Rockchip RK1820_RK1828 AI SDK 快速入门

文档标识：RK-JC-YF-A58

发布版本：V1.0.0

日期：2026-01-23

文件密级：□绝密 □秘密 □内部资料 ■公开

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

本文旨在帮助开发者快速熟悉 RK1820/RK1828 系列 AI 协处理器的硬件特性、软件环境、开发流程及性能优化方法，从而高效地将 AI 算法部署到 RK1820/RK1828 的硬件平台上。

产品版本

芯片名称	SDK版本
RK1820/RK1828	Release V1.0.0 版本

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

日期	版本	作者	修改说明
2025-08-15	V0.0.1	Zain Wang, Jair Wu, Hans Yang	初始版本
2025-09-19	V0.0.2	Zain Wang, Jiahang Zheng	Alpha V0.0.2 版本
2025-11-10	V0.4.0	Zack Huang	Beta V0.4.0 版本
2026-01-23	V1.0.0	Zack Huang	Release V1.0.0 版本

目录

Rockchip RK1820_RK1828 AI SDK 快速入门

1. 概述
   - 1.1 开发框架
   - 1.2 支持平台
2. 快速验证
   - 2.1 硬件准备
     - 2.1.1 硬件说明
     - 2.1.2 硬件连接
   - 2.2 固件准备
   - 2.3 模型准备
   - 2.4 环境部署
   - 2.5 模型测试
     - 2.5.1 CNN模型
     - 2.5.2 LLM模型
3. SDK介绍
   - 3.1 SDK目录
   - 3.2 SDK编译环境
   - 3.3 SDK编译
     - 3.3.1 配置
     - 3.3.2 编译
     - 3.3.3 安装
     - 3.3.4 测试
4. Host环境准备
   - 4.1 PCIe驱动安装
     - 4.1.1 驱动文件结构
     - 4.1.2 驱动编译与安装
       - 4.1.2.1 内建编译
       - 4.1.2.2 KO编译
   - 4.2 环境验证
   - 4.3 调试工具
     - 4.3.1 rknn-smi
       - 4.3.1.1 状态监控
       - 4.3.1.2 模组功耗
     - 4.3.2 rknn-console
   - 4.4 Docker
5. 开发流程
   - 5.1 模型转换
   - 5.2 应用开发
     - 5.2.1 C/C++ API
     - 5.2.2 Python API
     - 5.2.3 OpenAI API
6. 参考示例
   - 6.1 CNN模型推理
   - 6.2 LLM模型推理
   - 6.3 VLM模型推理
7. 系统功能介绍
   - 7.1 性能模式
   - 7.2 温控策略
     - 7.2.1 相关命令
   - 7.3 Prefill Mode
   - 7.4 多RK1820/RK1828适配
     - 7.4.1 固件烧录
       - 7.4.1.1 一般情况

7.4.1.2 异常情况
7.4.1.2.1 多 PCIe 连接下固件烧录失败
7.4.1.2.2 多 USB 连接下固件烧录失败
7.4.2 模型推理
7.4.2.1 推理流程
7.4.2.2 关键 API 及结构体
7.4.2.2.1 设备发现
7.4.2.2.2 Context 初始化
7.4.2.3 代码示例
8. 附录
8.1 PC环境ADB安装
8.1.1 Windows 系统
8.1.2 Linux 系统

1. 概述

本 SDK 支持 RK1820/RK1828 协处理器模式，即主控 SoC 通过 PCIe/USB 高速接口连接 RK1820/RK1828 协处理器。

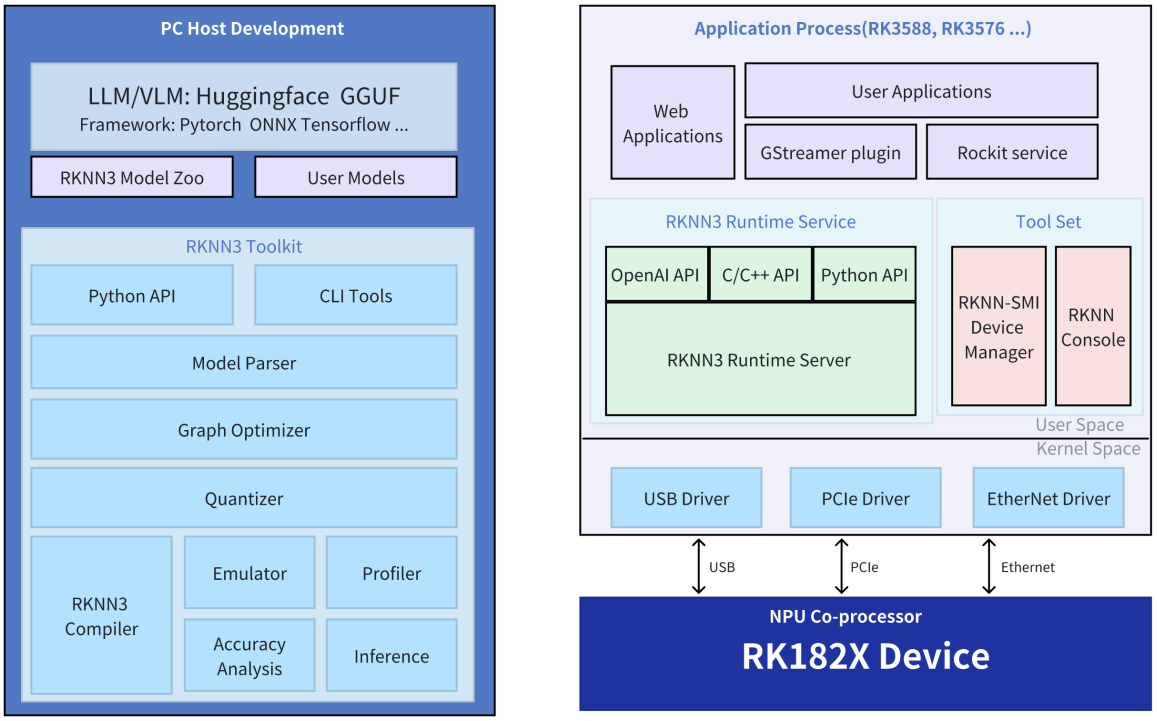
主控 SoC（Host）：作为系统核心，负责任务调度、资源分配和整体控制。

RK1820/RK1828 协处理器（Device）：作为 AI 计算加速单元，专注于高性能神经网络推理任务。

PCIe/USB 高速接口（通信接口）：实现主控与协处理器之间的低延迟、高带宽数据交互。

1.1 开发框架

下图展示了 RK1820/RK1828 平台的开发框架：



- PC 端开发环境，包含 RKNN3 Toolkit，RKNN3 Model Zoo 等。
  - RKNN3 Toolkit：是一个软件开发工具包，可将 PyTorch、ONNX 等深度学习框架训练的模型转换为 RKNN 格式，支持用户进行模型转换、推理和性能评估；
  - RKNN3 Model Zoo：提供了丰富的模型转换示例，涵盖多种 AI 模型类型。
- 板端开发环境，包含 RKNN3 Runtime，AI 应用 Examples，工具集合，驱动等。
  - RKNN3 Runtime：模型转换完成后，可在开发板上使用 RKNN3 API 加载和运行 RKNN 模型。除 RKNN3 API 外，还支持 OpenAI 兼容 API 调用 LLM 模型；
  - Examples：结合实际应用，提供模型应用参考示例；
  - 工具集合：如用于调试的 RKNN-SMI 和 RKNN Console；
  - 驱动：提供 RK1820/RK1828 PCIe EP 驱动。
- RK1820/RK1828 协处理器：提供固件，通过 PCIe/USB 高速接口与主控 SoC 互联。

