RKNN3 TOOLKIT LITE User Guide

Document ID: RK-KF-YF-432

Version: V1.0.0

Date: 2025-01-05

Confidentiality: □Top Secret □Secret □Internal ■Public

DISCLAIMER

THIS DOCUMENT IS PROVIDED “AS IS”. ROCKCHIP ELECTRONICS CO., LTD.(“ROCKCHIP”)DOES NOT PROVIDE ANY WARRANTY OF ANY KIND, EXPRESSED, IMPLIED OR OTHERWISE, WITH RESPECT TO THE ACCURACY, RELIABILITY, COMPLETENESS,MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE OR NON-INFRINGEMENT OF ANY REPRESENTATION, INFORMATION AND CONTENT IN THIS DOCUMENT. THIS DOCUMENT IS FOR REFERENCE ONLY. THIS DOCUMENT MAY BE UPDATED OR CHANGED WITHOUT ANY NOTICE AT ANY TIME DUE TO THE UPGRADES OF THE PRODUCT OR ANY OTHER REASONS.

Trademark Statement

"Rockchip", "瑞芯微", "瑞芯" shall be Rockchip’s registered trademarks and owned by Rockchip. All the other trademarks or registered trademarks mentioned in this document shall be owned by their respective owners.

All rights reserved. ©2025. Rockchip Electronics Co., Ltd.

Beyond the scope of fair use, neither any entity nor individual shall extract, copy, or distribute this document in any form in whole or in part without the written approval of Rockchip.

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 document is the user guide for RKNN3 TOOLKIT LITE.

Target Audience

This document is primarily intended for the following engineers:

- Technical Support Engineers
- Software Development Engineers

Revision History

Version	Author	Date	Description	Approved by
V0.5.0	HPC Team	2025-12-05	Initial version	Vincent
V1.0.0	HPC Team	2025-01-05	Improve API documentation	Vincent

Table of Contents

1 Overview

RKNN3-Toolkit Lite provides Python interfaces for coprocessor model inference, making it easier for users to develop AI large model applications or validate models using Python.

1.1 Hardware Platforms

This document is applicable to the following hardware platforms:

- RK1820
- RK1828

1.2 Applicable system

- Debian: 10 (aarch64)
- Debian: 11 (aarch64)
- Debian: 12 (aarch64)

2 Development environment build

2.1 Requirements/Dependencies

RKNN3-Toolkit Lite requires the following environment:

Operating System	**Debian 10 / 11 / 12 (aarch64)**
Python Version	3.9 / 3.10 / 3.11 / 3.12
Python Libraries	'transformers', 'numpy', 'jinja2'

2.2 Installation

Use the `pip3 install` command to install RKNN3-Toolkit Lite. The installation steps are as follows:

If `python3/pip3` is not installed, use the following command to install it:

```
sudo apt-get update
sudo apt-get install -y python3 python3-dev python3-pip gcc
```

Note:

- Some dependent modules require source code compilation. In this case, `python3-dev` and `gcc` will be used. This step will install both packages simultaneously to avoid compilation failures during dependency installation.

Install the dependencies `opencv-python` and `numpy`:

```
sudo apt-get install -y python3-opencv
sudo apt-get install -y python3-numpy
```

or

```
pip3 install opencv-python
pip3 install numpy
```

Note:

- RKNN3-Toolkit Lite itself does not depend on `opencv-python`, but this module is needed for image processing in the examples.
- Installing `numpy` via `pip3` on Debian 10 firmware may fail. It is recommended to use the method described above.

Install RKNN3-Toolkit Lite:

```
# Python 3.9
pip3 install rknn3_toolkit_lite-x.y.z-cp39-cp39-linux_aarch64.whl
# Python 3.10
pip3 install rknn3_toolkit_lite-x.y.z-cp310-cp310-linux_aarch64.whl
# Python 3.11
pip3 install rknn3_toolkit_lite-x.y.z-cp311-cp311-linux_aarch64.whl
# Python 3.12
pip3 install rknn3_toolkit_lite-x.y.z-cp312-cp312-linux_aarch64.whl
```

3 Usage

RKNN3-Toolkit Lite is mainly used for deploying RKNN models on Rockchip NPU. Before using RKNN3-Toolkit Lite, users need to convert the models exported from various deep learning frameworks into RKNN models using RKNN3-Toolkit.

3.1 Basic Usage Process

The basic process for deploying RKNN models using RKNN3-Toolkit Lite is shown in the following figure:

