currently on RV1106/RV1103/RV1106B/RV1103B/RK2118**. The definition of the structure is shown in the following table:

Field	Type	Meaning
ctx	rknn_context	The initialized rknn_context object.
real_model_offset	int32_t	The real offset that is in .rknn model file. Only valid when using file path as initialized parameter or zero-copy model buffer.
real_model_size	uint32_t	The real size of rknn model file. Only valid when using file path as initialized parameter or zero-copy model buffer.
model_buffer_fd	int32_t	The fd represented by the model memory allocated by the NPU after initialization with the RKNN_FLAG_MODEL_BUFFER_ZERO_COPY flag.
model_buffer_flags	uint32_t	The memory flag represented by the model memory allocated by the NPU after initialization with the RKNN_FLAG_MODEL_BUFFER_ZERO_COPY flag.
reserved	uint8_t[]	The reserved data.

rknn_run_extend

The structure rknn_run_extend represents the extended information during model inference. **It is not available currently**. The definition of the structure is shown in the following table:

Field	Type	Meaning
frame_id	uint64_t	The index of frame of current inference.
non_block	int32_t	0 means blocking mode, 1 means non-blocking mode, non-blocking mode means that the rknn_run call returns immediately.
timeout_ms	int32_t	The timeout of inference in milliseconds.
fence_fd	int32_t	For the non-blocking inference (**Not Available**) .

rknn_output_extend

The structure rknn_output_extend means to obtain the extended information of the output. **It is not available currently**. The definition of the structure is shown in the following table:

Field	Type	Meaning
frame_id	int32_t	The frame index of the output result.

rknn_custom_string

The structure rknn_custom_string represents the custom string set by the user when converting the RKNN model. The definition of the structure is shown in the following table:

Field	Type	Meaning
string	char[]	User-defined string.

4.3 Description of Basic API

rknn_init

The function of the rknn_init initialization function is to create a rknn_context object, load the RKNN model, and perform specific initialization behaviors based on the flag and rknn_init_extend structures.

API	rknn_init
Description	Initialize rknn context.
Parameters	rknn_context *context: The pointer of rknn_context object.
	void *model: Binary data for the RKNN model or the path of RKNN model. When the parameter size is greater than 0, model represents binary data; when the parameter size is equal to 0, model represents the RKNN model path.
	uint32_t size: When model is stored in binary data, it indicates the size of the model. The size is 0 when model is given as the path.
	uint32_t flag: Initialization flag, the default initialization behavior needs to be set to 0.
	rknn_init_extend: The extended information during specific initialization. It is disabled at the moment, which indicates this must be passed by the NULL. If using share weight， it should pass the pointer of another rknn_context pointing to another model.
Return	int: Error code (See RKNN Error Code).

Sample Code:

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

Each initialization flag is explained as follows:

RKNN_FLAG_COLLECT_PERF_MASK: used to query the time of each layer of the network at runtime;

RKNN_FLAG_MEM_ALLOC_OUTSIDE: used to indicate that model input, output, weight, and intermediate tensor memory are all allocated by the user. It mainly has two functions:

1. All memory is allocated by the user, which facilitates the overall arrangement of the entire system memory.
2. Used for memory reuse, especially for situations like RV1106/RV1103/RV1106B/RV1103B/RK2118 where memory is extremely tight.

Assume that there are two models, model A and model B. These two models are designed to run in series, so the memory of the intermediate tensors of the two models can be reused. The sample code is as follows:

```
rknn_context ctx_a, ctx_b;

rknn_init(&ctx_a, model_path_a, 0, RKNN_FLAG_MEM_ALLOC_OUTSIDE, NULL);
rknn_query(ctx_a, RKNN_QUERY_MEM_SIZE, &mem_size_a, sizeof(mem_size_a));

rknn_init(&ctx_b, model_path_b, 0, RKNN_FLAG_MEM_ALLOC_OUTSIDE, NULL);
rknn_query(ctx_b, RKNN_QUERY_MEM_SIZE, &mem_size_b, sizeof(mem_size_b));
```

```
max_internal_size = MAX(mem_size_a.total_internal_size, mem_size_b.total_internal_size);
internal_mem_max = rknn_create_mem(ctx_a, max_internal_size);

internal_mem_a = rknn_create_mem_from_fd(ctx_a, internal_mem_max->fd,
    internal_mem_max->virt_addr, mem_size_a.total_internal_size, 0);
rknn_set_internal_mem(ctx_a, internal_mem_a);

internal_mem_b = rknn_create_mem_from_fd(ctx_b, internal_mem_max->fd,
    internal_mem_max->virt_addr, mem_size_b.total_internal_size, 0);
rknn_set_internal_mem(ctx_b, internal_mem_b);
```

RKNN_FLAG_SHARE_WEIGHT_MEM: Weight used to share another model's weight. Mainly used to simulate variable length model input (this function is replaced by the dynamic shape function after the RKNPU runtime library version is greater than or equal to 1.5.0). For example, for some speech models, the input length is variable, but because the NPU cannot support variable-length input, it is necessary to generate several RKNN models with different resolutions. Among them, only one RKNN model retains complete weights, and the other RKNN models do not have weights. When initializing an unweighted RKNN model, using this flag allows the current context to share the weights of the complete RKNN model. Assuming that two models with resolutions A and B are required, the usage process is as follows:

1. Use RKNN-Toolkit2 to generate a model with resolution A.
2. Use RKNN-Toolkit2 to generate a model with resolution B without weight. In rknn.config(), remove_weight should be set to True. The main purpose is to reduce the size of model B.
3. On the board, initialize model A normally.
4. Initialize model B through the flags of RKNN_FLAG_SHARE_WEIGHT_MEM.
5. Others should be used in the original way. The board reference code is as follows:

```
rknn_context ctx_a, ctx_b;
rknn_init(&ctx_a, model_path_a, 0, 0, NULL);

rknn_init_extend extend;
extend.ctx = ctx_a;
rknn_init(&ctx_b, model_path_b, 0, RKNN_FLAG_SHARE_WEIGHT_MEM, &extend);
```

RKNN_FLAG_COLLECT_MODEL_INFO_ONLY: used to initialize an empty context. It can only call the rknn_query interface to query the total size of the model weight memory and the total size of the intermediate tensor, but cannot perform inference;

RKNN_FLAG_INTERNAL_ALLOC_OUTSIDE: Indicates that the intermediate tensor of the model is allocated by the user. It is often used by the user to manage and reuse the intermediate tensor memory between multiple models;

RKNN_FLAG_EXECUTE_FALLBACK_PRIOR_DEVICE_GPU: Indicates that all layers not supported by the NPU are preferred to run on the GPU, but it is not guaranteed to run on the GPU. The actual running back-end device depends on the runtime support for the operator;

RKNN_FLAG_ENABLE_SRAM: Indicates that the intermediate tensor memory is allocated on SRAM as much as possible;

RKNN_FLAG_SHARE_SRAM: used for the current context to try to share the SRAM memory address space of another context. It is required that the RKNN_FLAG_ENABLE_SRAM flag must be enabled when the current context is initialized;

RKNN_FLAG_DISABLE_PROC_HIGH_PRIORITY: Indicates that the current context uses the default process priority. If this flag is not set, the nice value of the process is -19;

RKNN_FLAG_DISABLE_FLUSH_INPUT_MEM_CACHE: The runtime does not flush input tensor cache, the user must ensure that the input tensor has flushed the cache before calling rknn_run.

RKNN_FLAG_DISABLE_FLUSH_OUTPUT_MEM_CACHE: The runtime does not actively clear the output tensor cache. In this case, direct access to output_mem->virt_addr by the user can lead to cache consistency issues. If the user intends to use output_mem->virt_addr, it is necessary to utilize rknn_mem_sync(ctx, mem, RKNN_MEMORY_SYNC_FROM_DEVICE) to refresh the cache. This flag is typically used when the output data from the NPU is not accessed by the CPU, such as when the output data is accessed by the GPU or RGA.

