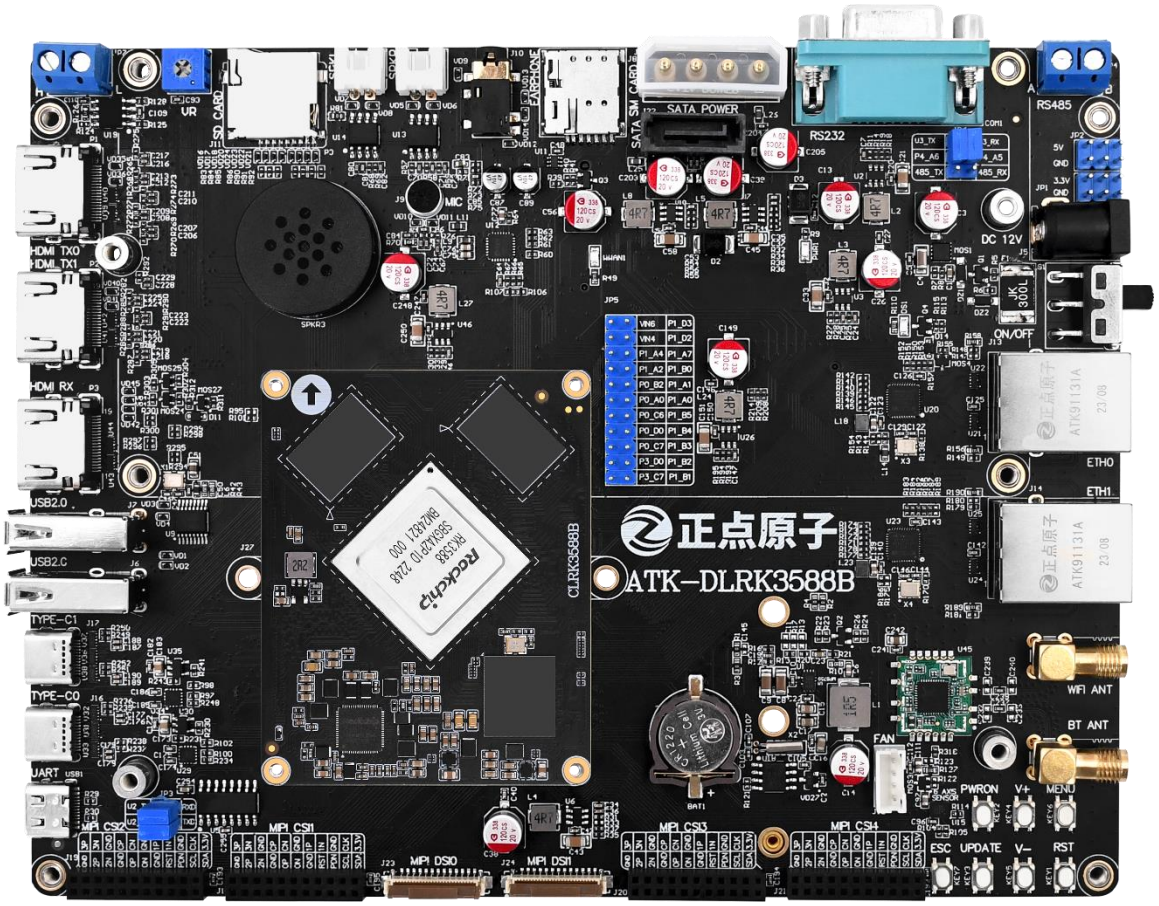
ATK-DLRK3588

AI Routine Supplementary Test Manual

V1.0



ALIENTEK

1. Shopping:

TMALL: https://zhengdianyuanzi.tmall.com

TAOBAO: https://openedv.taobao.com

2. Download

Address: http://www.openedv.com/docs/index.html

3. FAE

Website : www.alientek.com
Forum : http://www.openedv.com/forum.php
Videos : www.yuanzige.com
Fax : +86 - 20 - 36773971
Phone : +86 - 20 - 38271790

Disclaimer

The product specifications and instructions mentioned in this document are for reference only and subject to update without prior notice; Unless otherwise agreed, this document is intended as a product guide only, and none of the representations made herein constitutes a warranty of any kind.

In order to get the latest version of product information, please regularly visit the download center or contact the customer service of Taobao ALIENTEK flagship store. Thank you for your tolerance and support.

Revision History:

Version	Version Update Notes	Responsible person	Proofreading	Date
V1.0	release officially	ALIENTEK	ALIENTEK	2025.01.21

Catalogue

Brief Please read this document carefully before proceeding!

Differences Explanation:

Please watch this supplementary manual before proceeding!!!

Due to the rapid update speed of the rknn-toolkit2 tool and the rknpu driver, and the requirement for using compatible versions of both the rknpu driver and the rknn-toolkit, please experience the ai testing manual and the development board before watching this manual. This manual is intended to supplement the updates made by the RKNNPU team for new model adaptation and usage experience, making it convenient for readers to test and use the updated tools and new supported models. This document will be updated gradually. In simple terms, to keep up with the latest, please refer to this document.

This time, the PC-end model conversion and routine compilation environment:

- Ubuntu 24.04.1 LTS
- Anaconda3-2024.10-1-Linux-x86_64
- python 3.12(In Conda environment)
- rknn-toolkit2 2.3.0(In Conda environment)

The deployment environment for the development board side:

- python 3.10.5
- rknn-toolkitlite2

This document will be merged with the AI Test Manual at an appropriate time. If you have any questions, you can send an email to luominghui@alientek.com or join the ALIENTEK QQ group chat for consultation and discussion.

QQ group:

Users of the ALIENTEK RK3588 development board: 966712463(Please provide the order number)

ALIENTEK Rockchip RK Communication Group: 286877688(It's full. Please join Group 2.)

ALIENTEK Rockchip RK Communication Group2: 775783660

Chapter 1. Setup of the RKNN AI Routine Testing Development Environment

Setup of the RKNN AI Routine Testing Development Environment

Before conducting the routine testing with the latest sdk_r8 system, it is necessary to install a new compiler and related software for model conversion. This chapter is divided into the following sections:

1. Installing the cross-compilation toolchain
2. Installing Anaconda software
3. Updating the adbd service to the file system

1.1 Install the cross-compilation toolchain

Before compiling the program, we need to install the cross-compilation toolchain on Ubuntu. This toolchain is compiled using the RK3588 Buildroot SDK and includes the libraries required by our AI routines, which can be directly used on the board. Copy the data disk "Development Board CD A Disk - Basic Data → 05_tools → 03, Cross Compilation Tools → atk-dlrk3588-toolchain-aarch64-buildroot-linux-gnu-x86_64_5_10_r8_20250120-v1.1.run" to any directory in Ubuntu (the cross-compilation toolchain will be updated continuously, use the actual directory toolchain name as the reference).

If there is no executable permission, you can execute the following command to give it executable permission.

```
chmod a+x atk-dlrk3588-toolchain-aarch64-buildroot-linux-gnu-x86_64_5_10_r8_20250120-v1.1.run
```

Open the terminal and execute the following commands to install the compilation toolchain. The installation process is as shown in Figure 2.2.2.1.

```
./atk-dlrk3588-toolchain-aarch64-buildroot-linux-gnu-x86_64_5_10_r8_20250120-v1.1.run
```

When the prompt "Enter the target directory for the toolchain (default: /opt/atk-dlrk3588-toolchain):" appears, it indicates whether to select the default installation in the /opt/atk-dlrk3588-toolchain directory. It is recommended to directly select the default installation path (note!!!! This will overwrite the previously installed compiler, so if the original compiler is still useful, please modify the path or reinstall the original compiler next time you need it). Simply press the Enter key. When the prompt "You are about to install the toolchain to "/opt/atk-dlrk3588-toolchain". Proceed[Y/n]?" appears, directly press "Y" and then press Enter. After entering the Ubuntu password and pressing Enter, when the prompt "$. source /opt/atk-dlrk3588-toolchain/environment-setup" appears, it indicates that the installation is complete.

1.2 Install Anaconda software

If you have previously used a general version, and if you have already installed Anaconda, there is no need to reinstall it. If you haven't installed it, refer to Section 1.3 of "ATK-DLRK3588 "Development Board CD-ROM A Disk - Basic Materials → 10_user_manual → 01, Test Documents → 03 [ALIENTEK] ATK-DLRK3588_AI Routine Test Manual V1.1.pdf" of "ATK-DLRK3588".

1.3 Update the adbd service to the file system

After completing the above tasks, it is necessary to update the adbd tool to the root file system. Otherwise, it may cause subsequent inability to perform board connection reasoning. First, copy "Development Board CD-ROM A Disk - Basic Data → 01_codes → 04_AI_Routine → 04, r8 Source Code → adbd.zip" to the Ubuntu directory for transmission to the development board, and connect the OTG interface of the development board to the virtual machine. Open the terminal in the copied directory, and execute the following commands to transfer them to the /usr/bin directory of the development board and make the executable files available.

```
unzip adbd.zip
adb push adbd/linux-aarch64/adbd /usr/bin
adb shell
chmod +x /usr/bin/adbd
```

Just restart the development board.

Chapter 2. Update the npu driver

RKNN-Toollkit2 is a model conversion tool implemented in Python language, which can convert models exported from other training frameworks into RKNN models and provides relatively limited inference interfaces to assist users in testing the model conversion effect. The website link is: https://github.com/rockchip-linux/rknn-toolkit2. RKNPU2 is a component on the board, providing NPU driver, and providing model loading, model inference and other functions based on C language. RKNPU2 has a consistent version number. There may be incompatibility issues between different versions, to avoid causing unnecessary troubles, it is recommended that users use the same version of RKNN-Toolkit2 and RKNPU2. Version updates usually include bug fixes and performance optimizations. It is recommended that users use the latest version.

This chapter is divided into the following sections:

1. Download the rknpu2 driver compression package
2. Update the inference runtime library for the npu board connection

2.1 Download the rknpu2 driver compression package

In order to facilitate the subsequent use of OTG and the development board for AI-related development performance tests and board connection inference and other functions, it is necessary to update the NPU driver of rk3588 (precisely speaking, it is not a driver, but a board-end board connection service. Regarding the rknpu driver, we have updated it to the current latest version 0.9.6). You can view the NPU driver through the following command.

```
dmesg | grep rknpu
```

As shown in the figure below, the current RKNPU2 driver version is 0.9.6.

```
root@ATK-DLRK3588:/# dmesg  | grep rknpu
[    4.870800] RKNPU fdab0000.npu: RKNPU: rknpu iommu is enabled, using iommu mode
[    4.872352] [drm] Initialized rknpu 0.9.6 20240322 for fdab0000.npu on minor 1
[    4.877634] debugfs: Directory 'fdab0000.npu-rknpu' with parent 'vdd_npu_s0' already present!
```

Figure 2.1-1 Check the NPU driver version

Here is the practical operation on how to update the NPU driver and board service for rk3588. Open the GitHub download website https://github.com/airockchip/rknn-toolkit2, click on the v2.3.0 in the right side of the Releases column, and download it. You will find that the compressed file contains the rknn-toolkit2 conversion tool and rknpu driver files. There is also a downloaded compressed file on the development board data CD. You can directly use the rknn-toolkit2-2.3.0.zip in the A drive of the development board CD-ROM - Basic Data → 01_codes → 04_AI_Examples → 04, r8 source code.

