RKNN3 TOOLKIT LITE 参考手册

文件标识：RK-KF-YF-432

发布版本：V1.0.0

日期：2025-01-05

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按"现状"提供，瑞芯微电子股份有限公司（"本公司"，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

"Rockchip"、"瑞芯微"、"瑞芯"均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文是 RKNN3 TOOLKIT LITE 参考手册。

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本	修改人	修改日期	修改说明	核定人
V0.5.0	HPC团队	2025-12-05	初始版本	熊伟
V1.0.0	HPC团队	2025-01-05	完善API说明	熊伟

目录

1 主要功能说明

RKNN3-Toolkit Lite为用户提供协处理器模型推理的Python接口，方便用户使用Python语言进行AI大模型应用开发或者前期模型验证。

1.1 硬件平台

本文档适用如下硬件平台：

- RK1820
- RK1828

1.2 主控芯片适用系统

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

2 开发环境部署

2.1 系统依赖说明

使用RKNN3-Toolkit Lite需满足以下运行环境要求：

表2-1 RKNN3-Toolkit Lite运行环境

操作系统版本	**Debian 10 / 11 / 12 (aarch64)**
Python版本	3.9 / 3.10 / 3.11 / 3.12
Python依赖库	'transformers'、'numpy'、'jinja2'

2.2 工具安装

请通过`pip3 install`命令安装 RKNN3-Toolkit Lite。安装步骤如下：

- 如果系统中没有安装`python3/pip3`等程序，请先通过`apt-get`方式安装，参考命令如下：

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

注：部分依赖模块需要编译源码，此时将用到`python3-dev`和`gcc`，此步骤将同时安装这两个包，以避免在安装依赖时编译失败。

- 安装依赖模块：`opencv-python`和`numpy`，参考命令如下：

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

或者

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

注：

1. RKNN3-Toolkit Lite本身并不依赖`opencv-python`，但是在示例中需要使用该模块对图像进行处理。
2. 在Debian10固件上通过`pip3`安装`numpy`可能失败，建议用上述方法安装。

- 安装RKNN3-Toolkit Lite
  各平台的安装包都放在SDK的`rknn3-toolkit-lite/packages`文件夹下。进入该文件夹后，执行以下命令来安装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 使用说明

RKNN3-Toolkit Lite主要用于RKNN模型在Rockchip NPU上的部署。
在使用RKNN3-Toolkit Lite之前，用户需要先通过RKNN3-Toolkit将各深度学习框架导出的模型转成RKNN模型。