模块名	功能	部署环境
RKNN3 Toolkit	PC 端开发套件，用户可通过 RKNN3 Toolkit 将 PyTorch、ONNX 等深度学习框架训练的模型转换为 RKNN 格式	PC
RKNN3 Model Zoo	模型转换部署示例，涵盖多种 AI 模型类型，包括 CNN 模型、LLM 模型、VLM 模型	PC
RKNN3 Runtime	板端运行时库，RKNN 硬件加速接口库，使用 RKNN3 API 加载和运行 RKNN 模型	Host
RKNN-SMI	系统级调试和管理工具	Host
RKNN Console	调试终端工具	Host
Drivers	RK1820/RK1828 PCIe EP 驱动，其他接口则为通用驱动	Host
Examples	模型应用参考示例	Host
Firmware	预编译 RK1820/RK1828 固件	Device

1.2 支持平台

SDK版本	Host平台	Device平台（通信接口）
Alpha V0.0.1	RK3588 RK3576	RK1820/RK1828（PCIe）
Alpha V0.0.2	RK3588 RK3576	RK1820/RK1828（PCIe）
Beta V0.4.0	RK3588 RK3576	RK1820/RK1828（PCIe/USB）
Release V1.0.0	RK3588 RK3576	RK1820/RK1828（PCIe/USB）

2. 快速验证

本文档提供基于 RK3588 与 RK1820/RK1828 的完整运行环境，并配套预转换的 RKNN 模型，用户无需编译 SDK 即可高效完成模型推理验证。如希望自行转换模型并编译 SDK 进行测试，可跳过此章节内容。

本文档将以 RK3588_EVB10 搭配 RK1820/RK1828 设备进行开发说明，需要准备以下环境：

- Host：RK3588 EVB10
- Device：以下 RK1820/RK1828 硬件之一
    - SODIMM 模组

    - M.2 模组
- RK3588 EVB10 预编译固件
- RK1820/RK1828 SODIMM 预编译软件安装包（或参考SDK介绍章节编译）

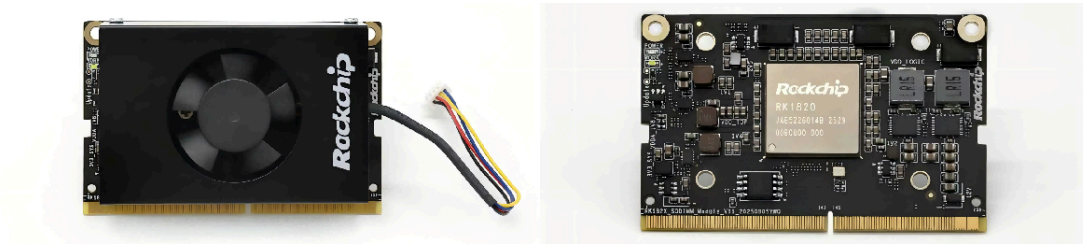
2.1 硬件准备

2.1.1 硬件说明

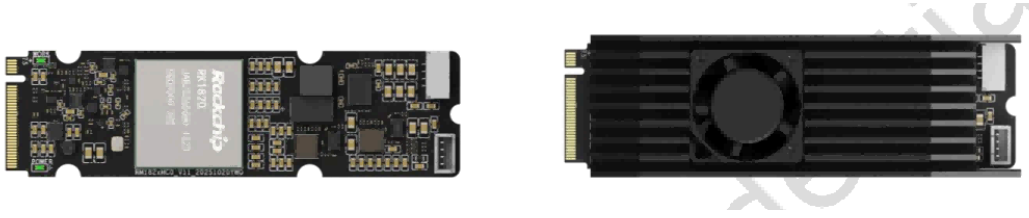
配套该 SDK 的官方参考硬件有：

序号	硬件名	描述	支持的连接方式
1	RK1820/RK1828 SODIMM	RK1820/RK1828 SODIMM 模组，模组采用 260PIN 内存条金手指接口，配套 RK3588 EVB10 做集成验证，也可设计开发底板配套使用，实现快速开发与应用	USB/PCIe
2	RK1820/RK1828 M.2	RK1820/RK1828 M.2 模组	PCIe
3	RK3588 EVB10	RK3588 EVB10 可作为 Host 平台使用	NA

1. RK1820/RK1828 SODIMM 模组实物示意图：



2. RK1820/RK1828 M.2模组示意图：



3. RK1820/RK1828 SODIMM + RK3588 EVB10 组装示意图：

2.1.2 硬件连接

RK1820/RK1828 硬件连接：断电情况下，将 RK1820/RK1828 硬件连接到 RK3588 EVB10 SODIMM 插槽。如 RK182X SODIMM + RK3588 EVB10 组装示意图 所示。

确认 RK1820/RK1828 连接状态：在 RK3588 EVB10 上检查设备连接状态。若为其他 Host 平台，可参考Host环境准备章节安装 PCIe EP 驱动。

```
# RK3588 EVB10 终端执行
root@linaro-alip:/# lspci
0003:31:00.0 Processing accelerators: Rockchip Electronics Co., Ltd Device 182a
(rev 01)
```

2.2 固件准备

从网盘下载 RK3588 EVB10 固件和 RK1820/RK1828 预编译软件安装包。

1. RK3588 EVB10 预编译固件下载地址：https://console.box.lenovo.com/l/7oOghG
2. RK1820/RK1828 预编译软件安装包下载地址：https://console.box.lenovo.com/l/PHk0PF

固件	说明
rknn3_rk182x_sodimm_installer_arm64.tgz	RK1820/RK1828 SODIMM 模组
rknn3_rk182x_sodimm-usb_installer_arm64.tgz	RK1820/RK1828 SODIMM 模组（使用 USB 连接模式）
rknn3_rk182x_m2_installer_arm64.tgz	RK1820/RK1828 M.2 模组

2.3 模型准备

可从网盘下载需要快速验证的模型。

> 网盘链接：https://console.box.lenovo.com/l/H1fig1 提取码: rknn
> 模型位置位于：RKNN3_SDK/rknn3_models/v1.0.0

常用模型：

类型	路径
LLM 模型	RKNN3_SDK/rknn3_models/v1.0.0/Qwen2.5-instruct-0.5B
CNN 模型	RKNN3_SDK/rknn3_models/v1.0.0/MobilenetV2
VLM 模型	RKNN3_SDK/rknn3_models/v1.0.0/InternVL3_2B

2.4 环境部署

RK3588 EVB10 固件烧录：通过 USB 烧录网盘下载的 RK3588 EVB10 固件。

RK1820/RK1828 环境配置：在 RK3588 EVB10 上安装 RK1820/RK1828 SODIMM 预编译套件

```
# PC 端执行
adb push rknn3_rk182x_sodimm_installer_arm64.tgz /tmp/installer.tgz
adb shell "cd /tmp && tar xzf installer.tgz && ./install.sh"
```

> ADB工具安装参考PC环境ADB安装完成
>
> 如果PC端未安装adb工具，可使用其他方式将文件推送至板端，在板端解压后执行安装命令

这将安装以下依赖项：

- RK1820/RK1828 PCIe EP 驱动
- RK1820/RK1828 固件
- RKNN3 运行库
- GStreamer 插件库
- 调试工具集
- RKNN3 开机自启服务

重启 RK3588 EVB10 使依赖项生效，重启后查看 RK1820/RK1828 状态，如果看到以下内容，则表示已成功安装 RK1820/RK1828 依赖项。

```
# RK3588 EVB10 终端执行
# 如果当前非root用户，建议切换到root用户执行
linaro@linaro-alip:/$ [ $(whoami) != "root" ] && sudo su -
root@linaro-alip:/# rknn-smi info
+----------------------+--------------+--------------+----------------------+
| rknn-smi     Version: 1.1.0                                               |
+======================+==============+==============+======================+
| Device               | Health       | Power(mW)    | Npu(%)               |
| Chip          Name   | Bus-Id       | Temp(C)      | Memory-Usage(MB)     |
+======================+==============+==============+======================+
| 0                    | OK           | 1631         | 0                    |
| 0             RK1820 | 0003:31:00.0 | 26           | 31   / 2560          |
+======================+==============+==============+======================+
```