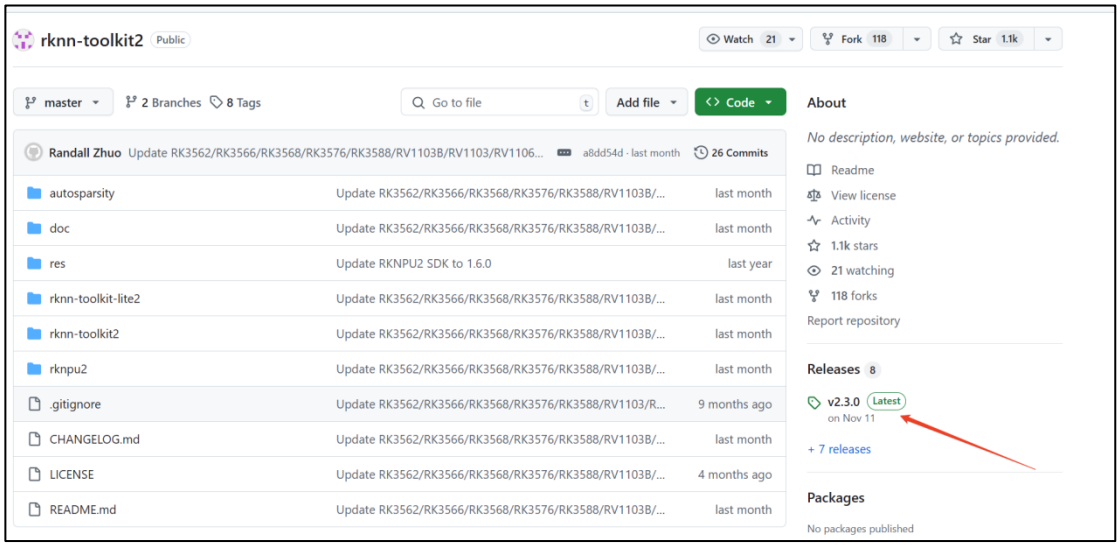


Figure 2.1-2 The GitHub official website of the rknn-toolkit2 tool

Click on "Source code (zip)" under v2.3.0 to download the compressed file.

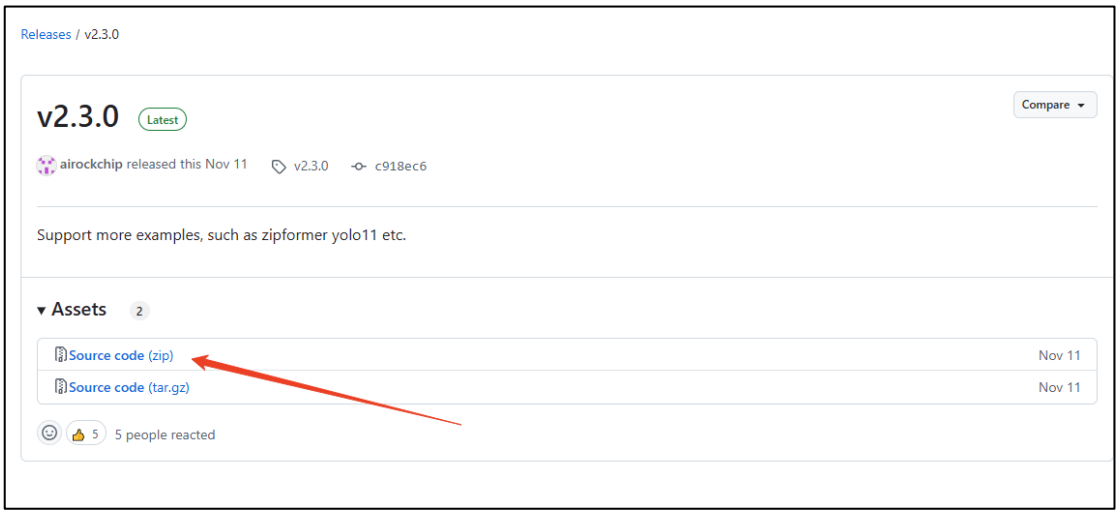


Figure 2.1-3 Download the compressed package of the rknn-toolkit2 tool.

After downloading, copy and unzip the compressed package to the "software" directory under Ubuntu (it is recommended to use the unzip command for unzipping, otherwise errors may occur). We can refer to the "doc/rknn_server_proxy.md" file in the "rknn-toolkit2-2.3.0" folder to update the driver files, as shown in the following picture.

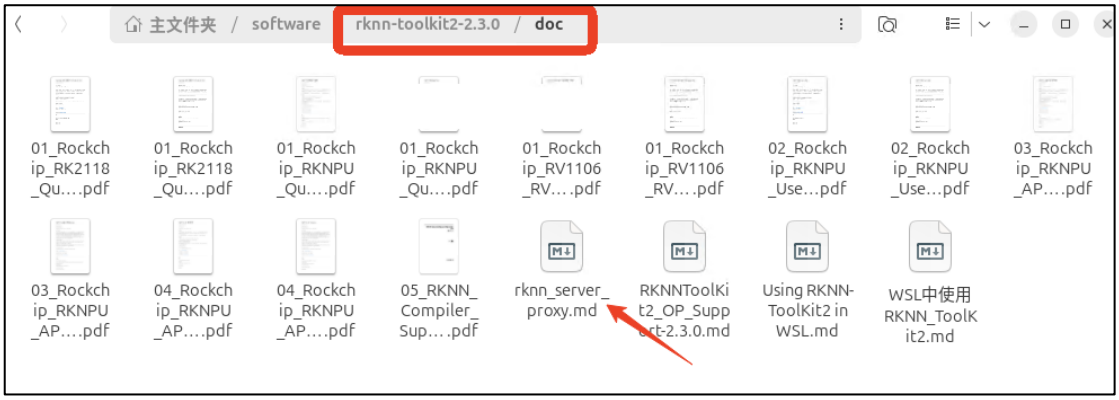


Figure 2.1-4 rknn_server_proxy.md file

2.2 Update the inference runtime library for the NPU board

To update the NPU board inference runtime library, we need to transfer the librknnrt.so file in the runtime directory of the downloaded rknpu2 folder, as well as the rknn_server file and scripts to the development board. The main files in these two directories are as follows.

One is the directory rknpu2/runtime/Linux/librknn_api/aarch64/.

Figure 2.2-1 The librknnrt.so file

The other one is in the directory: rknpu2/runtime/Linux/rknn_server/aarch64/usr/bin/

Figure 2.2-2 The rknn_server file and related scripts

Before updating the NPU driver service, you need to first connect the power supply of the development board and connect the development board's OTG via a Type-C cable to the computer. Then, enter the rknpu2 driver directory, open the terminal in the NPU driver directory, and use adb for data transfer.

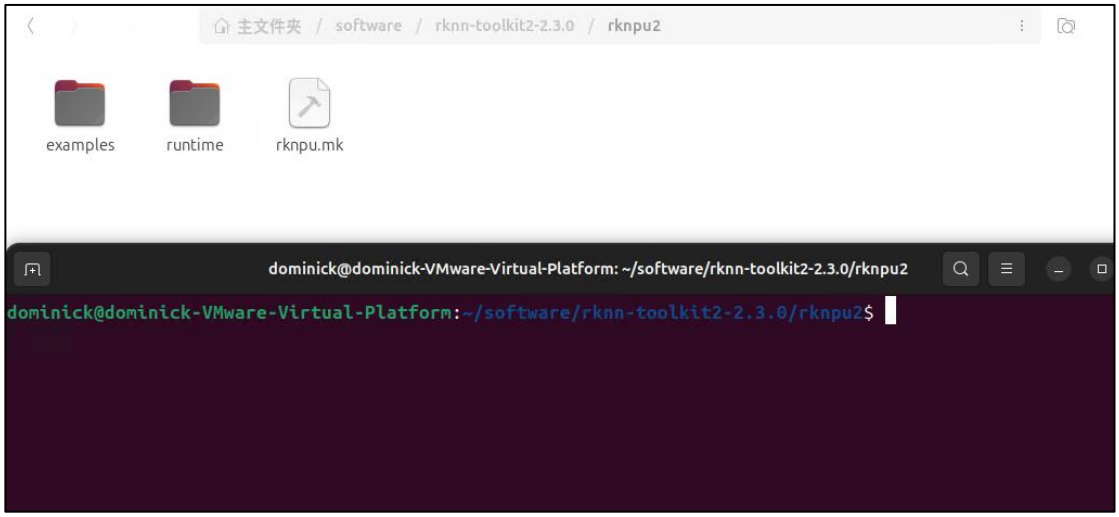


Figure 2.2-3 Open the terminal in the rknpu2 directory.

First, check if adb is installed. If not, you can execute the following command to install it.

```
sudo apt install adb -y
```

Execute the following commands one by one in the Ubuntu terminal to transfer the files to the development board.

```
adb push runtime/Linux/librknn_api/aarch64/librknnrt.so /usr/lib
```

```
dominick@dominick-VMware-Virtual-Platform:~/software/rknn-toolkit2-2.3.0/rknpu2$ adb push runtime/Linux/librknn_api/aarch64/lib
rknnrt.so /usr/lib
adb server version (40) doesn't match this client (41); killing...
* daemon started successfully
runtime/Linux/librknn_api/aarch64/librknnrt.so: 1 f...hed, 0 skipped. 24.5 MB/s (7259064 bytes in 0.283s)
```

Figure 2.2-4 Transfer the RKNN inference library via ADB to the development board.

```
Adb push runtime/Linux/rknn_server/aarch64/usr/bin/* /usr/bin
```

```
dominick@dominick-VMware-Virtual-Platform:~/software/rknn-toolkit2-2.3.0/rknpu2$ adb push runtime/Linux/rknn_server/aarch64/usr
/bin/* /usr/bin
runtime/Linux/rknn_server/aarch64/usr/bin/restart_rknn.sh: 1 file pushed, 0 skipped. 0.0 MB/s (252 bytes in 0.029s)
runtime/Linux/rknn_server/aarch64/usr/bin/rknn_server: 1 file pushed, 0 skipped. 42.6 MB/s (455600 bytes in 0.010s)
runtime/Linux/rknn_server/aarch64/usr/bin/start_rknn.sh: 1 file pushed, 0 skipped. 0.2 MB/s (71 bytes in 0.000s)
3 files pushed, 0 skipped. 3.8 MB/s (455923 bytes in 0.114s)
```

Figure 2.2-5 Transfer the rknn_server and scripts to the development board via adb.

Enter the shell environment of the development board in the Ubuntu terminal, grant execution permissions, and restart the service.

```
adb shell
chmod +x /usr/bin/rknn_server
chmod +x /usr/bin/start_rknn.sh
chmod +x /usr/bin/restart_rknn.sh
restart_rknn.sh
```

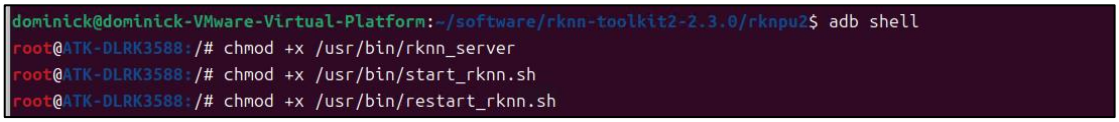


Figure 2.2-6 Enter the shell command line of the development board.