RKNN_FLAG_MODEL_BUFFER_ZERO_COPY: It means that the incoming model buffer of the rknn_init interface is the memory allocated by the rknn_create_mem or rknn_create_mem2 interface, and the model buffer does not need to be copied once inside the runtime to reduce the memory occupation of the runtime, but the user needs to ensure that the model memory is valid before the context is destroyed. And free that memory after destroying the context. When you initialize a context, rknn_init the rknn_init_extend parameters of the interface, whose members real_model_offset, real_model_size, model_buffer_fd, and model_buffer_flags rknn_tensor_mem returned based on the rknn_create_mem2 interface.

RKNN_MEM_FLAG_ALLOC_NO_CONTEXT: When allocating memory using rknn_create_mem2 interfaces, after setting this flag, the ctx parameter is allowed to be 0 or NULL. Thus, the user can obtain the memory allocated by the NPU driver before initializing any of the contexts, and the returned memory structure needs to be released using the rknn_destroy_mem interface, and the released interface can use any context as a parameter. Example code such as:

```
 rknn_tensor_mem* model_mem = rknn_create_mem2(ctx, model_size,
RKNN_MEM_FLAG_ALLOC_NO_CONTEXT);
 memcpy(model_mem->virt_addr, model_data, model_size);
 rknn_init_extend init_ext;
 memset(&init_ext, 0, sizeof(rknn_init_extend));
 init_ext.real_model_offset  = 0;
 init_ext.real_model_size    = model_size;
 init_ext.model_buffer_fd    = model_mem->fd;
 init_ext.model_buffer_flags = model_mem->flags;
 int ret = rknn_init(&ctx, model_mem->virt_addr, model_size, RKNN_FLAG_MODEL_BUFFER_ZERO_COPY,
&init_ext);

 // do rknn inference...

 rknn_destroy_mem(ctx, model_mem);
 rknn_destroy(ctx);
```

rknn_set_core_mask

This function sets the specific cores running inside the NPU. For now, it only works at the RK3588 and RK3576. It will return the error code if it is set on platform with a single-core NPU architecture.

API	rknn_set_core_mask
Description	Set the cores for the NPU.
Parameters	rknn_context context: The object of rknn context.

API	rknn_set_core_mask
	rknn_core_mask core_mask: The specification of NPU core setting. It has the following choices: **RKNN_NPU_CORE_AUTO** : Referring to automatic mode, meaning that it will select the idle core inside the NPU; **RKNN_NPU_CORE_0** : Running on the NPU0 core; **RKNN_NPU_CORE_1**: Runing on the NPU1 core; **RKNN_NPU_CORE_2**: Runing on the NPU2 core; **RKNN_NPU_CORE_0_1**: Running on both NPU0 and NPU1 core simultaneously; **RKNN_NPU_CORE_0_1_2**: Running on both NPU0, NPU1 and NPU2 simultaneously. **RKNN_NPU_CORE_ALL**: Running on all of NPU cores depending on the platform;
Return	int: Error code (See RKNN Error Code).

```
rknn_context ctx;
rknn_core_mask core_mask = RKNN_NPU_CORE_0;
int ret = rknn_set_core_mask(ctx, core_mask);
```

For multi-cores mode (when enabling RKNN_NPU_CORE_0_1 and RKNN_NPU_CORE_0_1_2), the following ops have better acceleration : Conv, DepthwiseConvolution, Add, Concat, Relu, Clip, Relu6, ThresholdedRelu. Prelu, LeakyRelu. Other type of op will fallback to Core0 to continue running. In the future update, some ops like Pool or ConvTranpose will be supported.

rknn_set_batch_core_num

The rknn_set_batch_core_num function specifies the number of NPU cores of the multi-batch RKNN model (the model exported by setting rknn_batch_size greater than 1 during RKNN-Toolkit2 conversion). This function only supports the RK3588 and RK3576 platform.

API	rknn_set_batch_core_num
Description	Set the number of NPU cores for multi-batch RKNN model running.
Parameters	rknn_context context: The object of rknn context.
	int core_num: Specifies the number of cores to run.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_context ctx;
int ret = rknn_set_batch_core_num(ctx, 2);
```

rknn_dup_context

The rknn_dup_context creates a new context object referring to the same model. The new context is useful in the condition where the weight of the model need to be reused during executing the same model on the multi-thread (**Unavailable for RV1106/RV1103/RV1106B/RV1103B/RK2118**) .

API	rknn_dup_context
Description	Creates a new context for the same model, to reuse the weight of the model.
Parameters	rknn_context *context_in: The pointer of rknn_context object. After the function is called, the obejct of the input context will be initialized.
	rknn_context *context_out: The pointer of a rknn_context output object where information about this new created object is returned.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_context ctx_in;
rknn_context ctx_out;
int ret = rknn_dup_context(&ctx_in, &ctx_out);
```

rknn_destroy

This function is used to release the rknn_context and its related resources.

API	rknn_destroy
Description	Destroy the rknn_context object and its related resources.
Parameters	rknn_context context: The rknn_context object that is going to be destroyed.
Return	int: Error code (See RKNN Error Code).

Sample Code:

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

rknn_query

The rknn_query function can query the information of models and SDK, including model input and output information, layer-by-layer running time, total model inference time, SDK version, memory usage information, user-defined strings and other information.

API	rknn_query
Description	Query the information about the model and the SDK.
Parameters	rknn_context context: The object of rknn_contex.
	rknn_query_cmd : The query command.
	void* info: The structure object that stores the result of the query.
	uint32_t size: The size of the info structure object.
Return	int: Error code (See RKNN Error Code).

Currently, the SDK supports following query commands:

Query command	Return result structure	Function
RKNN_QUERY_IN_OUT_NUM	rknn_input_output_num	Query the number of input and output Tensors.
RKNN_QUERY_INPUT_ATTR	rknn_tensor_attr	Query the input Tensor attribute.
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	Query the output Tensor attribute.
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	Query the running time of each layer of the network. It only works when the flag of **RKNN_FLAG_COLLECT_PERF_MASK** is set via using the rknn_init.
RKNN_QUERY_PERF_RUN	rknn_perf_run	Query the total time of inference (excluding time in setting input/output) in microseconds .
RKNN_QUERY_SDK_VERSION	rknn_sdk_version	Query the SDK version.
RKNN_QUERY_MEM_SIZE	rknn_mem_size	Query the memory size allocated for the weights and internal tensors in the network.
RKNN_QUERY_CUSTOM_STRING	rknn_custom_string	Query the user-defined strings in the RKNN model.
RKNN_QUERY_NATIVE_INPUT_ATTR	rknn_tensor_attr	When using the zero-copy API interface, it queries the native input Tensor attribute, which is the model input attribute directly read by the NPU.
RKNN_QUERY_NATIVE_OUTPUT_ATTR	rknn_tensor_attr	When using the zero-copy API interface, query the native output Tensor attribute, which is the model output attribute directly from the NPU.
RKNN_QUERY_NATIVE_NC1HWC2_INPUT_ATTR	rknn_tensor_attr	When using the zero-copy API interface, it queries the native input Tensor attribute, which is the model input attribute directly read by the NPU.it is same with RKNN_QUERY_NATIVE_INPUT_ATTR.
RKNN_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR	rknn_tensor_attr	When using the zero-copy API interface, it queries the native output Tensor attribute, which is the model input attribute directly read by the NPU.it is same with RKNN_QUERY_NATIVE_OUTPUT_ATTR.