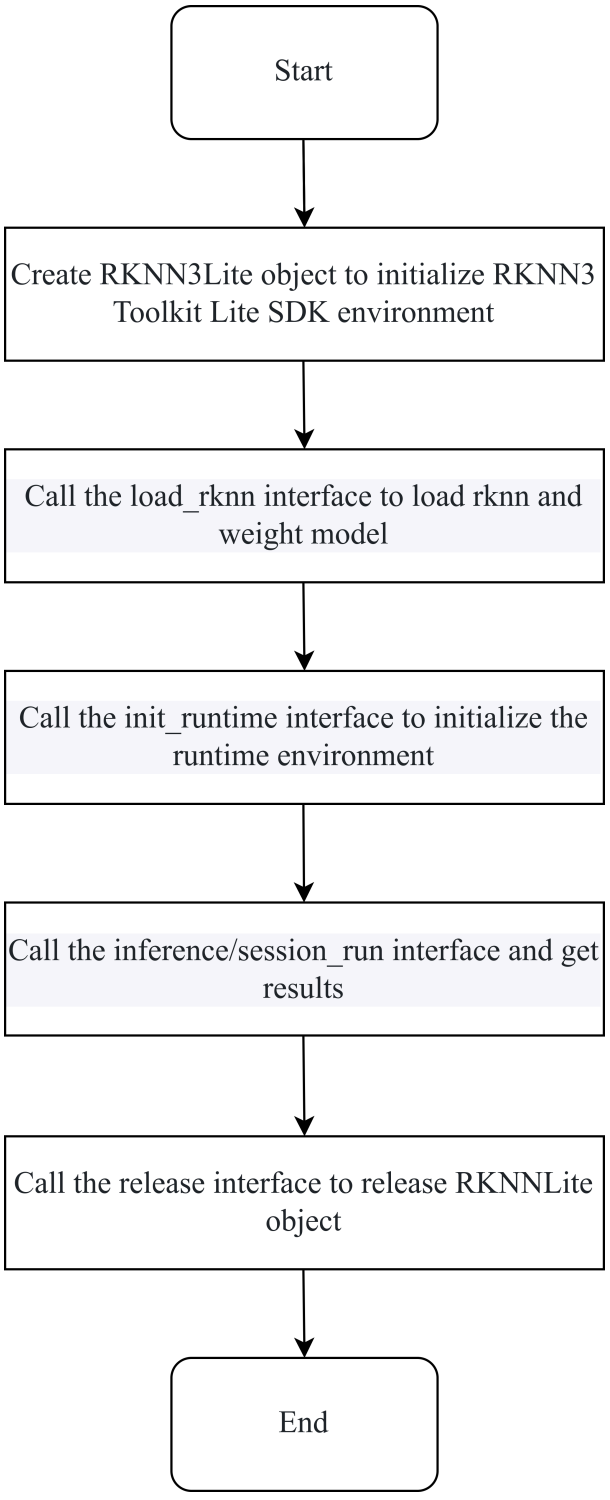

图3-1 Usage flow of RKNN3 Toolkit Lite

Note:

1. Before calling the `inference` function, input data must be prepared and preprocessed. For pure LLMs, you need to obtain `input_ids`, and for multimodal models, you need to obtain `embeds`. Then, set the appropriate parameters for the `inference` function.
2. After calling the `inference` function, the inference result usually requires further processing to complete the application.

3.2 Example

An example application based on RKNN3-Toolkit Lite is provided in the `SDK/rknn3-toolkit-lite/examples` directory. This application loads the Qwen2.5VL-3B RKNN model and Tokenizer, performs inference on input images and prompts, and outputs the analysis results.

Run the example by following these steps:

1. Prepare a development board with RKNN3-Toolkit Lite installed.
2. Push the `examples` folder to the board.
3. On the board, navigate to `examples/Qwen2.5VL` and run the example:

```
python test.py \
     --rknn_vision_path XXX/Qwen2.5-VL-3B-vision.rknn \
     --rknn_llm_path XXX/Qwen2.5-VL-3B-llm.rknn \
     --tokenizer_path XXX/Tokenizer_Path \
     --embed_path XXX/Qwen2.5-VL-3B-llm.embed.bin
```

Note: Please ensure that the specified RKNN model path matches the corresponding weight file path, as discrepancies may lead to inference failure.

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

```
--> Running vision model
This image depicts a space scene. A large planet, possibly the Moon or Mars, is seen in the
background, with craters and mountains. The sky is dotted with stars.

In the foreground, an astronaut in a white suit is lying on the ground resting, holding a
drink (perhaps beer), with a small box nearby. Other objects like a ladder, a stone, and
various unidentified objects are scattered on the ground.

The entire scene feels like a space exploration or lunar mission, but with a relaxed
atmosphere as the astronaut appears to be taking a break and enjoying some relaxation time.

--------------------Finished--------------------
```

Note: Due to differences in model versions, input images, or runtime environments, the actual output may slightly differ from the example above.

4 RKNN3 Toolkit Lite API Detailed Description

This section details the usage of APIs provided by RKNN3-Toolkit Lite.

4.1 RKNNLite Initialization and Object Release

When using RKNN3-Toolkit Lite, you first create an `RKNN3Lite` object by calling the `RKNN3Lite()` method, and after usage, release the resources by calling `release()`. For LLM models, the parameter `llm_mode=True` should be set during initialization.
Furthermore, when initializing the RKNN3Lite object, the log output behavior can be controlled through the verbose and verbose_file parameters:

- verbose=True indicates that detailed logs are printed in the terminal;
- If verbose_file (e.g., './inference.log') is also specified, log information will also be written to that file.

