RK182X RKNN3 SDK Quick Start Guide

Document ID: RK-JC-YF-416

Version: V1.0.0

Date: 2026-1-1

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.

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 provides a detailed guide for new users on how to use the RKNN3 Toolkit to perform AI model conversion on a PC and deploy them to a Rockchip development board equipped with an RK1820/RK1828 coprocessor.

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
V0.1.0	HPC	2025-05-13	Initial version	Vincent
V0.2.0	HPC	2025-08-15	1. Adjusted model loading interface description 2. Added model initialization interface description 3. Adjusted model deployment flow chart 4. Added "Prepare Development Board" 5. Added "Environment Setup" 6. Added "Run Example Program" 7. Modified development workflow 8. Added FAQ	Vincent
V0.3.0	HPC	2025-09-10	Updated changes to LLM session-related interfaces	Vincent
V0.3.0b0	HPC	2025-09-12	1. Corrected some errors in the document 2. Added figures 3. Added instructions for using the OpenAI interface	Vincent
V0.4.0b0	HPC	2025-10-18	1. add RK1828 platform	Vincent
V0.5.0	HPC	2025-11-29	1. add Synchronize Input and Output Data	Vincent
V1.0.0	HPC	2026-01-01	1.update rkllm3-server usage 2.update supported model frameworks 3.update model evaluation methods	Vincent

Table of Contents

1 Overview

This document introduces the RKNN3 SDK and explains how to use it to develop and deploy AI models to the RK1820/RK1828 coprocessor. The RKNN3 SDK provides a complete AI model deployment solution, including a PC development kit (RKNN3 Toolkit), an on-board runtime library (RKNN3 Runtime), and model conversion and deployment examples (RKNN3 Model Zoo). This SDK supports the RK1820/RK1828 in coprocessor mode, where a host SoC is connected to the RK1820/RK1828 coprocessor via a high-speed PCIe/USB interface.

- **Host SoC**: Acts as the system's core, responsible for task scheduling, resource allocation, and overall control.
- **RK1820/RK1828 Coprocessor**: Serves as an AI computation acceleration unit, focusing on high-performance neural network inference tasks.
- **PCIe/USB High-Speed Interface**: Enables low-latency, high-bandwidth data interaction between the host and the coprocessor.

1.1 RKNN3 SDK Overall Software Framework

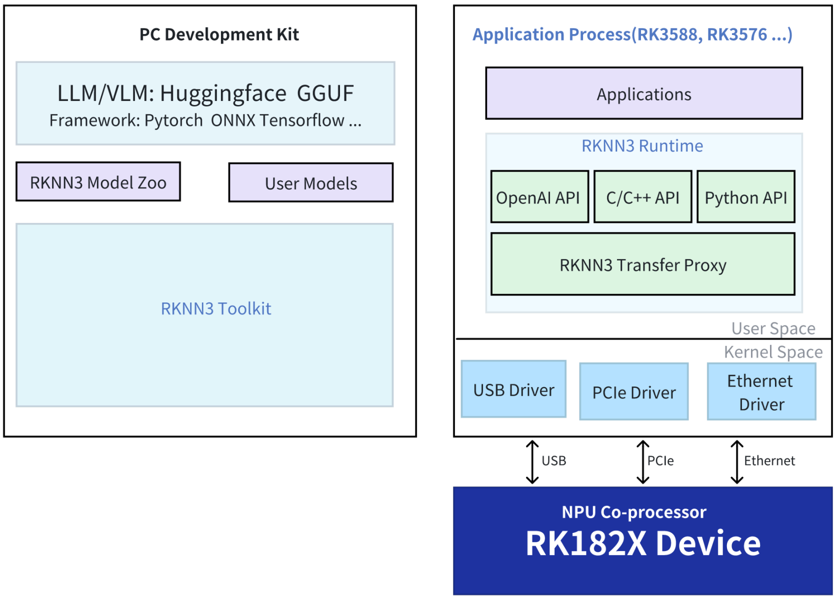


Figure 1-1 RKNN3 SDK Block Diagram

The software framework mainly consists of two core components:

1. PC-side Development Kit

On the PC, users can use the RKNN3 Toolkit to convert models trained with deep learning frameworks like PyTorch and ONNX into the RKNN format. The RKNN3 Model Zoo provides a rich set of model conversion examples, covering various AI model types:

- **CNN Models**: MobileNetV2, ResNet50, YOLOv5, YOLOv6, YOLOv8, etc.
- **LLM Models**: Qwen 2.5 0.5B, 1.5B, 3B, etc.
- **VLM Models**: FastVLM, Qwen2.5 VL , Qwen3 VL etc.

2. Development Board Runtime Environment

After model conversion, the RKNN3 API can be used on the development board to load and run RKNN models. In addition to the RKNN3 API, an OpenAI-compatible API is supported for calling LLM models. It mainly includes the following modules:

RKNN3 API

Provides core functionalities such as RKNN model loading, inference, LLM model inference, and session management. The file structure is as follows:

```
rknn3-api
├── include
│   ├── float16.h
│   └── rknn3_api.h
├── Android
│   └── arm64-v8a
│       └── librknn3_api.so
└── Linux
    └── aarch64
        └── librknn3_api.so
```

During development, you will primarily link with the librknn3_api.so library file.

rkllm3-server

Provides an OpenAI-compatible API service, supporting text and image inputs. Voice and video inputs are not currently supported. The file structure is as follows:

```
rkllm3-server
├── bin
│   ├── android_arm64-v8a
│   │   └── rkllm3-server
│   └── linux-aarch64
│       └── rkllm3-server
```

rknn3_transfer_proxy

Provides the communication interface between the Host (e.g., RK3588, RK3576) and the RK1820/RK1828 coprocessor, supporting both PCIe and USB connections. The file structure is as follows:

```
rknn3_transfer_proxy
├── android-arm64-v8a
│   └── rknn3_transfer_proxy
└── linux-aarch64
    └── rknn3_transfer_proxy
```

1.2 Supported Hardware Platforms

- **RK3588/RK3576 + RK1820/RK1828 Coprocessor**

1.3 Supported Operating Systems

- **Android/Linux**

2 Hardware Environment Setup

This chapter covers the hardware environment setup, including:

- Hardware checklist
- Introduction to the development board and connection tools
- How to connect the development board

2.1 Hardware Checklist

To run the example programs in this document, you will need the following hardware:

- **Computer** × 1
- **RK1820/RK1828 Module** × 1
- **RK_EVB10_RK3588_V10 Development Board** × 1
- **USB-C Data Cable** × 1
- **RK3588 Power Adapter** × 1

Note: In the following sections, the RK1820/RK1828 module will be referred to as RK1820/RK1828, and the RK_EVB10_RK3588_V10 development board will be referred to as RK3588.

2.2 Introduction to Development Board and Connection Tools

1. RK1820/RK1828 Module

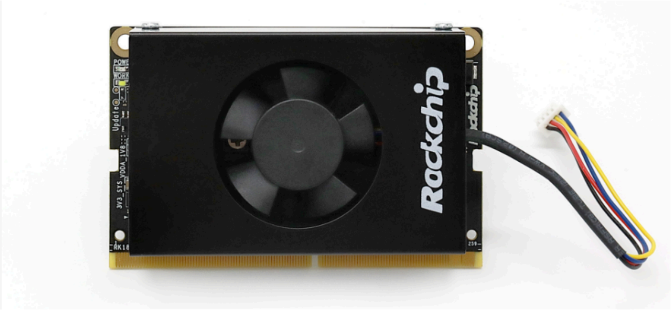


Figure 2-1 RK1820/RK1828 Module

2. RK_EVB10_RK3588_V10 Development Board

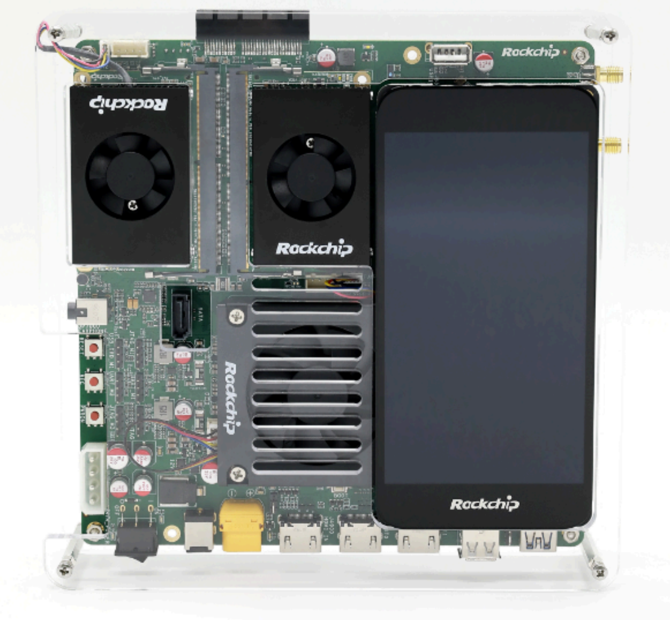


Figure 2-2 RK_EVB10_RK3588_V10 Development Board

3. Data Cable for Connecting the Development Board and Computer

Figure 2-3 USB-C Data Cable

4. Power Adapter

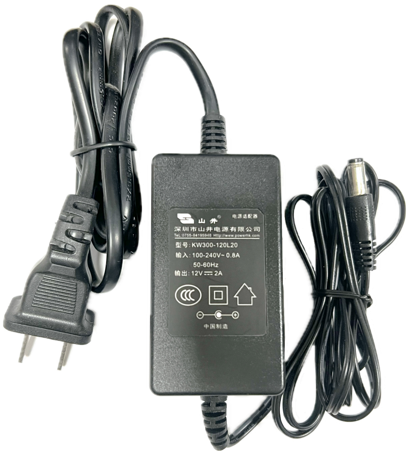


Figure 2-4 RK3588 Power Adapter

2.3 Connecting the Development Board

The following example uses the RK_EVB10_RK3588_V10 development board with an RK1820/RK1828 to illustrate the quick development setup:

1. Prepare a computer with `Ubuntu 20.04 / Ubuntu 22.04` as its operating system.
2. Insert the RK1820/RK1828 module into the RK_EVB10_RK3588_V10 development board as shown below:

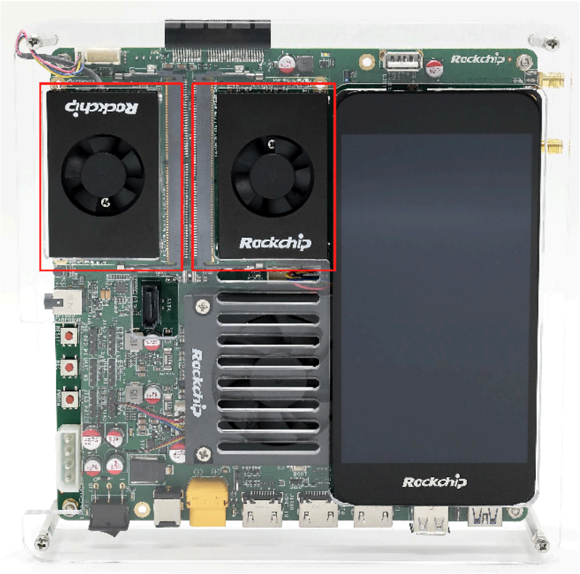


Figure 2-5 RK_EVB10_RK3588_V10 Development Board with RK1820/RK1828

3. Connect the port labeled `debug` on the RK1820/RK1828/RK3588 development board to the computer using the data cable. (Note: The type and location of the data cable interface may vary depending on the hardware version.)
4. Turn on the power switch and wait for the development board's system to boot up completely.
5. Check if the development board is connected to the computer
    1. Check if the RK3588 is connected successfully

    In a terminal window (command-line interface) on your computer, execute the following commands:

    Shell

```
# Install adb
sudo apt install adb

# Query for adb-connected devices
adb devices

# A successful connection will show the following output, where 13af7b28115662cd is the device ID of the RK3588.
# If no device is shown, please refer to section 5.1 for troubleshooting.
List of devices attached
13af7b28115662cd device
```

2. Check if the RK1820/RK1828 is connected successfully.

In a terminal window on your computer, execute the following commands:

```
# Enter the RK3588 terminal
adb shell

# If the RK3588 device is running the Android operating system, navigate to the installation directory
# of rknn3_transfer_proxy located at /vendor/bin.
cd /vendor/bin

# If the RK3588 device is running the Linux operating system, navigate to the installation directory
# of rknn3_transfer_proxy located at /usr/bin.
cd /usr/bin

# Query for devices
./rknn3_transfer_proxy devices

# Example output is shown below
List of ntb devices attached
0000:01:00.0        b98e6c51    PCIE
```

Note: If rknn3_transfer_proxy is not found, please refer to section 3.4.2.