核对版本信息

```
# RK3588 EVB10 终端执行
# 如果当前非root用户，建议切换到root用户执行
linaro@linaro-alip:/$ [ $(whoami) != "root" ] && sudo su -

# 软件版本信息
root@linaro-alip:/# rknn-smi -v
  rknn-smi version             : 1.1.0 # rknn-smi 工具版本
  PCIe driver version           : 3.3.0 # PCIe ep 驱动版本
  RC chips connect version     : 3.2.0 # rc 端 libchips_connect 组件版本
  EP chips connect version     : 3.2.0 # ep 端 libchips_connect 组件版本
  PCIe Device 0 firmware version: 1.0.0 # RK1820/RK1828 设备0 固件版本
  PCIe Device 1 firmware version: 1.0.0 # RK1820/RK1828 设备1 固件版本
  rknn3 API version             : 1.0.0 # RK1820/RK1828 rknn runtime 版本

# 硬件版本信息
root@linaro-alip:/# rknn-smi info -l
  Device Count                  : 2
    Device ID                    : 0
    Communication mode           : PCIe
    Product Name                 : RK1820SO-DIMMModule-G128-I
    Serial Number                : V10SZ531A00014
    Chip Count                   : 0

    Device ID                    : 1
    Communication mode           : PCIe
    Product Name                 : RK1820SO-DIMMModule-G128-I
    Serial Number                : V10SZ531A00015
    Chip Count                   : 0
```

2.5 模型测试

2.5.1 CNN模型

CNN 模型完整验证（以 MobilenetV2 为例，模型位置）：

```
# PC 端执行
# 将模型push到板端
adb push MobilenetV2 /userdata/
```

```
# RK3588 EVB10 终端执行
# 如果当前非root用户，建议切换到root用户执行
[ $(whoami) != "root" ] && sudo su -
# 模型推理命令
cd /userdata/MobilenetV2
rknn3_model_test mobilenetv2-12.rknn mobilenetv2-12.weight '' '' 0x01 10
```

以下日志输出表明模型已成功完成推理：

```
root@rk3588-buildroot:/userdata/cnn/MobilenetV2# rknn3_model_test mobilenetv2-
12.rknn mobilenetv2-12.weight '' '' 0x01 10
Input paths or golden output paths not provided, using random input and skipping
golden comparison
Found 1 RK182X devices
  Device 0: transfer_type=PCIE, id=0000:01:00.0
Info: Only one device found (id=0000:01:00.0), init_extend can be NULL
rknn3_init success, cost 523.398 ms
Device Memory Info: total=19 MB, free=18 MB
  Node 0: total=639 MB, free=639 MB
  Node 1: total=639 MB, free=639 MB
  Node 2: total=619 MB, free=619 MB
  Node 3: total=639 MB, free=639 MB
  Node 4: total=619 MB, free=619 MB
  Node 5: total=619 MB, free=619 MB
  Node 6: total=639 MB, free=639 MB
  Node 7: total=639 MB, free=639 MB
model_len=63680, model=0x25d3080
weight_len=3714048, weight_data=0x7f84ce9010
rknn3_load_model_from_data success, cost 4.909 ms
Core number: 1
model_config:0xf8c0
rknn3_model_init success, cost 28.556 ms
input tensors:
  name=input, n_dims=4, shape=[1, 224, 224, 3], stride=[150528, 672, 3, 1],
aligned_size=150528, layout=NHWC, dtype=UINT8, qnt_type=PER_LAYER_ASYMMETRIC,
scale=0.01866, zero_point=-14
output tensors:
  name=output, n_dims=2, shape=[1, 1000], stride=[1000, 1], aligned_size=1024,
layout=UNDEFINED, dtype=INT8, qnt_type=PER_LAYER_ASYMMETRIC, scale=0.14192,
zero_point=-55
Generating random input data for input 0 (name: input)
Input 0 size check passed: 150528 elements
```

```
Input[0] first values (type: 3):
[0] UINT8: 103
[1] UINT8: 198
[2] UINT8: 105
[3] UINT8: 115
[4] UINT8: 81
[5] UINT8: 255
[6] UINT8: 74
[7] UINT8: 236
[8] UINT8: 41
[9] UINT8: 205
Running model 10 times...
Syncing input[0] to device...
Syncing output[0] from device...
loop: 1/10, model inference cost 4.921 ms
...
loop: 10/10, model inference cost 5.109 ms
All 10 loops completed successfully
rknn3_destroy success, cost 1.975 ms
```

2.5.2 LLM模型

LLM类模型验证（以qwen2.5_0.5b_instruct为例，模型位置）：

```
# PC 端执行
# 将模型push到板端
adb push Qwen2.5-0.5B /userdata/
```

```
# RK3588 EVB10 终端执行
# 如果当前非root用户，建议切换到root用户执行
[ $(whoami) != "root" ] && sudo su -
# 模型推理命令
cd /userdata/Qwen2.5-0.5B
rknn3_session_test Qwen2.5-0.5B.rknn Qwen2.5-0.5B.weight Qwen2.5-
0.5B.tokenizer.gguf Qwen2.5-0.5B.embed.bin 1024 256 0xff
```

LLM模型能够正确对话，表示模型推理成功，如下日志：

```
root@rk3588-buildroot:/userdata/llm/Qwen2.5-0.5B# rknn3_session_test Qwen2.5-
0.5B.rknn Qwen2.5-0.5B.weight Qwen2.5-0.5B.tokenizer.gguf Qwen2.5-0.5B.embed.bin
1024 256 0xff
**********NEW TEST**********
llama_model_loader: loaded meta data with 23 key-value pairs and 0 tensors from
Qwen2.5-0.5B.tokenizer.gguf (version GGUF V3 (latest))
llama_model_loader: Dumping metadata keys/values. Note: KV overrides do not apply
in this output.
...

============================================================
                        Model Config
============================================================
Max Context Length              : 2048
```

```
Max Position Embeddings          : 8192
Model Type                       :
Task Type                        : RKNN3_LLM_TASK_GENERATE
============================================================

--------------------Input[0]--------------------
请解释一下相对论的基本概念。

--------------------Output----------------------
相对论是爱因斯坦在20世纪初提出的物理学理论，它改变了我们对时间、空间和物质的理解。以下是相对论的基本概念：

1. **狭义相对论** (Special Relativity)：
   - **光速不变原理**：所有惯性参考系中的光速都是一样的，无论光源的运动速度如何。
   - **相对性原理**：在不同的惯性参考系中，物理定律是相同的。这意味着时间、空间和长度都是相对的。

2. **广义相对论** (General Relativity)：
   - **引力场**：物质和能量通过引力场影响时空，使物体运动轨迹发生弯曲。
   - **时空曲率**：时空中的曲线反映了物质分布和引力场的强度。这种曲率与物质的质量成正比。

3. **时间膨胀** (Time Dilation)：
   - 物质在加速时会缩短时间，即时间流逝得更快。

4. **空间膨胀** (Space Dilation)：
   - 物质在加速时也会导致空间的弯曲，使距离物体更远的地方看起来变长。

5. **引力红移**：由于物质和能量的影响，光谱线向

--------------Max new token reached-------------

Performance Statistics:
-------------------------------------------------------------------------------------------
--------
 Stage      | Total Time (ms)  | Tokens    | Time per Token (ms)  | Tokens per
Second
-------------------------------------------------------------------------------------------
--------
 Prefill    | 63.28            | 48        | 1.32                 | 758.59
 Generate   | 1520.90          | 255       | 5.96                 | 167.66
-------------------------------------------------------------------------------------------
--------

--------------------Input[1]--------------------
Please explain the basic concept of relativity.

--------------------Output----------------------
Relativity is a set of theories developed by Albert Einstein, which fundamentally
changed our understanding of time, space, and the nature of physical objects.
Here are the key concepts of relativity:

1. **Special Relativity** (Special Relativity):
   - **Einstein's first law**: The laws of physics are the same for all observers
in uniform motion relative to each other.
   - **Einstein's second law**: The speed of light in a vacuum is constant,
regardless of the motion of the light source or observer.
```

```
2. **General Relativity** (General Relativity):
   - **Gravitational Field**: The gravitational field is caused by mass and
energy. This field affects the curvature of spacetime.
   - **Curvature of Space-Time**: The geometry of space-time changes when objects
are accelerated, leading to phenomena like time dilation and length contraction.

3. **Time Dilation** (Time Dilation):
   - When an object accelerates, its clocks run slower compared to a stationary
observer. This effect is known as time dilation.

4. **Space Dilation** (Space Dilation):
   - The same principle applies when objects are accelerated in space. Objects
moving at high speeds will appear to move faster than they actually do, due to

--------------Max new token reached------------

Performance Statistics:
-------------------------------------------------------------------------------
--------
 Stage      | Total Time (ms)  | Tokens    | Time per Token (ms)  | Tokens per
Second
-------------------------------------------------------------------------------
--------
 Prefill    | 35.97            | 17        | 2.12                 | 472.63
 Generate   | 1502.08          | 255       | 5.89                 | 169.76
-------------------------------------------------------------------------------
--------
**********************************END TEST*********************************
```