Query command	Return result structure	Function
RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR	rknn_tensor_attr	When using the zero-copy API interface, it queries the native input Tensor attribute, which is the model input attribute directly read by the NPU.it is same with RKNN_QUERY_NATIVE_INPUT_ATTR.
RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR	rknn_tensor_attr	When using the zero-copy API interface, it queries the native output NHWC Tensor attribute, which is the model input attribute directly read by the NPU.
RKNN_QUERY_DEVICE_MEM_INFO	rknn_tensor_mem	Query the memory properties of the model buffer.
RKNN_QUERY_INPUT_DYNAMIC_RANGE	rknn_input_range	When using the RKNN model that supports dynamic shapes, query the model to support input information such as the number of shapes, list, data layout and name corresponding to the shape.
RKNN_QUERY_CURRENT_INPUT_ATTR	rknn_tensor_attr	When using the RKNN model that supports dynamic shapes, query the input attributes used by the model for current inference.
RKNN_QUERY_CURRENT_OUTPUT_ATTR	rknn_tensor_attr	When using the RKNN model that supports dynamic shapes, query the output attributes used by the model for current inference.
RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR	rknn_tensor_attr	When using the RKNN model that supports dynamic shapes, query the NPU native input attributes used by the model's current inference.
RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR	rknn_tensor_attr	When using the RKNN model that supports dynamic shapes, query the NPU native output attributes used by the model's current inference.

Next we will explain each query command in detail.

1. **Query the SDK version**

   The RKNN_QUERY_SDK_VERSION command can be used to query the version information of the RKNN SDK. You need to create the rknn_sdk_version structure object first.

   Sample Code:

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

2. **Query the number of input and output Tensor**

   The RKNN_QUERY_IN_OUT_NUM command can be used to query the number of model input and output Tensor. You need to create the rknn_input_output_num structure object first.

   Sample Code:

```
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. **Query input Tensor attribute (for general API interface)**

The RKNN_QUERY_INPUT_ATTR command can be used to query the attribute of the model input Tensor. You need to create the rknn_tensor_attr structure object first (**Note: the tensor queried by RV1106/RV1103/RV1106B/RV1103B/RK2118 is the tensor originally entered as native**) .

Sample Code:

```
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));
}
```

4. **Query output Tensor attribute (for general API interface)**

   The RKNN_QUERY_OUTPUT_ATTR command can be used to query the attribute of the model output Tensor. You need to create the rknn_tensor_attr structure object first.

   Sample Code:

```
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));
}
```

5. **Query layer-by-layer inference time of model**

   After the rknn_run interface is called, the RKNN_QUERY_PERF_DETAIL command can be used to query the layer-by-layer inference time in microseconds. The premise of using this command is that the flag parameter of the rknn_init interface needs to include the RKNN_FLAG_COLLECT_PERF_MASK flag.

   Sample Code:

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, RKNN_FLAG_COLLECT_PERF_MASK, NULL);
...
ret = rknn_run(ctx,NULL);
...
rknn_perf_detail perf_detail;
ret = rknn_query(ctx, RKNN_QUERY_PERF_DETAIL, &perf_detail, sizeof(perf_detail));
```

6. **Query total inference time of model**

   After the rknn_run interface is called, the RKNN_QUERY_PERF_RUN command can be used to query the inference time of the model (not including setting input/output) in microseconds.

   Sample Code:

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
...
ret = rknn_run(ctx,NULL);
...
rknn_perf_run perf_run;
ret = rknn_query(ctx, RKNN_QUERY_PERF_RUN, &perf_run, sizeof(perf_run));
```

7. **Query the memory allocation of the model**

   After the rknn_init interface is called, when the user needs to allocate memory for network by themself, the RKNN_QUERY_MEM_SIZE command can be used to query the weights of the model and the internal memory (excluding input and output) in the network. The requirement of using this command is that the flag parameter of the rknn_init interface needs to enable the RKNN_FLAG_MEM_ALLOC_OUTSIDE flag.

   Sample Code:

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, RKNN_FLAG_MEM_ALLOC_OUTSIDE , NULL);
rknn_mem_size mem_size;
ret = rknn_query(ctx, RKNN_QUERY_MEM_SIZE, &mem_size, sizeof(mem_size));
```

8. **Query User-defined string in the model**

   After the rknn_init interface is called, if the user has added custom strings when generating the RKNN model, the RKNN_QUERY_CUSTOM_STRING command can be used to retrieve user-defined strings. For example, when converting the RKNN model, the user fills in the custom characters of "RGB" to identify that the RKNN model input is a three-channel image in RGB format instead of a three-channel image in BGR format. At runtime, the data is converted into an RGB image based on the queried "RGB" information.

   Sample Code:

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
rknn_custom_string custom_string;
ret = rknn_query(ctx, RKNN_QUERY_CUSTOM_STRING, &custom_string, sizeof(custom_string));
```

9. **Query native input tensor attribute (for zero-copy API interface)**

   The RKNN_QUERY_NATIVE_INPUT_ATTR command(the same to RKNN_QUERY_NATIVE_NC1HWC2_INPUT_ATTR) can be used to query the native attribute of the model input Tensor. User need to create the rknn_tensor_attr structure object first.

   Sample Code:

```
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_NATIVE_INPUT_ATTR, &(input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

10. **Query native output tensor attribute (for zero-copy API interface)**

    The RKNN_QUERY_NATIVE_OUTPUT_ATTR command(the same to RKNN_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR) can be used to query the native attribute of the model output Tensor. User need to create the rknn_tensor_attr structure object first.

    Sample Code:

```
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_NATIVE_OUTPUT_ATTR,
&(output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

11. **Query native output tensor attribute (for zero-copy API interface)**

    The RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR command can be used to query the NHWC attribute of the model input Tensor. User need to create the rknn_tensor_attr structure object first.

    Sample Code:

```
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_NATIVE_NHWC_INPUT_ATTR,
&(input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

12. **Query native output tensor attribute (for zero-copy API interface)**

    The RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR command can be used to query the NHWC attribute of the model output Tensor. User need to create the rknn_tensor_attr structure object first.

    Sample Code:

```
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_NATIVE_NHWC_OUTPUT_ATTR,
&(output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

13. **Query the dynamic input shape information supported by the RKNN model (Note: RV1106/RV1103/RV1106B/RV1103B/RK2118 does not support this interface)**

    After the rknn_init interface is called, pass in the RKNN_QUERY_INPUT_DYNAMIC_RANGE command to query the input shape information supported by the model, including the number of input shapes, the list of input shapes, the layout and name of the input shapes, and other information. The rknn_input_range structure object needs to be created first.

    The sample code is as follows:

```
rknn_input_range dyn_range[io_num.n_input];
memset(dyn_range, 0, io_num.n_input * sizeof(rknn_input_range));
for (uint32_t i = 0; i < io_num.n_input; i++) {
  dyn_range[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_INPUT_DYNAMIC_RANGE, &dyn_range[i],
sizeof(rknn_input_range));
}
```

14. **Query the memory information of model buffer(Note: RV1106/RV1103/RV1106B/RV1103B/RK2118 does not support this interface)**

    After the rknn_init interface is called, the RKNN_QUERY_DEVICE_MEM_INFO command can query the attributes of the model buffer created inside the runtime, including fd and physical address.