If you execute the restart_rknn.sh script on the development board and restart the service, the version number will be printed.

Figure 2.2-7 Check the version number of rknn

Execute the following command to query the version of rknn_server, and it will print the version of rknn_server.

```
strings /usr/bin/rknn_server | grep build
```

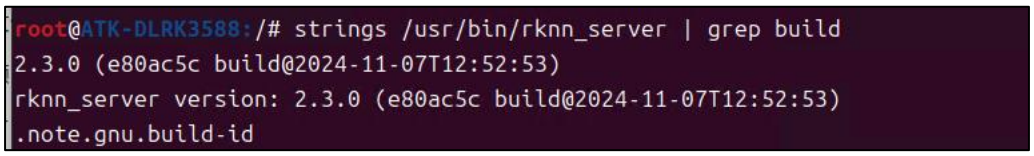


Figure 2.2-8 Check the version of the rknn_server service

Execute the following command to query the version of librknnrt.so, and it will print the version of librknnrt.

```
strings /usr/lib/librknnrt.so | grep version
```

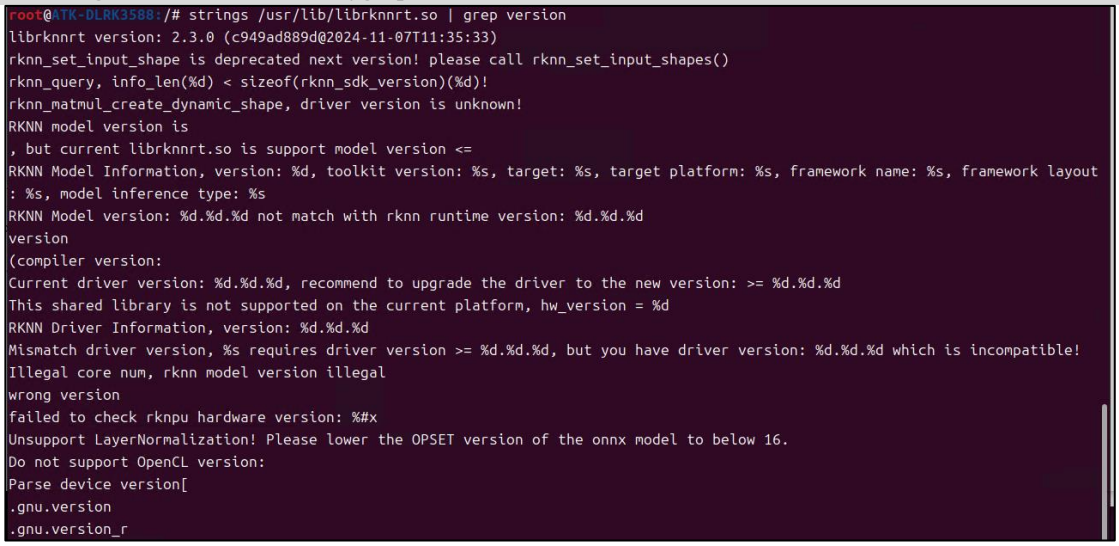


Figure 2.2-9 Check the runtime version of librknnrt.

Input "exit" to exit the development board system and return to the Ubuntu terminal.

Chapter 3. Install the rknn-toolkit2 conversion environment

As the application of deep learning becomes increasingly widespread in various fields, it is becoming increasingly important for embedded and edge computing devices to be able to process deep learning tasks in real time and efficiently. However, due to the hardware limitations of these devices (such as computing power, memory size, etc.), directly running traditional deep learning models on them is often impractical.

The emergence of rknn-toolkit2 has solved this problem. It provides a complete solution that allows developers to convert already trained deep learning models (such as traditional TensorFlow, Pytorch, onnx format models) into optimized rknn format. These format models can fully utilize the hardware characteristics of devices like RK3568, RK3588, etc., thereby ensuring the model performance while reducing the demand for computing resources.

This chapter will be divided into the following sections:

1. Create a new Conda environment
2. Install the rknn-toolkit2 tool

3.1 Create a new Conda environment

Previously, we downloaded Anaconda. To prevent confusion in the subsequent compilation environment, this comes in handy here. First, we create a new Conda virtual environment and then install the rknn-toolkit2. Open the terminal and execute the following command to create a Conda environment named "python3.12-tk2-2.0" with a Python version of Python 3.12.

```
conda create --name python3.12-tk2-2.3 python=3.12
```

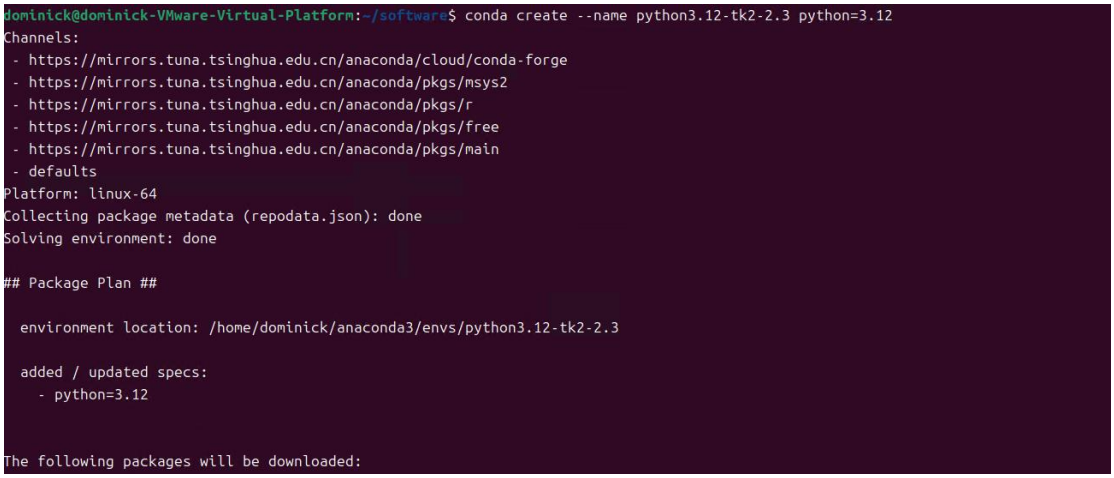


Figure 3.1-1 Create a Conda environment

The program stops at "Proceed". Press "y" and then press "Enter".

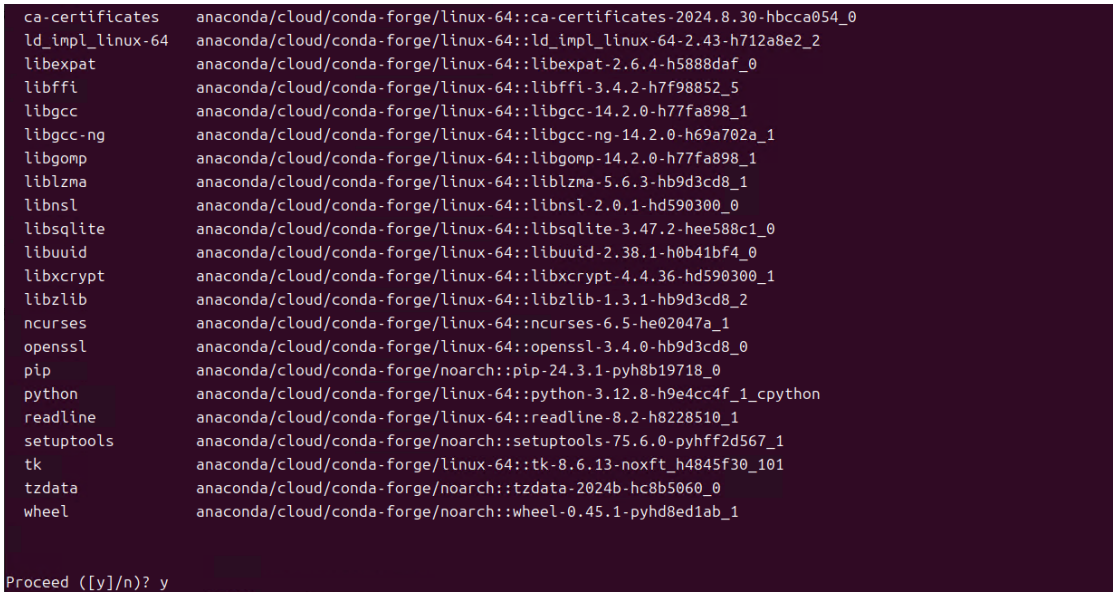


Figure 3.1-2 Select "y" and press Enter

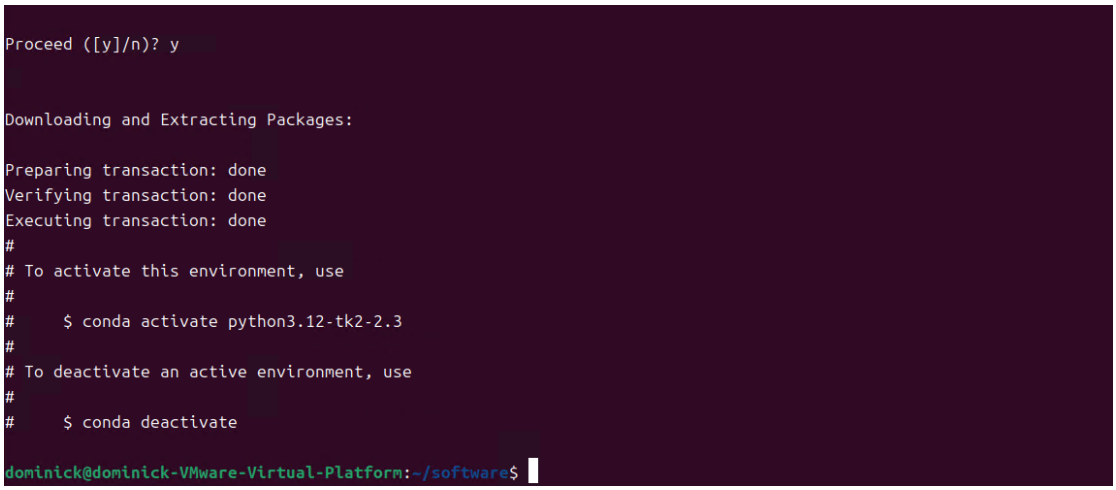


Figure 3.1-3 The conda environment has been successfully created.

At this point, the conda environment has been successfully set up. You can enter the conda environment by entering the following command in the command line terminal.

```
conda activate python3.12-tk2-2.3
```

To exit the conda environment, use the following command in the conda environment.

```
conda deactivate
```

3.2 Install the rknn-toolkit2 tool

3.2.1 Install the dependencies for rknn-toolkit2

Enter the rknn-toolkit2-2.3.0/rknn-toolkit2/packages folder, open the terminal and execute the following command to activate the conda environment.

```
conda activate python3.12-tk2-2.3
```

If the terminal command line shows "(python3.12-tk2-2.3)", it means that the newly created conda environment named "py3.12" has been successfully entered.