Example:

```
# Output detailed logs to the screen and write to inference.log file
rknn_lite = RKNN3Lite(verbose=True, verbose_file='./inference.log')
# Only output detailed logs to the screen
rknn_lite = RKNN3Lite(verbose=True)
# Initialize LLM model
rknn_lite = RKNN3Lite(llm_mode=True)
...
rknn_lite.release()
```

4.2 Loading RKNN Model

API	**load_rknn**
Description	Loads an RKNN model
Parameters	`model_path`: Path to RKNN model
	`weight_path`: Path to RKNN weight file
Return Value	0: Success; -1: Failure

Example:

```
# Load the mobilenetv2-12.rknn model from the current directory
ret = rknn_lite.load_rknn(model_path='mobilenetv2-12.rknn', weight_path='mobilenetv2-
12.weight')
```

4.3 Initializing the Runtime Environment

Before performing model inference, initialize the runtime environment.

API	**init_runtime**
Description	Initializes the runtime environment
Parameters	`target`: Specifies the coprocessor; currently, RK1820/RK1828 are supported.
	`core_mask`: NPU core mask. The co-processor contains 8 NPU cores, and this parameter specifies the enabled cores using a bitmask. For example, 0x0f (binary 0b00001111) indicates the use of the first 4 NPU cores. Note that this parameter must be consistent with the core_mask used when converting the RKNN model; otherwise, an error will occur.
	`llm_args`: A dictionary of configuration parameters for the LLM model.
	`llm_callback`: A callback function for the LLM model, which can be used for streaming inference.
	`llm_embed_path`: The embedding vocabulary path, required to initialize the LLMGetEmbedCallback.
	`device_id`: Device ID, used to distinguish multiple devices. If only one device is connected, it can be set to None
Return Value	0: Success; -1: Failure

Notes:

- The configurable parameters for `llm_args` currently include and their meanings are as follows:
  - `top_k`: The number of highest probability vocabulary to consider during sampling, integer type, default value depends on the model.
  - `top_p`: Nucleus sampling probability threshold, float type, range [0, 1], used to control sampling diversity.
  - `temperature`: Temperature parameter controlling randomness, float type, usually greater than 0, smaller values make generation more deterministic.
  - `repeat_penalty`: Repeat penalty coefficient, float type, used to reduce the penalty for repeated generation.
  - `frequency_penalty`: Frequency penalty, float type, penalty based on vocabulary occurrence frequency.
  - `presence_penalty`: Presence penalty, float type, penalty based on whether vocabulary has appeared.
  - `vocab_size`: Vocabulary size, integer type, represents the total size of the model's vocabulary.
  - `special_bos_id`: ID of the Beginning of Sequence token, integer type.

    - `special_eos_id`: ID of the End of Sequence token, integer type.
    - `linefeed_id`: ID of the linefeed token, integer type.
    - `skip_special_token`: Whether to skip special tokens, boolean type, True means skip.
    - `ignore_eos_token`: Whether to ignore the end of sequence token, boolean type, True means ignore.
    - `keep_history`: Whether to keep conversation history, boolean type, True means keep.
    - `max_new_tokens`: Maximum number of new tokens to generate, integer type, controls the maximum length of generated text.
    - `logits_name`: Name of the logits output, string type, used to specify the output layer name.
    - `max_context_len`: Maximum context length, integer type, represents the maximum context length the model can handle.
- The `llm_callback` mainly includes 5 callback functions for handling different stages of LLM inference:
    - `LLMResultCallback`: Callback function for handling inference results, used to receive and process generated text results.
    - `LLMSamplingCallback`: Sampling callback function, used for custom sampling strategies or processing the sampling process.
    - `LLMTokenizerCallback`: Tokenizer callback function, used for processing input text tokenization and encoding.
    - `LLMGetEmbedCallback`: Callback function for getting embeddings, used to get input embedding vectors.
    - `LLMGetLastHiddenLayerCallback`: Callback function for getting the last hidden layer, used to get the model's last hidden layer output.
      Note: The specific implementation of these callback functions and their parameters correspond one-to-one with the functions and parameters provided in the API manual. Users can customize implementations as needed.
- `device_id` can be used to obtain the device IDs of all coprocessors through the `get_devices_id` function.

Example for Traditional Model:

```
...
# Get device ID
device_id = rknn_lite.get_devices_id()
# Initialize runtime environment
ret = rknn_lite.init_runtime(target='rk1820', core_mask=0x01, device_id = device_id[0])
if ret != 0:
    print('Init runtime environment failed')
    exit(ret)
```

Example for LLM:

```
ARGS = [{"max_new_tokens":256, "top_k":1, "repeat_penalty":1.1, "special_eos_id": 151645,
...}]
callback = RKLLMCallback()
...

ret = rknn_lite.init_runtime(target='rk1820', core_mask=0xff, llm_args=ARGS,
llm_callback=callback)
if ret != 0:
    print('Init runtime environment failed')
    exit(ret)
```

4.4 Get Device IDs

This interface retrieves the device IDs of all connected coprocessors in the current system.

API	**get_devices_id**
Description	Retrieves a list of IDs for all connected devices.
Return Value	A list of device IDs, of type List[str]

Example:

```
# Get device id
rknn_lite = RKNN3Lite()
device_id = rknn_lite.get_devices_id()
```

Note: The above interface is suitable for multi-device deployment scenarios. If only a single device is used, the first element of the list can be retrieved directly or None can be passed in (as supported by init_runtime).