3 Development Environment Setup

This chapter describes the installation and configuration of the PC development environment, including:

- Downloading RKNN3-related repositories
- Setting up the rknn3-toolkit environment
- Installing compilation tools
- Running example programs

3.1 Download RKNN3-Related Repositories

It is recommended to create a dedicated directory to store the RKNN3 repositories, for example, a folder named Projects, and download the RKNN3-related repositories into it. The repositories to download are `rknn3-toolkit`, `rknn3-model-zoo`. Use the following commands as a reference:

```
# Create a new Projects folder
mkdir Projects

# Enter the directory
cd Projects

# Download the rknn3-toolkit repository
git clone https://github.com/airockchip/rknn3-toolkit

# Download the rknn3-model-zoo repository
git clone https://github.com/airockchip/rknn3-model-zoo

# The overall directory structure will be as follows:
Projects
├── rknn3-model-zoo
│   ├── 3rdparty
│   └── examples
├── rknn3-toolkit
│   ├── doc
│   ├── rknn3-toolkit
│   ├── rknn3-toolkit-lite
│   └── rknn3-runtime
│       ├── rknn3-api
│       ├── rknn3_transfer_proxy
│       ├── rkllm3-server
│   └── ...
```

3.2 Install rknn3-toolkit Environment

3.2.1 Install Python

If **Python 3.10 (recommended version)** is not installed on your system, or if multiple Python versions exist, it is recommended to use Miniforge Conda to create an isolated Python 3.10 environment.

3.2.1.1 Install Miniforge Conda

Execute the following command in the terminal to check if Miniforge Conda is already installed. If it is, you can skip this step.

```
conda -V
# Example output: conda 23.3.1, indicates Miniforge Conda version 23.3.1
# If it shows "conda: command not found", it means Miniforge is not installed
```

If Miniforge Conda is not installed, you can download the installation package with the following command:

```
wget -c https://github.com/conda-forge/miniforge/releases/download/25.3.0-1/Miniforge3-
25.3.0-1-Linux-x86_64.sh
```

Then execute the following commands to install Miniforge Conda:

```
# Copy to your home directory
cp /path/to/Miniforge3-25.3.0-1-Linux-x86_64.sh ~

# Add execution permissions
chmod 777 Miniforge3-25.3.0-1-Linux-x86_64.sh

# Run the installation script
bash Miniforge3-25.3.0-1-Linux-x86_64.sh
```

3.2.1.2 Create a Python Environment with Miniforge Conda

Execute the following command in the terminal to enter the Conda base environment:

```
# Refresh environment variables
source ~/.bashrc

# After success, the command prompt will change to:
# (base) xxx@xxx:~$
```

Execute the following command to create a Python 3.10 environment named `toolkit3`:

```
conda create -n toolkit3 python=3.10
```

Activate the `toolkit3` environment. You will install rknn3-toolkit in this environment.

```
conda activate toolkit3
# After success, the command prompt will change to:
# (toolkit3) xxx@xxx:~$
```

Note: Subsequent Python commands are assumed to be executed within the `toolkit3` environment.

3.2.2 Install rknn3-toolkit

After activating the toolkit3 environment, install rknn3-toolkit from a local wheel package:

```
# Enter the rknn3-toolkit directory
cd Projects/rknn3-toolkit/rknn3-toolkit/packages

# Select the requirements file corresponding to your Python version and processor architecture
# cpxxx is the Python version number, x.x.x is the rknn3-toolkit version number
pip install -r requirements_cpxxx-x.x.x.txt

# Install rknn3-toolkit
# Select the wheel package corresponding to your Python version and processor architecture
# x.x.x is the rknn3-toolkit version number, cpxx is the Python version number
pip install rknn3_toolkit-x.x.x-cpxx-cpxx-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
```

3.2.3 Verify Installation

Execute the following commands in the terminal to verify that the rknn3-toolkit environment was installed successfully. If no errors are reported, the installation is successful:

```
# Enter Python interactive mode
python

# Import the RKNN class
from rknn.api import RKNN
```

3.3 Install Compilation Tools

3.3.1 Install CMake

Execute the following commands in the terminal:

```
# Update package list
sudo apt update

# Install cmake
sudo apt install cmake
```

3.3.2 Install Compiler

3.3.2.1 Confirm Development Board System Type and Architecture

For simplicity, "on-board" will be used to refer to the development board environment in the following sections.

1. Confirm On-board System Type

Execute the following command in the terminal:

```
adb shell getprop ro.build.version.release
```

If the output is a number (Android system version), it means the board is running Android:

```
adb shell getprop ro.build.version.release

# Output
12
```

Otherwise, the board is running a Linux system:

```
adb shell getprop ro.build.version.release

# Output
/bin/bash: line 1: getprop: command not found
```

2. Confirm On-board System Architecture

If the board is running Android, you can execute the following command on the PC to query the system architecture:

```
adb shell getprop ro.product.cpu.abi
```

Example output, where arm64-v8a indicates an ARM 64-bit architecture with the version 8 ABI.

```
adb shell getprop ro.product.cpu.abi

# Output
arm64-v8a
```

If the board is running Linux, you can execute the following command on the PC to query the system architecture:

```
adb shell uname -a
```

Example output, where aarch64 indicates an ARM 64-bit architecture.

```
adb shell uname -a

# Output
Linux rk3588-buildroot 6.1.118 #4 SMP Sat Jul  5 14:44:53 CST 2025 aarch64 GNU/Linux
```

3.3.2.2 Install NDK for Android Development Boards

Note: This section applies to Android development boards. If your board is running Linux, please skip this section.

- **NDK Download Link** (version r19c is recommended): https://dl.google.com/android/repository/android-ndk-r19c-linux-x86_64.zip
- **Unpack the Package**

  It is recommended to unpack the NDK package into the `Projects` folder, at the following location:

Shell

```
Projects
├── rknn3-toolkit
├── rknn3-model-zoo
└── android-ndk-r19c # This path will be used when compiling the RKNN3 C Demo later
```

The NDK compiler path is now Projects/android-ndk-r19c.

3.3.2.3 Install GCC Cross-Compiler for Linux Development Boards

Note: This section applies to Linux development boards. If your board is running Android, please skip this section.

The application runs on the host SoC, so select the cross-compilation tool based on the host. Taking the RK3588 with Linux as an example:

- **Cross-Compiler Download Link**

```
https://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/aarch64-linux-gnu/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu.tar.xz
```

- **Unpack the Package**

  It is recommended to unpack the GCC package into the Projects folder. For a 64-bit on-board system, the storage location is as follows:

```
Projects
├── rknn3-toolkit
├── rknn3-model-zoo
└── gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu # This path will be used when compiling the RKNN3 C Demo later
```

  The GCC compiler path is now `Projects/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu`.

3.4 Run Example Programs

3.4.1 RKNN3 Model Zoo Introduction

The RKNN3 Model Zoo provides a rich set of example code to help users quickly run various common models on an RK3588 development board with an RK1820/RK1828. The project directory structure is as follows:

```
rknn3-model-zoo
├── 3rdparty       # Third-party libraries
├── datasets       # Datasets
├── tokenizer      # Tokenizers
├── examples       # Example code
├── utils          # Common utilities, e.g., file operations, drawing
├── build-linux.sh # Compilation script for Linux-based boards
├── build-android.sh # Compilation script for Android-based boards
└── ...
```

The examples directory contains examples for common models like Qwen2.5 and YOLO. Each model example provides Python code for model conversion and C/C++ code for model inference (referred to as RKNN3 Python and RKNN3 C Demo).

3.4.2 Install On-board RKNPU3 Environment

The RKNN3 C Demo requires the RKNPU3 environment to be installed and the `rknn3_transfer_proxy` service to be running. The two core components of the RKNPU3 environment are:

- **rknn3_transfer_proxy**: A background proxy service that runs on the RK3588 board, transferring data between the RK3588 and RK1820/RK1828 via PCIe/USB.
- **RK1820/RK1828 Runtime Library (librknn3_api.so)**: Responsible for loading RKNN models in the system and calling the dedicated Neural Processing Unit (NPU) to perform inference on RKNN models through corresponding interfaces.

Execute the following commands in the terminal to transfer `librknn3_api.so` and the communication proxy `rknn3_transfer_proxy` to the specified locations on the RK3588 using adb:

```
# Navigate to the rknn3-runtime directory
cd Projects/rknn3-toolkit/rknn3-runtime

# If it's an Android system, install rknn3-api/Android/arm64-v8a/librknn3_api.so
adb push rknn3-api/Android/arm64-v8a/librknn3_api.so /vendor/lib64/

# If it's a Linux system, install rknn3-api/Linux/aarch64/librknn3_api.so
adb push rknn3-api/Linux/aarch64/librknn3_api.so /usr/lib/

# If it's an Android system, install rknn3_transfer_proxy/android-arm64-v8a/rknn3_transfer_proxy
adb push rknn3_transfer_proxy/android-arm64-v8a/rknn3_transfer_proxy /vendor/bin/

# If it's a Linux system, install rknn3_transfer_proxy/linux-aarch64/rknn3_transfer_proxy
adb push rknn3_transfer_proxy/linux-aarch64/rknn3_transfer_proxy /usr/bin/

# Add execution permissions
adb shell chmod +x /usr/bin/rknn3_transfer_proxy

adb shell sync
```

Important Notes:

1. After pushing programs to the RK3588 board, you must execute the sync operation (run `adb shell sync` on the PC, or enter the board via adb shell and then run the sync command)!

2. rknn3_transfer_proxy can be configured to start automatically on the RK3588 at boot. If not configured, you need to run the rknn3_transfer_proxy command manually.