- 另可参考 `docs/Examples` 目录下的应用文档集成应用进行模型测试

3. SDK介绍

3.1 SDK目录

获取的 SDK 是一个功能完备的软件开发套件，主要包含以下部分：

```
├── build.sh -> scripts/common/build.sh      # SDK 编译脚本
├── docs                                     # SDK 文档
├── driver                                   # 驱动程序，用于 Host 与 RK1820/RK1828
硬件通信
├── examples                                 # 示例代码，覆盖 CNN 和 LLM 等多种应用场
景
│   ├── gstreamer
│   ├── rockit_yolo_detect
│   └── web
├── pkgroot                                  # 预编译固件、工具集
├── rknn                                     # RKNN3 工具集
│   ├── rknn3-model-zoo                      # 提供LLM/VLM/CNN模型转换示例等
```

```
│   ├── rknn3-runtime                    # 运行时库，提供 C++ 和 Python API，供应用程序调用
│   ├── rknn3-toolkit                    # 核心开发工具链，用于模型转换、量化和优化
│   └── rknn-gstreamer-plugins           # RKNN GStreamer 插件源码
└── scripts                              # SDK 脚本集合
```

3.2 SDK编译环境

系统推荐使用 Ubuntu 22.04 或更高版本的系统 PC 或服务器进行编译，文档下述操作均基于 Ubuntu 22.04 验证。

使用如下 apt-get 命令安装后续操作所需的库和工具：

```
sudo apt-get update && sudo apt-get install git make gcc python-is-python3 python2
```

3.3 SDK编译

3.3.1 配置

SDK 默认配置为 SODIMM 模组。

如果配置方案和实际不符，可以通过`./build.sh config`重新配置。

```
Select board type:
1) RK182X EVB1
2) RK182X SODIMM
3) RK182X SODIMM USB
4) RK182X M.2
5) Cancel
#?
```

3.3.2 编译

进入 SDK 根目录，执行编译脚本：

```
# PC端执行
./build.sh
```

生成的软件安装包在:

```
output/firmware/rknn3_rk182x_sodimm_installer_arm64.tgz
output/firmware/rknn3_rk182x_sodimm-usb_installer_arm64.tgz
output/firmware/rknn3_rk182x_m2_installer_arm64.tgz
```

3.3.3 安装

编译出来的软件安装包，可以使用 adb 工具进行安装，命令如下：

```
# PC端执行
cd output/firmware/
adb push rknn3_rk182x_sodimm_installer_arm64.tgz /tmp/installer.tgz
adb shell "cd /tmp && tar xzf installer.tgz && ./install.sh"
```

> 如果PC端没有adb工具，可使用其他方式将文件推送至板端，在板端解压后执行安装命令

安装重启后，Host 端系统会在启动后，自动下载 RK1820/RK1828 固件，并启动后台服务程序。

3.3.4 测试

SDK 提供两个模型测试工具代码，可定制模型加载参数和运行：

名称	简介	代码位置
rknn3_model_test	CNN 模型测试工具	rknn/rknn3-runtime/examples/rknn3_model_test_demo
rknn3_session_test	LLM 模型测试工具	rknn/rknn3-runtime/examples/rknn3_session_test_demo

如果需要自行编译 demo，需要设置工具链并且通过以下命令编译，并打包到软件安装包中：

```
# 编译Android版本测试工具
ANDROID_NDK_PATH="$SDK_PATH/prebuilts/android-ndk/" ./build.sh

# 编译Linux版本测试工具
HOST_TOOLCHAIN_PATH="$pwd/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu" \
HOST_TOOLCHAIN_NAME="aarch64-none-linux-gnu" \
./build.sh

# 同时编译Linux和Android版本测试工具
HOST_TOOLCHAIN_PATH="$pwd/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu" \
HOST_TOOLCHAIN_NAME="aarch64-none-linux-gnu" \
ANDROID_NDK_PATH="$SDK_PATH/prebuilts/android-ndk/" \
./build.sh
```

> 默认 Rockchip Host 平台 SDK 均有集成编译工具链，如 `<host sdk>/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu`。或下载 Rockchip 平台通用工具链 https://meta.zbox.filez.com/v/link/view/583256c6a5d341928a9ea1ff4c41a76c，推荐使用 6.3.1 版本。
>
> Android NDK 可从 https://console.box.lenovo.com/l/e1v3Ny 下载。

4. Host环境准备

4.1 PCIe驱动安装

4.1.1 驱动文件结构

```
.
├── pcie-rkep
│   ├── Makefile
│   ├── pcie-dw-dmatest.c
│   ├── pcie-dw-dmatest.h
│   ├── pcie-rkep.c
│   ├── rk-pcie-ep.h
│   └── rockchip-pcie-dma.h
├── README_CN.md
└── README_EN.md
```

4.1.2 驱动编译与安装

驱动的编译分为两种方式，KO 方式与内建方式，可根据实际情况选择合适的编译方式。

4.1.2.1 内建编译

若您使用的是 Rockchip SoC 的官方 SDK，请检查 SDK 是否已支持 PCIe EP 驱动，Kernel 仓库下是否已集成如下提交：

```
commit 9edd3c389717c176d9c25bc1cad73dcd66ae2e33 (HEAD)
Author: Jon Lin <jon.lin@rock-chips.com>
Date:   Tue Nov 11 20:01:52 2025 +0800

    misc: rockchip: pcie-rkep: Add another DMA finished detection mechanism

    If ep enables DMA interrupt management but uses MSI to notify rc, this
    method is needed to obtain interrupt completion information.

    Change-Id: I1ea024cfaec8d5a42290fa4fe2c0943776ff9cb2
    Signed-off-by: Jon Lin <jon.lin@rock-chips.com>
```

若已有相关补丁，开启以下开关以开启相关功能支持：

```
# 启用 Rockchip PCIe EP 功能支持
CONFIG_PCIE_FUNC_RKEP=y
CONFIG_PCIE_DW_DMATEST=y
```

4.1.2.2 KO编译

若 Kernel 仓库下没有相关提交，则可以使用 SDK 根目录下 driver/pcie-rkep 驱动代码编译 KO 文件。

1. 修改驱动目录下的 `Makefile`，指定已编译好的内核源码路径：

```
obj-m := pcie-rkep.o                          # 目标驱动模块名
KERNELDIR := /path/to/your/host/kernel        # 指向已编译的内核源码目录
PWD := $(shell pwd)                           # 当前驱动源码路径

modules:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules # 编译模块

clean:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) clean   # 清理编译输出
```

2. 使用交叉编译工具链进行编译（根据目标架构替换 `CROSS_COMPILE` 和 `ARCH`）：

```
# PC端执行
make CROSS_COMPILE=aarch64-none-linux-gnu- ARCH=arm64
```