4.5 Set Session Template

This interface is used to set the chat template for LLM, including system prompts, prefixes, and suffixes. `session` is the session handle, and `system_prompt`, `prompt_prefix`, and `prompt_postfix` are the template content.

API	**set_chat_template**
Description	Sets the chat template for LLM, including system prompts, prefixes, and suffixes.
	`prompt_prefix`: The prefix added before user input in the chat.
	`prompt_postfix`: The suffix added after user input in the chat.
Return Value	0: Successful setting; -1: Failed setting.

```
...
system_prompt  = "<|im_start|>system\nYou are a helpful assistant.<|im_end|>\n"
prompt_prefix  = "<|im_start|>user\n"
prompt_postfix = "<|im_end|>\n<|im_start|>assistant\n"

ret = rknn_lite.set_chat_template(system_prompt, prompt_prefix, prompt_postfix)
if ret != 0:
    print('Set chat template failed!')
    exit(ret)
```

4.6 Model Inference

API	**inference**
Description	Performs inference on traditional models.
Parameters	`inputs`: Input data list, each element is numpy.ndarray.
	`data_format`: Data format list, each element is string, optional 'undefined', 'nhwc' or 'nchw'. 'nhwc' and 'nchw' are only valid for 4D inputs. For non-4D inputs, set to 'undefined' or leave empty. Default is None.
Return Value	Output data list, each element is numpy.ndarray; failure returns None.

API	**session_run**
Description	Performs inference on LLM models, supporting streaming output.
Parameters	`inputs`: Input data list for multimodal models (e.g., images, audio, etc.). Type: list, elements can be RKNN3Image, RKNN3Audio, RKNN3Video, or RKNN3AuxTensor.
	`prompt`: Text prompt as input for LLM.
	`embeds`: Input embedding vectors for scenarios skipping tokenization. Type: numpy.ndarray, usually 3D tensor (batch_size, sequence_length, hidden_size). Note: prompt and embeds are mutually exclusive—you cannot provide both simultaneously.
Return Value	Returns two values: • `ret`: 0 indicates success, -1 indicates failure. • Second value: Performance statistics list in the format [n_decode_tokens, n_prefill_tokens, llm_start_time, llm_end_time].

Traditional model inference example:

```
...
# Run inference
outputs = rknn_lite.inference(inputs=[img])
if outputs is None:
    print('Inference failed')
    exit(-1)
# Process output
print(outputs)
```

LLM model inference example:

```
...
prompt = "Please explain how LLMs work."
ret, [n_decode_tokens, n_prefill_tokens, llm_start_time, llm_end_time] =
rknn_llm.session_run(prompt=prompt)
if ret != 0:
  print('RKNN llm inference failed!')
  exit(ret)
```

VLM inference example:

```
...
# Get vision model output and convert to required format
outputs = np.float16(np.expand_dims(rknn_vision.inference(inputs=[feature])[0], 0))

prompt = "<image>Please describe the image content."
inputs = []
llm_input = RKNN3Image()
llm_input.image_embed = outputs.ctypes.data_as(ctypes.POINTER(Float16))
llm_input.n_image_tokens = outputs.shape[1]
llm_input.n_image = outputs.shape[0]
llm_input.image_width = 392
llm_input.image_height = 392
llm_input.image_start = "<|vision_start|>".encode('utf-8')
llm_input.image_end = "<|vision_end|>".encode('utf-8')
llm_input.image_content = "<|image_pad|>".encode('utf-8')
inputs.append(llm_input)

# Run LLM inference
ret, [n_decode_tokens, n_prefill_tokens, llm_start_time, llm_end_time] =
rknn_llm.session_run(inputs=inputs, prompt=prompt)
if ret != 0:
  print('RKNN LLM inference failed!')
  exit(ret)
### 4.7 Get SDK Version

| API | get_sdk_version |
| :- | :- |
| Description | Gets the version information of RKNN SDK. |
| Parameters | None |
| Return Value | SDK version string; failure returns None. |
```

Example:

```py
# Get SDK version
version = rknn_lite.get_sdk_version()
if version is not None:
    print('SDK version:', version)
else:
    print('Failed to get SDK version')
```

4.8 Get Input Tensor Attributes

API	get_inputs_tensor_attr
Description	Gets the attribute information of the model's input tensors.
Parameters	None
Return Value	Input tensor attributes list; failure returns None.

Example:

```
# Get input tensor attributes
input_attrs = rknn_lite.get_inputs_tensor_attr()
```

4.9 Get Output Tensor Attributes

API	get_outputs_tensor_attr
Description	Gets the attribute information of the model's output tensors.
Parameters	None
Return Value	Output tensor attributes list; failure returns None.

Example:

```
# Get output tensor attributes
output_attrs = rknn_lite.get_outputs_tensor_attr()
```

5 RKNN3 Type Definitions

This section details the type definitions, structures, enumerations, and constants used in RKNN3 Toolkit Lite. These definitions are based on the `rknn3_types.py` file and are used to describe the data structures and parameters of the RKNN3 API.