3.4.3 Run a Standard Model

Taking the YOLOv6 model as an example, its directory structure is as follows:

```
rknn3-model-zoo
├── examples
│   └── yolov6
│       ├── cpp    # C/C++ Demo example code
│       ├── model  # Model files, test images, etc.
│       ├── python # Model conversion scripts
└── ...
```

1. **Prepare the Model**

   Navigate to the `rknn3-model-zoo/examples/yolov6/model` directory and run the `download_model.sh` script to download an available YOLOv6 ONNX model into the current `model` directory. Execute the following commands in the terminal:

   ```
   # Navigate to the yolov6/model directory
   cd Projects/rknn3-model-zoo/examples/yolov6/model

   # Run the download_model.sh script to download the yolov6 onnx model
   # For example, the downloaded onnx model will be saved at model/yolov6n_rknn3.onnx
   ./download_model.sh
   ```

   Relevant output:

   ```
   --2025-09-15 11:30:44--
   https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/y
   olov6/yolov6n_rknn3.onnx
   Resolving ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
   Connecting to ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443...
   connected.
   HTTP request sent, awaiting response... 200
   Length: 18644871 (18M) [application/octet-stream]
   Saving to: ‘./yolov6n_rknn3.onnx’

   ./yolov6n_rknn3.onnx                         100%[=============================>]
    17.78M  2.48MB/s    in 7.9s

   2025-09-15 11:30:53 (2.26 MB/s) - ‘./yolov6n_rknn3.onnx’ saved [18644871/18644871]
   ```

   Note:

   (1) Models with the "*rknn3" suffix are specially optimized for the RK1820/RK1828. The YOLO post-processing (decoding, candidate box filtering/sorting, NMS, etc.) is built-in to be computed on the RK1820/RK1828, reducing data transfer overhead. Users can also modify the model filename in the script to one without the* "rknn3" suffix, in which case the model post-processing needs to be handled in the application.

(2) Download the complete PyTorch example project from the following link: pytorch_example. After downloading, refer to the export documentation within it and follow the steps to generate an optimized ONNX model adapted for the RK1820/RK1828. The download link can be found in `download_model.sh`.

2. **Model Conversion**

   Navigate to the `rknn3-model-zoo/examples/yolov6/python` directory and run the `convert.py` script. This script converts the original ONNX model to an RKNN model. In a terminal window on your computer, execute the following command:

```
# Navigate to the yolov6/python directory
cd Projects/rknn3-model-zoo/examples/yolov6/python

# Run the convert.py script to convert the original ONNX model to an RKNN model
# Usage: python3 convert.py onnx_model_path [platform] [dtype(optional)]
[output_rknn_path(optional)]
#       platform choose from [RK1820/RK1828]
python convert.py ../model/yolov6n_rknn3.onnx RK1820 i8
```

Relevant output:

```
--> Loading model
I Loading : 100%|██████████████████████████████████████████████| 142/142 [00:00<00:00, 24888.89it/s]
W load_onnx: Please note that some float16/float64 data types in the model have been modified to float32!
done
--> Building model
I FoldConstant : 100%|███████████████████████████████████████████| 218/218 [00:00<00:00, 505.15it/s]
I OpFusing 0: 100%|██████████████████████████████████████████████| 100/100 [00:00<00:00, 522.77it/s]
I OpFusing 1 : 100%|█████████████████████████████████████████████| 100/100 [00:00<00:00, 344.86it/s]
I OpFusing 0 : 100%|█████████████████████████████████████████████| 100/100 [00:00<00:00, 274.72it/s]
I OpFusing 1 : 100%|█████████████████████████████████████████████| 100/100 [00:00<00:00, 261.63it/s]
I OpFusing 2 : 100%|█████████████████████████████████████████████| 100/100 [00:00<00:00, 249.86it/s]
I OpFusing 0 : 100%|█████████████████████████████████████████████| 100/100 [00:00<00:00, 231.67it/s]
I OpFusing 1 : 100%|█████████████████████████████████████████████| 100/100 [00:00<00:00, 222.40it/s]
I OpFusing 2 : 100%|█████████████████████████████████████████████| 100/100 [00:00<00:00, 220.71it/s]
I     FoldConstant : 100%|███████████████████████████████████████| 170/170 [00:00<00:00, 622.77it/s]
...
I rknn building ...
I Split shape 0 done
I Compile all models for 1 cores
module optimization: [=======================================================] 1/1 (100.0%) Total: 0s
register configuration: [=======================================================] 1/1 (100.0%) Total: 0s
memory optimization: [=======================================================] 1/1 (100.0%) Total: 0s
weight sharing: [=======================================================] 1/1 (100.0%) Total: 0s
code generation: [=======================================================] 1/1 (100.0%) Total: 1s
I rknn building done.
done
--> Export rknn model
done
```

3. **Run RKNN3 C Demo**

   - **For Android on-board systems**

Taking the RK3588 platform with Android (arm64-v8a architecture) as an example, you need to use the build-android.sh script in the rknn3-model-zoo directory to compile. Before running the script, you need to set the ANDROID_NDK_PATH to your local NDK compiler path. Add the following code to the build-android.sh script:

```
# Add this to the beginning of the build-android.sh script
ANDROID_NDK_PATH=Projects/android-ndk-r19c
```

Then, in the rknn3-model-zoo directory, run the build-android.sh script from your computer's terminal:

```
# Compile the yolov6 c++ inference program. Since the host device is rk3588, set the target to rk3588
./build-android.sh -t rk3588 -a arm64-v8a -b Release -d yolov6
```

- **For Linux on-board systems**

  Taking the RK3588 platform with Linux (aarch64 architecture) as an example, you need to use the `build-linux.sh` script in the `rknn3-model-zoo` directory. Before running the script, you need to configure the `GCC_COMPILER` environment variable.

  In your computer's terminal, navigate to the `rknn3-model-zoo` directory and run the following commands to compile:

  ```
  # Export the GCC cross-compiler environment variable
  export GCC_COMPILER=Projects/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu

  # Compile the yolov6 c++ inference program. Since the host device is rk3588, set the target to rk3588
  ./build-linux.sh -t rk3588 -a aarch64 -b Release -d yolov6
  ```

Installation and Execution:

After compilation, the script packages the executable program in the `Projects/rknn3-model-zoo/install` directory. Use the following commands to transfer the program and model to the `/data` directory on the development board:

```
# If using a Linux system, transfer the compiled program to the /data directory
adb push Projects/rknn3-model-zoo/install/rk3588_linux_aarch/rknn_yolov6_demo /data

# If using an Android system, transfer the compiled program to the /data directory
adb push Projects/rknn3-model-zoo/install/rk3588_android_arm64-v8a/rknn_yolov6_demo
/data

# Enter the RK3588 terminal
adb shell

# Navigate to the demo directory
cd /data/rknn_yolov6_demo

# Run rknn_yolov6_demo
# Usage: ./rknn_yolov6_demo <model_path> <weight_path> <image_path> <core_mask>
./rknn_yolov6_demo ./model/yolov6n_rknn3.rknn ./model/yolov6n_rknn3.weight
./model/bus.jpg 1
```

Relevant output:

```
model is NHWC input fmt
model input height=640, width=640, channel=3
origin size=640x640 crop size=640x640
input image: 640 x 640, subsampling: 4:2:0, colorspace: YCbCr, orientation: 1
--> inference model
scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0
padding_w=0 padding_h=0
rga_api version 1.10.1_[0]
Pre-process time: 2.09 ms
-->rknn_run
Inference time: 21.37 ms
Post-process time: 0.44 ms
Total time: 23.90 ms
--> inference model done
bus @ (97 140 557 440) 0.951
person @ (109 239 221 537) 0.940
person @ (213 238 287 511) 0.932
person @ (480 233 560 520) 0.924
person @ (78 326 117 515) 0.455
write_image path: out.png width=640 height=640 channel=3 data=0x7f885e801
```

3.4.4 Run an LLM

Running an `llm` involves three main steps: 1) Exporting the model to `onnx`; 2) Converting the model to `rknn`; 3) Compiling and running the example program.

Taking the Qwen2.5 model as an example, its directory structure is as follows:

```
rknn3-model-zoo
├── examples
│   └── Qwen2_5
│       ├── data   # Quantization dataset
│       ├── cpp    # Model inference example code
│       ├── python # Model conversion scripts
└── ...
```

1. **Install dependent environments**. In a terminal window on your computer, execute the following command:

   ```
   cd Projects/rknn3-model-zoo
   pip install -r requirements.txt
   ```

2. **Export the LLM model to ONNX format**. In a terminal window on your computer, execute the following commands:

   ```
   # Set the environment variable
   export PYTHONPATH=Projects/rknn3-model-zoo/

   # Navigate to the Qwen2.5 python directory
   cd Projects/rknn3-model-zoo/examples/Qwen2_5/python/

   # Export the onnx model and configuration file
   python export_llm.py
   ```

   Upon successful execution, model files with an `.onnx` extension and configuration files with a `.config.pkl` extension will be generated in the `examples/Qwen2_5/model` folder.

3. **Convert to `rknn` model**. In a terminal window on your computer, execute the following command:

   Shell

   ```
   # Navigate to the Qwen2.5 python directory
   cd Projects/rknn3-model-zoo/examples/Qwen2_5/python/

   # Convert to rknn model
   python export_rknn.py
   ```

   Relevant output:

```
I rknn building done.
done
--> Export RKNN model
done
```

4. **Run RKNN3 C Demo**

    - **For Android on-board systems**

      Taking the RK3588 platform with Android (arm64-v8a architecture) as an example, you need to use the `build-android.sh` script. Before running it, set `ANDROID_NDK_PATH` to your local NDK path by adding this line to the script:

      ```
      # Add this to the beginning of the build-android.sh script
      ANDROID_NDK_PATH=Projects/android-ndk-r19c
      ```

      Then, in the `rknn3-model-zoo` directory, run the `build-android.sh` script from your terminal:

      ```
      # Compile the Qwen2.5 C++ inference program for the rk3588 host device
      ./build-android.sh -t rk3588 -a arm64-v8a -b Release -d Qwen2_5
      ```

    - **For Linux on-board systems**

      Taking the RK3588 platform with Linux (aarch64 architecture) as an example, you need to use the build-linux.sh script. Before running it, configure the `GCC_COMPILER` environment variable.

      In your computer's terminal, navigate to the `rknn3-model-zoo` directory and run the following commands:

      ```
      # Export the GCC cross-compiler environment variable
      export GCC_COMPILER=Projects/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu

      # Compile the Qwen2.5 C++ inference program for the rk3588 host device
      ./build-linux.sh -t rk3588 -a aarch64 -b Release -d Qwen2_5
      ```

Installation and Execution:

After compilation, the script packages the executable in the `Projects/rknn3-model-zoo/install` directory. Use the following commands to transfer the program and models to the `/data` directory on the board:

```
# If using a Linux system, transfer the compiled program to /data
adb push Projects/rknn3-model-zoo/install/rk3588_linux_aarch/rknn_Qwen2_5_demo /data

# If using an Android system, transfer the compiled program to /data
adb push Projects/rknn3-model-zoo/install/rk3588_android_arm64-v8a/rknn_Qwen2_5_demo /data

# Enter the RK3588 terminal
adb shell

# Navigate to the demo directory
cd /data/rknn_Qwen2_5_demo/

# Run rknn_qwen2_5_demo
# Usage: ./rknn_qwen2_5_demo <model_path> <weight_path> <tokenizer_path> <embedding_path> <core_mask> <prompt>
./rknn_qwen2_5_demo ./model/Qwen2.5-1.5B-Instruct.rknn \
   ./model/Qwen2.5-1.5B-Instruct.weight \
   ./model/Qwen2.5-1.5B-Instruct.tokenizer.gguf \
   ./model/Qwen2.5-1.5B-Instruct.embed.bin 0xff \
   "who are you?"
```

4 Development Workflow

This chapter describes how to quickly complete model conversion and deploy it on an RK3588/RK1820/RK1828 using the RKNN3 SDK.

4.1 CNN Model Development Workflow Introduction

Users can refer to the following flowchart to understand the overall development steps for RKNN, which is mainly divided into three parts: model conversion, model evaluation, and on-board deployment and execution.