3.1 基本使用流程

使用RKNN3-Toolkit Lite部署RKNN模型的基本流程如下图所示：

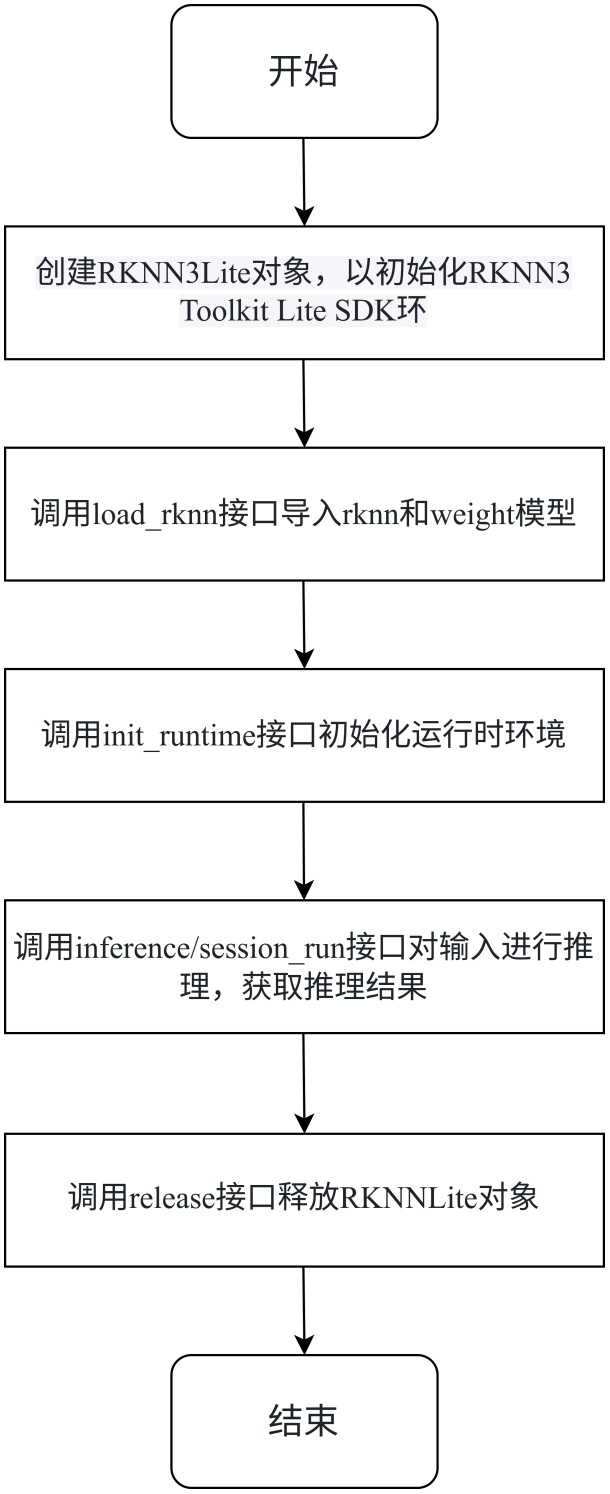

图3-1 RKNN3-Toolkit Lite基本使用流程

注：

1.在调用`inference`接口进行推理之前，需要准备输入数据并进行相应的预处理。如果是纯LLM需要获取`input_ids`数据，若是多模态模型则要获取`embeds`输入，然后根据输入信息设置`inference`接口中的各项参数。
2.在调用`inference`接口后，通常需要对推理结果进行相应的处理，以完成应用程序相关功能。

3.2 运行参考示例

在`SDK/rknn3-toolkit-lite/examples`目录，提供了一个基于RKNN3-Toolkit Lite开发的VLM应用。该应用使用RKNN3-Toolkit Lite接口加载Qwen2.5VL-3B RKNN模型和Tokenizer，进行输入图片和Prompt的推理，并输出分析结果。

运行该示例的步骤：

1. 准备一块安装有RKNN3-Toolkit Lite的开发板；
2. 将`SDK/rknn3-toolkit-lite/examples`完整推送到开发板；
3. 在开发板上进入`examples/Qwen2.5VL`目录，执行如下命令运行该示例：

```
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
```

注意：请确保所指定的 RKNN 模型路径与配套权重文件路径一致，否则可能导致推理失败。

示例运行结果（部分）：

```
--> Running vision model
这张图片展示了一个太空场景。背景中可以看到一个巨大的行星，可能是月球或火星的表面，上面有一些坑洞和山脉。天空中有许多星星点缀其中。

前景中有一个穿着宇航服的人躺在地上休息。宇航服看起来是白色的，并且有多个口袋和按钮。这个人似乎在放松，手里拿着一瓶饮料（可能是啤酒），旁边还有一个小盒子。地面上还有一些物品散落着，包括一个梯子、一块石头和其他一些不明物体。

整个场景给人一种太空探索或月球任务的感觉，但同时也带有一种轻松的氛围，因为宇航员看起来像是在休息和享受片刻的放松时间。

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

注意：由于模型版本、输入图像或运行环境的差异，实际输出结果可能与上述示例略有不同。

4 RKNN3 Toolkit Lite API详细说明

本章节将详细说明RKNN3-Toolkit Lite提供的所有API的用法。

4.1 RKNNLite初始化及对象释放

在使用 RKNN3-Toolkit Lite 时，需要先调用 RKNN3Lite() 方法创建一个 RKNN3Lite 对象，并在使用完毕后调用该对象的 release() 方法释放相关资源。如果加载的是大语言模型（LLM），则需在初始化时设置参数 llm_mode=True。

参数：

- `llm_mode`：布尔类型，是否为LLM模式，默认False
- `verbose`：布尔类型，是否在终端中打印详细日志，默认False
- `verbose_file`：字符串类型，日志文件路径，如果指定，将日志写入该文件，默认None

此外，初始化 RKNN3Lite 对象时，可通过 verbose 和 verbose_file 参数控制日志输出行为：

- verbose=True 表示在终端中打印详细日志；
- 若同时指定了 verbose_file（例如 './inference.log'），则日志信息还会被写入该文件中。

举例如下：

```
# 将详细的日志信息输出到屏幕，并写到inference.log文件中
rknn_lite = RKNN3Lite(verbose=True, verbose_file='./inference.log')
# 只在屏幕打印详细的日志信息
rknn_lite = RKNN3Lite(verbose=True)
#初始化LLM模型
rknn_lite = RKNN3Lite(llm_mode=True)
…
rknn_lite.release()
```

4.2 加载RKNN模型

表4-1 load_rknn接口详细说明

API	load_rknn
描述	加载RKNN模型
参数	`model_path`: RKNN模型路径
	`weight_path`: RKNN模型路径
返回值	0：加载成功；-1：加载失败。

举例如下：