```
rknn_tensor_mem mem_info;
memset(&mem_info, 0, sizeof(mem_info));
ret = rknn_query(ctx, RKNN_QUERY_DEVICE_MEM_INFO, &mem_info, sizeof(mem_info));
```

15. **Query the input dynamic shape currently used by the RKNN model**

    After the rknn_set_input_shapes interface is called, pass in the RKNN_QUERY_CURRENT_INPUT_ATTR command to query the input attribute information currently used by the model. The rknn_tensor_attr structure needs to be created first (Note: RV1106/RV1103/RV1103B/RV1106B/RK2118 does not support this command).

    The sample code is as follows:

```
rknn_tensor_attr cur_input_attrs[io_num.n_input];
memset(cur_input_attrs, 0, io_num.n_input * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_input; i++) {
  cur_input_attrs[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_CURRENT_INPUT_ATTR, &(cur_input_attrs[i]),
sizeof(rknn_tensor_attr));
}
```

16. **Query the output dynamic shape currently used by the RKNN model**

    After the rknn_set_input_shapes interface is called, pass in the RKNN_QUERY_CURRENT_OUTPUT_ATTR command to query the output attribute information currently used by the model. The rknn_tensor_attr structure needs to be created first (Note: RV1106/RV1103/RV1103B/RV1106B/RK2118 does not support this command).

    The sample code is as follows:

```
rknn_tensor_attr cur_output_attrs[io_num.n_output];
memset(cur_output_attrs, 0, io_num.n_output * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_output; i++) {
  cur_output_attrs[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_CURRENT_OUTPUT_ATTR, &(cur_output_attrs[i]),
sizeof(rknn_tensor_attr));
}
```

17. **Query the native input dynamic shape currently used by the RKNN model**

    After the rknn_set_input_shapes interface is called, pass in the RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR command to query the native input attribute information currently used by the model. The rknn_tensor_attr structure needs to be created first (Note: RV1106/RV1103/RV1103B/RV1106B/RK2118 does not support this command).

    The sample code is as follows:

```
rknn_tensor_attr cur_input_attrs[io_num.n_input];
memset(cur_input_attrs, 0, io_num.n_input * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_input; i++) {
  cur_input_attrs[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR,
&(cur_input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

18. **Query the native output dynamic shape currently used by the RKNN model**

After the rknn_set_input_shapes interface is called, pass in the RKNN_QUERY_CURRENT_OUTPUT_ATTR command to query the native output attribute information currently used by the model. The rknn_tensor_attr structure needs to be created first (Note: RV1106/RV1103/RV1103B/RV1106B/RK2118 does not support this command).

The sample code is as follows:

```
rknn_tensor_attr cur_output_attrs[io_num.n_output];
memset(cur_output_attrs, 0, io_num.n_output * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_output; i++) {
  cur_output_attrs[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR, &(cur_output_attrs[i]),
sizeof(rknn_tensor_attr));
}
```

rknn_inputs_set

The input data of the model can be set by the rknn_inputs_set function. This function can support multiple inputs, each of which is a rknn_input structure object. The user needs to set these object field before passing in rknn_inputs_set function (**Note: unavailable on RV1106/RV1103/RV1103B/RV1106B/RK2118**).

API	**rknn_inputs_set**
Description	Set the model input data.
Parameter	rknn_context context: The object of rknn_contex.
	uint32_t n_inputs: Number of inputs.
	rknn_input inputs[]: Array of rknn_input.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
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].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;
inputs[0].pass_through = 0;

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

rknn_run

The rknn_run function will perform a model inference. The input data need to be configured by the *rknn_inputs_set* function or zero-copy interface before *rknn_run is* called.

API	**rknn_run**
Description	Perform a model inference.
Parameter	rknn_context context: The object of rknn_contex.
	rknn_run_extend* extend: Reserved for extension. It is not used currently and accepted NULL only.
Return	int: Error code (See RKNN Error Code).

Sample Code:

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

rknn_outputs_get

The rknn_outputs_get function can get the output data from the model. This function can get multiple output data, each of which is a rknn_output structure object that needs to be created and initialized in turn before the function is called (**Note: RV1106/RV1103/RV1106B/RV1103B/RK2118 unsupported**) .

There are two ways to store buffers for output data:

1. The user allocate and release buffers themselves. In this case, the rknn_output.is_prealloc needs to be set to 1, and the rknn_output.buf points to users' allocated buffer.
2. The other is allocated by rknn. At this time, the rknn_output .is_prealloc needs to be set to 0. After the function is executed, rknn_output.buf will be created and store the output data.

API	rknn_outputs_get
Description	Get model inference output data.
Parameter	rknn_context context: The object of rknn_context.
	uint32_t n_outputs: Number of output.
	rknn_output outputs[]: Array of rknn_output.
	rknn_run_extend* extend: Reserved for extension, currently not used yet. Accepting NULL only.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_output outputs[io_num.n_output];
memset(outputs, 0, sizeof(outputs));
for (int i = 0; i < io_num.n_output; i++) {
    outputs[i].index = i;
    outputs[i].is_prealloc = 0;
    outputs[i].want_float = 1;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

rknn_outputs_release

The rknn_outputs_release function will release the relevant resources of the rknn_output object.

API	rknn_outputs_release
Description	Release the rknn_output object.
Parameter	rknn_context context: rknn_context object.
	uint32_t n_outputs: Number of output.
	rknn_output outputs[]: The array of rknn_output to be released.
Return	int: Error code (See RKNN Error Code).

Sample Code:

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

rknn_create_mem_from_phys

When the **user** wants to allocate memory for NPU, the rknn_create_mem_from_phys function can create a rknn_tensor_mem structure and return its pointer. This function will pass the physical address, virtual address and size, and the information related to the external memory to the rknn_tensor_mem structure.

API	rknn_create_mem_from_phys
Description	Create rknn_tensor_mem structure and allocate memory through physical address.
Parameter	rknn_context context: rknn_context object.
	uint64_t phys_addr: The physical address of buffer.
	void *virt_addr: The virtual address of buffer, **it is the start address of the memory corresponding to fd**.
	uint32_t size: The size of buffer.
Return	rknn_tensor_mem*: The tensor memory information structure pointer.

Sample Code:

```
//suppose we have got buffer information as input_phys, input_virt and size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem_from_phys(ctx, input_phys, input_virt, size);
```

rknn_create_mem_from_fd

When the user wants to allocate the memory for NPU, the rknn_create_mem_from_fd creates a rknn_tensor_mem structure and return its pointer. This function filled with the file descriptor, logical address and size, and the information related to the external memory will be assigned to the rknn_tensor_mem structure.

API	rknn_create_mem_from_fd
Description Parameter	Create rknn_tensor_mem structure and allocate memory through file descriptor.
	rknn_context context: rknn_context object.
	int32_t fd: The file descriptor of buffer.
	void *virt_addr: The virtual address of buffer, which indicates the beginning of fd.
	uint32_t size: The size of buffer.
	int32_t offset: The offset corresponding for file descriptor and virtual address.
Return	rknn_tensor_mem*: The tensor memory information structure pointer.

Sample Code:

```
//suppose we have got buffer information as input_fd, input_virt and size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem_from_fd(ctx, input_fd, input_virt, size, 0);
```

rknn_create_mem

When the user wants to allocate memory which can be used directly in the NPU, the rknn_create_mem function can allocate the memory size specified by the user and return a rknn_tensor_mem structure.

API	rknn_create_mem
Description	Create rknn_tensor_mem structure internally and allocate memory during runtime.
Parameter	rknn_context context: rknn_context object.
	uint32_t size: The size of buffer.
Return	rknn_tensor_mem*: The tensor memory information structure pointer.

Sample Code:

```
//suppose we have got buffer size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem(ctx, size);
```

rknn_create_mem2

When the user wants to allocate memory which can be used directly in the NPU, the rknn_create_mem2 function can allocate the memory size and memory type specified by the user and return a rknn_tensor_mem structure.

API	**rknn_create_mem2**
Description	Create rknn_tensor_mem structure internally and allocate memory during runtime.
Parameter	rknn_context context: rknn_context object.
	uint64_t size: The size of buffer.
	uint64_t alloc_flags: Controls whether the allocated memory is cacheable. RKNN_FLAG_MEMORY_CACHEABLE： Create cacheable memory. RKNN_FLAG_MEMORY_NON_CACHEABLE: Create non-cacheable memory. RKNN_FLAG_MEMORY_FLAGS_DEFAULT： same with RKNN_FLAG_MEMORY_CACHEABLE
Return	rknn_tensor_mem*: The tensor memory information structure pointer.