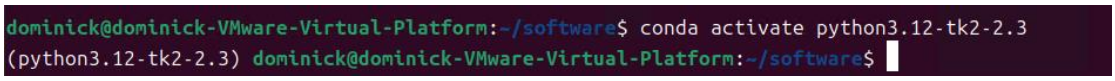


Figure 3.2-1 Enter the Conda environment

Before installing the rknn-toolkit2, it is necessary to install the related dependent libraries first. Before executing the following commands, it is necessary to check whether you have entered the rknn-toolkit2/packages folder. If not, you need to enter the packages folder first.

```
pip install -r requirements_cp312-2.3.0.txt -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

```
(python3.12-tk2-2.3) dominick@dominick-VMware-Virtual-Platform:~/software/rknn-toolkit2-2.3.0/rknn-toolkit2/packages/x86_64$ pip install -
r requirements_cp312-2.3.0.txt -i https://pypi.tuna.tsinghua.edu.cn/simple/
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple/
Collecting protobuf<=4.25.4,>=4.21.6 (from -r requirements_cp312-2.3.0.txt (line 4))
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/ca/6c/cc7ab2fb3a4a7f07f211d8a7bbb76bba633eb09b148296dbd4281e217f95/protobuf-4.25.
4-cp37-abi3-manylinux2014_x86_64.whl (294 kB)
Collecting psutil>=5.9.0 (from -r requirements_cp312-2.3.0.txt (line 7))
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/58/4d/8245e6f76a93c98aab285a43ea71ff1b171bcd90c9d238bf81f7021fb233/psutil-6.1.0-c
p36-abi3-manylinux_2_12_x86_64.manylinux2010_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (287 kB)
Collecting ruamel.yaml>=0.17.21 (from -r requirements_cp312-2.3.0.txt (line 8))
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/73/67/8ece580cc363331d9a53055130f86b096bf16e38156e33b1d3014fffda6b/ruamel.yaml-0.
18.6-py3-none-any.whl (117 kB)
Collecting scipy>=1.9.3 (from -r requirements_cp312-2.3.0.txt (line 9))
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/8e/ee/8a26858ca517e9c64f84b4c7734b89bda8e63bec85c3d2f432d225bb1886/scipy-1.14.1-c
p312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (40.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.8/40.8 MB 7.9 MB/s eta 0:00:00
Collecting tqdm>=4.64.1 (from -r requirements_cp312-2.3.0.txt (line 10))
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/d0/30/dc54f88dd4a2b5dc8a0279bdd7270e735851848b762aeb1c1184ed1f6b14/tqdm-4.67.1-py
3-none-any.whl (78 kB)
Collecting opencv-python>=4.5.5.64 (from -r requirements_cp312-2.3.0.txt (line 11))
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/3f/a4/d2537f47fd7fcfba966bd806e3ec18e7ee1681056d4b0a9c8d983983e4d5/opencv_python-
4.10.0.84-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (62.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 MB 8.4 MB/s eta 0:00:00
```

Figure 3.2-2 Dependencies required for installing rknn-toolkit2

The -i https://pypi.tuna.tsinghua.edu.cn/simple/ parameter here means specifying the source as pip and the mirror source as https://pypi.tuna.tsinghua.edu.cn/simple/. By opening requirements_cp312-2.3.0.txt, you can see that this mirror source is the recommended download source by Rockchip Microelectronics. Therefore, we will try to perform the download and installation operation according to the official recommended source.

打开(O) requirements_cp312-2.3.0.txt
~/rknn-modelzoo-2.3.0/rknn-toolkit2-2.3.0/rknn-toolkit2/packages/x86_64

```
# if install failed, please change the pip source to 'https://pypi.tuna.tsinghua.edu.cn/simple/'

# base deps
protobuf>=4.21.6,<=4.25.4

# utils
psutil>=5.9.0
ruamel.yaml>=0.17.21
scipy>=1.9.3
tqdm>=4.64.1
opencv-python>=4.5.5.64
fast-histogram>=0.11
numpy<=1.26.4

# base
onnx>=1.10.0
onnxruntime>=1.17.0
torch>=1.10.1,<=2.4.0
```

Figure 3.2-3 Installation source for the required libraries of rknn-toolkit2

When the following message "Successfully installed" is displayed, it indicates that the related dependent libraries have been installed successfully.

```
Installing collected packages: mpmath, flatbuffers, typing-extensions, tqdm, sympy, ruamel.yaml.clib, psutil, protobuf, packaging, nvidia-
nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu1
2, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, numpy, networkx, MarkupSafe, humanfriendly, fsspec, filelock, triton, scipy, ruamel.yaml, o
pencv-python, onnx, nvidia-cusparse-cu12, nvidia-cudnn-cu12, jinja2, fast-histogram, coloredlogs, onnxruntime, nvidia-cusolver-cu12, torch
Successfully installed MarkupSafe-3.0.2 coloredlogs-15.0.1 fast-histogram-0.14 filelock-3.16.1 flatbuffers-24.3.25 fsspec-2024.10.0 humanf
riendly-10.0 jinja2-3.1.4 mpmath-1.3.0 networkx-3.4.2 numpy-1.26.4 nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda
-nvrtc-cu12-12.1.105 nvidia-cuda-runtime-cu12-12.1.105 nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu12-11.0.2.54 nvidia-curand-cu12-10.3.2.10
6 nvidia-cusolver-cu12-11.4.5.107 nvidia-cusparse-cu12-12.1.0.106 nvidia-nccl-cu12-2.20.5 nvidia-nvjitlink-cu12-12.6.85 nvidia-nvtx-cu12-1
2.1.105 onnx-1.17.0 onnxruntime-1.20.1 opencv-python-4.10.0.84 packaging-24.2 protobuf-4.25.4 psutil-6.1.0 ruamel.yaml-0.18.6 ruamel.yaml.
clib-0.2.12 scipy-1.14.1 sympy-1.13.3 torch-2.4.0 tqdm-4.67.1 triton-3.0.0 typing-extensions-4.12.2
```

Figure 3.2-4 The required libraries for rknn-toolkit2 have been installed successfully.

3.2.2 Installation of the rknn-toolkit2 tool

Stay in the conda environment python3.12-tk2-2.3.0 and go to the rknn-toolkit2/packages/x86_64 directory. Execute the following command to enter the directory where the corresponding whl file is located, and use the pip tool to install rknn-toolkit2-2.3.0.

```
cd ~/software/rknn-toolkit2-2.3.0/rknn-toolkit2/packages/x86_64
pip install rknn_toolkit2-2.3.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
```

After the installation is completed, the message "Successfully installed rknn-toolkit2-2.3.0" will appear on the last line, indicating that the installation was successful.

Figure 3.2-5 rknn-toolkit2 installation completed

3.2.3 Testing rknn-toolkit2

We enter the routine directory of examples/tflite/mobilenet_v1, to test if the mobilenet_v1 routine works properly. If it can be converted and inferred normally, then it indicates that the rknn-toolkit2 installation is successful. Execute the following command.

```
cd ../../examples/tflite/mobilenet_v1
python test.py
```

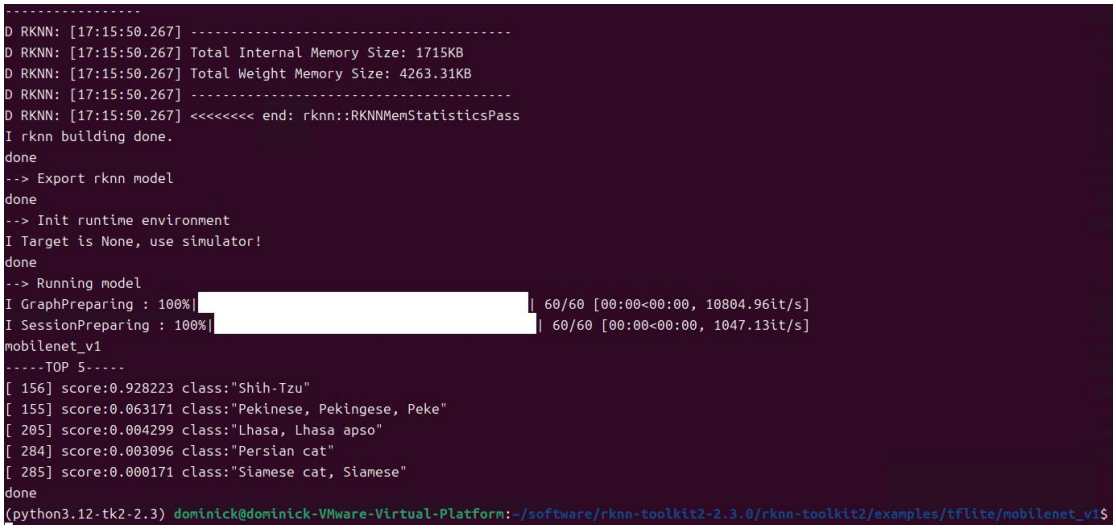


Figure 3.2-6 Run the conversion routine for the mobilenet_v1 model.

It can print out the top-5 classification results of the model. The test is normal. If the test is not normal, please refer to the following.

Note: In the dependency description file of the rknn-toolkit2 version 2.3.0, it is not required to install TensorFlow by default (it is guessed that this is because it needs to reduce the size. If you need to convert a TensorFlow model, it is recommended to install it. If you do not need to convert a TensorFlow model, you can choose not to install it. You can then test in other framework type directories). Therefore, an error will occur. Simply execute the following command to install TensorFlow and then repeat the above test operation to successfully run.

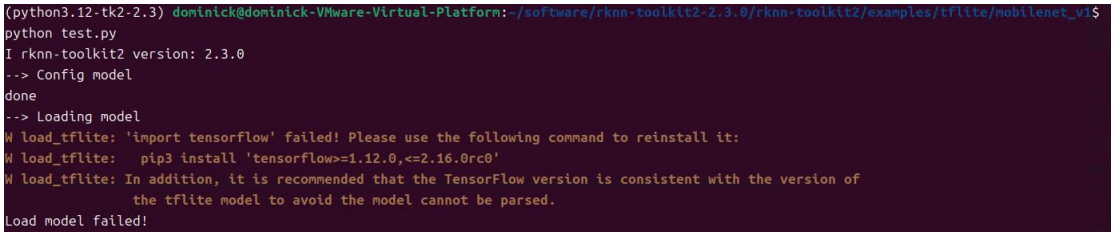


Figure 3.2-7 The test for converting the TensorFlow model failed.

```
pip3 install 'tensorflow>=1.12.0,<=2.16.0rc0' -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

Chapter 4. Model Zoo Testing and Introduction

The environment has been prepared previously. Next, we will test the corresponding AI routines and deploy them to the development board for testing.

This chapter is divided into the following sections.

1. Introduction to modelzoo and Difference Explanation
2. Testing the yolov8_obb Model
3. Testing the yolov10 Model
4. Testing the yolov11 Model
5. Testing the yolo_world Model
6. Testing the yolov8_pose Model
7. Testing the mobilesam Model
8. Testing the clip Model
9. Testing the wav2vec2 Model
10. Testing the whisper Model
11. Testing the zipformer Model
12. Testing the yamnet Model
13. Testing the mms_tts Model

4.1 ModelZoo Introduction and Difference Explanation

The RKNN Model Zoo provides a large number of example codes for model conversion and deployment, aiming to help users quickly run various models on development boards equipped with Rockchip chips.

4.1.1 Difference Explanation

There are certain differences between the rknn_model_zoo-2.3.0 version and the 2.0.0 version. Firstly, some conversion model scripts and test model scripts are no longer unified (mobilenet has not changed and still uses a single script to convert and test the model), and they have been separated into two scripts, namely convert.py and the inference script for the corresponding model.