5.1 Constants

RKNN3 Toolkit Lite defines the following constants:

- `RKNN3_MAX_DIMS = 16`: Maximum number of tensor dimensions
- `RKNN3_MAX_STRIDE_DIMS = RKNN3_MAX_DIMS + 1`: Maximum number of tensor stride dimensions
- `RKNN3_MAX_NAME_LEN = 256`: Maximum name length
- `RKNN3_MAX_DYNAMIC_SHAPE_NUM = 512`: Maximum number of dynamic shapes
- `RKNN3_MAX_DEVS = 64`: Maximum number of devices
- `RKNN3_MAX_DEV_LEN = 64`: Maximum device ID length
- `RKNN3_MAX_NPU_NODE_NUM = 128`: Maximum number of NPU nodes
- `RKNN3_MAX_SPECIAL_BOS_ID_NUM = 64`: Maximum number of special BOS token IDs
- `RKNN3_MAX_SPECIAL_EOS_ID_NUM = 64`: Maximum number of special EOS token IDs
- `LIBRKNN3RT_PATH = '/usr/lib/librknn3_api.so'`: RKNN3 runtime library path

5.2 Enumerations

5.2.1 RKNN3QueryCmd

Query command type enumeration:

- `RKNN3_QUERY_IN_OUT_NUM = 0`: Query input/output number
- `RKNN3_QUERY_INPUT_ATTR = 1`: Query input attributes
- `RKNN3_QUERY_OUTPUT_ATTR = 2`: Query output attributes
- `RKNN3_QUERY_PERF_DETAIL = 3`: Query performance details
- `RKNN3_QUERY_PERF_RUN = 4`: Query run performance
- `RKNN3_QUERY_SDK_VERSION = 5`: Query SDK version
- `RKNN3_QUERY_MEM_SIZE = 6`: Query memory size
- `RKNN3_QUERY_CUSTOM_STRING = 7`: Query custom string
- `RKNN3_QUERY_NATIVE_INPUT_ATTR = 8`: Query native input attributes
- `RKNN3_QUERY_NATIVE_OUTPUT_ATTR = 9`: Query native output attributes
- `RKNN3_QUERY_NATIVE_NC1HWC2_INPUT_ATTR = 8`: Query NC1HWC2 input attributes
- `RKNN3_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR = 9`: Query NC1HWC2 output attributes
- `RKNN3_QUERY_NATIVE_NHWC_INPUT_ATTR = 10`: Query NHWC input attributes
- `RKNN3_QUERY_NATIVE_NHWC_OUTPUT_ATTR = 11`: Query NHWC output attributes

- `RKNN3_QUERY_DEVICE_MEM_INFO = 12`: Query device memory info
- `RKNN3_QUERY_CORE_NUMBER = 13`: Query core number
- `RKNN3_QUERY_ALLOCATION_INFO = 14`: Query allocation info
- `RKNN3_QUERY_DYNAMIC_SHAPE_CONFIG = 15`: Query dynamic shape config
- `RKNN3_QUERY_DYNAMIC_SHAPE_INFO = 16`: Query dynamic shape info
- `RKNN3_QUERY_LLM_CONFIG = 17`: Query LLM config
- `RKNN3_QUERY_POSTPROCESS_IN_OUT_NUM = 18`: Query postprocess input/output number
- `RKNN3_QUERY_POSTPROCESS_OUTPUT_ATTR = 19`: Query postprocess output attributes
- `RKNN3_QUERY_POSTPROCESS_DYNAMIC_SHAPE_INFO = 20`: Query postprocess dynamic shape info

5.2.2 RKNN3TensorType

Tensor data type enumeration:

- `RKNN3_TENSOR_FLOAT32 = 0`: 32-bit float
- `RKNN3_TENSOR_FLOAT16 = 1`: 16-bit float
- `RKNN3_TENSOR_INT8 = 2`: 8-bit signed integer
- `RKNN3_TENSOR_UINT8 = 3`: 8-bit unsigned integer
- `RKNN3_TENSOR_INT16 = 4`: 16-bit signed integer
- `RKNN3_TENSOR_UINT16 = 5`: 16-bit unsigned integer
- `RKNN3_TENSOR_INT32 = 6`: 32-bit signed integer
- `RKNN3_TENSOR_UINT32 = 7`: 32-bit unsigned integer
- `RKNN3_TENSOR_INT64 = 8`: 64-bit signed integer
- `RKNN3_TENSOR_UINT64 = 9`: 64-bit unsigned integer
- `RKNN3_TENSOR_BOOL = 10`: Boolean
- `RKNN3_TENSOR_INT4 = 11`: 4-bit signed integer

5.2.3 RKNN3TensorQntType

Tensor quantization type enumeration:

- `RKNN3_TENSOR_QNT_NONE = 0`: No quantization
- `RKNN3_TENSOR_PER_LAYER_SYMMETRIC = 1`: Per-layer symmetric quantization
- `RKNN3_TENSOR_PER_LAYER_ASYMMETRIC = 2`: Per-layer asymmetric quantization
- `RKNN3_TENSOR_PER_CHANNEL_SYMMETRIC = 3`: Per-channel symmetric quantization
- `RKNN3_TENSOR_PER_CHANNEL_ASYMMETRIC = 4`: Per-channel asymmetric quantization
- `RKNN3_TENSOR_PER_GROUP_SYMMETRIC = 5`: Per-group symmetric quantization
- `RKNN3_TENSOR_PER_GROUP_ASYMMETRIC = 6`: Per-group asymmetric quantization