The main difference between rknn_create_mem2 and rknn_create_mem is that rknn_create_mem2 brings an alloc_flags, which can specify whether the allocated memory is cacheable, while rknn_create_mem cannot be specified, and the default is cacheable.

Sample Code:

```
//suppose we have got buffer size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem2(ctx, size, RKNN_FLAG_MEMORY_NON_CACHEABLE);
```

rknn_destroy_mem

The rknn_destroy_mem function destroys the rknn_tensor_mem structure. However, the memory allocated by the user needs to be released manually.

API	**rknn_destroy_mem**
Description	Destroy rknn_tensor_mem structure.
Parameter	rknn_context context: rknn_context object.
	rknn_tensor_mem*: The tensor memory information structure pointer.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_tensor_mem* input_mems [1];
int ret = rknn_destroy_mem(ctx, input_mems[0]);
```

rknn_set_weight_mem

If the user has allocated memory for the network weights, after initializing the corresponding rknn_tensor_mem structure, the NPU can use the memory through the rknn_set_weight_mem function. This function must be called before calling rknn_run.

API	rknn_set_weight_mem
Description	Set up the rknn_tensor_mem structure containing weights memory information.
Parameter	rknn_context context: rknn_context object.
	rknn_tensor_mem*: The tensor memory information structure pointer.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_tensor_mem* weight_mems [1];
int ret = rknn_set_weight_mem(ctx, weight_mems[0]);
```

rknn_set_internal_mem

If the user has allocated memory for the internal tensor in network, after initializing the corresponding rknn_tensor_mem structure, the NPU can use the memory through the rknn_set_internal_mem function. This function must be called before calling rknn_run.

API	rknn_set_internal_mem
Description	Set up the rknn_tensor_mem structure containing internal tensor memory information in network.
Parameter	rknn_context context: rknn_context object.
	rknn_tensor_mem*: The pointer to the tensor memory information structure.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_tensor_mem* internal_tensor_mems [1];
int ret = rknn_set_internal_mem(ctx, internal_tensor_mems[0]);
```

rknn_set_io_mem

If the user has allocated memory for the input/output tensor in network, after initializing the corresponding rknn_tensor_mem structure, the NPU can use the memory through the rknn_set_io_mem function. This function must be called before calling rknn_run.

API	rknn_set_io_mem
Description	Set up the rknn_tensor_mem structure containing input/output tensor memory information in network.

API	rknn_set_io_mem
Parameter	rknn_context context: rknn_context object.
	rknn_tensor_mem*: The pointer to the tensor memory information structure .
	rknn_tensor_attr *: The attribute of input or output tensor buffer.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_tensor_attr output_attrs[1];
rknn_tensor_mem* output_mems[1];
ret = rknn_query(ctx, RKNN_QUERY_NATIVE_OUTPUT_ATTR, &(output_attrs[0]), sizeof(rknn_tensor_attr));
output_mems[0] = rknn_create_mem(ctx, output_attrs[0].size_with_stride);
rknn_set_io_mem(ctx, output_mems[0], &output_attrs[0]);
```

rknn_set_input_shape（deprecated）

This interface has been deprecated. Please use the rknn_set_input_shapes interface to bind input shapes. The current version is unavailable. If you want to continue using this interface, please use version 1.5.0 SDK and refer to the usage guide document of version 1.5.0.

rknn_set_input_shapes

For dynamic shape input RKNN models, the currently used input shape must be specified before inference. The interface passes in the number of inputs and the rknn_tensor_attr array, which contains each input shape and corresponding data layout information. The index, name, shape (dims) and memory layout information (fmt) of each rknn_tensor_attr structure object must be filled. Other members of the rknn_tensor_attr structure do not need to be set. Before using this interface, you can use the rknn_query function to query the number of input shapes and the list of dynamic shapes supported by the RKNN model, and the shape of the input data is required to be in the list of input shapes supported by the model. When running for the first time or switching a new input shape every time, you need to call this interface to set a new shape, otherwise, you don’t need to call this interface repeatedly.

API	**rknn_set_input_shapes**
Description	Set the input shape currently used by the model.
Parameter	rknn_context context: rknn_context object.
	uint32_t n_inputs: The number of input tensors。
	rknn_tensor_attr *: The attribute array pointer of tensor to pass all the input shape information. The user needs to set the index, name, dims, fmt, and n_dims members of each input attribute structure, and other members do not need to be set.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
for (int i = 0; i < io_num.n_input; i++) {
    for (int j = 0; j < input_attrs[i].n_dims; ++j) {
        input_attrs[i].dims[j] = dyn_range[i].dyn_range[0][j];
    }
}
ret = rknn_set_input_shapes(ctx, io_num.n_input, input_attrs);
if (ret < 0) {
    fprintf(stderr, "rknn_set_input_shapes error! ret=%d\n", ret);
    return -1;
}
```

rknn_mem_sync

The memory created by the rknn_create_mem function has the cacheable flag by default. For the memory created with the cacheable flag, when it is used by the CPU and NPU at the same time, the cache behavior will cause data consistency problems. This interface is used to synchronize a memory created with the cacheable flag to ensure that the data accessed by the CPU and NPU in this memory is consistent.

API	rknn_mem_sync
Description	Synchronize CPU cache and DDR data.
Parameter	rknn_context context: rknn_context object.
	rknn_tensor_mem* mem: tensor memory information structure pointer.
	rknn_mem_sync_mode mode: Indicates the mode for refreshing CPU cache and DDR data. **RKNN_MEMORY_SYNC_TO_DEVICE**: Indicates that the CPU cache data is synchronized to the DDR. This mode is usually used to write the data in the cache back to the DDR before the NPU accesses the same memory after the CPU writes to the memory. **RKNN_MEMORY_SYNC_FROM_DEVICE**: Indicates that DDR data is synchronized to the CPU cache. It is usually used after the NPU writes to the memory. Use this mode to make the cache data invalid the next time the CPU accesses the same memory, and the CPU re-reads the data from the DDR. **RKNN_MEMORY_SYNC_BIDIRECTIONAL**: Indicates that the CPU cache data is synchronized to the DDR and the CPU re-reads the data from the DDR.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
ret =rknn_mem_sync(ctx, &outputs[0].mem,
RKNN_MEMORY_SYNC_FROM_DEVICE);
if (ret < 0) {
    fprintf(stderr, " rknn_mem_sync error! ret=%d\n", ret);
    return -1;
}
```

4.4 Definition of Matrix Multiplication Data Structure

rknn_matmul_info

rknn_matmul_info indicates the specification information for performing matrix multiplication, which includes the size of matrix multiplication, data type and memory arrangement of input and output matrices. The definition of the structure is shown in the following table:

Field	Type	Meaning
M	int32_t	The number of rows of the A matrix.
K	int32_t	The number of columns of the A matrix.
N	int32_t	The number of columns of the B matrix.
type	rknn_matmul_type	Data type of input and output matrices: **RKNN_FLOAT16_MM_FLOAT16_TO_FLOAT32**: Indicates that matrices A and B are of type float16, and matrix C is of type float; **RKNN_INT8_MM_INT8_TO_INT32**: Indicates that matrices A and B are of type int8, and matrix C is of type int32; **RKNN_INT8_MM_INT8_TO_INT8**: Indicates that matrices A and B are of type int8, and matrix C is of type int8; **RKNN_FLOAT16_MM_FLOAT16_TO_FLOAT16：**： Indicates that matrices A and B are of type float16, and matrix C is of type float16; **RKNN_FLOAT16_MM_INT8_TO_FLOAT32**: Indicates that matrix A is of type float16, matrix B is type int8, and matrix C is type float; **RKNN_FLOAT16_MM_INT8_TO_FLOAT16**: Indicates that matrix A is of type float16, matrix B is type int8, and matrix C is type float16; **RKNN_FLOAT16_MM_INT4_TO_FLOAT32**: Indicates that matrix A is of type float16, matrix B is type int4, and matrix C is type float; **RKNN_FLOAT16_MM_INT4_TO_FLOAT16**: Indicates that matrix A is of type float16, matrix B is type int4, and matrix C is type float16; **RKNN_INT8_MM_INT8_TO_FLOAT32**: Indicates that matrices A and B are of type int8, and matrix C is of type float; **RKNN_INT4_MM_INT4_TO_INT16**: Indicates that matrices A and B are of type int4, and matrix C is of type int16; **RKNN_INT8_MM_INT4_TO_INT32**: Indicates that matrix A is of type int8, matrix B is type int4, and matrix C is type int32;