Some model examples will provide different precision conversion options. This chapter only provides the test methods for the new model routines.

4.1.2 Preparation before testing

First, the following files from the development board CD (A disk - Basic Data\01_codes\01_AI_Routine\04, r8 Source Code) need to be copied to the software directory of Ubuntu.

- gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu.tar.xz
- rknn_model_zoo-2.3.0.zip

The first compressed file gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu.tar.xz should be decompressed to the ~/software directory in Ubuntu for the subsequent routine compilation.

Similarly, copy the second compressed file to the ~/software directory in Ubuntu and decompress it. Note that after decompressing the compressed file, please change the name of the rknn_model_zoo-2.3.0 folder to rknn_model_zoo. Otherwise, there may be errors later. The decompression is as follows.

After decompression, go to the "software" directory.

Right-click in the current directory to open the terminal, enter the "modelzoo" directory, and execute the following command to grant execution permissions for easier compilation of subsequent routines.

```
cd rknn_model_zoo
chmod +x build-linux.sh
```

```
dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo$ chmod +x build-linux.sh
dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo$ ls -l
总计 148
drwxrwxr-x 14 dominick dominick  4096 12月 13 17:57 3rdparty
drwxrwxr-x  2 dominick dominick  4096 12月 13 17:57 asset
-rw-rw-r--  1 dominick dominick  4949 12月 13 17:57 build-android.sh
-rwxrwxr-x  1 dominick dominick  5293 12月 13 17:57 build-linux.sh
drwxrwxr-x  6 dominick dominick  4096 12月 13 17:57 datasets
drwxrwxr-x  2 dominick dominick  4096 12月 13 17:57 docs
drwxrwxr-x 30 dominick dominick  4096 12月 13 17:57 examples
-rw-rw-r--  1 dominick dominick  8719 12月 13 17:57 FAQ_CN.md
-rw-rw-r--  1 dominick dominick 10212 12月 13 17:57 FAQ.md
-rw-rw-r--  1 dominick dominick 11357 12月 13 17:57 LICENSE
drwxrwxr-x  3 dominick dominick  4096 12月 16 14:50 py_utils
-rw-rw-r--  1 dominick dominick 27350 12月 13 17:57 README_CN.md
-rw-rw-r--  1 dominick dominick 28033 12月 13 17:57 README.md
-rw-rw-r--  1 dominick dominick 11713 12月 13 17:57 scaling_frequency.sh
drwxrwxr-x  2 dominick dominick  4096 12月 13 17:57 utils
dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo$
```

Figure 4.1-1 Grant execute permission to the compilation script

4.2 Test the yolov8_obb model

First, you need to download the model. Download the model by executing the model download script. Before execution, you need to check if the Ubuntu network is normal. Execute the following command in the rknn_model_zoo directory after decompressing Ubuntu to download the model.

```
cd examples/yolov8_obb/model
sh download_model.sh
```

```
dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo$ cd examples/yolov8_obb/model/
dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo/examples/yolov8_obb/model$ sh download_model.sh
--2024-12-16 15:38:05--  https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/yolov8
_obb/yolov8n-obb.onnx
正在解析主机 ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
正在连接 ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度： 12382727 (12M) [application/octet-stream]
正在保存至：‘./yolov8n-obb.onnx’

./yolov8n-obb.onnx        100%[===================================>]  11.81M  10.0MB/s    用时 1.2s

2024-12-16 15:38:06 (10.0 MB/s) - 已保存 ‘./yolov8n-obb.onnx’ [12382727/12382727])
```

Figure 4.2-1 Download the yolov8_obb model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the yolov8_obb routine directory under the modelzoo directory.

```
cd ~/software/rknn_model_zoo/examples/yolov8_obb/python
```

If you are using the rk3588 development board, execute the following command to convert the yolov8_obb model.

```
python3 convert.py ../model/yolov8n-obb.onnx rk3588 i8
```

The last parameter i8 refers to the quantization precision. It can also be changed to fp as per requirements (note: if it is changed to another quantization precision, the quantization precision of the subsequent inference must be modified to ensure the correct result can be obtained during inference).

Note: i8 refers to the int8 type.

fp refers to the float type. Generally, no quantization precision is applied, and the inference speed will be slower.

```
(python3.12-tk2-2.3) dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo/examples/yolov8_obb/python$ python3 convert.
py ../model/yolov8n-obb.onnx rk3588 i8
I rknn-toolkit2 version: 2.3.0
--> Config model
done
--> Loading model
I Loading : 100%|| 147/147 [00:00<00:00, 47365.96it/s]
done
--> Building model
I OpFusing 0: 100%|| 100/100 [00:00<00:00, 316.27it/s]
I OpFusing 1 : 100%|| 100/100 [00:00<00:00, 270.42it/s]
I OpFusing 2 : 100%|| 100/100 [00:00<00:00, 133.84it/s]
I GraphPreparing : 100%|| 175/175 [00:00<00:00, 9794.02it/s]
I Quantizating : 100%|| 175/175 [00:06<00:00, 26.11it/s]
W build: The default input dtype of 'images' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '/model.22/Concat_1_output_0' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '/model.22/Concat_2_output_0' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '/model.22/Concat_3_output_0' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '/model.22/Sigmoid_output_0' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
I rknn building ...
I rknn building done.
done
--> Export rknn model
done
```

Figure 4.2-2 Convert the yolov8_obb model to a rknn model and complete the process.

Carry out model testing to see if it can perform normal inference. Pay attention to specifying the device ID of the rk3588 development board to the corresponding device through the "device_id" parameter. If there are multiple rk3588 devices, you can also specify the rk3588 devices for inference through the "--devices_id" parameter. After connecting the development board, execute "adb devices" under Ubuntu to obtain the device ID.

```
adb devices
```

```
(python3.12-tk2-2.3) dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo/examples/yolov8_obb/python$ adb devices
List of devices attached
11a48dc6b976db80        device
```

Figure 4.2-3 Obtain the device ID

```
python3 yolov8_obb.py --model_path ../model/yolov8n_obb.rknn --target rk3588 --device_id Your device ID
```

For example, if the device number here is 11a48dc6b976db80, then the command for the connection board is as follows.

```
python3 yolov8_obb.py --model_path ../model/yolov8n_obb.rknn --target rk3588 --device_id 11a48dc6b976db80
```

The reasoning is completed as shown in the following figure, and the result is saved in "result.jpg".

```
D NPUTransfer(8558): Transfer spec = local:transfer_proxy
D NPUTransfer(8558): Transfer interface successfully opened, fd = 3
I NPUTransfer(8558): TransferBuffer: min aligned size: 1024
D RKNNAPI: ==========================================
D RKNNAPI: RKNN VERSION:
D RKNNAPI:   API: 2.3.0 (e80ac5c build@2024-11-07T12:57:35)
D RKNNAPI:   DRV: rknn_server: 2.3.0 (e80ac5c build@2024-11-07T12:52:53)
D RKNNAPI:   DRV: rknnrt: 2.3.0 (c949ad889d@2024-11-07T11:35:33)
D RKNNAPI: ==========================================
D RKNNAPI: Input tensors:
D RKNNAPI:   index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, w_stride = 0, siz
e_with_stride = 0, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
D RKNNAPI: Output tensors:
D RKNNAPI:   index=0, name=/model.22/Concat_1_output_0, n_dims=4, dims=[1, 79, 80, 80], n_elems=505600, size=505600,
 w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=65, scale=0.209463
D RKNNAPI:   index=1, name=/model.22/Concat_2_output_0, n_dims=4, dims=[1, 79, 40, 40], n_elems=126400, size=126400,
 w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=68, scale=0.202605
D RKNNAPI:   index=2, name=/model.22/Concat_3_output_0, n_dims=4, dims=[1, 79, 20, 20], n_elems=31600, size=31600, w
_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=42, scale=0.142794
D RKNNAPI:   index=3, name=/model.22/Sigmoid_output_0, n_dims=3, dims=[1, 1, 8400], n_elems=8400, size=8400, w_strid
e = 0, size_with_stride = 0, fmt=UNDEFINED, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.002982
done
--> Running model
W inference: The 'data_format' is not set, and its default value is 'nhwc'!
save image in ./result.jpg
D NPUTransfer(8558): Transfer client closed, fd = 3
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/yolov8_obb/python$
```

Figure 4.2-4 Testing the rknn model of yolov8_obb

When using a python script to infer the model, the following issues may occur.

```
Traceback (most recent call last):
  File "/home/dominick/software/rknn_model_zoo-2.3.0/examples/yolov8_obb/python/yolov8_obb.py", line 11, in <module>
    from shapely.geometry import Polygon
ModuleNotFoundError: No module named 'shapely'
```

If the module 'shapely' cannot be found, execute the following command to install it. Please note that this should be done in the previously installed conda environment.

```
pip install shapely
```

The installation is completed as shown in the following picture.

```
(python3.12-tk2-2.3) dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo-2.3.0/examples/yolov8_obb/python$ cd ~/softw
are/rknn_model_zoo-2.3.0/examples/yolov8_obb/python
(python3.12-tk2-2.3) dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo-2.3.0/examples/yolov8_obb/python$ pip instal
l shapely
Collecting shapely
  Downloading shapely-2.0.6-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (7.0 kB)
Requirement already satisfied: numpy<3,>=1.14 in /home/dominick/anaconda3/envs/python3.12-tk2-2.3/lib/python3.12/site-packages (from
shapely) (1.26.4)
Downloading shapely-2.0.6-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 154.5 kB/s eta 0:00:00
Installing collected packages: shapely
Successfully installed shapely-2.0.6
```

Figure 4.2-5 Install the shapely module

After the previous tests confirm that the model has been successfully converted, first, the routines in the CPP directory need to be compiled. Then, the compiled folder should be pushed to the development board for inference on the board side. Enter the initial directory of modelzoo and open the terminal. First, enable the compiler and execute the following command to compile the routines.

```
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolov8_obb
```

```
dominick@ubuntu24:~/software/rknn_model_zoo$ export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d yolov8_obb
./build-linux.sh -t rk3588 -a aarch64 -d yolov8_obb
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=yolov8_obb
BUILD_DEMO_PATH=examples/yolov8_obb/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_yolov8_obb_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_yolov8_obb_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
```

Figure 4.2-6 Compile the yolov8_obb program.

After the compilation is completed, executable files and packaged models, etc. will be generated in the modelzoo directory under the install/rk3588_linux_aarch64/rknn_yolov8_obb_demo directory. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolov8_obb_demo/ /userdata/aidemo
```

```
dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_yolov8_obb_demo/ /userdata/aidemo
install/rk3588_linux_aarch64/rknn_yolov8_obb_demo/: 6 files pushed, 0 skipped. 1.0 MB/s (13103368 bytes in 12.004s)
```

Figure 4.2-7 Push the yolov8_obb program and model to the development board.

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_yolov8_obb_demo that was pushed to the development board, and execute the following commands in the development board terminal. Use test.jpg for inference. The execution result is as shown in the figure below.