```
ret = rknn_lite.load_rknn(model_path='mobilenetv2-12.rknn', weight_path='mobilenetv2-12.weight')
```

4.3 初始化运行时环境

在模型推理之前，必须先初始化运行时环境。

表4-2 init_runtime接口详细说明

API	init_runtime
描述	初始化运行时环境。
参数	`target`：指定的协处理器，目前支持RK1820/RK1828
	`core_mask`：npu核心掩码，协处理器共包含 8 个 NPU 核心，该参数通过位掩码指定启用的核心。例如 0x0f(即二进制0b00001111)，表示使用低4位npu核心。注意该参数需要和转rknn模型的core_mask保持一致，否则会报错
	`llm_args`：LLM 模型的配置参数字典
	`llm_callback`：LLM模型的回调函数可用于流式推理
	`llm_embed_path`：embedding词表路径，用于初始化LLMGetEmbedCallback 所需的参数
	`device_id`：设备id,用于区分多个设备，如果只连接了一个设备可以置为None
返回值	0：初始化运行时环境成功；-1：初始化运行时环境失败。

注：

- `llm_args` 目前可配置的参数及其含义如下：
  - `top_k`：采样时考虑的最高概率词汇的数量，整数类型，默认值根据模型而定。
  - `top_p`：核采样概率阈值，浮点数类型，取值范围 [0, 1]，用于控制采样的多样性。
  - `temperature`：控制随机性的温度参数，浮点数类型，通常大于 0，越小生成越确定。
  - `repeat_penalty`：重复惩罚系数，浮点数类型，用于减少重复生成的惩罚。
  - `frequency_penalty`：频率惩罚，浮点数类型，基于词汇出现频率的惩罚。
  - `presence_penalty`：存在惩罚，浮点数类型，基于词汇是否出现过的惩罚。
  - `vocab_size`：词汇表大小，整数类型，表示模型词汇表的总大小。
  - `special_bos_id`：开始标记（Beginning of Sequence）的ID，整数类型。
  - `special_eos_id`：结束标记（End of Sequence）的ID，整数类型。
  - `linefeed_id`：换行符的ID，整数类型。
  - `skip_special_token`：是否跳过特殊标记，布尔类型，True 表示跳过。
  - `ignore_eos_token`：是否忽略结束标记，布尔类型，True 表示忽略。
  - `keep_history`：是否保持对话历史，布尔类型，True 表示保持。
  - `max_new_tokens`：最大新生成标记数，整数类型，控制生成文本的最大长度。
  - `logits_name`：logits 输出的名称，字符串类型，用于指定输出层名称。
  - `max_context_len`：最大上下文长度，整数类型，表示模型能处理的上下文最大长度。

- `llm_callback` 主要包含5个回调函数，用于处理LLM推理过程中的不同阶段：
  - `LLMResultCallback`：处理推理结果的回调函数，用于接收和处理生成的文本结果。
  - `LLMSamplingCallback`：采样回调函数，用于自定义采样策略或处理采样过程。
  - `LLMTokenizerCallback`：分词器回调函数，用于处理输入文本的分词和编码。
  - `LLMGetEmbedCallback`：获取嵌入回调函数，用于获取输入的嵌入向量。
  - `LLMGetLastHiddenLayerCallback`：获取最后隐藏层回调函数，用于获取模型的最后隐藏层输出。
    注意：这些回调函数的具体实现和参数与 API 手册中提供的函数和参数一一对应，用户可根据需要自定义实现。
- `device_id` 可以通过 `get_devices_id` 函数获取所有协处理器的设备ID。

传统模型举例如下：

```
...
# 获取device id
device_id = rknn_lite.get_devices_id()
# 初始化运行时环境
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)
```

大模型举例如下：

```
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 获取设备ID

用于获取当前系统中所有已连接协处理器的设备ID。

表4-3 get_devices_id接口说明

API	get_devices_id
描述	获取所有已连接设备的 ID 列表。
返回值	设备 ID 列表，类型为List[str]

举例如下：

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

注：以上接口适用于多设备部署场景。若仅使用单设备，可直接取列表首元素或传入 None（如 init_runtime 所支持）。

4.5 设置会话模板

该接口用于设置LLM的聊天模板，包括系统提示、前缀和后缀。session为会话句柄，system_prompt、prompt_prefix、prompt_postfix为模板内容。

表4-4 set_chat_template接口说明

API	set_chat_template
描述	设置LLM的聊天模板，包括系统提示、前缀和后缀。
参数	`system_prompt`：定义语言模型上下文或行为的系统提示
	`prompt_prefix`：聊天中用户输入前添加的前缀
	`prompt_postfix`：聊天中用户输入后添加的后缀
返回值	0：设置成功；-1：设置失败
举例如下：	

```
...
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 模型推理

表4-5 inference接口详细说明

API	inference
描述	运行传统模型推理。
参数	`inputs`：输入数据列表，每个元素为numpy.ndarray。
	`data_format`：数据格式列表，每个元素为字符串，可选 'undefined', 'nhwc' 或 'nchw'。'nhwc' 和 'nchw'仅对4维输入有效。对于非四维输入，应设为 undefined 或留空。默认值为 None。

API	inference
返回值	输出数据列表，每个元素为numpy.ndarray；失败时返回None。

表4-6 session_run接口详细说明

API	session_run
描述	运行LLM模型推理，支持流式输出。
参数	`inputs`：输入数据列表，用于多模态模型（如图像、音频等）。类型为列表，元素可以是RKNN3Image、RKNN3Audio、RKNN3Video 或 RKNN3AuxTensor 等专用类型。
	`prompt`：文本提示，作为LLM模型的输入。
	`embeds`：输入嵌入向量，用于跳过tokenization的场景。类型为numpy.ndarray，通常为3D张量(batch_size, sequence_length, hidden_size)。注意：prompt 和 embeds 互斥，不能同时提供。
返回值	返回两个值： • `ret`：0 表示推理成功，-1 表示失败； • 第二个值为性能统计信息列表，格式为 `[n_decode_tokens, n_prefill_tokens, llm_start_time, llm_end_time]`。

传统模型推理代码参考如下：

```
...
# 运行推理
outputs = rknn_lite.inference(inputs=[img])
if outputs is None:
    print('Inference failed')
    exit(-1)
# 处理输出
print(outputs)
```

LLM模型推理代码参考如下：