Field	Type	Meaning
B_layout	int16_t	The memory layout of the B matrix. 0: Indicates [K,N] shape arrangement. **RKNN_MM_LAYOUT_NORM**: Indicates that matrix B is arranged according to the original shape, i.e. the shape of KxN; **RKNN_MM_LAYOUT_NATIVE**: Indicates that matrix B is arranged in a high-performance shape **RKNN_MM_LAYOUT_TP_NORM**: Matrix B is arranged according to the shape after Transpose, i.e., the shape row of NxK
B_quant_type	int16_t	Specifies the type of quantization method for matrix B. **RKNN_QUANT_TYPE_PER_LAYER_SYM**: Matrix B is symmetrically quantized according to the Per-Layer method; **RKNN_QUANT_TYPE_PER_LAYER_ASYM**: Matrix B is asymmetrically quantized according to the Per-Layer method; **RKNN_QUANT_TYPE_PER_CHANNEL_SYM**: Matrix B is symmetrically quantized according to the Per-Channel method ; **RKNN_QUANT_TYPE_PER_CHANNEL_ASYM**: indicates that matrix B is asymmetrically quantized in the Per-Channel mode; **RKNN_QUANT_TYPE_PER_GROUP_SYM**: indicates that matrix B is symmetrically quantized in the Per-Group mode; **RKNN_QUANT_TYPE_PER_GROUP_ASYM**: Matrix B is asymmetrically quantified according to the Per-Group method
AC_layout	int16_t	Specifies how the data is arranged for Matrix A and Matrix C. **RKNN_MM_LAYOUT_NORM**: Indicates that matrices A and C are arranged in their original shapes; **RKNN_MM_LAYOUT_NATIVE**: Represents matrices A and C arranged in high-performance shapes
AC_quant_type	int16_t	Specifies the quantization type of matrices A and C. **RKNN_QUANT_TYPE_PER_LAYER_SYM**: Represents the symmetrical quantization of matrices A and C in a Per-Layer method; **RKNN_QUANT_TYPE_PER_LAYER_ASYM**: Represents the asymmetric quantization of matrices A and C in the Per-Layer method
iommu_domain_id	int32_t	The index of the IOMMU address space domain where the matrix context resides. The IOMMU address space corresponds to the context one-to-one, and the size of each IOMMU address space is 4 GB. This parameter is mainly used when the parameters of matrices A, B, and C are large and need to be switched to another domain after the NPU allocated more than 4 GB of memory in one domain.
group_size	int16_t	The number of elements in a group takes effect only when group quantization is enabled.
reserved	int8_t[]	Reservation field.

rknn_matmul_tensor_attr

rknn_matmul_tensor_attr represents the attribute of each matrix tensor, which includes the name, shape, size and data type of the matrix. The definition of the structure is shown in the following table:

Field	Type	Meaning
name	char[]	The name of the matrix.
n_dims	uint32_t	The number of dimensions of the matrix.
dims	uint32_t[]	The shape of the matrix.
size	uint32_t	the size of the matrix.
type	rknn_tensor_type	The data type of matrix

rknn_matmul_io_attr

rknn_matmul_io_attr represents the attributes of all input and output tensors of the matrix, which includes the attributes of matrices A, B, and C. The definition of the structure is shown in the following table:

Field	Type	Meaning
A	rknn_matmul_tensor_attr	The tensor attribute of matrix A.
B	rknn_matmul_tensor_attr	The tensor attribute of matrix B.
C	rknn_matmul_tensor_attr	The tensor attribute of matrix C.

rknn_quant_params

rknn_quant_params represents the quantization parameters of the matrix, including name and the pointers and lengths of the scale and zero_point arrays. name is used to identify the name of the matrix, which can be obtained from the rknn_matmul_io_attr structure obtained when initializing the matrix context. The structure definition is shown in the following table:

Field	Type	Meaning
name	char[]	The name of the matrix.
scale	float*	The scale array pointer of matrix.
scale_len	int32_t	The scale array length of the matrix
zp	int32_t*	The zero_point array pointer of matrix
zp_len	int32_t	The zero_point array length of matrix

rknn_matmul_shape

rknn_matmul_shape represents the M, K and N of matrix multiplication of a specific shape. When initializing the matrix multiplication context of a dynamic shape, you need to provide the number of shapes and use the rknn_matmul_shape structure array to represent all input shapes. The structure definition is shown in the following table:

Field	Type	Meaning
M	int32_t	The number of rows of the A matrix.
K	int32_t	The number of columns of matrix A.
N	int32_t	The number of columns of matrix C.

4.5 Description of Matrix Multiplication API

rknn_matmul_create

The function of this function is to complete the initialization of the matrix multiplication context based on the incoming matrix multiplication specification and other information, and return the shape, size, data type and other information of the input and output tensor. Here, the rknn_matmul_ctx pointer and rknn_context are the same data structure.

API	rknn_matmul_create
Description	Initializes the matrix multiplication context.
Parameters	rknn_matmul_ctx* ctx: Matrix multiplication context pointer.
	rknn_matmul_info* info: Pointer to the specification information structure of matrix multiplication.
	rknn_matmul_io_attr* io_attr: Pointer to the matrix multiplication input and output tensor attribute structure.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_matmul_info info;
memset(&info, 0, sizeof(rknn_matmul_info));
info.M          = 4;
info.K          = 64;
info.N          = 32;
info.type       = RKNN_INT8_MM_INT8_TO_INT32;
info.B_layout   = RKNN_MM_LAYOUT_NORM;
info.AC_layout  = RKNN_MM_LAYOUT_NORM;

rknn_matmul_io_attr io_attr;
memset(&io_attr, 0, sizeof(rknn_matmul_io_attr));

int ret = rknn_matmul_create(&ctx, &info, &io_attr);
if (ret < 0) {
  printf("rknn_matmul_create fail! ret=%d\n", ret);
  return -1;
}
```

rknn_matmul_set_io_mem

This function is used to set the input/output memory for the matrix multiplication operation. Before calling this function, first use the rknn_tensor_mem structure pointer created by the rknn_create_mem interface, and then pass it into the function with the rknn_matmul_tensor_attr structure pointer of the matrix A, B or C returned by the rknn_matmul_create function, and set the input and output memory to the matrix multiplication in context. Before calling this function, the data of matrix A and matrix B should be prepared according to the memory arrangement configured in rknn_matmul_info.

API	**rknn_matmul_set_io_mem**
Description	Sets the input/output memory for matrix multiplication.
Parameters	rknn_matmul_ctx ctx: Matrix multiplication context.
	rknn_tensor_mem* mem: Pointer to tensor memory information structure.
	rknn_matmul_tensor_attr* attr: Pointer to matrix multiplication input and output tensor attribute structure.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
// Create A
rknn_tensor_mem* A = rknn_create_mem(ctx, io_attr.A.size);
if (A == NULL) {
  printf("rknn_create_mem fail!\n");
  return -1;
}
memset(A->virt_addr, 1, A->size);
rknn_matmul_io_attr io_attr;
memset(&io_attr, 0, sizeof(rknn_matmul_io_attr));

int ret = rknn_matmul_create(&ctx, &info, &io_attr);
if (ret < 0) {
  printf("rknn_matmul_create fail! ret=%d\n", ret);
  return -1;
}
// Set A
ret = rknn_matmul_set_io_mem(ctx, A, &io_attr.A);
if (ret < 0) {
  printf("rknn_matmul_set_io_mem fail! ret=%d\n", ret);
  return -1;
}
```

rknn_matmul_set_core_mask

This function is used to set the available NPU cores for matrix multiplication (only RK3588 and RK3576 platform is supported). Before calling this function, you need to initialize the matrix multiplication context through the rknn_matmul_create function. The mask value that can be set by this function specifies the cores that need to be used to improve the performance and efficiency of matrix multiplication operations.