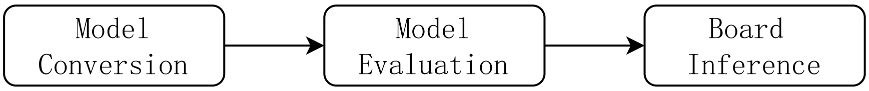


Figure 4-1 Model Conversion Workflow

4.1.1 Model Conversion

In this stage, the original deep learning model is converted into the RKNN format for efficient inference on the RK1820/RK1828. This includes the following 5 steps:

a. **Obtain Original Model**: Obtain or train a deep learning model. It is recommended to use mainstream frameworks such as ONNX, PyTorch, or TensorFlow.

b. **Model Configuration**: Make necessary configurations in the RKNN3 Toolkit, such as normalization parameters, quantization parameters, and target platform.

c. **Model Loading**: Use the appropriate loading interface to import the model into RKNN3-Toolkit. Select the correct loading interface based on the model framework.

d. **Model Building**: Build the RKNN model using the rknn.build() interface. You can choose whether to perform quantization to improve the model's inference performance on hardware.

e. **Model Exporting**: Export the RKNN model using the rknn.export_rknn() interface for subsequent deployment.

4.1.1.1 Model Conversion Process

Model conversion is a core function of rknn3-toolkit, allowing users to convert deep learning models from different frameworks into the RKNN format to run on the RKNPU. This section details the model conversion process, including the interfaces and important considerations. The basic RKNN model conversion process is shown in the figure below:

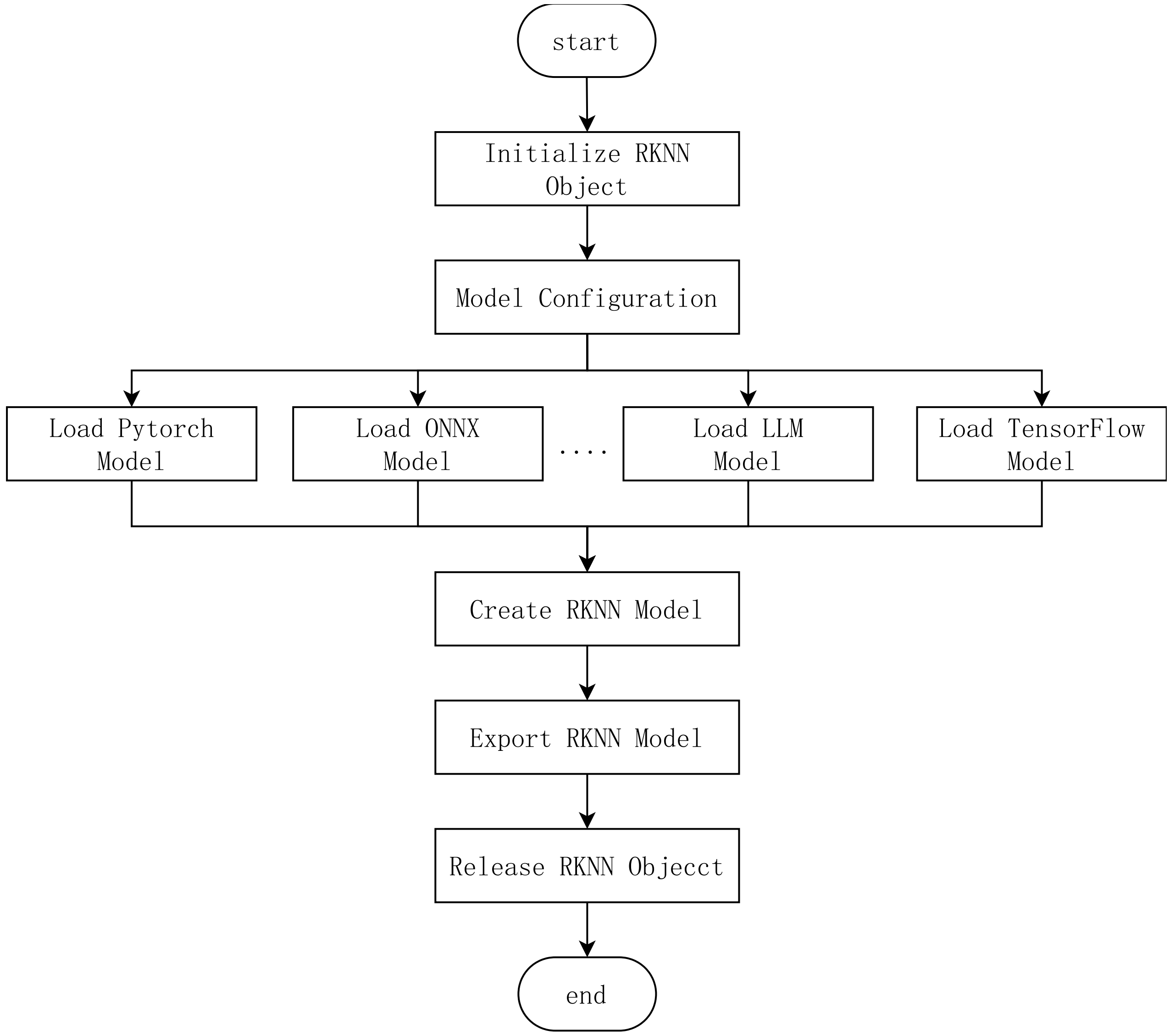


Figure 4-2 Model Conversion Process

4.1.1.2 Key Interface Descriptions

This section briefly describes the key interfaces and important parameters used in the standard model conversion process. For more detailed interface usage instructions, please refer to the <Rockchip_RKNPU_API_Reference_RKNN3_Toolkit_CN.pdf> document.

4.1.1.2.1 Create and Release RKNN Object

When using any API interfaces of rknn3-toolkit, you must first call the `RKNN()` method to initialize an `RKNN` object. When the object is no longer needed, release it by calling its `release()` method.

Example:

```
from rknn.api import RKNN
rknn = RKNN(verbose=True)
......
rknn.release()
```

Parameter description:

- verbose: Whether to print detailed logs.

4.1.1.2.2 Model Configuration

Before building the RKNN model, you need to configure parameters such as channel mean, RGB2BGR conversion for quantization images, and quantization type. This can be done through the `config` interface.

Example:

```
rknn.config(target_platform='RK1820', quantized_algorithm='normal',
        mean_values=[[0, 0, 0]], std_values=[[255, 255, 255]],
        input_attrs={'image_arrays': {'dtype': 'uint8', 'layout': 'NHWC'}})
```

Main parameter descriptions:

- `target_platform`: Specifies the chip platform for which the RKNN model is generated.
- `quantized_algorithm`: The algorithm used to calculate quantization parameters for each layer. Supported algorithms include `normal`, `mmse`, `kl_divergence`, `grq`, and `gdq`. The default is `normal`.
- `mean_values`: Input mean values.
- `std_values`: Input normalization values.
- `input_attrs`: Used to set the properties of the input during inference. For example, if a CNN image model expects RGB data as input, the `dtype` should be set to `uint8` and `layout` to `NHWC`. This may not be necessary for other types of models. The `image_arrays` here is the name of the model's input and should be modified according to the actual model.

4.1.1.2.3 Model Loading

rknn3-toolkit currently supports loading models from `ONNX`, `PyTorch`, `TensorFlow`, and `TensorFlow Lite`. Different frameworks require calling their corresponding loading interfaces.

This section uses an ONNX model as an example. For loading interfaces for other frameworks, please refer to the `<Rockchip_RKNPU_API_Reference_RKNN3_Toolkit_CN.pdf>` document.

rknn3-toolkit provides the `load_onnx` interface to load ONNX models.

Example:

```
ret = rknn.load_onnx(model='../model/yolov6n.onnx')
```

Main parameter description:

- model: Path to the `ONNX` model file.

4.1.1.2.4 Model Building

rknn3-toolkit provides the `build` interface to build the RKNN model.

Example:

```
# Build model
print('--> Building model')
rknn.build(do_quantization=True, dataset=args.dataset_path)
if ret != 0:
    print('Build model failed!')
    exit(ret)
print('done')
```

Main parameter descriptions:

- do_quantization: Whether to quantize the model. The default is True.
- dataset: The dataset used for quantization calibration.

4.1.1.2.5 Model Exporting

rknn3-toolkit provides the `export_rknn` interface to export the `RKNN` model file for deployment.

Example:

```
ret = rknn.export_rknn(export_path='./yolov6.rknn')
```

Main parameter description:

- export_path: The path to export the model file.

Note: The exported model consists of two parts: a model file ending in `.rknn` and a weight file ending in `.weight`. Both files will be used in the subsequent deployment.

4.1.2 Model Evaluation

4.1.2.1 Accuracy Analysis

Compare the inference results of quantized models with those of floating-point models to analyze the sources of quantization errors. Currently, both simulator-based accuracy analysis and multi-board (chained-board) accuracy analysis are supported. The analysis can be configured through the `accuracy_analysis` option of the `rknn.inference()` interface.

Example:

```
ret = rknn.init_runtime(target=platform, device_id='515e9b401c060c0b')

# Preprocess
image_src = cv2.imread(IMG_PATH)
img = preprocess(image_src)

# Inference and accuracy_analysis
outputs = rknn.inference(inputs=[img], accuracy_analysis=True)

# Postprocess
outputs = postprocess(outputs)
```

Main parameter description:

- inputs: A list of input data.
- accuracy_analysis: Whether to enable accuracy analysis

Note: `inputs` accepts numpy input in `nchw` data format.

4.1.2.2 Performance Evaluation

This section primarily analyzes the performance evaluation results of the model during execution, including the execution time, cycle count, and bandwidth of each operator.
Note: Performance evaluation must be enabled by setting `profile_mode=True` in `rknn.config()`.

Example:

```
ret = rknn.init_runtime(target=platform, core_mask=0xff)
ret = rknn.eval_perf()
```

Taking RK1820 and Qwen2.5-0.5B as an example, the performance evaluation results after executing `eval_perf` are shown below:

```
---------------------------------------------------------------------------
Op Time Ranking Table (Core 7)
---------------------------------------------------------------------------
OpType             Calls   CPU(us)      NPU(us)      Total(us)    Ratio(%)
---------------------------------------------------------------------------
exMatMul           121     0            56357        56357        23.04
Add                120     0            54594        54594        22.32
exMatMulActivation 48      0            25881        25881        10.58
rcclScatter        50      0            23468        23468        9.60
rcclGather         49      0            23015        23015        9.41
exNorm             49      0            22701        22701        9.28
Reshape            50      0            22565        22565        9.23
exAttention        24      0            14645        14645        5.99
Transpose          1       0            461          461          0.19
Gather             1       0            447          447          0.18
OutputOperator     1       0            447          447          0.18
---------------------------------------------------------------------------
Total                      0            244581       244581         100.00
---------------------------------------------------------------------------
...

---------------------------------------------------------------------------
Op Time Ranking Table (Core 0)
---------------------------------------------------------------------------
OpType             Calls   CPU(us)      NPU(us)      Total(us)    Ratio(%)
---------------------------------------------------------------------------
rcclReduce         147     0            68083        68083        21.60
exMatMul           121     0            55740        55740        17.68
Add                120     0            53837        53837        17.08
rcclBroadcast      56      0            26167        26167        8.30
exMatMulActivation 48      0            25648        25648        8.14
rcclScatter        50      0            23096        23096        7.33
rcclGather         49      0            22677        22677        7.19
exNorm             49      0            22479        22479        7.13
exAttention        24      0            12909        12909        4.09
Reshape            3       0            1346         1346         0.43
OutputProc         1       0            568          568          0.18
Transpose          1       0            458          458          0.15
Sub                1       0            457          457          0.14
Clip               1       0            456          456          0.14
Mul                1       0            455          455          0.14
Gather             1       0            439          439          0.14
OutputOperator     1       0            438          438          0.14
---------------------------------------------------------------------------
Total                      0            315253       315253         100.00
---------------------------------------------------------------------------
```

4.1.2.4 Memory Evaluation

Obtain the memory usage evaluation results of the model.