```
./rknn_yolov8_obb_demo model/yolov8n_obb.rknn model/test.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yolov8_obb_demo# ./rknn_yolov8_obb_demo model/yolov8n_obb.rknn model/test.jpg
load lable ./model/yolov8_obb_labels_list.txt
model input num: 1, output num: 4
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128,
scale=0.003922
output tensors:
  index=0, name=/model.22/Concat_1_output_0, n_dims=4, dims=[1, 79, 80, 80], n_elems=505600, size=505600, fmt=NCHW, type=INT8, qnt_type
=AFFINE, zp=65, scale=0.209463
  index=1, name=/model.22/Concat_2_output_0, n_dims=4, dims=[1, 79, 40, 40], n_elems=126400, size=126400, fmt=NCHW, type=INT8, qnt_type
=AFFINE, zp=68, scale=0.202605
  index=2, name=/model.22/Concat_3_output_0, n_dims=4, dims=[1, 79, 20, 20], n_elems=31600, size=31600, fmt=NCHW, type=INT8, qnt_type=A
FFINE, zp=42, scale=0.142794
  index=3, name=/model.22/Sigmoid_output_0, n_dims=3, dims=[1, 1, 8400, 0], n_elems=8400, size=8400, fmt=UNDEFINED, type=INT8, qnt_type
=AFFINE, zp=-128, scale=0.002982
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
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]
rknn_run
rknn_run time=23.02ms, FPS = 43.43
post_process time=271.07ms, FPS = 3.69
ship @ (289 539 20 68 angle=0.507482) 0.835
ship @ (152 474 20 75 angle=0.507482) 0.835
ship @ (602 91 23 78 angle=0.460638) 0.835
ship @ (362 10 54 19 angle=0.488744) 0.835
ship @ (274 447 18 54 angle=0.470007) 0.835
ship @ (349 83 17 54 angle=0.507482) 0.835
ship @ (252 522 22 62 angle=0.470007) 0.835
ship @ (46 79 20 57 angle=0.423164) 0.835
ship @ (258 36 18 55 angle=0.470007) 0.835
ship @ (80 437 19 65 angle=0.498113) 0.835
```

Figure 4.2-8 The program reasoning is executed at the board end.

From the result shown in the above figure, it can be seen that the corresponding detection target, the corresponding position information, rotation angle and the corresponding confidence level will be printed out. The final output result image is saved as out.png and can be transmitted to the PC end for viewing.

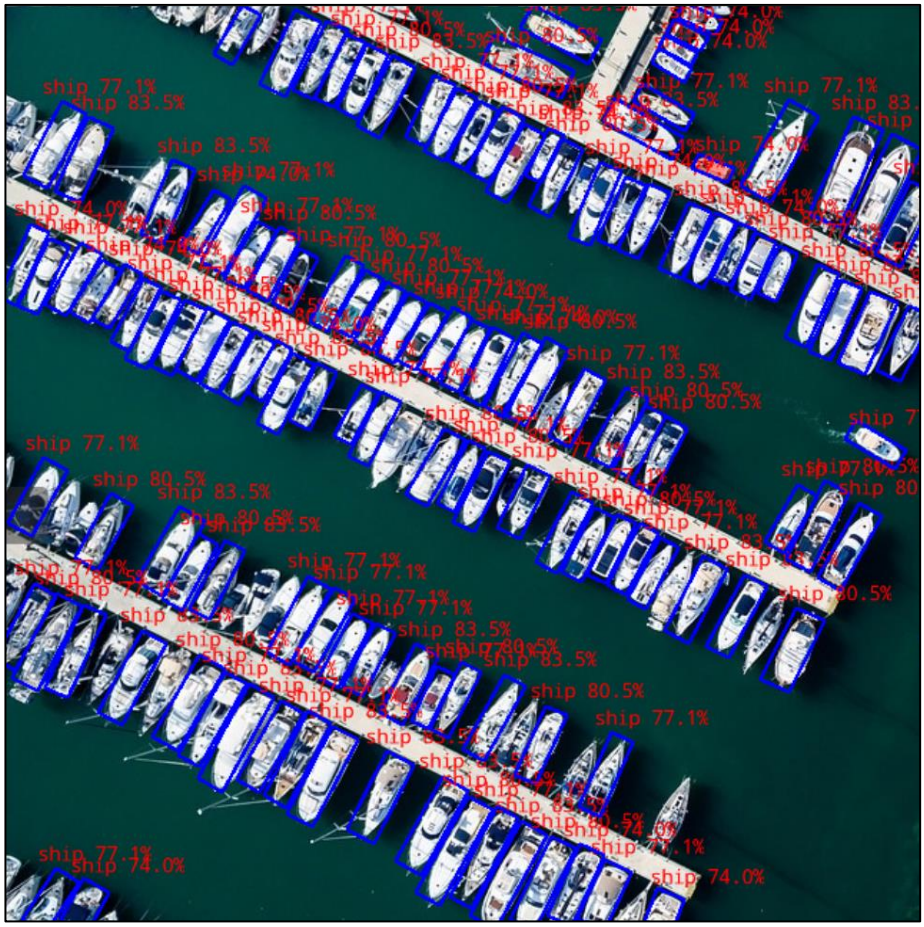


Figure 4.2-9 Output image out.png

4.3 Testing the yolov10 model

First, the model needs to be downloaded. This can be done by executing the model download script. Before execution, it is necessary to check if the network of Ubuntu is normal. In the rknn_model_zoo directory after decompressing Ubuntu, execute the following command to download the model. This will download the yolov10s and yolov10n models of two scales.

```
cd examples/yolov10/model
sh download_model.sh
```

Figure 4.3-1 Download the yolov10 model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the yolov10 sample program directory under the modelzoo.

```
cd ~/software/rknn_model_zoo/examples/yolov10/python
```

If you are using the rk3588 development board, execute the following command to convert the yolov10n model.

```
python3 convert.py ../model/yolov10n.onnx rk3588 i8
```

The last parameter i8 refers to the quantization precision. It can also be changed to fp as per requirements (note: if it is changed to another quantization precision, the quantization precision of the subsequent inference must be consistent to obtain the correct result).

Note: i8 refers to the int8 type.

fp refers to the float type. It generally does not have a quantization precision and the inference speed will be slower.

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/yolov10/python$ python3 convert.py ../model/yolov10n.onnx rk3
588 i8
I rknn-toolkit2 version: 2.3.0
--> Config model
done
--> Loading model
I Loading : 100%|| 164/164 [00:00<00:00, 38361.82it/s]
done
--> Building model
I OpFusing 0: 100%|| 100/100 [00:00<00:00, 214.26it/s]
I OpFusing 1 : 100%|| 100/100 [00:00<00:00, 137.73it/s]
I OpFusing 0 : 100%|| 100/100 [00:01<00:00, 88.59it/s]
I OpFusing 1 : 100%|| 100/100 [00:01<00:00, 86.49it/s]
I OpFusing 2 : 100%|| 100/100 [00:01<00:00, 62.10it/s]
W build: found outlier value, this may affect quantization accuracy
                        const name                                abs_mean    abs_std     outlier value
                        model.23.one2one_cv3.1.1.1.conv.weight    0.50        0.61        -11.375
                        model.23.one2one_cv3.0.1.1.conv.weight    0.61        0.62        -11.948
I GraphPreparing : 100%|| 197/197 [00:00<00:00, 11231.18it/s]
I Quantizating : 100%|| 197/197 [00:10<00:00, 19.51it/s]
W build: The default input dtype of 'images' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '487' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '501' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '508' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
```

Figure 4.3-2 Convert the yolov10n model to a rknn model

Perform model testing to see if it can perform normal inference. Pay attention to specifying the device ID of the rk3588 development board to the corresponding device through the "device_id" parameter. If there are multiple rk3588 devices, you can also specify the rk3588 devices for inference through the "--devices_id" parameter. After connecting the development board, execute "adb devices" under Ubuntu to obtain the device ID.

```
adb devices
```

```
(python3.12-tk2-2.3) dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo/examples/yolov8_obb/python$ adb devices
List of devices attached
11a48dc6b976db80        device
```

Figure 4.3-3 Obtain the device ID

```
python3 yolov10.py --model_path ../model/yolov10.rknn --target rk3588 --device_id Your device ID
```

For example, if the device number here is 11a48dc6b976db80, then the command for the connection board is as follows.

```
python3 yolov10.py --model_path ../model/yolov10.rknn --target rk3588 --device_id 11a48dc6b976db80
```

The result of the reasoning process is saved in result.jpg. Based on the previous tests, once it is confirmed that the model has been successfully converted, the routines in the CPP directory need to be compiled first, and then the compiled folder should be pushed to the development board for deployment and inference on the board. Enter the initial directory of modelzoo and open the terminal. First, enable the compiler, and then execute the following command to compile the routines.

```
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolov10
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d yolov10
./build-linux.sh -t rk3588 -a aarch64 -d yolov10
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=yolov10
BUILD_DEMO_PATH=examples/yolov10/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_yolov10_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_yolov10_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- !!!!!!!!!!CMAKE_SYSTEM_NAME: Linux
-- Configuring done (0.0s)
-- Generating done (0.0s)
-- Build files have been written to: /home/dominick/software/rknn_model_zoo/build/build_rknn_yolov10_demo_rk3588_linux_aarch64_Release
[ 33%] Built target imageutils
[ 33%] Built target fileutils
[ 66%] Built target imagedrawing
[ 66%] Built target audioutils
[100%] Built target rknn_yolov10_demo
[ 16%] Built target imageutils
[ 33%] Built target fileutils
[ 50%] Built target imagedrawing
```

Figure 4.3-4 Compile the yolov10 program

After the compilation is completed, executable files and packaged models, etc. will be generated in the install/rk3588_linux_aarch64/rknn_yolov10_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolov10_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_yolov10_demo/ /userdata/ai
demo
install/rk3588_linux_aarch64/rknn_yolov10_demo/: 6 files pushed, 0 skipped. 1.0 MB/s (12410802 bytes in 11.502s)
```

Figure 4.3-5 Push the yolov10 program and model to the development board.

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_yolov10_demo that was pushed to the development board, and execute the following commands in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the figure below.