5.2.4 RKNN3TensorLayout

Tensor layout format enumeration:

- `RKNN3_TENSOR_UNDEFINED = 0`: Undefined
- `RKNN3_TENSOR_NCHW = 1`: NCHW format
- `RKNN3_TENSOR_NHWC = 2`: NHWC format
- `RKNN3_TENSOR_NC1HWC2 = 3`: NC1HWC2 format
- `RKNN3_TENSOR_CHWN = 4`: CHWN format
- `RKNN3_TENSOR_HWIO = 5`: HWIO format
- `RKNN3_TENSOR_OIHW = 6`: OIHW format
- `RKNN3_TENSOR_O1I1HWI2O2 = 7`: O1I1HWI2O2 format

5.2.5 RKNN3MemAllocFlags

Memory allocation flags enumeration:

- `RKNN3_FLAG_MEMORY_FLAGS_DEFAULT = 0`: Default memory flags
- `RKNN3_FLAG_MEMORY_CACHEABLE = 1`: Cacheable memory
- `RKNN3_FLAG_MEMORY_NON_CACHEABLE = 2`: Non-cacheable memory

5.2.6 RKNN3MemSyncMode

Memory synchronization mode enumeration:

- `RKNN3_MEMORY_SYNC_TO_DEVICE = 1`: Sync to device
- `RKNN3_MEMORY_SYNC_FROM_DEVICE = 2`: Sync from device
- `RKNN3_MEMORY_SYNC_BIDIRECTIONAL = 3`: Bidirectional sync

5.2.7 RKNN3CoreMask

NPU core mask enumeration:

- `RKNN3_NPU_CORE_AUTO = 0`: Auto select core
- `RKNN3_NPU_CORE_0 = 1`: Core 0
- `RKNN3_NPU_CORE_1 = 2`: Core 1
- `RKNN3_NPU_CORE_2 = 4`: Core 2
- `RKNN3_NPU_CORE_3 = 8`: Core 3
- `RKNN3_NPU_CORE_4 = 16`: Core 4
- `RKNN3_NPU_CORE_5 = 32`: Core 5
- `RKNN3_NPU_CORE_6 = 64`: Core 6
- `RKNN3_NPU_CORE_7 = 128`: Core 7
- `RKNN3_NPU_CORE_ALL = 255`: All cores

5.2.8 RKNN3KVCachePolicy

KV cache policy enumeration:

- `RKNN3_KVCACHE_POLICY_AUTO = 0`: Auto policy
- `RKNN3_KVCACHE_POLICY_RECURRENT = 1`: Recurrent policy
- `RKNN3_KVCACHE_POLICY_NORMAL = 2`: Normal policy

5.2.9 RKNN3KVCacheClearPolicy

KV cache clear policy enumeration:

- `RKNN3_KVCACHE_CLEAR_ALL = 0`: Clear all
- `RKNN3_KVCACHE_KEEP_SYSTEM_PROMPT = 1`: Keep system prompt

5.2.10 RKNN3LLMInputType

LLM input type enumeration:

- `RKNN3_LLM_INPUT_PROMPT = 0`: Prompt input
- `RKNN3_LLM_INPUT_TOKEN = 1`: Token input
- `RKNN3_LLM_INPUT_EMBED = 2`: Embed input
- `RKNN3_LLM_INPUT_MULTIMODAL = 3`: Multimodal input
- `RKNN3_LLM_INPUT_AUX = 4`: Auxiliary input

5.2.11 LLMCallState

LLM callback state enumeration:

- `RKLLM_RUN_NORMAL = 0`: Normal run
- `RKLLM_RUN_WAITING = 1`: Waiting
- `RKLLM_RUN_FINISH = 2`: Run finished
- `RKLLM_RUN_STOP = 3`: Run stopped
- `RKLLM_RUN_MAX_NEW_TOKEN_REACHED = 4`: Max new token reached
- `RKLLM_RUN_ERROR = 5`: Run error

5.3 Basic Structures

5.3.1 RKNN3InputOutputNum

Input/output number information:

- `n_input`: Number of inputs (uint32)
- `n_output`: Number of outputs (uint32)

5.3.2 RKNN3QuantInfo

Quantization information:

- `scale`: Scale factor (float)
- `zero_point`: Zero point (int32)

5.3.3 RKNN3TensorAttr

Tensor attribute information:

- `index`: Index (uint32)
- `name`: Name (char[256])
- `n_dims`: Number of dimensions (uint32)
- `shape`: Shape (uint32[16])
- `aligned_size`: Aligned size (uint64)
- `n_stride`: Number of stride dimensions (uint32)
- `stride`: Stride (uint64[17])
- `n_elems`: Number of elements (uint32)
- `dtype`: Data type (int)
- `layout`: Layout (int)
- `qnt_type`: Quantization type (int)
- `qnt_info`: Quantization info (RKNN3QuantInfo)
- `core_id`: Core ID (int32)