API	rknn_matmul_set_core_mask
Description	Sets the NPU core mask for matrix multiply operations.
Parameters	rknn_matmul_ctx ctx: Matrix multiplication context.
	rknn_core_mask core_mask: The NPU core mask value of the matrix multiplication operation, which is used to specify the available NPU cores. Each bit of the mask represents a core. If the corresponding bit is 1, it means that the core is available; otherwise, it means that the core is unavailable (see rknn_set_core_mask API parameters for detailed mask description).
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_matmul_set_core_mask(ctx, RKNN_NPU_CORE_AUTO);
```

rknn_matmul_set_quant_params

rknn_matmul_set_quant_params is used to set the quantization parameters of each matrix, and supports two methods of quantization parameter settings: Per-Channel quantization and Per-Layer quantization. When using Per-Channel quantization, the length of the scale and zp arrays in rknn_quant_params is equal to N. When using Per-Layer quantization, the scale and zp arrays in rknn_quant_params have length 1. Call this interface before rknn_matmul_run to set the quantization parameters of all matrices. If this interface is not called, the default quantization method is Per-Layer quantization, scale=1.0, zero_point=0.

API	rknn_matmul_set_quant_params
Description	Set the quantization parameters of the matrix.
Parameters	rknn_matmul_ctx ctx: matrix multiplication context.
	rknn_quant_params* params: quantization parameter information of the matrix.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_quant_params params_a;
memcpy(params_a.name, io_attr.A.name, RKNN_MAX_NAME_LEN);
params_a.scale_len = 1;
params_a.scale = (float *)malloc(params_a.scale_len * sizeof(float));
params_a.scale[0] = 0.2;
params_a.zp_len = 1;
params_a.zp = (int32_t *)malloc(params_a.zp_len * sizeof(int32_t));
params_a.zp[0] = 0;
rknn_matmul_set_quant_params(ctx, &params_a);
```

rknn_matmul_get_quant_params

rknn_matmul_get_quant_params is used when the rknn_matmul_type type is equal to RKNN_INT8_MM_INT8_TO_INT32 and the Per-Channel quantization mode is used to obtain the scale values of all channels of matrix B after scale normalization. The obtained scale value is multiplied by the original scale value of A to obtain the scale value of C. It can be used to approximately calculate the scale of C when the matrix C does not have a real scale.

API	rknn_matmul_get_quant_params
Description	Get the quantization parameters of matrix B.
Parameters	rknn_matmul_ctx ctx: matrix multiplication context.
	rknn_quant_params* params: quantization parameter information of matrix B.
	float* scale: scale pointer of matrix B.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
float b_scale;
rknn_matmul_get_quant_params(ctx, &params_b, &b_scale);
```

rknn_matmul_create_dynamic_shape

rknn_matmul_create_dynamic_shape is used to create a dynamic shape matrix multiplication context. This interface needs to pass in the rknn_matmul_info structure, the number of shapes and the corresponding shape array. The shape array will record multiple M, K and N values. After successful initialization, you will get the array of rknn_matmul_io_attr, which contains the shape, size and data type of all input and output matrices. Currently, it only supports setting multiple different M, K and N.

API	rknn_matmul_create_dynamic_shape
Description	Initialize the context for dynamic shape matrix multiplication.
Parameters	rknn_matmul_ctx *ctx: matrix multiplication context pointer.
	rknn_matmul_info* info: matrix multiplication specification information structure pointer. Among them, M, K and N do not need to be set.
	int shape_num: The number of shapes supported by the matrix context.
	rknn_matmul_shape dynamic_shapes[]: Shape array supported by matrix context.
	rknn_matmul_io_attr io_attrs[]: matrix multiplication input and output tensor attribute structure array.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
const int          shape_num = 2;
rknn_matmul_shape shapes[shape_num];
for (int i = 0; i < shape_num; ++i) {
  shapes[i].M = i+1;
  shapes[i].K = 64;
  shapes[i].N = 32;
}
rknn_matmul_io_attr io_attr[shape_num];
memset(io_attr, 0, sizeof(rknn_matmul_io_attr) * shape_num);

int ret = rknn_matmul_create_dynamic_shape(&ctx, &info, shape_num, shapes, io_attr);
if (ret < 0) {
  fprintf(stderr, " rknn_matmul_create_dynamic_shape fail! ret=%d\n", ret);
  return -1;
}
```

rknn_matmul_create_dyn_shape(deprecated)

This interface has been deprecated and the rknn_matmul_create_dynamic_shape interface has been used instead.

rknn_matmul_set_dynamic_shape

rknn_matmul_set_dynamic_shape is used to specify a certain shape used in matrix multiplication. After creating the matrix multiplication context of the dynamic shape, select one of the rknn_matmul_shape structures as the input parameter, and call this interface to set the shape used for the operation.

API	**rknn_matmul_set_dynamic_shape**
Description	Set matrix multiplication shape.
Parameters	rknn_matmul_ctx ctx: matrix multiplication context.
	rknn_matmul_shape* shape: Specifies the shape used for matrix multiplication.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
ret = rknn_matmul_set_dynamic_shape(ctx, &shapes[0]);
if (ret != 0) {
  fprintf(stderr, "rknn_matmul_set_dynamic_shapes fail!\n");
  return -1;
}
```

rknn_B_normal_layout_to_native_layout

rknn_B_normal_layout_to_native_layout is used to convert data arranged in the original shape of matrix B (KxN) into data arranged in a high-performance data arrangement.

API	rknn_B_normal_layout_to_native_layout
Description	Convert the data arrangement of matrix B from the original shape to a high-performance shape.
Parameters	void* B_input: Matrix B data pointer of original shape.
	void* B_output: High-performance shape matrix B data pointer.
	int K: The number of rows of matrix B.
	int N: The number of columns of matrix B.
	int subN: equal to B.dims[2] in the rknn_matmul_io_attr structure.
	int subK: equal to B.dims[3] in the rknn_matmul_io_attr structure.
	rknn_matmul_info* info: The information struct pointer for matrix multiplication.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
int32_t subN = io_attr.B.dims[2];
int32_t subK = io_attr.B.dims[3];
rknn_B_normal_layout_to_native_layout(B_Matrix, B->virt_addr, K, N, subN, subK, &info);
```

rknn_matmul_run

This function is used to run a matrix multiplication operation and save the result in the output matrix C. Before calling this function, the input matrices A and B need to prepare data first, and set them to the input buffer through the rknn_matmul_set_io_mem function. The output matrix C needs to be set to the output buffer through the rknn_matmul_set_io_mem function, and the tensor attribute of the output matrix is obtained through the rknn_matmul_create function.

API	rknn_matmul_run
Description	Perform a matrix multiplication operation.
Parameters	rknn_matmul_ctx ctx: Matrix multiplication context.
Return	int: Error code (See RKNN Error Code).