```
./rknn_yolov10_demo model/yolov10.rknn model/bus.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yolov10_demo# ./rknn_yolov10_demo model/yolov10.rknn model/bus.jpg
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 6
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128,
scale=0.003922
output tensors:
  index=0, name=487, n_dims=4, dims=[1, 64, 80, 80], n_elems=409600, size=409600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-38, scale=0
.114574
  index=1, name=501, n_dims=4, dims=[1, 80, 80, 80], n_elems=512000, size=512000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=
0.002001
  index=2, name=508, n_dims=4, dims=[1, 64, 40, 40], n_elems=102400, size=102400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-57, scale=0
.095044
  index=3, name=522, n_dims=4, dims=[1, 80, 40, 40], n_elems=128000, size=128000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=
0.003505
  index=4, name=529, n_dims=4, dims=[1, 64, 20, 20], n_elems=25600, size=25600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-58, scale=0.0
61253
  index=5, name=543, n_dims=4, dims=[1, 80, 20, 20], n_elems=32000, size=32000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.
003792
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
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]
rknn_run
bus @ (88 137 556 437) 0.929
person @ (109 234 226 536) 0.895
person @ (210 241 284 512) 0.891
person @ (477 233 560 519) 0.796
person @ (79 331 114 518) 0.344
write_image path: out.png width=640 height=640 channel=3 data=0x7fb4e47010
```

Figure 4.3-6 Execute program reasoning at the board end

From the result shown in the above figure, it can be seen that the corresponding detection target, the corresponding position information, and the corresponding confidence level will be printed out. The final output result image is saved as out.png and can be transmitted to the PC end for viewing.

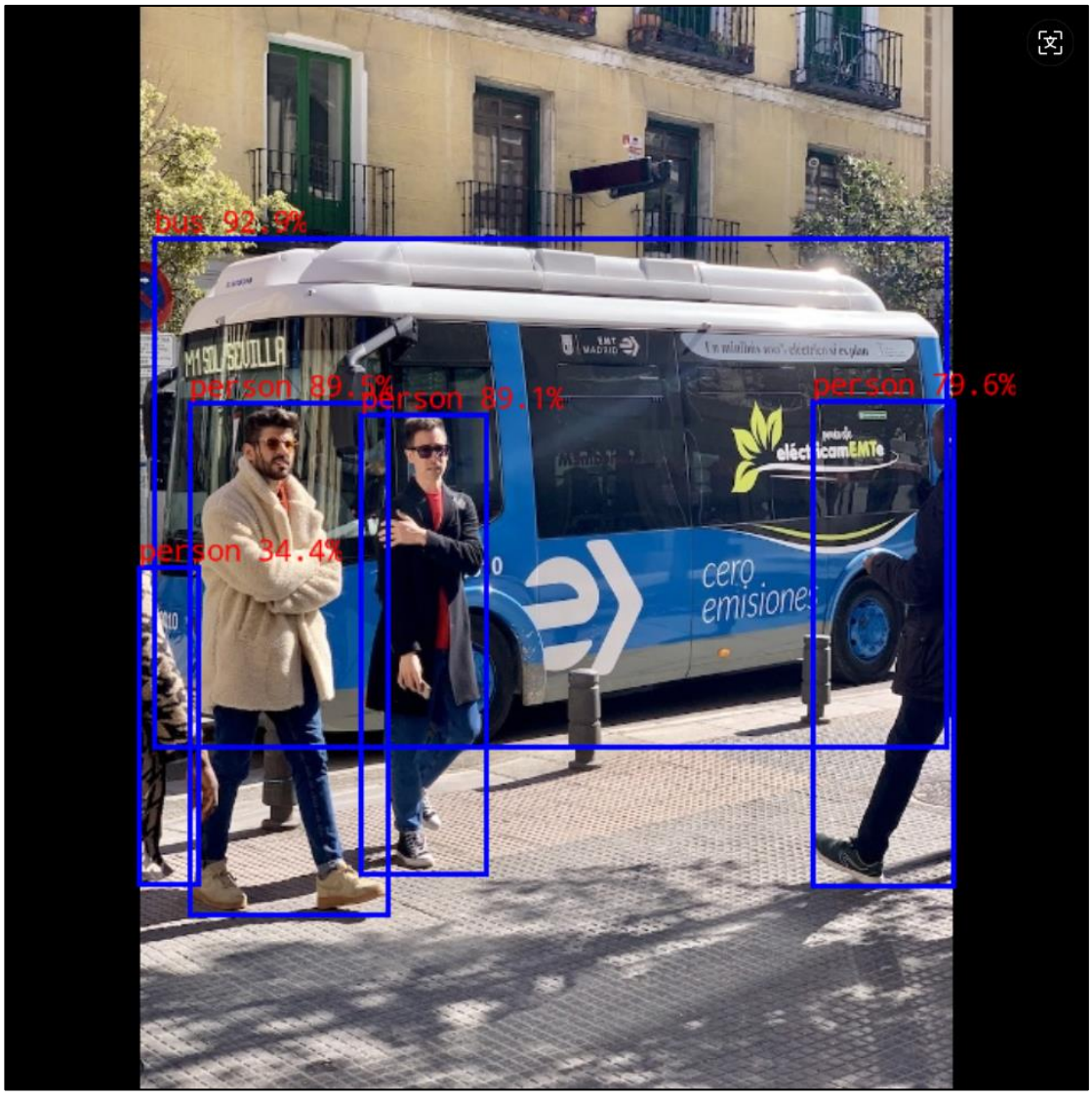


Figure 4.3-7 Output image named "out.png"

4.4 Test the yolo11 model

First, you need to download the model. This can be done by executing the model download script. Before execution, it is necessary to check if the Ubuntu network is functioning properly. Then, execute the following command in the rknn_model_zoo directory after decompressing Ubuntu to download the ONNX format model of yolov11.

```
cd examples/yolo11/model
sh download_model.sh
```

```
dominick@ubuntu24:~/software/rknn_model_zoo/examples/yolo11/model$ sh download_model.sh
--2024-12-26 17:12:26--  https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/yolo11/yolo11n.onnx
正在解析主机 ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
正在连接 ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度： 10527859 (10M) [application/octet-stream]
正在保存至：‘./yolo11n.onnx’

./yolo11n.onnx      100%[===================>]  10.04M  10.0MB/s    用时 1.0s

2024-12-26 17:12:28 (10.0 MB/s) - 已保存 ‘./yolo11n.onnx’ [10527859/10527859])
```

Figure 4.4-1 Download the yolo11 model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the YOLO 11 example program directory under the modelzoo.

```
cd ~/software/rknn_model_zoo/examples/yolo11/python
```

If you are using the rk3588 development board, execute the following command to convert the yolo11 model.

```
python3 convert.py ../model/yolo11n.onnx rk3588 i8
```

Then, execute the conversion model Python script again for the conversion. The last parameter, i8, refers to the quantization precision. It can also be changed to fp (note: if it is changed to another quantization precision, the quantization precision for the subsequent inference must be modified to be consistent to obtain the correct result for inference).

Note: i8 refers to the int8 type.

fp refers to the float type. Generally, no quantization precision is applied, and the inference speed will be slower.

```
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'onnx::ReduceSum_501' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '505' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '512' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'onnx::ReduceSum_526' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '530' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
I rknn building ...
I rknn building done.
done
--> Export rknn model
done
```

Figure 4.4-2 Convert the yolo11n model to a rknn model.

Perform model testing to see if it can perform normal inference. Pay attention to specifying the device ID of the RK3588 development board to the corresponding device. If there are multiple RK3588 devices, you can also specify the RK3588 devices through the --devices_id parameter for inference. After connecting the development board, execute "adb devices" under Ubuntu to obtain the device ID.

```
adb devices
```

```
(python3.12-tk2-2.3) dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo/examples/yolov8_obb/python$ adb devices
List of devices attached
11a48dc6b976db80        device
```

Figure 4.4-3 Obtain the device ID

```
python3 yolo11.py --model_path ../model/yolo11.rknn --target rk3588 --device_id Your device id
```

For example, if the device number here is 11a48dc6b976db80, then the command for the connection board is as follows.

```
python3 yolo11.py --model_path ../model/yolo11.rknn --target rk3588 --device_id 11a48dc6b976db80
```

The result of the reasoning process is saved in result.jpg. Based on the previous tests, once it is confirmed that the model has been successfully converted, the routines in the CPP directory need to be compiled first, and then the compiled folder should be pushed to the development board for deployment and inference on the board. Enter the initial directory of modelzoo and open the terminal. First, enable the compiler, and then execute the following command to compile the routines.

```
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolo11
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a
aarch64 -d yolo11
./build-linux.sh -t rk3588 -a aarch64 -d yolo11
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=yolo11
BUILD_DEMO_PATH=examples/yolo11/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_yolo11_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_yolo11_demo_rk3588_linux_aarch
64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
```

Figure 4.4-4 Compile the yolo11 program

After the compilation is completed, executable files and packaged models and other files will be generated in the modelzoo directory under the install/rk3588_linux_aarch64/rknn_yolo11_demo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolo11_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_
aarch64/rknn_yolo11_demo/ /userdata/aidemo
install/rk3588_linux_aarch64/rknn_yolo11_demo/... skipped. 1.0 MB/s (13866286 bytes in 12.868s)
```

Figure 4.4-5 Push the yolo11 program and model to the development board.

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_yolo11_demo on the development board, and execute the following commands in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the figure below.

```
./rknn_yolo11_demo model/yolo11.rknn model/bus.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yolo11_demo# ./rknn_yolo11_demo model/yolo11.rknn model/bus.jpg
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 9
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128,
scale=0.003922
output tensors:
  index=0, name=462, n_dims=4, dims=[1, 64, 80, 80], n_elems=409600, size=409600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-44, scale=0
.135466
  index=1, name=onnx::ReduceSum_476, n_dims=4, dims=[1, 80, 80, 80], n_elems=512000, size=512000, fmt=NCHW, type=INT8, qnt_type=AFFINE,
 zp=-128, scale=0.003027
  index=2, name=480, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400, size=6400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003
036
  index=3, name=487, n_dims=4, dims=[1, 64, 40, 40], n_elems=102400, size=102400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-38, scale=0
.093536
  index=4, name=onnx::ReduceSum_501, n_dims=4, dims=[1, 80, 40, 40], n_elems=128000, size=128000, fmt=NCHW, type=INT8, qnt_type=AFFINE,
 zp=-128, scale=0.003476
  index=5, name=505, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600, size=1600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003
922
  index=6, name=512, n_dims=4, dims=[1, 64, 20, 20], n_elems=25600, size=25600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-42, scale=0.0
84287
  index=7, name=onnx::ReduceSum_526, n_dims=4, dims=[1, 80, 20, 20], n_elems=32000, size=32000, fmt=NCHW, type=INT8, qnt_type=AFFINE, z
p=-128, scale=0.003903
  index=8, name=530, n_dims=4, dims=[1, 1, 20, 20], n_elems=400, size=400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.00392
2
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
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]
rknn_run
bus @ (95 136 553 438) 0.948
person @ (108 236 223 536) 0.898
person @ (212 240 284 509) 0.839
person @ (477 229 559 521) 0.835
person @ (79 359 117 515) 0.473
write image path: out.png width=640 height=640 channel=3 data=0x55a99496f0
```

Figure 4.4-6 Execute program reasoning at the board end

From the result shown in the above figure, it can be seen that the corresponding detection target, the corresponding position information, and the corresponding confidence level will be printed out. The final output result image is saved as out.png and can be transmitted to the PC end for viewing.