- `CROSS_COMPILE`: 指定交叉编译器前缀 (e.g., `aarch64-none-linux-gnu-`)
- `ARCH`: 指定目标 CPU 架构 (e.g., `arm64`)

3. 成功编译后，在当前目录生成驱动模块文件：

```
$ file pcie-rkep.ko
pcie-rkep.ko: ELF 64-bit LSB relocatable, ARM aarch64, version 1 (SYSV),
BuildID[sha1]=... , with debug_info, not stripped
```

4. 将编译生成的 `pcie-rkep.ko` 驱动文件部署至：`/usr/lib/modules/pcie-rkep.ko`，加载驱动：

```
adb push pcie-rkep.ko /usr/lib/modules/pcie-rkep.ko
insmod /usr/lib/modules/pcie-rkep.ko
```

5. 驱动加载成功后，系统将创建设备节点：

```
root@rk3588-buildroot:~# ls -l /dev/pcie-rkep-*
crw------- 1 root root 10, 121 Jan  1 12:02 /dev/pcie-rkep-0001:11:00.0
```

6. 系统启动过程中，`rknn3_startup` 脚本会在执行 RK1820/RK1828 固件下载操作前，自动完成驱动安装，因此后续启动将不需要再手动加载 KO 模块

4.2 环境验证

RK1820/RK1828 启动后，可以使用 `rknn-smi` 来获取信息：

```
root@linaro-alip:/# rknn-smi info
+-----------------------+--------------+--------------+----------------------+
| rknn-smi     Version: 1.1.0                                                 |
+=======================+==============+==============+======================+
| Device                | Health       | Power(mW)    | Npu(%)               |
| Chip          Name    | Bus-Id       | Temp(C)      | Memory-Usage(MB)     |
+=======================+==============+==============+======================+
| 0                     | OK           | 1631         | 0                    |
| 0             RK1820  | 0003:31:00.0 | 26           | 31   / 2560          |
+=======================+==============+==============+======================+
```

4.3 调试工具

由于调试工具需要 root 权限操作硬件设备，建议直接切换为 root 用户，以避免在调试中遇到权限问题导致命令执行失败，可使用命令`sudo su -`切换为 root 用户。

4.3.1 rknn-smi

rknn-smi (System Management Interface) 工具用于 RK1820/RK1828 设备信息收集、功能配置、日志管理等功能，且支持多 RK1820/RK1828 同时连接。

完整功能使用参考`docs/tools/Rockchip_User_Guide_RKNN-SMI_Tool_CN.pdf`。

4.3.1.1 状态监控

```
# 打印所有RK1820/RK1828设备信息
root@rk3588-buildroot:/# rknn-smi info -w
Device(Idx)  Chip(Idx) Power(mW)   Temp(C)   CPU(%)   NPU(%)   Memory(%)
Health
0            0              NA        41       0%       0%           2%     OK
1            0              NA        43       0%       0%           2%     OK
0            0              NA        41       0%       0%           2%     OK
1            0              NA        43       0%       0%           2%     OK
# ... 持续打印
```

多设备场景可通过`-d <Device ID>`指定目标设备（Device ID 可由`rknn-smi info -l`获取）。

```
# 打印特定RK1820/RK1828设备信息
root@rk3588-buildroot:/# rknn-smi info -w -d 1
Device(Idx)  Chip(Idx) Power(mW)   Temp(C)   CPU(%)   NPU(%)   Memory(%)
Health
1            0              NA        43       0%       0%           2%     OK
1            0              NA        44       0%       0%           2%     OK
1            0              NA        43       0%       0%           2%     OK
# 持续打印
```

4.3.1.2 模组功耗

使用 RK1820/RK1828 EVB1 或 RK3588 EVB10 搭配 RK1820/RK1828 SODIMM 模组时，可通过 rknn-smi 工具查询模组功耗。

```
root@linaro-alip:/# rknn-smi info -t power
  Device ID                     : 0
  Device Power (mW)             : 1681
```

> **注：** RK1820/RK1828 EVB1 **已板载功率传感器**。对于 RK3588 EVB10，请注意仅 **靠近屏幕方向** 的 SODIMM 插槽配备有功率传感器。

4.3.2 rknn-console

rknn-console 工具用于在 HOST 端通过 PCIe 和 USB 来模拟RK1820/RK1828端的控制台，实现对 RK1820/RK1828 的控制和调试

```
root@linaro-alip:/# rknn-console --help    # 查看可用命令
Usage: rknn-console [OPTION]... SOC [shell] [COMMAND]
Commands:
    rknn-console -h                     Show this help message
    rknn-console rk1820 -l              list device
    rknn-console rk1820 -d device_id    Connect to device
        rknn-console rk1820 -d 0        Connect to device_id 0
    rknn-console rk1820 shell "ls -l"   Execute command on rk1820 and exit
    rknn-console --version              Display version

root@linaro-alip:/# rknn-console rk1820 -l    # 查询可用设备
Found PCIe device: 0001:11:00.0 -> dev_t=0
Found USB device: /dev/ttyACM1 -> dev_t=1
Found USB device: /dev/ttyACM3 -> dev_t=2

Available devices:
================================
DEVICE ID: 2 (USB): /dev/ttyACM3
DEVICE ID: 1 (USB): /dev/ttyACM1
DEVICE ID: 0 (PCIE): 0001:11:00.0
================================
```

在多 RK1820/RK1828 连接时需加上`-d <Device ID>`来指定连接的设备。

```
root@linaro-alip:/# rknn-console rk1820 -d 0    # 连接 device id 0
18:47:03.824 D [hugepage_mem: hugepage_mem_init: 183] huagepage mem init success
hugepage_manager 0x3ef0f460
PCIE VUART RC (Version 0x10000000) started
Line buffered mode: Input will be sent to EP on newline
Press Ctrl+X to exit
18:47:03.825 D [cmp_pcie_rc: pcie_task_rc_msg_t: 293] [0001:11:00.0] start
RX polling thread started

msh />help              # RK1820/RK1828 可执行命令
```

执行`Ctrl+X`退出控制台

```
msh />
Ctrl+X pressed, exiting...
RX polling thread exiting
17:04:20.064 D [cmp_pcie_rc: pcie_task_rc_msg_t: 321] [0001:11:00.0] end
PCIE VUART RC application terminated
```

4.4 Docker

如果需要在 Host 端搭建 Docker 容器去运行 RK1820/RK1828 相关服务，可参考`examples/docker/README_CN.md`。

5. 开发流程

RK1820/RK1828 支持模型、模型性能、模型精度、使用限制、性能测试、精度测试方案，可参考`rknn/rknn3-runtime/doc/00_Rockchip_RK182X_ReleaseNote_RKNN3_SDK_V1.0.0_CN.pdf`

RK1820/RK1828 模型部署流程可参考`rknn/rknn3-runtime/doc/01_Rockchip_RK182X_Quick_Start_RKNN3_SDK_V1.0.0_CN.pdf`。

5.1 模型转换

模型转换工具路径为`rknn/rknn3-toolkit/`，部署时可参考该目录下的文档。强烈建议使用提供的 Dockerfile 来配置 Docker 环境，以减少环境搭建过程中可能出现的问题。Dockerfile 文件路径为`rknn/rknn3-toolkit/docker/docker_file/`。

在 rknn3-model-zoo 内 examples 各自目录下均有提供转换示例代码，如`rknn/rknn3-model-zoo/examples/mobilenet/python/convert.py`。CNN 模型转换步骤如下：

```
# 进入模型目录下，先下载模型
$ cd rknn/rknn3-model-zoo/examples/mobilenet/python/model
$ ./download_model.sh
# 再进入python目录进行转换
$ cd ../python
$ python3 convert.py
```

大语言模型转换可参考`rknn/rknn3-model-zoo/README.md`。

更多模型转换说明可参考`rknn/rknn3-toolkit/doc/02_Rockchip_RKNPU_API_Reference_RKNN3_Toolkit_V1.0.0_CN.pdf`。