```
...
prompt = "介绍一下LLM模型的工作原理。"
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模型推理代码参考如下：

```
...
# 获取视觉模型的输出并转换为所需格式
outputs = np.float16(np.expand_dims(rknn_vision.inference(inputs=[feature])[0], 0))
```

```
prompt = "<image>请描述图片内容"
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)

# 运行LLM推理
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 获取SDK版本

表4-7 get_sdk_version接口详细说明

API	get_sdk_version
描述	获取RKNN SDK的版本信息。
参数	无
返回值	SDK版本字符串；失败时返回None。

举例如下：

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

4.8 获取输入张量属性

表4-8 get_inputs_tensor_attr接口详细说明

API	get_inputs_tensor_attr
描述	获取模型输入张量的属性信息。
参数	无

API	get_inputs_tensor_attr
返回值	输入张量属性列表；失败时返回None。

举例如下：

```
# 获取输入张量属性
input_attrs = rknn_lite.get_inputs_tensor_attr()
```

4.9 获取输出张量属性

表4-9 get_outputs_tensor_attr接口详细说明

API	get_outputs_tensor_attr
描述	获取模型输出张量的属性信息。
参数	无
返回值	输出张量属性列表；失败时返回None。

举例如下：

```
# 获取输出张量属性
output_attrs = rknn_lite.get_outputs_tensor_attr()
```

5 RKNN3 类型定义

本章节详细说明 RKNN3 Toolkit Lite 中使用的类型定义、结构、枚举和常量。这些定义基于 `rknn3_types.py` 文件，用于描述 RKNN3 API 的数据结构和参数。

5.1 常量定义

RKNN3 Toolkit Lite 定义了以下常量：

- `RKNN3_MAX_DIMS = 16`：张量最大维度数
- `RKNN3_MAX_STRIDE_DIMS = RKNN3_MAX_DIMS + 1`：张量步长最大维度数
- `RKNN3_MAX_NAME_LEN = 256`：名称最大长度
- `RKNN3_MAX_DYNAMIC_SHAPE_NUM = 512`：动态形状最大数量
- `RKNN3_MAX_DEVS = 64`：最大设备数
- `RKNN3_MAX_DEV_LEN = 64`：设备ID最大长度
- `RKNN3_MAX_NPU_NODE_NUM = 128`：NPU节点最大数量
- `RKNN3_MAX_SPECIAL_BOS_ID_NUM = 64`：特殊开始标记ID最大数量
- `RKNN3_MAX_SPECIAL_EOS_ID_NUM = 64`：特殊结束标记ID最大数量
- `LIBRKNN3RT_PATH = '/usr/lib/librknn3_api.so'`：RKNN3运行时库路径

5.2 枚举类型

5.2.1 RKNN3QueryCmd

查询命令类型枚举：

- `RKNN3_QUERY_IN_OUT_NUM = 0`：查询输入输出数量
- `RKNN3_QUERY_INPUT_ATTR = 1`：查询输入属性
- `RKNN3_QUERY_OUTPUT_ATTR = 2`：查询输出属性
- `RKNN3_QUERY_PERF_DETAIL = 3`：查询性能详情
- `RKNN3_QUERY_PERF_RUN = 4`：查询运行性能
- `RKNN3_QUERY_SDK_VERSION = 5`：查询SDK版本
- `RKNN3_QUERY_MEM_SIZE = 6`：查询内存大小
- `RKNN3_QUERY_CUSTOM_STRING = 7`：查询自定义字符串
- `RKNN3_QUERY_NATIVE_INPUT_ATTR = 8`：查询原生输入属性
- `RKNN3_QUERY_NATIVE_OUTPUT_ATTR = 9`：查询原生输出属性
- `RKNN3_QUERY_NATIVE_NC1HWC2_INPUT_ATTR = 8`：查询NC1HWC2输入属性
- `RKNN3_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR = 9`：查询NC1HWC2输出属性
- `RKNN3_QUERY_NATIVE_NHWC_INPUT_ATTR = 10`：查询NHWC输入属性
- `RKNN3_QUERY_NATIVE_NHWC_OUTPUT_ATTR = 11`：查询NHWC输出属性
- `RKNN3_QUERY_DEVICE_MEM_INFO = 12`：查询设备内存信息

- `RKNN3_QUERY_CORE_NUMBER = 13`：查询核心数量
- `RKNN3_QUERY_ALLOCATION_INFO = 14`：查询分配信息
- `RKNN3_QUERY_DYNAMIC_SHAPE_CONFIG = 15`：查询动态形状配置
- `RKNN3_QUERY_DYNAMIC_SHAPE_INFO = 16`：查询动态形状信息
- `RKNN3_QUERY_LLM_CONFIG = 17`：查询LLM配置
- `RKNN3_QUERY_POSTPROCESS_IN_OUT_NUM = 18`：查询后处理输入输出数量
- `RKNN3_QUERY_POSTPROCESS_OUTPUT_ATTR = 19`：查询后处理输出属性
- `RKNN3_QUERY_POSTPROCESS_DYNAMIC_SHAPE_INFO = 20`：查询后处理动态形状信息

5.2.2 RKNN3TensorType

张量数据类型枚举：

- `RKNN3_TENSOR_FLOAT32 = 0`：32位浮点数
- `RKNN3_TENSOR_FLOAT16 = 1`：16位浮点数
- `RKNN3_TENSOR_INT8 = 2`：8位有符号整数
- `RKNN3_TENSOR_UINT8 = 3`：8位无符号整数
- `RKNN3_TENSOR_INT16 = 4`：16位有符号整数
- `RKNN3_TENSOR_UINT16 = 5`：16位无符号整数
- `RKNN3_TENSOR_INT32 = 6`：32位有符号整数
- `RKNN3_TENSOR_UINT32 = 7`：32位无符号整数
- `RKNN3_TENSOR_INT64 = 8`：64位有符号整数
- `RKNN3_TENSOR_UINT64 = 9`：64位无符号整数
- `RKNN3_TENSOR_BOOL = 10`：布尔类型
- `RKNN3_TENSOR_INT4 = 11`：4位有符号整数

5.2.3 RKNN3TensorQntType

张量量化类型枚举：

- `RKNN3_TENSOR_QNT_NONE = 0`：无量化
- `RKNN3_TENSOR_PER_LAYER_SYMMETRIC = 1`：每层对称量化
- `RKNN3_TENSOR_PER_LAYER_ASYMMETRIC = 2`：每层非对称量化
- `RKNN3_TENSOR_PER_CHANNEL_SYMMETRIC = 3`：每通道对称量化
- `RKNN3_TENSOR_PER_CHANNEL_ASYMMETRIC = 4`：每通道非对称量化
- `RKNN3_TENSOR_PER_GROUP_SYMMETRIC = 5`：每组对称量化
- `RKNN3_TENSOR_PER_GROUP_ASYMMETRIC = 6`：每组非对称量化

5.2.4 RKNN3TensorLayout

张量布局格式枚举：

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

5.2.5 RKNN3MemAllocFlags

内存分配标志枚举：

- `RKNN3_FLAG_MEMORY_FLAGS_DEFAULT = 0`：默认内存标志
- `RKNN3_FLAG_MEMORY_CACHEABLE = 1`：可缓存内存
- `RKNN3_FLAG_MEMORY_NON_CACHEABLE = 2`：不可缓存内存