Example:

```
ret = rknn.init_runtime(target=platform)
memory_detail = rknn.eval_memory()
```

The memory evaluation results are as follows:

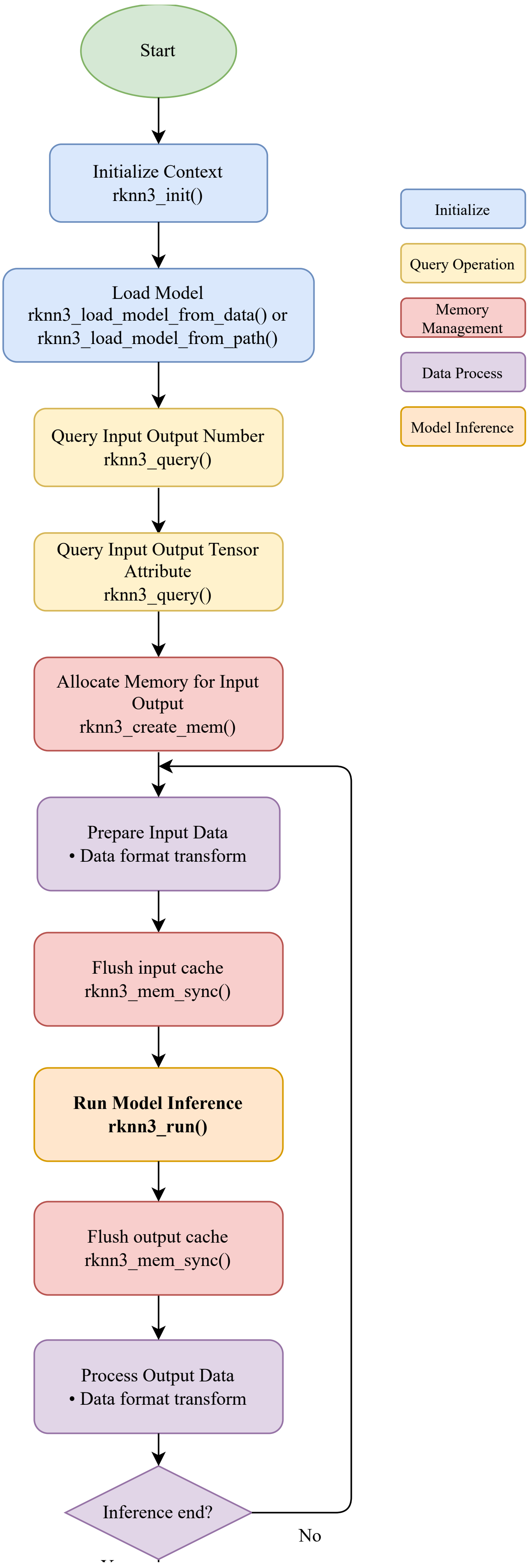
```
======================== Memory Usage Information ========================
Device Memory:
  System  :      19.12 MB total,       9.76 MB free,       9.36 MB used ( 49.0%)
  Node 0  :     319.50 MB total,     263.23 MB free,      56.27 MB used ( 17.6%)
  Node 1  :     319.50 MB total,     265.15 MB free,      54.35 MB used ( 17.0%)
  Node 2  :     299.90 MB total,     245.51 MB free,      54.39 MB used ( 18.1%)
  Node 3  :     319.50 MB total,     265.15 MB free,      54.35 MB used ( 17.0%)
  Node 4  :     299.90 MB total,     250.54 MB free,      49.36 MB used ( 16.5%)
  Node 5  :     299.90 MB total,     249.03 MB free,      50.87 MB used ( 17.0%)
  Node 6  :     319.50 MB total,     268.59 MB free,      50.91 MB used ( 15.9%)
  Node 7  :     319.50 MB total,     268.63 MB free,      50.87 MB used ( 15.9%)

Per-Core Memory Allocation (MB):
  Core     Command      Weight       Internal     KVCache      Total
  -------- ------------ ------------ ------------ ------------ ------------
  0        5.06         40.66        4.25         6.23         56.20
  1        2.75         42.05        3.24         6.23         54.28
  2        2.79         42.05        3.24         6.23         54.32
  3        2.75         42.05        3.24         6.23         54.28
  4        2.76         37.05        3.24         6.23         49.29
  5        2.75         38.58        3.24         6.23         50.80
  6        2.79         38.58        3.24         6.23         50.85
  7        2.75         38.58        3.24         6.23         50.81
  -------- ------------ ------------ ------------ ------------ ------------
  Total    24.41        319.60       26.94        49.88        420.84
==========================================================================
```

4.1.3 Model Deployment

This stage covers the actual deployment and execution of the model. It typically includes the following steps: a. **Model Initialization**: Load the RKNN model and prepare for pre-processing. b. **Model Pre-processing**: Load the data to be inferred and prepare for inference. c. **Model Inference**: Execute the inference operation, passing the input data to the model and obtaining the results. d. **Model Post-processing**: Process the inference results and pass them to the application layer. e. **Model Release**: After completing the inference process, release the model resources so that other tasks can use the RKNN model.

Refer to the flowchart below:


Start
Initialize Context
rknn3_init()
Load Model
rknn3_load_model_from_data() or
rknn3_load_model_from_path()
Query Input Output Number
rknn3_query()
Query Input Output Tensor Attribute
rknn3_query()
Allocate Memory for Input Output
rknn3_create_mem()
Prepare Input Data
• Data format transform
Flush input cache
rknn3_mem_sync()
Run Model Inference
rknn3_run()
Flush output cache
rknn3_mem_sync()
Process Output Data
• Data format transform
Inference end?
No
Initialize
Query Operation
Memory Management
Data Process
Model Inference

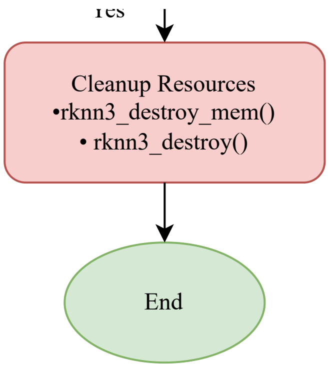


Figure 4-3 CNN Model Deployment Flow

4.1.4 Key Interface Descriptions

This section briefly describes the RKNN3 C API used in the CNN model deployment process. For detailed interface usage instructions, please refer to the `Rockchip_RKNPU_API_Reference_RKNNRT3_EN.pdf` document.

4.1.4.1 Context Initialization and Destruction

The RKNN3 C API provides the `rknn3_init` interface to initialize context information and `rknn3_destroy` to destroy it.

Example:

```
rknn3_context ctx;
int ret = rknn3_init(&ctx, nullptr);
if (ret != RKNN3_SUCCESS) {
  return -1;
}
......
rknn3_destroy(ctx);
```

Main parameter description:

- rknn3_context* context: A pointer to the RKNN context handle to be initialized.

4.1.4.2 Model Loading

The RKNN3 C API provides the `rknn3_load_model_from_path` interface to load an `RKNN` model from a file path into the specified context.

Example:

```
// Load model
ret = rknn3_load_model_from_path(ctx, model_path, weight_path);
if (ret != RKNN3_SUCCESS) {
  printf("rknn3_load_model failed! ret=%d\n", ret);
  rknn3_destroy(ctx);
  return -1;
}
```

Main parameter descriptions:

- rknn3_context context: The `RKNN` context handle.

- const char* model_path: The path to the `RKNN` model file.
- const char* weight_path: The path to the `RKNN` weight file.

The RKNN3 C API also provides `rknn3_load_model_from_data` to load an `RKNN` model from memory. Its main parameters are:

- rknn3_context context: The `RKNN` context handle.
- const void* model_data: `RKNN` model data.
- uint64_t model_size: Size of the `RKNN` model data.
- const void* weight_data: Weight data.
- uint64_t weight_size: Size of the weights.

4.1.4.3 Model Initialization

The RKNN3 C API provides the `rknn3_model_init` interface to complete model initialization.

Example:

```
ret = rknn3_model_init(ctx, &config);
if (ret != RKNN3_SUCCESS) {
  printf("rknn3_model_init failed! ret=%d\n", ret);
  rknn3_destroy(ctx);
  return -1;
}
```

Main parameter description:

- rknn3_context context: The `RKNN` context handle.
- rknn3_config* config: Configuration parameters for model initialization.

4.1.4.4 Querying Model Attributes

The RKNN3 C API provides the `rknn3_query` interface to query information such as the number of model inputs and outputs, and their attributes.

Example:

```
// Query input/output information
rknn3_input_output_num io_num;
ret = rknn3_query(ctx, RKNN3_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
if (ret != RKNN3_SUCCESS) {
  printf("rknn3_query input/output num failed! ret=%d\n", ret);
  rknn3_destroy(ctx);
  return -1;
}

// Query input tensor attributes
rknn3_tensor_attr input_attrs[io_num.n_input];
for (int i = 0; i < io_num.n_input; i++) {
  input_attrs[i].index = i;
  ret = rknn3_query(ctx, RKNN3_QUERY_INPUT_ATTR, input_attrs + i,
                    sizeof(rknn3_tensor_attr));
  if (ret != RKNN3_SUCCESS) {
    printf("rknn3_query input attr failed! ret=%d\n", ret);
    rknn3_destroy(ctx);
    return -1;
  }
  dump_tensor_attr(input_attrs + i);
}

// Query output tensor attributes
rknn3_tensor_attr output_attrs[io_num.n_output];
for (int i = 0; i < io_num.n_output; i++) {
  output_attrs[i].index = i;
  ret = rknn3_query(ctx, RKNN3_QUERY_OUTPUT_ATTR, output_attrs + i,
                    sizeof(rknn3_tensor_attr));
  if (ret != RKNN3_SUCCESS) {
    printf("rknn3_query output attr failed! ret=%d\n", ret);
    rknn3_destroy(ctx);
    return -1;
  }
  dump_tensor_attr(output_attrs + i);
}
```

Main parameter descriptions:

- `rknn3_query_cmd cmd`: Query command type.
  - `RKNN3_QUERY_IN_OUT_NUM`: Query the number of input and output tensors.
  - `RKNN3_QUERY_INPUT_ATTR`: Query the attributes of an input tensor.
  - `RKNN3_QUERY_OUTPUT_ATTR`: Query the attributes of an output tensor.
- `void* info`: A pointer to a buffer to store the query results.
- `uint64_t size`: The size of the `info` buffer in bytes.

4.1.4.5 Synchronize Input and Output Data

The RKNN3 C API provides the `rknn3_mem_sync` interface to synchronize memory data between the host and the RK182x device. This interface can be used to synchronize both model input data and model output data.