5.2 应用开发

5.2.1 C/C++ API

可参考 rknn3-model-zoo 的 examples 各自目录下的 cpp 目录进行开发，如`rknn/rknn3-model-zoo/examples/mobilenet/cpp/`。

完整的API说明可参考`rknn/rknn3-runtime/doc/Rockchip_RKNPU_API_Reference_RKNNRT3_V1.0.0_CN.pdf`。

5.2.2 Python API

完整的 API 说明可参考 `rknn/rknn3-runtime/doc/05_Rockchip_User_Guide_RKNN3_Toolkit_Lite_V1.0.0_CN.pdf`

5.2.3 OpenAI API

OpenAI API 能力通过 rkllm3-server 提供，具体可参考`rknn/rknn3-runtime/rkllm3-server/README_CN.md`。

6. 参考示例

6.1 CNN模型推理

C/C++ 接口基础测试可参考`rknn/rknn3-runtime/examples/rknn3_model_test_demo/`。

GStreamer 应用可参考`docs/Examples/Rockchip_Developer_Guide_RKNN_Stream_CN.pdf`。

6.2 LLM模型推理

C/C++ 接口基础测试可参考`rknn/rknn3-runtime/examples/rknn3_session_test_demo/`。

OpenAI API 应用可参考`docs/Examples/Rockchip_User_Guide_RKNN_RKLLM3_Server_DEMO_CN.pdf`。

6.3 VLM模型推理

C/C++ 接口应用可参考`rknn3-model-zoo/examples`目录各自cpp子目录，如`rknn/rknn3-model-zoo/examples/FastVLM/cpp/`。

OpenAI API 应用可参考`docs/Examples/Rockchip_User_Guide_RKNN_RKLLM3_Server_DEMO_CN.pdf`。

7. 系统功能介绍

7.1 性能模式

可通过 rknn-smi 调整RK1820/RK1828 NPU 性能档位，支持 0-EFFICIENT, 1-NORMAL, 2-PERFORMANCE。

板型/性能档位	EFFICIENT	NORMAL	PERFORMANCE
RK1820/RK1828 SODIMM Module	500M	850M	1000M
RK1820/RK1828 M.2 Module	400M	500M	850M

```
# 例如调整到NORMAL模式
root@linaro-alip:/# rknn-smi set -t work_mode -s 1
Successfully set device 0 chip 0 work mode to 1
```

在多 RK1820/RK1828 连接时需加上`-d Device ID`来指定设备设置。

```
root@rk3588-buildroot:/# rknn-smi set -t work_mode -s 1 -d 1
Successfully set device 1 chip 0 work mode to 1

# 未加-d Device ID 则报错
root@rk3588-buildroot:/# rknn-smi set -t work_mode -s 1
Device ID is required if multiple devices are online.
```

7.2 温控策略

RK1820/RK1828 模组默认采用风扇主动散热方案，通过软件温控策略精准调节系统运行状态，保障设备在全工况下的稳定性与可靠性。

随着系统温度升高，温控策略将逐级触发风扇调速动作；当温度达到临界阈值时，将进一步启动 NPU 降频机制，实现分层级热管理。

Tmargin(℃)	Fan Speed Ratio(%)	NPU Max Freq(MHz)
0	0	1000 [1]
46	**51**	1000 [1]
60	**63**	1000 [1]
68	**78**	1000 [1]
76	**100**	1000 [1]
83	100	**850**
90	100	**500**
100 [2]	--	--

> [1]RK1820/RK1828 M.2 最高频率不超过850M。
>
> [2]当系统温度超过 100℃ 时，芯片存在运行不稳定风险，系统将立即触发**强制重启**，以保护硬件安全。

参数释义：

- Tmargin：表示温度阈值，当实时温度超过该阈值时，风扇转速比率（Fan Speed Ratio）与 NPU 最高频率（NPU Max Freq）将同步切换至对应档位。
- Fan Speed Ratio：表示风扇转速比率。当前转速占最大转速的比率。
- NPU Max Freq: 表示 NPU 允许运行的最高工作频率。在硬件能力范围内，频率越高，NPU 数据处理性能越强。

温控调节采用滞回控制逻辑，避免因温度小幅波动导致频繁档位切换，具体规则如下：

- 升温触发：当系统温度上升至 `Tmargin` 阈值时，立即执行对应档位的参数配置。
- 降温恢复：当系统温度下降至 `Tmargin - 7` 阈值时，自动切换至上一档位的参数配置。

> 调节示例（以 46℃ 温度阈值档位为例）：
>
> - 升温阶段：当实时温度从低温区间上升并超过 46℃ 时，风扇转速将立即调整为 51%；
> - 降温阶段：经散热后，当实时温度下降至 39℃（46-7） 时，风扇转速将恢复至 0%。

7.2.1 相关命令

Host 环境下，可以通过以下命令，获取温控相关参数。

```
rknn-smi info -t temp       # 获取当前系统温度
rknn-smi info -t fan        # 获取当前风扇转速
rknn-smi info -t npu_freq   # 获取 NPU 频率信息
```

7.3 Prefill Mode

LLM 模型推理 Prefill 阶段的峰值电流较大。通过调整 Prefill Mode，可以对此峰值电流进行抑制与优化。

模型	NPU频率 (MHz)	NPU 平均电流 (A)	NPU 峰值电流 (A)	整机平均电流 (A)	整机峰值电流 (A)	TTFT (ms)	TPS (Token/s)	Prefill Mode
Qwen2.5 3B LLM	1000	7.25	28.37	0.916	2.7	341.5	97.4	PERFORMANCE
Qwen2.5 3B LLM	1000	7.12	15.35	0.866	1.61	582.9	97.5	EFFICIENT

> Prefill Mode
>
> EFFICIENT： Prefill阶段NPU频率是Decode阶段的一半
>
> PERFORMANCE：Prefill阶段和Decode阶段 NPU频率同频（默认模式）

查询设备的 Prefill 模式

```
rknn-smi info [-t | --type] prefill_mode
```

设置设备的 Prefill 模式

```
rknn-smi set -t prefill_mode -d <id> -c <chip_id> -s <prefill_mode>
```

> prefill_mode： 0 --EFFICIENT，1 --PERFORMANCE
>
> 当只有1个设备/芯片在线时，可以不指定 `[-d | --device] <id> / [-c | --chip] <chip_id>` 参数

7.4 多RK1820/RK1828适配

7.4.1 固件烧录

7.4.1.1 一般情况

1. 确认拿到的 RK1820/RK1828 开发板的配置方案，如板卡类型：SODIMM、EVB1，连接方式：PCIe、USB。
2. 目前安装包对于不同板卡类型以及连接方式并不兼容，因此若需要在 HOST 上连接多块 RK1820/RK1828 请保证配置方案一致。
3. 参考快速验证章节，从网盘下载对应配置的软件包，并安装。或参考配置和编译章节，编译安装对应配置的软件包。
4. 固件烧录脚本已集成于安装包，开机时或检测到 USB 插入时自动触发，USB 和 PCIe 的烧录时机略有不同：

- PCIe：开机阶段通过 `/bin/rknn3_startup start` 执行烧录。
- USB：通过 udev 规则在检测到设备插入时调用 `rknn3_usb_startup.sh LocationID` 执行烧录。

5. 重启开发板使依赖项生效，重启后查看 RK1820/RK1828 状态，若看到以下内容，则表示已成功安装依赖项：

```
# 开发板终端运行
root@linaro-alip:/# rknn-smi info
+----------------------+--------------+--------------+----------------------+
| rknn-smi     Version: V1.1.0
  |
+======================+==============+==============+======================+
| Device                | Health       | Power(mW)    | Npu(%)               |
| Chip          Name    | Bus-Id       | Temp(C)      | Memory-Usage(MB)     |
+======================+==============+==============+======================+
| 0                     | OK           | 3023         | 0                    |
| 0             RK1820  | NA           | 31           | 29   / 2560          |
+======================+==============+==============+======================+
| 1                     | OK           | 4222         | 0                    |
| 0             RK1820  | NA           | 37           | 29   / 2560          |
+======================+==============+==============+======================+
```