5.2.6 RKNN3MemSyncMode

内存同步模式枚举：

- `RKNN3_MEMORY_SYNC_TO_DEVICE = 1`：同步到设备
- `RKNN3_MEMORY_SYNC_FROM_DEVICE = 2`：从设备同步
- `RKNN3_MEMORY_SYNC_BIDIRECTIONAL = 3`：双向同步

5.2.7 RKNN3CoreMask

NPU核心掩码枚举：

- `RKNN3_NPU_CORE_AUTO = 0`：自动选择核心
- `RKNN3_NPU_CORE_0 = 1`：核心0
- `RKNN3_NPU_CORE_1 = 2`：核心1
- `RKNN3_NPU_CORE_2 = 4`：核心2
- `RKNN3_NPU_CORE_3 = 8`：核心3
- `RKNN3_NPU_CORE_4 = 16`：核心4
- `RKNN3_NPU_CORE_5 = 32`：核心5
- `RKNN3_NPU_CORE_6 = 64`：核心6
- `RKNN3_NPU_CORE_7 = 128`：核心7
- `RKNN3_NPU_CORE_ALL = 255`：所有核心

5.2.8 RKNN3KVCachePolicy

KV缓存策略枚举：

- `RKNN3_KVCACHE_POLICY_AUTO = 0`：自动策略
- `RKNN3_KVCACHE_POLICY_RECURRENT = 1`：循环策略
- `RKNN3_KVCACHE_POLICY_NORMAL = 2`：普通策略

5.2.9 RKNN3KVCacheClearPolicy

KV缓存清除策略枚举：

- `RKNN3_KVCACHE_CLEAR_ALL = 0`：清除所有
- `RKNN3_KVCACHE_KEEP_SYSTEM_PROMPT = 1`：保留系统提示

5.2.10 RKNN3LLMInputType

LLM输入类型枚举：

- `RKNN3_LLM_INPUT_PROMPT = 0`：提示输入
- `RKNN3_LLM_INPUT_TOKEN = 1`：标记输入
- `RKNN3_LLM_INPUT_EMBED = 2`：嵌入输入
- `RKNN3_LLM_INPUT_MULTIMODAL = 3`：多模态输入
- `RKNN3_LLM_INPUT_AUX = 4`：辅助输入

5.2.11 LLMCallState

LLM回调状态枚举：

- `RKLLM_RUN_NORMAL = 0`：正常运行
- `RKLLM_RUN_WAITING = 1`：等待中
- `RKLLM_RUN_FINISH = 2`：运行完成
- `RKLLM_RUN_STOP = 3`：运行停止
- `RKLLM_RUN_MAX_NEW_TOKEN_REACHED = 4`：达到最大新标记数
- `RKLLM_RUN_ERROR = 5`：运行错误

5.3 基本结构

5.3.1 RKNN3InputOutputNum

输入输出数量信息：

- `n_input`：输入数量（uint32）
- `n_output`：输出数量（uint32）

5.3.2 RKNN3QuantInfo

量化信息：

- `scale`：缩放因子（float）
- `zero_point`：零点（int32）

5.3.3 RKNN3TensorAttr

张量属性信息：

- `index`：索引（uint32）
- `name`：名称（char[256]）
- `n_dims`：维度数（uint32）
- `shape`：形状（uint32[16]）
- `aligned_size`：对齐大小（uint64）
- `n_stride`：步长维度数（uint32）
- `stride`：步长（uint64[17]）
- `n_elems`：元素数量（uint32）
- `dtype`：数据类型（int）
- `layout`：布局（int）
- `qnt_type`：量化类型（int）
- `qnt_info`：量化信息（RKNN3QuantInfo）
- `core_id`：核心ID（int32）

5.3.4 RKNN3PerfDetail

性能详情：

- `perf_data`：性能数据（char*）
- `data_len`：数据长度（uint64）

5.3.5 RKNN3PerfRun

运行性能：

- `run_duration`：运行时长（int64）

5.3.6 RKNN3SDKVersion

SDK版本：

- `api_version`：API版本（char[256]）
- `drv_version`：驱动版本（char[256]）

5.3.7 RKNN3MemSize

内存大小：

- `total_const_size`：常量总大小（uint64）
- `total_internal_size`：内部总大小（uint64）
- `total_dma_allocated_size`：DMA分配总大小（uint64）
- `total_sram_size`：SRAM总大小（uint64）
- `free_sram_size`：可用SRAM大小（uint64）

5.3.8 RKNN3CustomString

自定义字符串：

- `string`：字符串（char[1024]）

5.3.9 RKNN3TensorMemory

张量内存：

- `virt_addr`：虚拟地址（void*）
- `phys_addr`：物理地址（uint64）
- `fd`：文件描述符（int32）
- `offset`：偏移（uint64）
- `size`：大小（uint64）
- `flags`：标志（uint64）
- `core_id`：核心ID（int32）
- `priv_data`：私有数据（void*）

5.3.10 RKNN3Config

RKNN3配置：

- `priority`：优先级（int32）
- `run_timeout`：运行超时（uint32）
- `run_core_mask`：运行核心掩码（uint32）
- `user_mem_weight`：用户内存权重（uint8）
- `user_mem_internal`：用户内部内存（uint8）
- `user_sram`：用户SRAM（uint8）

5.3.11 RKNN3Tensor

RKNN3张量：

- `mem`：内存（RKNN3TensorMemory*）
- `attr`：属性（RKNN3TensorAttr*）

5.3.12 RKNN3AllocationInfo

内存分配信息：

- `core_id`：核心ID（int32）
- `n_const`：常量数量（uint32）
- `n_internal`：内部数量（uint32）
- `n_input`：输入数量（uint32）
- `n_output`：输出数量（uint32）
- `const_mem`：常量内存（RKNN3TensorMemory*）
- `internal_mem`：内部内存（RKNN3TensorMemory*）
- `input_mem`：输入内存（RKNN3TensorMemory*）
- `output_mem`：输出内存（RKNN3TensorMemory*）

5.3.13 RKNN3ShapeInfo

形状信息：

- `shape_id`：形状ID（int32）
- `n_inputs`：输入数量（uint32）
- `input_attrs`：输入属性（RKNN3TensorAttr*）
- `n_outputs`：输出数量（uint32）
- `output_attrs`：输出属性（RKNN3TensorAttr*）
- `is_default`：是否默认（uint8）

5.3.14 RKNN3ShapeConfig

形状配置：

- `n_shapes`：形状数量（uint32）
- `current_shape_id`：当前形状ID（int32）

5.3.15 RKNN3InitExtend

扩展初始化参数：

- `device_id`：设备ID（char*）
- `reserved`：保留（uint8[128]）

5.3.16 RKN3NodeMemInfo

节点内存信息：

- `total`：总内存（uint64）
- `free`：可用内存（uint64）

5.3.17 RKNN3DevMemInfo

设备内存信息：

- `node_num`：节点数量（uint32）
- `sys_total`：系统总内存（uint64）
- `sys_free`：系统可用内存（uint64）
- `node_mem_info`：节点内存信息（RKN3NodeMemInfo[128]）

5.3.18 RKNN3Device

设备信息：

- `id`：设备ID（char[64]）