5.3.4 RKNN3PerfDetail

Performance details:

- `perf_data`: Performance data (char*)
- `data_len`: Data length (uint64)

5.3.5 RKNN3PerfRun

Run performance:

- `run_duration`: Run duration (int64)

5.3.6 RKNN3SDKVersion

SDK version:

- `api_version`: API version (char[256])
- `drv_version`: Driver version (char[256])

5.3.7 RKNN3MemSize

Memory size:

- `total_const_size`: Total constant size (uint64)
- `total_internal_size`: Total internal size (uint64)
- `total_dma_allocated_size`: Total DMA allocated size (uint64)
- `total_sram_size`: Total SRAM size (uint64)
- `free_sram_size`: Free SRAM size (uint64)

5.3.8 RKNN3CustomString

Custom string:

- `string`: String (char[1024])

5.3.9 RKNN3TensorMemory

Tensor memory:

- `virt_addr`: Virtual address (void*)
- `phys_addr`: Physical address (uint64)
- `fd`: File descriptor (int32)
- `offset`: Offset (uint64)
- `size`: Size (uint64)
- `flags`: Flags (uint64)
- `core_id`: Core ID (int32)
- `priv_data`: Private data (void*)

5.3.10 RKNN3Config

RKNN3 config:

- `priority`: Priority (int32)
- `run_timeout`: Run timeout (uint32)
- `run_core_mask`: Run core mask (uint32)
- `user_mem_weight`: User memory weight (uint8)
- `user_mem_internal`: User internal memory (uint8)
- `user_sram`: User SRAM (uint8)

5.3.11 RKNN3Tensor

RKNN3 tensor:

- `mem`: Memory (RKNN3TensorMemory*)
- `attr`: Attribute (RKNN3TensorAttr*)

5.3.12 RKNN3AllocationInfo

Memory allocation info:

- `core_id`: Core ID (int32)
- `n_const`: Number of constants (uint32)
- `n_internal`: Number of internals (uint32)
- `n_input`: Number of inputs (uint32)
- `n_output`: Number of outputs (uint32)
- `const_mem`: Constant memory (RKNN3TensorMemory*)
- `internal_mem`: Internal memory (RKNN3TensorMemory*)
- `input_mem`: Input memory (RKNN3TensorMemory*)
- `output_mem`: Output memory (RKNN3TensorMemory*)

5.3.13 RKNN3ShapeInfo

Shape info:

- `shape_id`: Shape ID (int32)
- `n_inputs`: Number of inputs (uint32)
- `input_attrs`: Input attributes (RKNN3TensorAttr*)
- `n_outputs`: Number of outputs (uint32)
- `output_attrs`: Output attributes (RKNN3TensorAttr*)
- `is_default`: Is default (uint8)

5.3.14 RKNN3ShapeConfig

Shape config:

- `n_shapes`: Number of shapes (uint32)
- `current_shape_id`: Current shape ID (int32)

5.3.15 RKNN3InitExtend

Extended init parameters:

- `device_id`: Device ID (char*)
- `reserved`: Reserved (uint8[128])

5.3.16 RKN3NodeMemInfo

Node memory info:

- `total`: Total (uint64)
- `free`: Free (uint64)

5.3.17 RKNN3DevMemInfo

Device memory info:

- `node_num`: Node number (uint32)
- `sys_total`: System total (uint64)
- `sys_free`: System free (uint64)
- `node_mem_info`: Node memory info (RKN3NodeMemInfo[128])

5.3.18 RKNN3Device

Device info:

- `id`: ID (char[64])
- `type`: Type (char[64])
- `core_num`: Core number (uint32)
- `mem_info`: Memory info (RKNN3DevMemInfo)

5.3.19 RKNN3Devices

Device list:

- `n_devices`: Number of devices (uint32)
- `devices`: Devices (RKNN3Device[64])

5.4 LLM Related Structures

5.4.1 Float16

16-bit float representation:

- `frac`: Fraction (uint16, 10 bits)
- `exp`: Exponent (uint16, 5 bits)
- `sign`: Sign (uint16, 1 bit)

5.4.2 RKNN3VocabInfo

Vocabulary info:

- `vocab_size`: Vocabulary size (int)
- `special_bos_id`: Special BOS token ID (int[64])
- `special_eos_id`: Special EOS token ID (int[64])
- `n_special_bos_id`: Number of special BOS IDs (int)
- `n_special_eos_id`: Number of special EOS IDs (int)
- `linefeed_id`: Linefeed ID (int)
- `skip_special_token`: Skip special token (bool)
- `ignore_eos_token`: Ignore EOS token (bool)

- `reserved`: Reserved (uint8[64])

5.4.3 RKNN3SamplingParams

Sampling parameters:

- `top_k`: Top-k (int32)
- `top_p`: Top-p (float)
- `temperature`: Temperature (float)
- `repeat_penalty`: Repeat penalty (float)
- `frequency_penalty`: Frequency penalty (float)
- `presence_penalty`: Presence penalty (float)

5.4.4 RECURRENT

Recurrent KV cache parameters:

- `n_keep`: Number to keep (int64)
- `n_keep_aligned`: Number to keep aligned (int64)

5.4.5 RKNN3KVCachePolicyParam

KV cache policy parameters:

- `recurrent`: Recurrent parameters (RECURRENT*)
- `reserved`: Reserved (uint8[64])

5.4.6 RKNN3Lora

LoRA config:

- `lora_name`: LoRA name (char*)
- `scale`: Scale (float)

5.4.7 RKNN3LLMTensor

LLM tensor input:

- `name`: Name (char*)
- `prompt`: Prompt (char*)
- `embed`: Embed (Float16*)
- `tokens`: Tokens (int32*)
- `n_tokens`: Number of tokens (uint64)
- `enable_thinking`: Enable thinking (bool)

5.4.8 RKNN3AuxTensor

Auxiliary tensor (same as RKNN3Tensor)

5.4.9 RKNN3Image

Image input:

- `image_embed`: Image embed (Float16*)
- `n_image_tokens`: Number of image tokens (uint64)
- `n_image`: Number of images (uint64)
- `image_start`: Image start (char*)
- `image_end`: Image end (char*)
- `image_content`: Image content (char*)
- `image_width`: Image width (uint64)
- `image_height`: Image height (uint64)

5.4.10 RKNN3Audio

Audio input:

- `audio_embed`: Audio embed (Float16*)
- `n_audio_tokens`: Number of audio tokens (uint64)
- `n_audio`: Number of audios (uint64)
- `audio_start`: Audio start (char*)
- `audio_end`: Audio end (char*)
- `audio_content`: Audio content (char*)

5.4.11 RKNN3Video

Video input:

- `video_embed`: Video embed (Float16*)
- `n_frame_tokens`: Number of frame tokens (uint64)
- `n_frame_per_video`: Number of frames per video (uint64)
- `n_video`: Number of videos (uint64)
- `video_start`: Video start (char*)
- `video_end`: Video end (char*)
- `video_content`: Video content (char*)
- `frame_width`: Frame width (uint64)
- `frame_height`: Frame height (uint64)

5.4.12 RKNN3LLMMultiModelTensor

Multimodal tensor input:

- `name`: Name (char*)
- `prompt`: Prompt (char*)
- `tokens`: Tokens (int32*)
- `n_tokens`: Number of tokens (uint64)
- `enable_thinking`: Enable thinking (bool)
- `image`: Image data (RKNN3Image)
- `audio`: Audio data (RKNN3Audio)
- `video`: Video data (RKNN3Video)

5.4.13 RKNN3LLMInputUnion

LLM input union:

- `llm_input`: LLM input (RKNN3LLMTensor)
- `multimodal_input`: Multimodal input (RKNN3LLMMultiModelTensor)
- `aux_input`: Auxiliary input (RKNN3AuxTensor)

5.4.14 RKNN3LLMInput

LLM input:

- `role`: Role (char*)
- `input_type`: Input type (int)
- `input_union`: Input union (RKNN3LLMInputUnion)

5.4.15 RKNN3LLMExtendParam

Extended LLM parameters:

- `reserved`: Reserved (uint8[128])

5.4.16 RKNN3LLMParam

LLM parameters:

- `logits_name`: Logits name (char*)
- `max_context_len`: Max context length (int32)
- `sampling_param`: Sampling parameters (RKNN3SamplingParams)
- `vocab_info`: Vocabulary info (RKNN3VocabInfo)
- `extend_param`: Extended parameters (RKNN3LLMExtendParam)

5.4.17 RKNN3LLMInferParam

LLM inference parameters:

- `keep_history`: Keep history (int)
- `max_new_tokens`: Max new tokens (int32)
- `reserved`: Reserved (uint8[128])

5.4.18 RKLLMResult

LLM generation result:

- `token_ids`: Token IDs (int*)
- `num_tokens`: Number of tokens (int)

5.4.19 RKLLMRunState

LLM run state:

- `n_total_tokens`: Total tokens (uint64)
- `n_max_tokens`: Max tokens (uint64)
- `n_decode_tokens`: Decode tokens (uint64)
- `n_prefill_tokens`: Prefill tokens (uint64)
- `kvcache_policy`: KV cache policy (int)
- `n_loras_enabled`: Number of LoRAs enabled (int32)
- `loras_enabled`: LoRAs enabled (RKNN3Lora*)

5.4.20 RKLLMResultLastHiddenLayer

Last hidden layer result:

- `hidden_states`: Hidden states (float*)
- `embd_size`: Embedding size (int)
- `num_tokens`: Number of tokens (int)

5.5 Callback Types

5.5.1 LLMResultCallback

LLM result callback function type:

```
int (*LLMResultCallback)(void* userdata, RKLLMResult* result, int state)
```

5.5.2 LLMSamplingCallback

Sampling callback function type:

```
int (*LLMSamplingCallback)(void* userdata, Float16* logits, char* token_str)
```

5.5.3 LLMGetEmbedCallback

Get embed callback function type:

```
int (*LLMGetEmbedCallback)(void* userdata, int32_t* tokens, uint64_t n_tokens, void* embeds,
uint64_t embd_size)
```