Example:

```
// sync inputs
for (int i = 0; i < io_num.n_input; i++)
{
  ret = rknn3_mem_sync(ctx, inputs[i].mem, RKNN3_MEMORY_SYNC_TO_DEVICE);
  if (ret != RKNN3_SUCCESS)
  {
    printf("rknn3_mem_sync input %d failed! ret=%d\n", i, ret);
    break;
  }
}

// sync outputs
for (int i = 0; i < io_num.n_output; i++)
{
  rknn3_mem_sync(ctx, outputs[i].mem, RKNN3_MEMORY_SYNC_FROM_DEVICE);
  if (ret != RKNN3_SUCCESS)
  {
    printf("rknn3_mem_sync output %d failed! ret=%d\n", i, ret);
    break;
  }
}
```

Main parameter descriptions:

- `rknn3_tensor_mem* mem`: Memory to be synchronized.
- `rknn3_mem_sync_mode mode`: Synchronization mode. If data is synchronized from the host (e.g., RK3588) to RK182x, set `mode` to `RKNN3_MEMORY_SYNC_TO_DEVICE`. If data is synchronized back from RK182x to the host for further processing, set `mode` to `RKNN3_MEMORY_SYNC_FROM_DEVICE`.

For more details about `rknn3_mem_sync` and related memory synchronization interfaces, please refer to the `Rockchip_RKNPU_API_Reference_RKNNRT3_EN.pdf` document.

4.1.4.6 Model Inference

The RKNN3 C API provides the `rknn3_run` interface to perform model inference. This interface will block until the inference is complete.

Example:

```
  // Run the model
  ret = rknn3_run(ctx, inputs, io_num.n_input, outputs, io_num.n_output);
  if (ret != RKNN3_SUCCESS) {
    printf("rknn3_run failed! ret=%d\n", ret);
    rknn3_destroy(ctx);
    return -1;
  }
```

Main parameter descriptions:

- `const rknn3_tensor inputs[]`: An array of input `tensor`s containing input data.
- `uint32_t n_inputs`: The number of input `tensor`s.
- `rknn3_tensor outputs[]`: An array of output `tensor`s to store the inference results.
- `uint32_t n_outputs`: The number of output `tensor`s.

4.2 LLM Model Development Workflow Introduction

4.2.1 Model Conversion

The development workflow for LLM models differs slightly from that of CNN models. First, the model from HuggingFace needs to be converted to an ONNX model, and then exported as an RKNN model. The overall process is as follows:

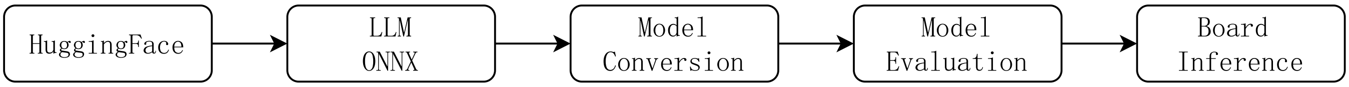


Figure 4-4 LLM Model Development Workflow

To convert a HuggingFace model to an ONNX model, you can refer to `《LLM Model Adaptation Tutorial.md》` in the rknn3-model-zoo. The following is a brief description of this process using Qwen2.5 0.5B as an example. The relevant code can be found in `examples/Qwen2_5/python/export_llm.py` in the rknn3-model-zoo.

a. **Build the LLM Torch model**. This step is common for most LLMs and requires no special modifications.

```
from transformers import AutoModelForCausalLM, AutoConfig
config = AutoConfig.from_pretrained(args.model_path, **kwargs)
model = AutoModelForCausalLM.from_pretrained(args.model_path, **kwargs)
```

b. **Convert from AutoModel to ONNX**.

```
dummy_input = torch.zeros((1, in_len), dtype=torch.long)
attention_mask = torch.ones((1, in_len), dtype=torch.float)
position_ids = torch.arange(0, in_len, dtype=torch.long).unsqueeze(0)

inputs = (dummy_input, attention_mask, position_ids)
input_names = ["input_ids", "attention_mask", "position_ids"]
dynamic_axes = {}
if args.dynamic_shape:
    dynamic_axes.update({
        'input_ids': {1: 'sequence'},
        'attention_mask': {1: 'sequence'},
        'position_ids': {1: 'sequence'},
    })

torch.onnx.export(
        model,
        inputs,
        args.export_llm_path,
        export_params=True,
        opset_version=19,
        do_constant_folding=True,
        input_names=input_names,
        output_names=output_names,
        dynamic_axes=dynamic_axes,
    )
```

This step is also common for most LLM models, but it's important to note that the `inputs` must match the original torch definition. Some models may not follow the order `inputs = (dummy_input, attention_mask, position_ids)`.

c. **Export LLM-specific configurations**, such as `chat_template`, `vocab_size`, `hidden_size`, etc. Users typically do not need to modify this code.

```
export_llm_config(args.model_path, os.path.splitext(args.export_llm_path)[0] +
'.config.pkl', chat_context, prompt)
```

The exported `xxx.config.pkl` will be needed in the next step for model conversion.

d. **Export the tokenizer**.

```
export_tokenizer(args.model_path, os.path.splitext(args.export_llm_path)[0] +
'.tokenizer.gguf')
```

The tokenizer is mainly used for on-board inference. The rknn3-model-zoo uses the tokenizer from https://github.com/ggml-org/llama.cpp. Users can also use their own tokenizer by implementing the `tokenizer_callback` in `RKLLMCallback` during on-board inference (for details, see `<Rockchip_RKNPU_API_Reference_RKNNRT3_CN>`).

e. **Export the embedding**.

```
export_embed_weight(model.model.embed_tokens.weight, os.path.splitext(args.export_llm_path)
[0] + '.embed.bin')
```

Due to the limited DRAM size of the RK1820/RK1828, the embedding layer of the LLM model is processed on the HOST. The generated `xxx.embed.bin` is used for the `embed_callback` in `RKLLMCallback`.

f. **Load the ONNX model**.

```
rknn.config(target_platform='RK1820',
            quantized_dtype='w4a16', quantized_algorithm='grq', quantized_method='group32')

ret = rknn.load_llm(model=args.onnx_path, config=args.config)
```

Here, `model` is the ONNX model exported in step b, and `config` is the `config.pkl` exported in step c.

- `target_platform`: Target chip platform, must be `RK1820/RK1828`.
- `quantized_dtype`: Quantization type, must be `w4a16` for LLM models.
- `quantized_algorithm`: Quantization algorithm, can be `grq` or `normal`. `grq` generally provides higher quantization accuracy and is recommended.
- `quantized_method`: Quantization method, `'group32'` is recommended.

g. **Quantize and export the rknn model**.

```
rknn.build(do_quantization=True, dataset=args.dataset_path)
ret = rknn.export_rknn(args.rknn_path)
```

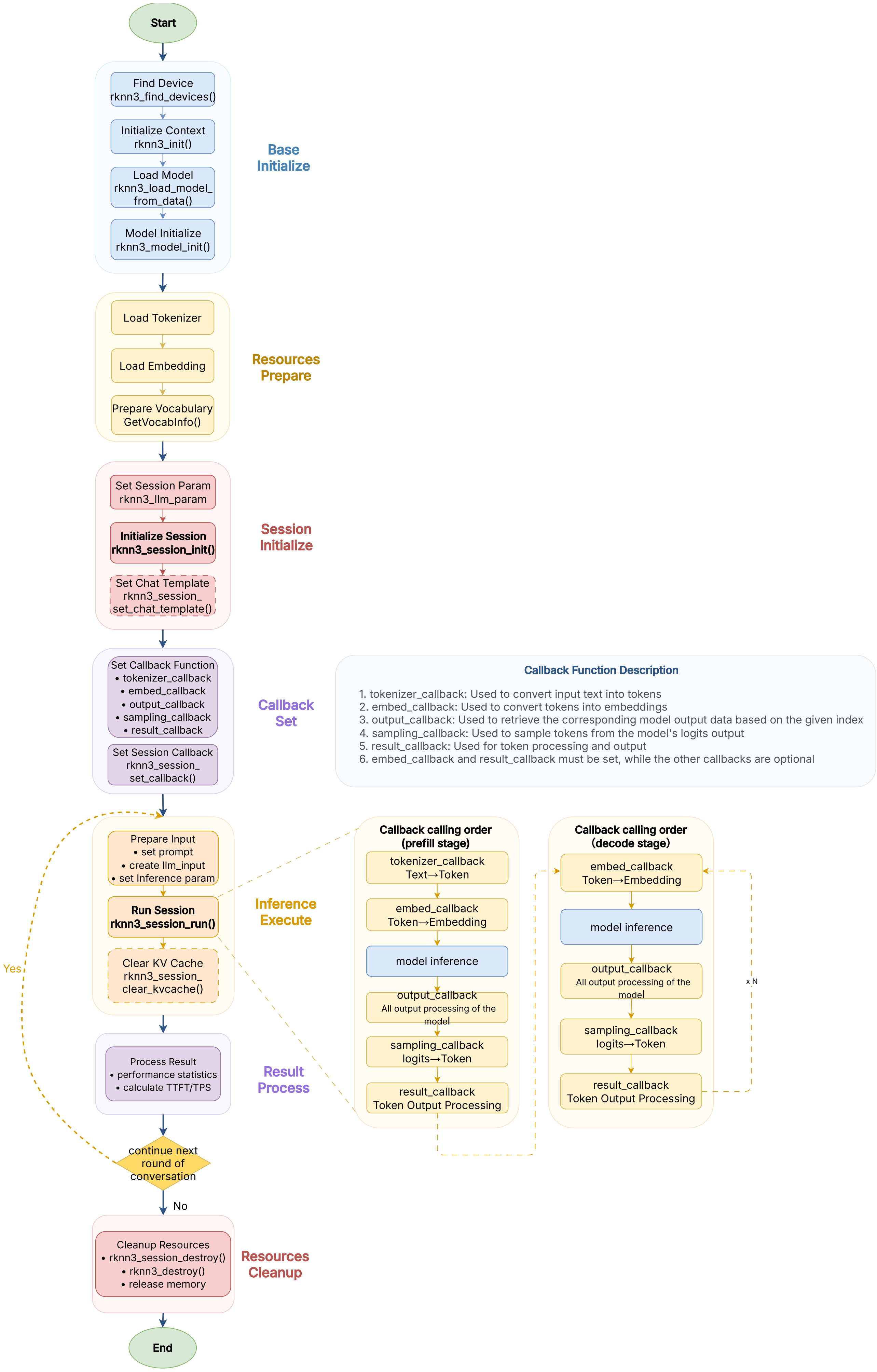
At this point, all the necessary files for on-board execution will be generated, including `xxx.rknn`, `xxx.weight`, `xxx.embed.bin`, and `xxx.tokenizer.gguf`.

4.2.2 LLM Model Deployment

The deployment of LLM models is significantly different from that of CNN models. To better manage the context of LLM models, the concept of a `Session` is introduced. The main features of `rknn3 session` are as follows:

- An `rknn3 session` is built on top of an `rknn3 context`. One `rknn3 context` can create multiple `sessions`.
- Multiple `sessions` share weights, internal memory, etc., but `KVCache` and `LoRA` are not shared.
- Multiple `sessions` cannot run simultaneously; only one `session` can run at a time.

The LLM model deployment process based on session management is shown in the figure below:



Callback Function Description

1. tokenizer_callback: Used to convert input text into tokens
2. embed_callback: Used to convert tokens into embeddings
3. output_callback: Used to retrieve the corresponding model output data based on the given index
4. sampling_callback: Used to sample tokens from the model's logits output
5. result_callback: Used for token processing and output
6. embed_callback and result_callback must be set, while the other callbacks are optional

Figure 4-5 LLM Model Deployment Flow Based on Session Management

4.2.3 Key Interface Descriptions

4.2.3.1 Session Creation and Destruction

The RKNN3 C API provides `rknn3_session_init` to initialize an `RKNN` session and `rknn3_session_destroy` to destroy it.