- `type`：设备类型（char[64]）
- `core_num`：核心数量（uint32）
- `mem_info`：内存信息（RKNN3DevMemInfo）

5.3.19 RKNN3Devices

设备列表：

- `n_devices`：设备数量（uint32）
- `devices`：设备信息（RKNN3Device[64]）

5.4 LLM相关结构

5.4.1 Float16

16位浮点数表示：

- `frac`：尾数（uint16, 10位）
- `exp`：指数（uint16, 5位）
- `sign`：符号（uint16, 1位）

5.4.2 RKNN3VocabInfo

词汇表信息：

- `vocab_size`：词汇表大小（int）
- `special_bos_id`：特殊开始标记ID（int[64]）
- `special_eos_id`：特殊结束标记ID（int[64]）
- `n_special_bos_id`：特殊开始标记数量（int）
- `n_special_eos_id`：特殊结束标记数量（int）
- `linefeed_id`：换行符ID（int）
- `skip_special_token`：是否跳过特殊标记（bool）
- `ignore_eos_token`：是否忽略结束标记（bool）

- `reserved`：保留（uint8[64]）

5.4.3 RKNN3SamplingParams

采样参数：

- `top_k`：top-k（int32）
- `top_p`：top-p（float）
- `temperature`：温度（float）
- `repeat_penalty`：重复惩罚（float）
- `frequency_penalty`：频率惩罚（float）
- `presence_penalty`：存在惩罚（float）

5.4.4 RECURRENT

循环KV缓存参数：

- `n_keep`：保留数量（int64）
- `n_keep_aligned`：对齐保留数量（int64）

5.4.5 RKNN3KVCachePolicyParam

KV缓存策略参数：

- `recurrent`：循环参数（RECURRENT*）
- `reserved`：保留（uint8[64]）

5.4.6 RKNN3Lora

LoRA配置：

- `lora_name`：LoRA名称（char*）
- `scale`：缩放因子（float）

5.4.7 RKNN3LLMTensor

LLM张量输入：

- `name`：名称（char*）
- `prompt`：提示（char*）
- `embed`：嵌入（Float16*）
- `tokens`：标记（int32*）
- `n_tokens`：标记数量（uint64）
- `enable_thinking`：启用思考（bool）

5.4.8 RKNN3AuxTensor

辅助张量（与RKNN3Tensor相同）

5.4.9 RKNN3Image

图像输入：

- `image_embed`：图像嵌入（Float16*）
- `n_image_tokens`：图像标记数量（uint64）
- `n_image`：图像数量（uint64）
- `image_start`：图像开始标记（char*）
- `image_end`：图像结束标记（char*）
- `image_content`：图像内容标记（char*）
- `image_width`：图像宽度（uint64）
- `image_height`：图像高度（uint64）

5.4.10 RKNN3Audio

音频输入：

- `audio_embed`：音频嵌入（Float16*）
- `n_audio_tokens`：音频标记数量（uint64）
- `n_audio`：音频数量（uint64）
- `audio_start`：音频开始标记（char*）
- `audio_end`：音频结束标记（char*）
- `audio_content`：音频内容标记（char*）

5.4.11 RKNN3Video

视频输入：

- `video_embed`：视频嵌入（Float16*）
- `n_frame_tokens`：帧标记数量（uint64）
- `n_frame_per_video`：每视频帧数量（uint64）
- `n_video`：视频数量（uint64）
- `video_start`：视频开始标记（char*）
- `video_end`：视频结束标记（char*）
- `video_content`：视频内容标记（char*）
- `frame_width`：帧宽度（uint64）
- `frame_height`：帧高度（uint64）

5.4.12 RKNN3LLMMultiModelTensor

多模态张量输入：

- `name`：名称（char*）
- `prompt`：提示（char*）
- `tokens`：标记（int32*）
- `n_tokens`：标记数量（uint64）
- `enable_thinking`：启用思考（bool）
- `image`：图像数据（RKNN3Image）
- `audio`：音频数据（RKNN3Audio）
- `video`：视频数据（RKNN3Video）

5.4.13 RKNN3LLMInputUnion

LLM输入联合：

- `llm_input`：LLM输入（RKNN3LLMTensor）
- `multimodal_input`：多模态输入（RKNN3LLMMultiModelTensor）
- `aux_input`：辅助输入（RKNN3AuxTensor）

5.4.14 RKNN3LLMInput

LLM输入：

- `role`：角色（char*）
- `input_type`：输入类型（int）
- `input_union`：输入联合（RKNN3LLMInputUnion）

5.4.15 RKNN3LLMExtendParam

扩展LLM参数：

- `reserved`：保留（uint8[128]）

5.4.16 RKNN3LLMParam

LLM参数：

- `logits_name`：logits名称（char*）
- `max_context_len`：最大上下文长度（int32）
- `sampling_param`：采样参数（RKNN3SamplingParams）
- `vocab_info`：词汇表信息（RKNN3VocabInfo）
- `extend_param`：扩展参数（RKNN3LLMExtendParam）

5.4.17 RKNN3LLMInferParam

LLM推理参数：

- `keep_history`：保持历史（int）
- `max_new_tokens`：最大新标记数（int32）
- `reserved`：保留（uint8[128]）

5.4.18 RKLLMResult

LLM生成结果：

- `token_ids`：标记ID（int*）
- `num_tokens`：标记数量（int）

5.4.19 RKLLMRunState

LLM运行状态：

- `n_total_tokens`：总标记数（uint64）
- `n_max_tokens`：最大标记数（uint64）
- `n_decode_tokens`：解码标记数（uint64）
- `n_prefill_tokens`：预填充标记数（uint64）
- `kvcache_policy`：KV缓存策略（int）
- `n_loras_enabled`：启用LoRA数量（int32）
- `loras_enabled`：启用LoRA（RKNN3Lora*）

5.4.20 RKLLMResultLastHiddenLayer

最后隐藏层结果：

- `hidden_states`：隐藏状态（float*）
- `embd_size`：嵌入大小（int）
- `num_tokens`：标记数量（int）

5.5 回调类型

5.5.1 LLMResultCallback

LLM结果回调函数类型：

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

5.5.2 LLMSamplingCallback

采样回调函数类型：

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

5.5.3 LLMGetEmbedCallback

获取嵌入回调函数类型：

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

5.5.4 LLMTokenizerCallback

分词器回调函数类型：