6. 对于 USB 连接的 RK1820/RK1828，首先连接时需要注意连接在 HOST 端的 USB3.0 接口，其次需要在设备上电时确认是否为 USB 3.0（SuperSpeed）：

```
# 开发板终端运行
root@linaro-alip:/# dmesg |grep SuperSpeed
[   17.530377] usb 2-1.1.2: new SuperSpeed USB device number 8 using xhci-hcd
```

7.4.1.2 异常情况

7.4.1.2.1 多 PCIe 连接下固件烧录失败

现象：通过 PCIe 连接多个 RK1820/RK1828，开机后风扇持续转动。

排查步骤：

1. 确认安装包为最新版本。
2. 采集信息用于定位:

- Debian 文件系统：

```
# 开发板终端运行
# 1. 先检查服务状态
systemctl status rknn3

# 2. 如果状态异常，查看详细日志
journalctl -u rknn3 -f
```

- Buildroot 文件系统：

```
# 开发板终端运行
# pcie_upgrade_tool ld 会显示所有通过PCIe连接的RK1820/RK1828的地址信息
```

```
root@rk3588-buildroot:/# pcie_upgrade_tool ld
Program directory: /usr/bin/
List of connected rkep devices
Addr=0003:31:00.0 [1d87:182a]
Addr=0001:11:00.0 [1d87:182a]

# 手动执行启动脚本
root@rk3588-buildroot:/# /bin/rknn3_startup start
find device 0003:31:00.0
Program directory: /usr/bin/
Downloading firmware...
Device boot state: 0xffffffff
Device is not in maskrom mode or loader mode
find device 0001:11:00.0
Program directory: /usr/bin/
Downloading firmware...
Device boot state: 0xffffffff
Device is not in maskrom mode or loader mode
```

3. 若无法分析对应问题，可将上述收集到的信息咨询对应技术支持工程师。

7.4.1.2.2 多 USB 连接下固件烧录失败

现象：通过 USB 连接多个 RK1820/RK1828，开机后风扇持续转动。

排查步骤：

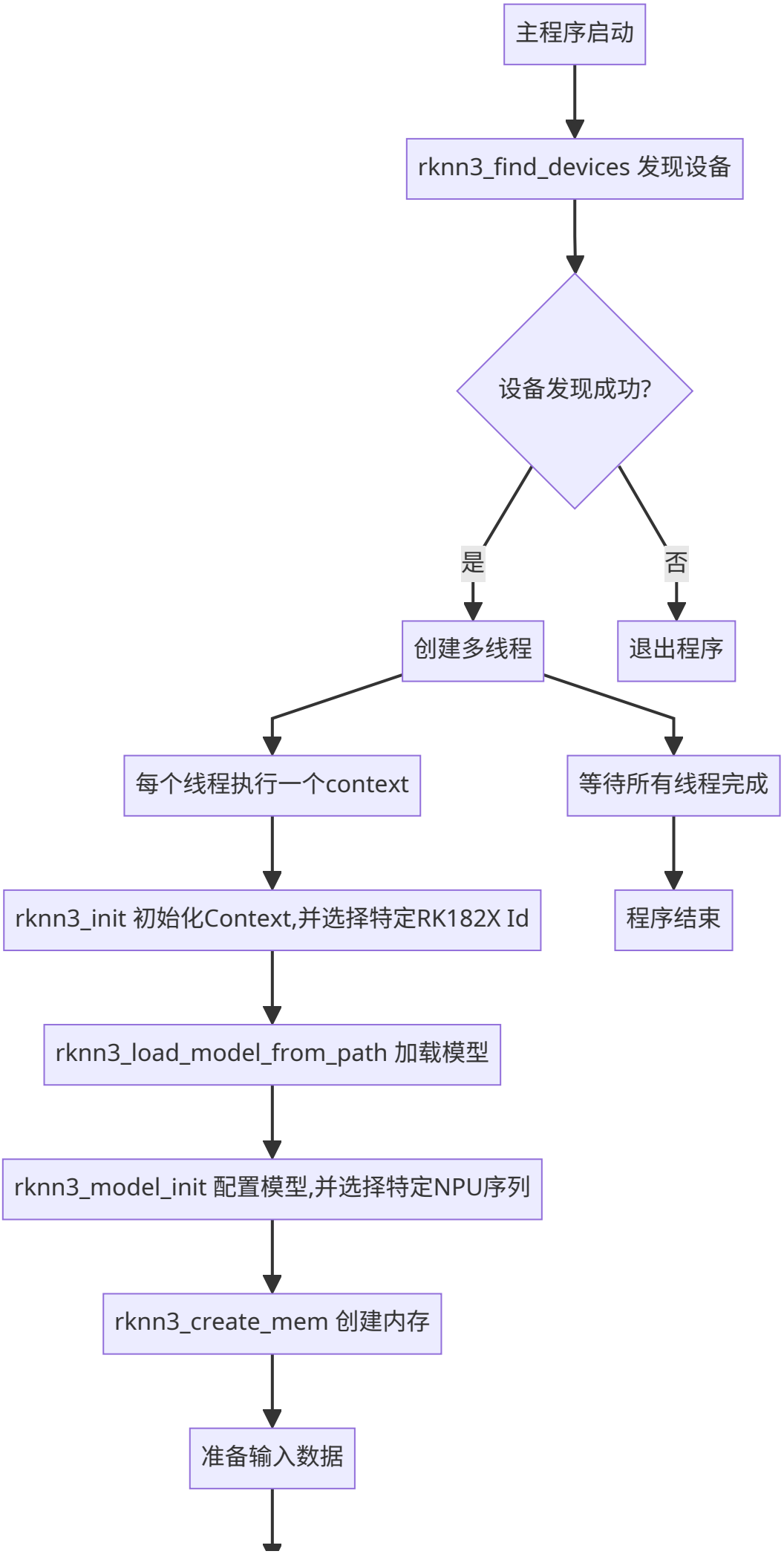
1. 确认安装包为最新版本。
2. 长按 MASKROM 按钮并同时按 Reset，使 RK1820/RK1828 进入 MASKROM 模式。
3. 采集信息用于定位：

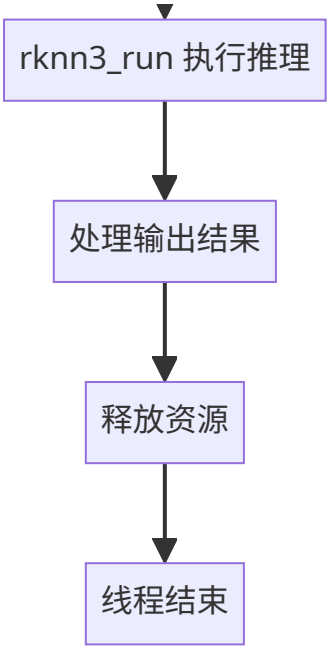
```
# 开发板终端运行

# upgrade_tool ld 会列出处于 MASKROM 模式的 USB 连接设备，应可见 Mode=Maskrom 与
LocationID
root@rk3588-buildroot:/# upgrade_tool ld
List of rockusb connected(1)
DevNo=1 Vid=0x2207,Pid=0x180b,LocationID=1111   Mode=Maskrom    SerialNo=rockchip

# 通过rknn3_usb_startup.sh <LocationID> 烧录对应设备
root@rk3588-buildroot:/# /bin/rknn3_usb_startup.sh 1111

# 查看烧录日志(/tmp/usb_upgrade_tool.log)
root@rk3588-buildroot:/# cat /tmp/usb_upgrade_tool.log
```

7.4.2 模型推理

本节说明多协处理器/多 context 并行开发要点。可参考 `rknn/rknn3-model-zoo/examples/` 下各示例的 cpp 目录进行开发。完整 API 参考 `rknn/rknn3-runtime/doc/` 下各文件。