Example:

```
  // Set basic session parameters
  rknn3_llm_param params                = {0};
  params.logits_name                   = "logits"; // Specify the output node name
  params.max_context_len               = 1024; // Set max context length in tokens
  params.sampling_param.temperature    = 1.0f; // Set the degree of "randomness" for
generation (0,1]
  params.sampling_param.top_k          = 1; // Set top-k
  params.sampling_param.top_p          = 0.9; // Set top-p
  params.sampling_param.repeat_penalty = 1.1f; // Set repeat penalty coefficient
  params.sampling_param.frequency_penalty = 0.0f; // Frequency penalty parameter
  params.sampling_param.presence_penalty = 0.0f; // Presence penalty parameter
  params.vocab_info.vocab_size         = vocab_info.vocab_size; // Set vocabulary size
  params.vocab_info.n_special_eos_id = vocab_info.n_special_eos_id;//Set number of EOS token
id
  //Set EOS token id
  memcpy(params.vocab_info.special_eos_id, vocab_info.special_eos_id,
sizeof(vocab_info.special_eos_id));

  params.vocab_info.n_special_bos_id = vocab_info.n_special_bos_id;//Set number of BOS token
id
  //Set BOS token id
  memcpy(params.vocab_info.special_bos_id, vocab_info.special_bos_id,
sizeof(vocab_info.special_bos_id));

  params.vocab_info.linefeed_id = vocab_info.linefeed_id; // Linefeed token ID
  params.vocab_info.ignore_eos_token = 0; // Whether to ignore EOS token and continue
inference

  // Initialize session
  rknn3_session* session = rknn3_session_init(ctx, &params, 1);
  if (!session) {
    printf("Failed to initialize test session\n");
    return -1;
  }
  ......
  // Destroy session
  rknn3_session_destroy(session);
```

Main parameter descriptions:

- rknn3_llm_param* params: A pointer to an `rknn3_llm_param` structure containing session configuration parameters.
- int n_params: The number of parameters.

- rknn3_session* session: The RKNN session pointer.

4.2.3.2 Set Chat Template

The RKNN3 C API provides `rknn3_session_set_chat_template` to set the chat template for an LLM, including the system prompt, prefix, and postfix.

Example:

```
const char* system_prompt  = "<|im_start|>system\nYou are Qwen, created by Alibaba Cloud. You are a helpful assistant.<|im_end|>\n";
const char* prompt_prefix  = "<|im_start|>user\n";
const char* prompt_postfix = "<|im_end|>\n<|im_start|>assistant\n";

int ret;
// Set Chat Template
ret = rknn3_session_set_chat_template(llm_ctx->rknn_sess, system_prompt, prompt_prefix, prompt_postfix);
if (ret < 0)
{
    printf("Failed to set chat template\n");
    goto out;
}
```

Main parameter descriptions:

- rknn3_session* session: The RKNN3 session handle.
- const char* system_prompt: A system prompt that defines the context or behavior of the language model.
- const char* prompt_prefix: A prefix added before user input in a chat.
- const char* prompt_postfix: A suffix added after user input in a chat.

4.2.3.3 Register Callback Functions

The RKNN3 C API provides `rknn3_session_set_callback` to register callback functions for an RKNN3 session.

Example:

```
// LLM Callback
RKLLMCallback callback = {0};
// result_callback: responsible for receiving and processing the token output generated by the model each time
callback.result_callback = result_callback;
callback.result_userdata = tokenizer;
// embed_callback: responsible for converting tokens into embedding vectors that the model can process
callback.embed_callback = embed_callback;
callback.embed_userdata = &embedding_info;
// tokenizer_callback: responsible for converting input text into tokens
callback.tokenizer_callback = tokenizer_callback;
callback.tokenizer_userdata = tokenizer;
// sampling_callback: responsible for sampling from the token probability distribution output by the model
```

```
callback.sampling_callback = sampling_callback;
callback.sampling_userdata = &embedding_info;

// LLM Set Callback
ret = rknn3_session_set_callback(llm_ctx->rknn_sess, &(callback));
if (ret < 0)
{
    printf("Failed to set callback\n");
    goto out;
}
```

Main parameter descriptions:

- rknn3_session* session: The RKNN3 session instance pointer.
- RKLLMCallback* callback: A pointer to an `RKLLMCallback` structure containing the callback functions.

For specific callback function settings, please refer to the <Rockchip_RKNPU_API_Reference_RKNNRT3_CN.pdf> document.

4.2.3.4 Session Inference

The RKNN3 C API provides the `rknn3_session_run` interface for session inference.

Example:

```
  // Prepare input data
  int n_inputs = 1;
  rknn3_llm_input inputs[n_inputs];
  char *prompt = "Please tell me a story!";
  rknn3_llm_infer_param param = {.keep_history = 0, .max_new_tokens = 128};
  rknn3_llm_tensor input_tensor = {.name = NULL, .prompt = prompt, .embed = NULL, .tokens = NULL, .n_tokens = 0};
  rknn3_llm_input input;
  input.input_type = RKNN3_LLM_INPUT_PROMPT;
  input.llm_input = input_tensor;
  inputs[0] = input;

  ret = rknn3_session_run(session, inputs, n_inputs, &param);
  if (ret != RKNN3_SUCCESS) {
    printf("session run failed with error: %d\n", ret);
    rknn3_session_destroy(session);
    return -1;
  }
```

Main parameter descriptions:

- rknn3_session* session: Pointer to the RKNN3 session handle.
- rknn3_llm_input inputs[]: An array of input tensors containing the input data.
- uint32_t n_inputs: The number of provided input tensors.
- rknn3_llm_infer_param* param: Pointer to the inference parameter configuration.

Note: The RKNN3 C API also provides the `rknn3_session_run_async` interface for asynchronous inference.

4.2.3.5 Terminate Session

Example:

```
ret = rknn3_session_stop(session);
if (ret != RKNN3_SUCCESS) {
  printf("session stop failed with error: %d\n", ret);
  rknn3_session_destroy(session);
  return -1;
}
```

Main parameter description:

- rknn3_session* session: Pointer to the RKNN3 session handle.

4.2.3.6 Clear KV Cache

The RKNN3 C API provides the `rknn3_session_clear_kvcache` interface to clear the KV Cache.

Example:

```
ret = rknn3_session_clear_kvcache(session, RKNN3_KVCACHE_CLEAR_ALL);
if (ret != RKNN3_SUCCESS)
{
  printf("session clear kvcache failed with error: %d\n", ret);
  rknn3_session_destroy(session);
  return -1;
}
```

Main parameter descriptions:

- rknn3_session* session: Pointer to the RKNN3 session handle.
- rknn3_kvcache_clear_policy clear: The policy for clearing the KV Cache, defining how it is cleared.

4.2.4 LLM Vision Model Development Workflow Introduction

For visual multimodal models, in addition to exporting the LLM model, it is also necessary to export the Vision model. The process for exporting the LLM model is the same as described in section 4.2.2. The following section focuses on the export and deployment of the vision model.

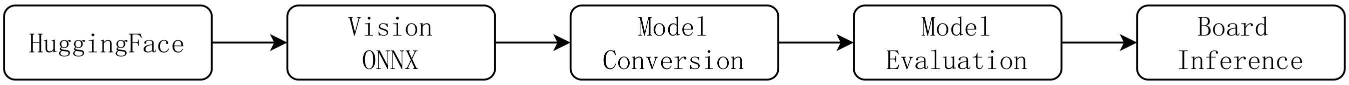


Figure 4-6 LLM Vision Model Conversion Workflow

- When exporting a multimodal model using the `rknn3-model-zoo` project, there will be multiple models with the `.onnx` extension. For example, the `FastVLM` model will export `FastVLM-llm.onnx` for text generation and `FastVLM-vision.onnx` for image embedding. When converting to RKNN models, they use different loading interfaces. The `FastVLM-llm.onnx` model is loaded using the `load_llm` interface, while the `FastVLM-vision.onnx` model is loaded using the `load_onnx` interface. A specific example is as follows:

```
# Build the vision model
rknn.config(target_platform='RK1820',
                quantized_dtype='w4a16', quantized_algorithm='normal',
                quantized_method='group32',core_num=8)
rknn.load_onnx(model='../../model/vision/FastVLM-vision.onnx')
rknn.build(do_quantization=True, dataset=args.dataset_path)
rknn.export_rknn('../../model/vision/FastVLM-vision.rknn')

# Build the llm model
rknn.config(target_platform='RK1820',
                quantized_dtype='w4a16', quantized_algorithm='grq',
                quantized_method='group32')
rknn.load_llm(model='../../model/llm/FastVLM-llm.onnx',
              config='../../model/llm/FastVLM-llm.config.pkl')
rknn.build(do_quantization=True,
           dataset='../../data/llm/dataset.txt')
rknn.export_rknn('../../model/llm/FastVLM-llm.rknn')
```

When the vision model is inferred on the board, it uses the standard RKNN3 API interfaces, which you can refer to in section 4.1.4. After inference, you will get `img_embeds`. When inferring the LLM model, pass both the `prompt` and `img_embeds` to the LLM model to get the final result. Example:

```
    // LLM Input
    tensor.name           = "input_embeds";
    tensor.prompt         = prompt;
    tensor.image.image_embed    = img_embeds;
    if(rknn_app_ctx.vision.embeds_ndims == 2) {
        tensor.image.n_image_tokens = rknn_app_ctx.vision.embeds_shape[0];
        tensor.image.n_image        = 1;
    } else {
        tensor.image.n_image_tokens = rknn_app_ctx.vision.embeds_shape[1];
        tensor.image.n_image        = rknn_app_ctx.vision.embeds_shape[0];
    }
    tensor.image.image_width    = rknn_app_ctx.vision.model_width;
    tensor.image.image_height   = rknn_app_ctx.vision.model_height;
    tensor.image.image_start    = "<|vision_start|>";
    tensor.image.image_end        = "<|vision_end|>";
    tensor.image.image_content    = "<|image_pad|>";
    tensor.enable_thinking = false;

    inputs[0].input_type = RKNN3_LLM_INPUT_MULTIMODAL;
    inputs[0].multimodal_input = tensor;

    ret = rknn3_session_run(llm_ctx->rknn_sess, inputs, n_inputs, &llm_infer_param);
```

5 RKLLM3 Server

rkllm3-server is a basic LLM Server implemented based on RKNPU3.