```
int (*LLMTokenizerCallback)(void* userdata, char* text, int32_t text_len, int32_t* tokens,
int32_t max_tokens)
```

5.5.5 LLMGetLastHiddenLayerCallback

获取最后隐藏层回调函数类型：

```
int (*LLMGetLastHiddenLayerCallback)(void* userdata, RKLLMResultLastHiddenLayer
hidden_layer, int state)
```

5.5.6 RKLLMCallback

LLM回调函数结构：

- `result_callback`：结果回调（LLMResultCallback）
- `result_userdata`：结果用户数据（void*）
- `sampling_callback`：采样回调（LLMSamplingCallback）
- `sampling_userdata`：采样用户数据（void*）
- `tokenizer_callback`：分词器回调（LLMTokenizerCallback）
- `tokenizer_userdata`：分词器用户数据（void*）
- `embed_callback`：嵌入回调（LLMGetEmbedCallback）
- `embed_userdata`：嵌入用户数据（void*）
- `hidden_layer_callback`：隐藏层回调（LLMGetLastHiddenLayerCallback）
- `hidden_layer_userdata`：隐藏层用户数据（void*）

5.6 包装类

5.6.1 RKNN3AuxTensorWrapper

辅助张量包装类：

- `aux_data`：辅助数据（numpy数组）
- `index`：索引（int）
- `align_size`：对齐大小（int）

5.7 工具函数

5.7.1 rknn_dtype_to_numpy_dtype

描述：将 RKNN3 张量数据类型枚举转换为对应的 NumPy 数据类型。该函数用于在 Python 环境中处理张量数据时，确保数据类型的正确映射。

参数：

- `dtype_enum`：RKNN3 张量类型枚举值（RKNN3TensorType）

返回值：对应的 NumPy 数据类型（numpy.dtype）。如果输入的枚举值未找到映射，则返回 numpy.float32。

示例：