7.4.2.1 推理流程


主程序启动
rknn3_find_devices 发现设备
设备发现成功?
是
否
创建多线程
退出程序
每个线程执行一个context
等待所有线程完成
rknn3_init 初始化Context,并选择特定RK182X Id
程序结束
rknn3_load_model_from_path 加载模型
rknn3_model_init 配置模型,并选择特定NPU序列
rknn3_create_mem 创建内存
准备输入数据



7.4.2.2 关键 API 及结构体

7.4.2.2.1 设备发现

rknn3_devices

包含 RKNN3 设备信息的结构体。该结构体包含了可用 RKNN3 设备的数量和每个设备的信息。

成员变量	数据类型	含义
n_devices	uint32_t	设备数量
devices	rknn3_device	设备信息

rknn3_device

表示 RKNN3 设备的结构体。该结构体包含了特定 RKNN3 设备的信息，包括其 ID、类型、NPU 核心数量和内存信息。

成员变量	数据类型	含义
id	char[RKNN3_MAX_DEV_LEN]	设备ID
type	char[RKNN3_MAX_DEV_LEN]	设备类型
core_num	uint32_t	NPU核心数量
mem_info	rknn3_dev_mem_info	设备的内存信息

rknn3_find_devices

功能	获取可用的 RKNN3 设备列表。
参数	rknn3_devices* pdevs：将接收设备列表的结构的指针
返回值	int 状态码：详见上述 API 文档
注意	此函数用系统上所有可用的 RKNN3 设备的信息填充提供的 rknn3_devices 结构

7.4.2.2.2 Context 初始化

rknn3_init_extend

设备特定初始化信息结构体。该结构体用于在 RKNN3 运行（rknn3_init 调用）时上下文初始化期间指定设备特定参数，包括设备 ID 和为将来使用保留的空间。

成员变量	数据类型	含义
device_id	char*	输入参数，指示选择哪个设备。如果只连接了一个设备，可以设置为 NULL
reserved	uint8_t	保留字段

rknn3_init

该接口用于初始化 RKNN3 运行时上下文，必须在加载和运行模型前调用。context 为输出参数，init_extend 可用于指定设备 ID 等扩展信息。初始化完成后，需在不再使用时调用 `rknn3_destroy` 释放资源。多次初始化需确保资源正确释放，避免内存泄漏。

功能	初始化一个新的 RKNN3 运行时上下文，该上下文用于在 Rockchip NPU 硬件上运行 RKNN3 模型
参数	rknn3_context* context：指向将被初始化的 RKNN3 上下文句柄的指针 rknn3_init_extend* init_extend ： 指向设备特定初始化信息的指针
返回值	int 状态码：详见上述 API 文档
注意	使用完成后，请调用 rknn3_destroy(context) 释放上下文资源

rknn3_config

模型加载的控制参数结构体。

成员变量	数据类型	含义
priority	int32_t	运行优先级
run_timeout	uint32_t	运行超时时间（毫秒），0 表示无超时
run_core_mask	uint32_t	模型执行的核心掩码
user_mem_weight	uint8_t	权重内存是否由用户分配
user_mem_internal	uint8_t	内部内存是否由用户分配
user_sram	uint8_t	SRAM 内存是否由用户分配
reserved	uint8_t	保留字段

在配置`run_core_mask`可以使用下面枚举内容进行连续或运算来赋值：

```
/**
 * @brief the mode of running on target NPU core.
 */
typedef enum _rknn3_core_mask
{
  RKNN3_NPU_CORE_AUTO = 0,           /* default, run on NPU core randomly. */
  RKNN3_NPU_CORE_0    = 1 << 0,      /* run on NPU core 0. */
  RKNN3_NPU_CORE_1    = 1 << 1,      /* run on NPU core 1. */
  RKNN3_NPU_CORE_2    = 1 << 2,      /* run on NPU core 2. */
  RKNN3_NPU_CORE_3    = 1 << 3,      /* run on NPU core 3. */
  RKNN3_NPU_CORE_4    = 1 << 4,      /* run on NPU core 4. */
  RKNN3_NPU_CORE_5    = 1 << 5,      /* run on NPU core 5. */
  RKNN3_NPU_CORE_6    = 1 << 6,      /* run on NPU core 6. */
  RKNN3_NPU_CORE_7    = 1 << 7,      /* run on NPU core 7. */
  RKNN3_NPU_CORE_ALL  = 0xffffffff, /* auto choice, run on NPU cores depending on
platform */
} rknn3_core_mask;
```

7.4.2.3 代码示例

1. 枚举已连接的 RK1820/RK1828，代码参考如下：

```
// 获取目前连接的RK1820/RK1828信息
rknn3_devices devices;
memset(&devices, 0 ,sizeof(devices));
int ret = rknn3_find_devices(&devices);
if (ret != RKNN3_SUCCESS || devices.n_devices <= 0) {
    printf("find devices failed: %d\n", ret);
    return -1;
}

// devices.n_devices代表了连接的RK1820/RK1828设备数量
// devices.devices[i].id后续用于rknn3_init设置特定的RK1820/RK1828设备
printf("find %d NPU devices:\n", devices.n_devices);
for (uint32_t i = 0; i < devices.n_devices; i++) {
    printf("  device %d: ID=%s, type=%s\n",
```

```
            i, devices.devices[i].id, devices.devices[i].type);
}
```

2. 指定协处理器并初始化 context，代码参考如下：

```
// 指定对应协处理器来执行当前context
// 将想要设置的device id设置到init_extend.device_id，下面是设置了第一个设备的device.id
rknn3_init_extend init_extend = {0};
init_extend.device_id = devices.devices[0].id;

int ret = rknn3_init(&ctx->context, &init_extend);
if (ret != RKNN3_SUCCESS) {
    printf("Context %d init failed: %d\n", ctx->context_id, ret);
    return NULL;
}
```

3. 指定特定 NPU 核心执行context，代码参考如下：

```
// 模型加载（略）
// 通过RKNN3_QUERY_CORE_NUMBER查询加载的模型需要配置几个核心
uint32_t core_num = 0;
ret = rknn3_query(ctx, RKNN3_QUERY_CORE_NUMBER, &core_num, sizeof(core_num));
if (ret != RKNN3_SUCCESS) {
   printf("Context %d rknn3_query failed: %d\n", ctx->context_id, ret);
   rknn3_destroy(ctx->context);
   return NULL;
}
rknn3_config config = {0};

// 根据模型要求的核心数量，从CORE_0开始设置run_core_mask
for (int i = 0; i < core_num; i++) {
   config.run_core_mask |= 1 << i;
}

// 如果有自己定义的core_mask，则需要检查设置核心的数量是否和模型要求的一致
int core_count = 0;
for (int i = 0; i < 32; i++) {
   if (self_set_core_mask & (1 << i)) {
      core_count++;
   }
}
if (core_count == core_num) {
   config.run_core_mask = self_set_core_mask;
}

ret = rknn3_model_init(ctx->context, &config);
if (ret != RKNN3_SUCCESS) {
   printf("Context %d model init failed: %d\n", ctx->context_id, ret);
   rknn3_destroy(ctx->context);
   return NULL;
}
```

8. 附录

8.1 PC环境ADB安装

8.1.1 Windows 系统

1. 下载 ADB 工具包
   点此下载 SDK Platform Tools
2. 解压文件
   将压缩包解压到 `C:\platform-tools`（或其他路径）
3. 配置环境变量
   - 右键 `此电脑` → `属性` → `高级系统设置` → `环境变量`
   - 在 `Path` 中添加解压路径（例如 `C:\platform-tools`）
4. 验证安装
   打开命令提示符（CMD/PowerShell），执行：

   ```
   adb --version
   ```

   安装成功标志：显示 `Android Debug Bridge version x.x.x`

8.1.2 Linux 系统

1. 安装 ADB 工具包

```
sudo apt update && sudo apt-get install adb
```

2. 验证安装，安装成功标志：显示版本号 `x.x.x`

```
adb --version
```