Sample Code:

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

rknn_matmul_destroy

This function is used to destroy the matrix multiplication operation context and release related resources. After using the matrix multiplication pointer created by the rknn_matmul_create function, you need to call this function to destroy it.

API	rknn_matmul_destroy
Description	Destroys the matrix multiply operation context.
Parameters	rknn_matmul_ctx ctx: Matrix multiplication context.
Return	int: Error code (See RKNN Error Code).

Sample Code:

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

4.6 Definition of Custom Operator Data Structure

rknn_gpu_op_context

rknn_gpu_op_context represents the context information of the custom operator run by the specified GPU. The definition of the structure is shown in the following table:

Field	Type	Meaning
cl_context	void*	OpenCL's cl_context object, please force type conversion to cl_context when using it.
cl_command_queue	void*	OpenCL's cl_command_queue object, please force type conversion to cl_command_queue when using it.
cl_kernel	void*	OpenCL's cl_kernel object, please force type conversion to cl_kernel when using it.

rknn_custom_op_context

rknn_custom_op_context represents the context information of the custom operator. The definition of the structure is shown in the following table:

Field	Type	Meaning
target	rknn_target_type	Backend device that executes custom operators: RKNN_TARGET_TYPE_CPU: CPU RKNN_TARGET_TYPE_CPU: GPU
internal_ctx	rknn_custom_op_interal_context	Private context inside the operator.
gpu_ctx	rknn_gpu_op_context	Contains the OpenCL context information of the custom operator. When the execution backend device is a GPU, OpenCL objects such as cl_context are obtained from this structure in the callback function.
priv_data	void*	Data pointers left to developers to manage.

rknn_custom_op_tensor

rknn_custom_op_tensor represents the input/output tensor information of the custom operator. The definition of the structure is shown in the following table:

Field	Type	Meaning
attr	rknn_tensor_attr	Contains the name, shape, size and other information of the tensor.
mem	rknn_tensor_mem	Contains tensor's memory address, fd, valid data offset and other information.

rknn_custom_op_attr

rknn_custom_op_attr represents the parameters or attribute information of the custom operator. The definition of the structure is shown in the following table:

Field	Type	Meaning
name	char[]	The parameter name of the custom operator.
dtype	rknn_tensor_type	The data type of each element.
n_elems	uint32_t	Number of elements.
data	void*	The virtual address of the parameter data memory segment.

rknn_custom_op

rknn_custom_op represents the registration information of the custom operator. The definition of the structure is shown in the following table:

Field	Type	Meaning
version	uint32_t	Custom operator version number.
target	rknn_target_type	Custom operator execution backend type.
op_type	char[]	Custom operator type.
cl_kernel_name	char[]	OpenCL kernel function name.
cl_kernel_source	char*	OpenCL resource name. When cl_source_size is equal to 0, it represents the absolute path of the file; when cl_source_size is greater than 0, it represents the string of kernel function code.
cl_source_size	uint64_t	When cl_kernel_source is a string, it indicates the length of the string; when cl_kernel_source is a file path, it is set to 0.
cl_build_options	char[]	Compilation options for the OpenCL kernel.
init	int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs);	Custom operator initialization callback function pointer. Called once during registration. It can be set to NULL when not needed.
prepare	int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs);	Preprocessing callback function pointer. Called once during rknn_run. It can be set to NULL when not needed.
compute	int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs);	The callback function pointer of the custom operator function. Called once during rknn_run. **Cannot be set to NULL.**
compute_native	int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs);	High-performance computing callback function pointer. The difference between it and the compute callback function is the format of the input and output tensor. Not supported yet, currently set to NULL.

Field	Type	Meaning
destroy	int (*)(rknn_custom_op_context* op_ctx);	The callback function pointer for destroying resources. Called once during rknn_destroy.

4.7 Description of Custom Operator API

rknn_register_custom_ops

After the context is successfully initialized, this function is used to register several custom operator information in the context, including custom operator types, running backend types, OpenCL kernel information, and callback function pointers. After successful registration, during the inference phase, the rknn_run interface will call the callback function implemented by the developer.

API	rknn_register_custom_ops
Description	Register several custom operators into the context.
Parameters	rknn_context *context: the rknn_context pointer. Before the function is called, the context must have been successfully initialized.
	rknn_custom_op* op: the custom operator information array, each element of the array is a rknn_custom_op structure object.
	uint32_t custom_op_num: custom operator information array length.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
// CPU operators
rknn_custom_op user_op[2];
memset(user_op, 0, 2 * sizeof(rknn_custom_op));
strncpy(user_op[0].op_type, "cstSoftmax", RKNN_MAX_NAME_LEN - 1);
user_op[0].version = 1;
user_op[0].target  = RKNN_TARGET_TYPE_CPU;
user_op[0].init    = custom_op_init_callback;
user_op[0].compute = compute_custom_softmax_float32;
user_op[0].destroy = custom_op_destroy_callback;

strncpy(user_op[1].op_type, "ArgMax", RKNN_MAX_NAME_LEN - 1);
user_op[1].version = 1;
user_op[1].target  = RKNN_TARGET_TYPE_CPU;
user_op[1].init    = custom_op_init_callback;
user_op[1].compute = compute_custom_argmax_float32;
user_op[1].destroy = custom_op_destroy_callback;

ret = rknn_register_custom_ops(ctx, user_op, 2);
if (ret < 0) {
  printf("rknn_register_custom_ops fail! ret = %d\n", ret);
  return -1;
}
```

rknn_custom_op_get_op_attr

This function is used to obtain the parameter information of the custom operator in the callback function of the custom operator, such as the axis parameter of the Softmax operator. It passes in the field name of the custom operator parameter and a rknn_custom_op_attr structure pointer. After calling this interface, the parameter value will be stored in the data member in the rknn_custom_op_attr structure. The developer forces the pointer according to the dtype member in the returned structure. Convert it into the first address of the array of a specific data type in C language, and then read out the complete parameter value according to the number of elements.

API	rknn_custom_op_get_op_attr
Description	Get the parameters or properties of a custom operator.
Parameters	rknn_custom_op_context* op_ctx: custom operator context pointer.
	const char* attr_name: field name of custom operator parameter.
	rknn_custom_op_attr* op_attr: a structure representing the custom operator parameter value.
Return	No value.

Sample Code:

```
rknn_custom_op_attr op_attr;
rknn_custom_op_get_op_attr(op_ctx, "axis", &op_attr);
if (op_attr.n_elems == 1 && op_attr.dtype == RKNN_TENSOR_INT64) {
    axis = ((int64_t*)op_attr.data)[0];
}
…
```

5. RKNN Error Code

The return code of the RKNN API function is defined as shown in the following table.

Error Code	Message
RKNN_SUCC(0)	Execution is successful.
RKNN_ERR_FAIL (-1)	Execution error.
RKNN_ERR_TIMEOUT (-2)	Execution timeout.
RKNN_ERR_DEVICE_UNAVAILABLE (-3)	NPU device is unavailable.
RKNN_ERR_MALLOC_FAIL (-4)	Memory allocation is failed.
RKNN_ERR_PARAM_INVALID (-5)	Parameter error.
RKNN_ERR_MODEL_INVALID (-6)	RKNN model is invalid.
RKNN_ERR_CTX_INVALID (-7)	rknn_context is invalid.
RKNN_ERR_INPUT_INVALID (-8)	rknn_input object is invalid.
RKNN_ERR_OUTPUT_INVALID (-9)	rknn_output object is invalid.
RKNN_ERR_DEVICE_UNMATCH (-10)	Version does not match.
RKNN_ERR_INCOMPATILE_OPTIMIZATION_LEVEL_VERSION (-12)	This RKNN model use optimization level mode, but not compatible with current driver.
RKNN_ERR_TARGET_PLATFORM_UNMATCH (-13)	This RKNN model doesn’t compatible with current platform.