```
from rknn3lite.api.rknn3_types import rknn_dtype_to_numpy_dtype, RKNN3TensorType

# 转换 FP16 类型
np_dtype = rknn_dtype_to_numpy_dtype(RKNN3TensorType.RKNN3_TENSOR_FLOAT16)
print(np_dtype)  # 输出: <class 'numpy.float16'>
```

5.7.2 rknn3_get_layout_string

描述：根据张量布局枚举值获取对应的布局字符串表示。该函数用于调试和日志输出，帮助用户理解张量的布局格式。

参数：

- `layout`：RKNN3 张量布局枚举值（RKNN3TensorLayout）

返回值：布局的字符串表示（如 "NCHW", "NHWC"）。如果枚举值无效，则返回 "UNKNOWN"。

示例：

```
from rknn3lite.api.rknn3_types import rknn3_get_layout_string, RKNN3TensorLayout

# 获取 NHWC 布局字符串
layout_str = rknn3_get_layout_string(RKNN3TensorLayout.RKNN3_TENSOR_NHWC)
print(layout_str)  # 输出: NHWC
```

5.7.3 rknn3_get_type_string

描述：根据张量数据类型枚举值获取对应的类型字符串表示。该函数用于调试和日志输出，帮助用户理解张量的数据类型。

参数：

- `dtype`：RKNN3 张量数据类型枚举值（RKNN3TensorType）

返回值：数据类型的字符串表示（如 "FP32", "INT8"）。如果枚举值无效，则返回 "UNKNOWN"。

示例：

```python
from rknn3lite.api.rknn3_types import rknn3_get_type_string, RKNN3TensorType

# 获取 INT8 类型字符串
type_str = rknn3_get_type_string(RKNN3TensorType.RKNN3_TENSOR_INT8)
print(type_str)  # 输出: INT8
```

5.7.4 dump_tensor_attr

描述：打印张量属性的详细信息到控制台。该函数用于调试目的，帮助开发者检查张量的各种属性信息，包括形状、数据类型、布局等。

参数：

- `attr`：RKNN3 张量属性结构（RKNN3TensorAttr）
- `prefix`：打印前缀字符串，默认为 "tensor"

返回值：无

示例：

```python
from rknn3lite.api.rknn3_types import dump_tensor_attr

# 假设有张量属性对象 attr
dump_tensor_attr(attr, prefix="input")
# 输出示例:
# input_0: name=input_tensor, n_dims=4, shape=[1, 3, 224, 224], layout=NHWC, dtype=FP32, ...
```

5.7.5 get_os_platform

描述：获取当前操作系统的平台字符串。该函数用于确定运行环境的架构（32位或64位），有助于进行平台相关的配置和兼容性检查。

参数：无

返回值：平台字符串，格式为 "{操作系统}_{架构}"，如 "Linux_x64" 或 "Windows_x32"。

示例：

```python
from rknn3lite.api.rknn3_types import get_os_platform

platform = get_os_platform()
print(platform)  # 输出: Linux_x64 (在64位Linux系统上)
```