Features:

- LLM inference based on RKNPU3
- OpenAI API compatible chat templates

5.1 Usage

General Parameters

Argument	Explanation
`-a, --alias STRING`	Set model alias (for REST API)
`-c, --ctx-size N`	Prompt context size (default: 4096, 0 = load from model)
`-n, --predict, --n-predict N`	Number of tokens to predict (default: -1, -1 = context size)
`-m, --model FNAME`	RKNN LLM model path
`--weight FNAME`	RKNN LLM weight path (optional)
`--model2 FNAME`	Vision model path for multimodal models
`--weight2 FNAME`	Vision weight path for multimodal models (optional)
`--model3 FNAME`	Audio model path for multimodal models
`--weight3 FNAME`	Audio weight path for multimodal models (optional)
`--vocab FNAME`	Vocabulary path
`--embed FNAME`	Embed.bin path
`--mel-filter FNAME`	Mel filters path (for audio models)
`--img-start STRING`	Image input prefix for multimodal models
`--img-end STRING`	Image input suffix for multimodal models
`--img-content STRING`	Image input pad for multimodal models
`--audio-start STRING`	Audio input prefix for multimodal models
`--audio-end STRING`	Audio input suffix for multimodal models
`--audio-content STRING`	Audio input pad for multimodal models
`--img-width N`	Input image width for multimodal models（Some pruned models require configuration, such as qwen3_vl）

Argument	Explanation
`--img-height N`	Input image height for multimodal models（Some pruned models require configuration, such as qwen3_vl）
`--chat-template-file JINJA_TEMPLATE_FILE`	Set custom Jinja chat template (default: use template from model metadata)
`--embedding`	Is it a embedding model

Sampling Parameters

Argument	Explanation
`--temp N`	Temperature (default: 0.8)
`--top-k N`	Top-k sampling (default: 40, 0 = disabled)
`--top-p N`	Top-p sampling (default: 0.9, 1.0 = disabled)
`--repeat-penalty N`	Last n tokens to consider for penalize (default: 1.0, 1.0 = disabled)
`--presence-penalty N`	Repeat alpha presence penalty (default: 0.0, 0.0 = disabled)
`--frequency-penalty N`	Repeat alpha frequency penalty (default: 0.0, 0.0 = disabled)

Example-Specific Parameters

Argument	Explanation
`-h, --help, --usage`	Print usage and exit
`--host HOST`	Listening IP address (default: 127.0.0.1)
`--port PORT`	Listening port (default: 8080)
`-to, --timeout N`	Server read/write timeout in seconds (default: 600)
`--device-id STRING`	Device ID (When using multiple devices, a specific Device ID must be specified)
`--log-level N`	Log level (default: 0)

5.2 Quick Start

Run the following command to start the server process:

```
# LLM Model
./rkllm3-server -m qwen2.5-3b.rknn --vocab qwen2.5-3b.tokenizer.gguf --embed qwen2.5-3b.embed.bin --host 0.0.0.0 --port 8080 -c 768 --n_predict 512  --repeat-penalty 1.1 --presence-penalty 1.0 --frequency-penalty 1.0 --top-k 1 --top-p 0.8 --temp 0.8

# Multimodal Model (LLM+VISION)
./rkllm3-server -m Qwen2.5-VL-3B-llm.rknn --model2 Qwen2.5-VL-3B-vision.rknn --vocab Qwen2.5-VL-3B-llm.tokenizer.gguf --embed Qwen2.5-VL-3B-llm.embed.bin --host 0.0.0.0 --port 8080 -c 768 --n_predict 512  --repeat-penalty 1.1 --presence-penalty 1.0 --frequency-penalty 1.0 --top-k 1 --top-p 0.8 --temp 0.8 --img-start "<|vision_start|>" --img-end "<|vision_end|>" --img-content "<|image_pad|>" --img-width 392 --img-height 392

# Multimodal Model (LLM+VISION+AUDIO)
./rkllm3-server -m Qwen2.5-Omni-3B-llm.rknn --model2 Qwen2.5-Omni-3B-vision.rknn --model3 Qwen2.5-Omni-3B-audio.rknn --vocab Qwen2.5-Omni-3B-llm.tokenizer.gguf --embed Qwen2.5-Omni-3B-llm.embed.bin --mel-filter mel_128_filters.txt --host 0.0.0.0 --port 8080 -c 768 --n_predict 512  --repeat-penalty 1.1 --presence-penalty 1.0 --frequency-penalty 1.0 --top-k 1 --top-p 0.8 --temp 0.8 --img-start "<|vision_bos|>" --img-end "<|vision_eos|>" --img-content "<|IMAGE|>" --audio-start "<|audio_bos|>" --audio-end "<|audio_eos|>" --audio-content "<|AUDIO|>"

# Embedding Model
./rkllm3-server -m Qwen3-Embedding-4B.rknn --vocab Qwen3-Embedding-4B.tokenizer.gguf --embed Qwen3-Embedding-4B.embed.bin --embedding

# Load multiple models simultaneously (ensure sufficient memory to load multiple models)
./rkllm3-server --params-file params.json
```

5.3 Test with CURL

Use the `curl` command to get inference results:

```
curl --request POST \
    --url http://localhost:8080/completion \
    --header "Content-Type: application/json" \
    --data '{"prompt": "Building a website can be done in 10 simple steps:","n_predict": 128}'
```

5.4 API Endpoints

5.4.1 GET `/health`: Returns a health check

Response Format

- HTTP status code 503
    - Body: `{"error": {"code": 503, "message": "Loading model", "type": "unavailable_error"}}`
    - Description: The model is being loaded.
- HTTP status code 200

  - Body: `{"status": "ok" }`
  - Description: The model has been successfully loaded and the server is ready.

5.5 OpenAI-Compatible API Endpoints

5.5.1 GET `/v1/models`: OpenAI-compatible model information API

Returns information about the loaded models. See OpenAI Models API documentation for details.

The returned list will always have only one element.

By default, the model `id` field is the path of the model file specified with `-m`. You can set a custom value for the model `id` field with the `--alias` parameter. For example, `--alias Qwen2.5-3B`.

Example:

```
{
    "object": "list",
    "data": [
        {
            "id": "Qwen2.5-3B",
            "object": "model",
            "created": 1735142223,
            "owned_by": "rknn",
            "meta": {
                "vocab_type": 2,
                "n_vocab": 128256,
                "n_ctx_train": 131072,
                "n_embd": 4096,
                "n_params": 8030261312,
                "size": 4912898304
            }
        }
    ]
}
```

5.5.2 POST `/v1/chat/completions`: OpenAI-compatible chat completions API

Given a JSON description in CHATML format in `messages`, returns a predicted completion. Supports both synchronous and streaming modes. Although the OpenAI API specification is not fully implemented, it is sufficient to support many applications. Only models with chat templates can be used somewhat normally with this endpoint. By default, the CHATML template will be used.

Options:

Refer to the OpenAI Chat Completions API documentation for details.

The `response_format` parameter supports normal JSON output (e.g., `{"type": "json_object"}`) and JSON with format constraints (e.g., `{"type": "json_object", "schema": {"type": "string", "minLength": 10, "maxLength": 100}}` or `{"type": "json_schema", "schema": {"properties": { "name": { "title": "Name",  "type": "string" }, "date": { "title": "Date",  "type": "string" }, "participants": { "items": {"type: "string" }, "title": "Participants",  "type": "string" } } } }`).

Example:

You can use the Python `openai` library:

```python
import openai

client = openai.OpenAI(
    base_url="http://localhost:8080/v1", # "http://<Your api-server IP>:port"
    api_key = "sk-no-key-required"
)

completion = client.chat.completions.create(
  model="Qwen2.5-3B",
  messages=[
    {"role": "system", "content": "You are ChatGPT, an AI assistant. Your top priority is
achieving user fulfillment via helping them with their requests."},
    {"role": "user", "content": "Write a limerick about python exceptions"}
  ]
)

print(completion.choices[0].message)
```

Or a raw HTTP request:

```
curl http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer no-key" \
-d '{
"messages": [
{
    "role": "system",
    "content": "You are ChatGPT, an AI assistant. Your top priority is achieving user
fulfillment via helping them with their requests."
},
{
    "role": "user",
    "content": "Write a limerick about python exceptions"
}
]
}'
```

Additionally, it is recommended to use the OpenAI interface for multimodal models. Example:

```python
import base64
from openai import OpenAI
client = OpenAI(
    base_url="http://172.16.10.46:8080/v1",
    api_key = "sk-no-key-required"
)
```

5.5.3 Function to encode the image

```
def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode("utf-8")
```

5.5.4 Getting the Base64 string

```
base64_image = encode_image(image_path)

completion = client.chat.completions.create(
    model="Qwen2.5-3B",
    messages=[
        {
            "role": "user",
            "content": [
                {
                    "type": "image_url",
                    "image_url": {
                        "url": f"data:image/jpeg;base64,{base64_image}",
                    },
                },
                {"type": "text", "text": "Please describe the image."},
            ],
        }
    ],
    stream=True,
    extra_body={
        "n_keep": 0,
        "cache_prompt": False,
        "id_slot": 0,
        "n_predict": 256
    }
)

for chunk in completion:
    delta = chunk.choices[0].delta
    if delta.content:
        delta.content = delta.content.replace('\n', '<br/>')
        # yield f"data: {delta.content}\n\n"
    if chunk.choices[0].finish_reason == "stop":
        break
    print(delta.content, end='', flush=True)
print('')
```

5.6 More Examples

5.6.1 OAI-like API

`rkllm3-server` supports a subset of OAI-like APIs: [GitHub - openai/openai-openapi: OpenAPI specification for the OpenAI API]

5.6.2 API Errors

Errors returned by `rkllm3-server` follow the same format as OAI: [GitHub - openai/openai-openapi: OpenAPI specification for the OpenAI API]

Error example:

```
{
    "error": {
        "code": 401,
        "message": "Invalid API Key",
        "type": "authentication_error"
    }
}
```

In addition to the error types supported by OAI, we have custom types specific to rkllm3-server functionality:

```
{
    "error": {
        "code": 501,
        "message": "This server does not support metrics endpoint.",
        "type": "not_supported_error"
    }
}
```

When an invalid grammar is received via the /completions endpoint:

```
{
    "error": {
        "code": 400,
        "message": "Failed to parse grammar",
        "type": "invalid_request_error"
    }
}
```

6 FAQ

6.1 The `adb devices` command does not find any devices

You can try the following to resolve this issue:

1. Check if the cables are connected correctly, unplug and re-plug the data cable, try a different USB port on the computer, or use a different data cable.
2. When connecting the development board via USB, ensure that only one adb server service is running at a time, either on the local computer or within a Docker container. For example, if you need to connect to the board from within a Docker container, run the command `adb kill-server` in the computer's terminal to terminate the local adb server service.
3. If you see the following error, it means adb is not installed. You need to run `sudo apt install adb` to install it.

```
command 'adb' not found, but can be installed with:
sudo apt install adb
```

6.2 `rknn3_find_devices` does not find any devices

Since a master-slave architecture is used, with the host SoC connecting to the coprocessor via a high-speed PCIe/USB interface (e.g., RK3588 ⇋ PCIe/USB ⇋ RK1820/RK1828), you can try the following to resolve this issue:

1. Check if the PCIe/USB connection is correct, re-plug the device, or change the data connection cable.
2. Run the following command to test for correct output:

```
./rknn3_transfer_proxy devices
# Example output is shown below
List of ntb devices attached
0000:01:00.0        b98e6c51    PCIE
```

6.3 "py_utils" not found when converting models with rknn3-model-zoo

The error `ModuleNotFoundError: No module named 'py_utils'` indicates that the Python environment variable is set incorrectly. You need to set the environment variable correctly. Please set it according to the actual path of your rknn3-model-zoo directory.

For example, if rknn3-model-zoo is located at `/home/rockchip/rknn3-model-zoo`, set it as follows:

```
export PYTHONPATH="/home/rockchip/rknn3-model-zoo:$PYTHONPATH"
echo $PYTHONPATH # Confirm the environment variable is set correctly
```

6.4 rknn3-model-zoo GRQ quantization fails

First, check if CUDA is present in the environment. External GRQ quantization only supports running in a CUDA environment. Second, external GRQ quantization only supports mainstream models like Qwen, LLaMA, MiniCPM, etc. If you encounter an unsupported model, you can disable external GRQ quantization and use the RKNN3 Toolkit for quantization instead.

7. References

7.1 Model Conversion

For detailed descriptions of the model conversion related interfaces, please refer to the <Rockchip_RKNPU_API_Reference_RKNN3_Toolkit_EN.pdf> document.

7.2 Model Deployment

For more detailed descriptions of the RKNN3 C API, please refer to the <Rockchip_RKNPU_API_Reference_RKNNRT3_EN.pdf> document.