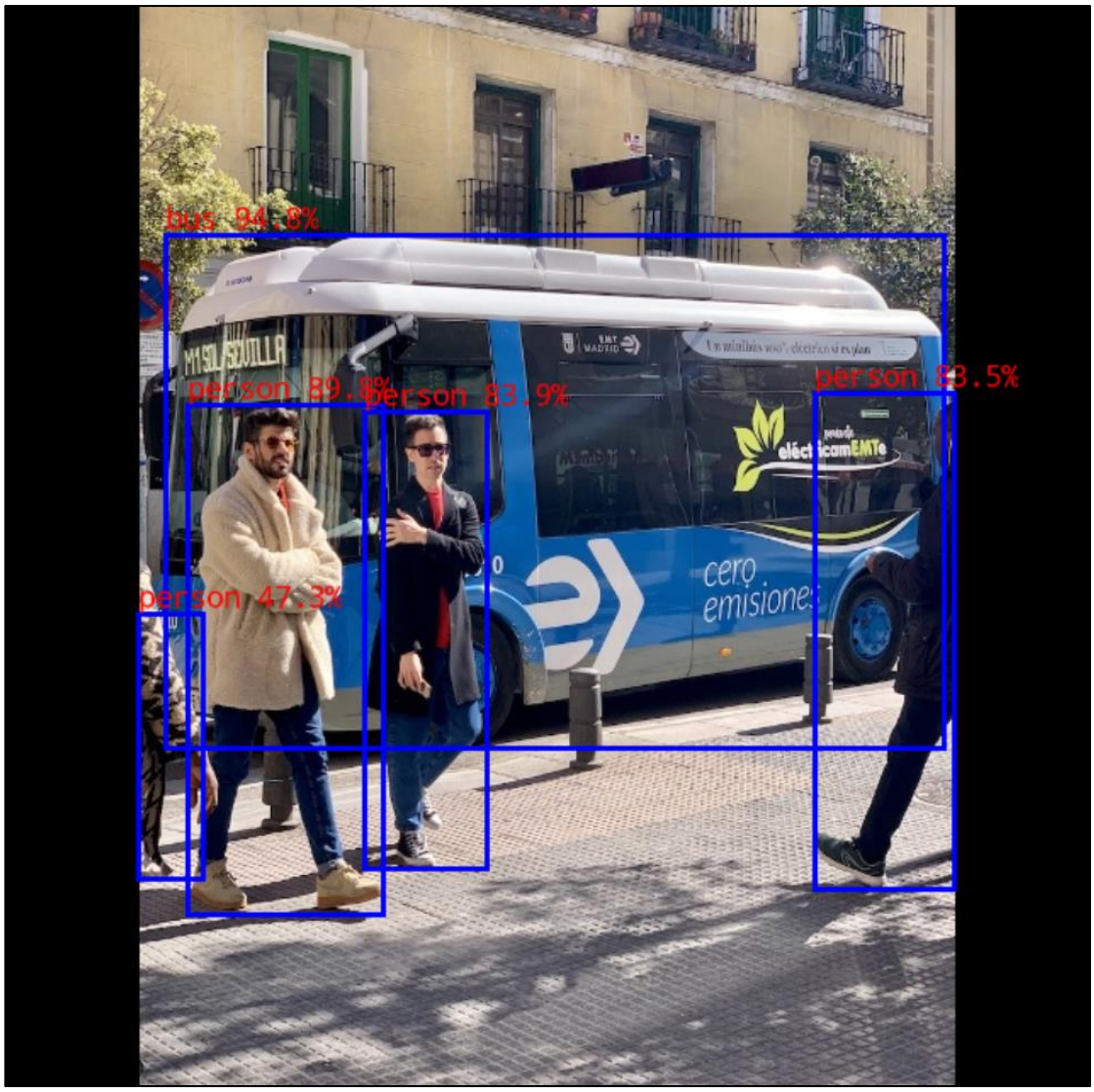


Figure 4.4-7 Output image out.png

4.5 Test the yolo_world model

First, you need to download the model. This can be done by executing the model download script. Before execution, it is necessary to check if the Ubuntu network is normal. In the rknn_model_zoo directory after decompressing Ubuntu, execute the following command to download the onnx format model of yolo_world.

```
cd examples/yolo_world/model
sh download_model.sh
```

```
dominick@ubuntu24:~/software/rknn_model_zoo/examples/yolo_world/model$ sh download_model.sh
--2024-12-26 17:36:28--  https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/
clip/clip_text.onnx
正在解析主机 ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
正在连接 ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度： 254185587 (242M) [application/octet-stream]
正在保存至：‘./clip_text.onnx’

./clip_text.onnx    100%[===================>] 242.41M  4.12MB/s    用时 30s

2024-12-26 17:36:59 (7.96 MB/s) - 已保存 ‘./clip_text.onnx’ [254185587/254185587])

--2024-12-26 17:36:59--  https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/
yolo_world/yolo_world_v2s.onnx
正在解析主机 ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
正在连接 ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度： 51068366 (49M) [application/octet-stream]
正在保存至：‘./yolo_world_v2s.onnx’

./yolo_world_v2s.on 100%[===================>]  48.70M  10.2MB/s    用时 4.9s

2024-12-26 17:37:05 (9.98 MB/s) - 已保存 ‘./yolo_world_v2s.onnx’ [51068366/51068366])
```

Figure 4.5-1 Download the yolo_world model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the directory of the YOLO_world routine under the modelzoo. If you are using the RK3588 development board, execute the following command to convert the YOLO_world model and the clip_text model.

```
cd ~/software/rknn_model_zoo/examples/yolo_world/python/yolo_world/
python3 convert.py ../../model/yolo_world_v2s.onnx rk3588 i8
cd ~/software/rknn_model_zoo/examples/yolo_world/python/clip_text/
python3 convert.py ../../model/clip_text.onnx rk3588
```

Then, execute the conversion model Python script again for the conversion. The last parameter, i8, refers to the quantization precision. It can also be changed to fp (note: if this is changed to another quantization precision, the quantization precision for the subsequent inference must be modified accordingly to obtain the correct result).

Note: i8 refers to the int8 type.

fp refers to the float type. Generally, no quantization precision is applied, and the inference speed will be slower.

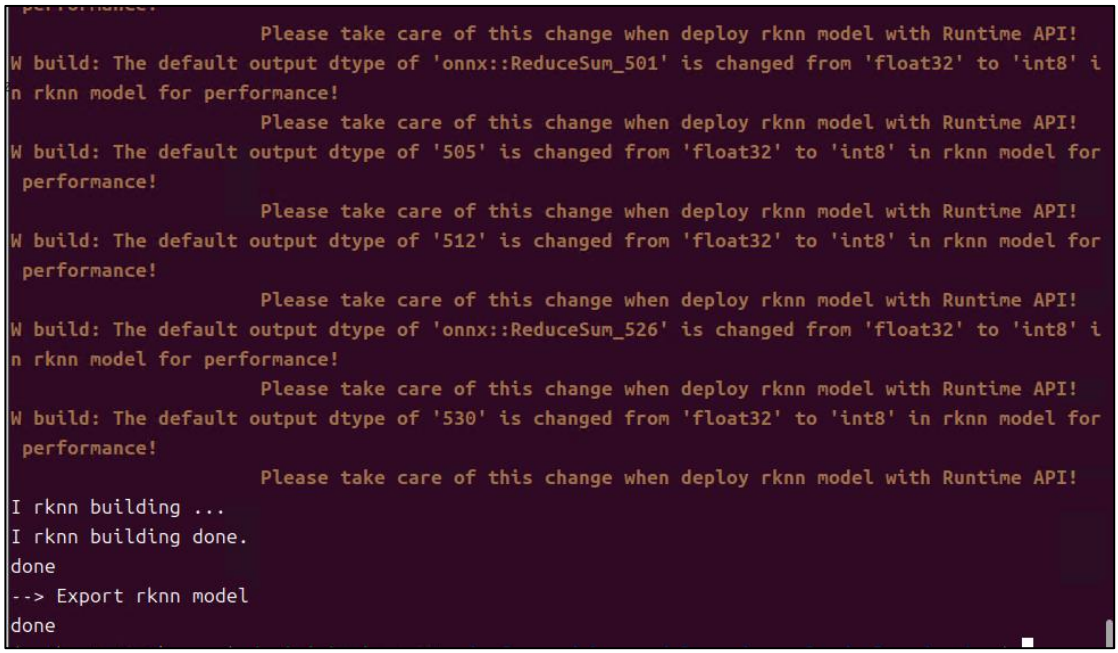


Figure 4.5-2 Convert the yolo11n model to a rknn model.

Perform model testing to see if it can perform normal inference. Pay attention to specifying the device ID of the RK3588 development board to the corresponding device through the "device_id" parameter. If there are multiple RK3588 devices, you can also specify the RK3588 devices for inference through the "--devices_id" parameter. After connecting the development board, execute "adb devices" under Ubuntu to obtain the device ID.

```
adb devices
```

Figure 4.5-3 Obtain the device ID

```
python3 yolo11.py --model_path ../model/yolo11.rknn --target rk3588 --device_id Your device ID
```

For example, if the device number here is 11a48dc6b976db80, then the command for the connection board is as follows.

```
python3 yolo11.py --model_path ../model/yolo11.rknn --target rk3588 --device_id 11a48dc6b976db80
```

The result of the reasoning process is saved in result.jpg. Based on the previous tests, once it is confirmed that the model has been successfully converted, the routines in the CPP directory need to be compiled first, and then the compiled folder should be pushed to the development board for deployment and inference on the board. Enter the initial directory of modelzoo and open the terminal. First, enable the compiler, and then execute the following command to compile the routines.

```
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolo11
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a
aarch64 -d yolo11
./build-linux.sh -t rk3588 -a aarch64 -d yolo11
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=yolo11
BUILD_DEMO_PATH=examples/yolo11/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_yolo11_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_yolo11_demo_rk3588_linux_aarch
64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
```

Figure 4.5-4 Compile the yolo11 program

After the compilation is completed, executable files and packaged models and other files will be generated in the modelzoo directory under the install/rk3588_linux_aarch64/rknn_yolo11_demo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolo11_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_
aarch64/rknn_yolo11_demo/ /userdata/aidemo
install/rk3588_linux_aarch64/rknn_yolo11_demo/... skipped. 1.0 MB/s (13866286 bytes in 12.868s)
```

Figure 4.5-5 Push the yolo11 program and model to the development board.

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_yolo11_demo on the development board, and execute the following commands in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the figure below.

```
./rknn_yolo11_demo model/yolo11.rknn model/bus.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yolo11_demo# ./rknn_yolo11_demo model/yolo11.rknn model/bus.jpg
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 9
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128,
scale=0.003922
output tensors:
  index=0, name=462, n_dims=4, dims=[1, 64, 80, 80], n_elems=409600, size=409600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-44, scale=0
.135466
  index=1, name=onnx::ReduceSum_476, n_dims=4, dims=[1, 80, 80, 80], n_elems=512000, size=512000, fmt=NCHW, type=INT8, qnt_type=AFFINE,
 zp=-128, scale=0.003027
  index=2, name=480, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400, size=6400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003
036
  index=3, name=487, n_dims=4, dims=[1, 64, 40, 40], n_elems=102400, size=102400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-38, scale=0
.093536
  index=4, name=onnx::ReduceSum_501, n_dims=4, dims=[1, 80, 40, 40], n_elems=128000, size=128000, fmt=NCHW, type=INT8, qnt_type=AFFINE,
 zp=-128, scale=0.003476
  index=5, name=505, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600, size=1600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003
922
  index=6, name=512, n_dims=4, dims=[1, 64, 20, 20], n_elems=25600, size=25600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-42, scale=0.0
84287
  index=7, name=onnx::ReduceSum_526, n_dims=4, dims=[1, 80, 20, 20], n_elems=32000, size=32000, fmt=NCHW, type=INT8, qnt_type=AFFINE, z
p=-128, scale=0.003903
  index=8, name=530, n_dims=4, dims=[1, 1, 20, 20], n_elems=400, size=400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.00392
2
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
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]
rknn_run
bus @ (95 136 553 438) 0.948
person @ (108 236 223 536) 0.898
person @ (212 240 284 509) 0.839
person @ (477 229 559 521) 0.835
person @ (79 359 117 515) 0.473
write_image path: out.png width=640 height=640 channel=3 data=0x55a99496f0
```

Figure 4.5-6 Execute program reasoning at the board end

From the result shown in the above figure, it can be seen that the corresponding detection target, the corresponding position information, and the corresponding confidence level will be printed out. The final output result image is saved as out.png and can be transmitted to the PC end for viewing.

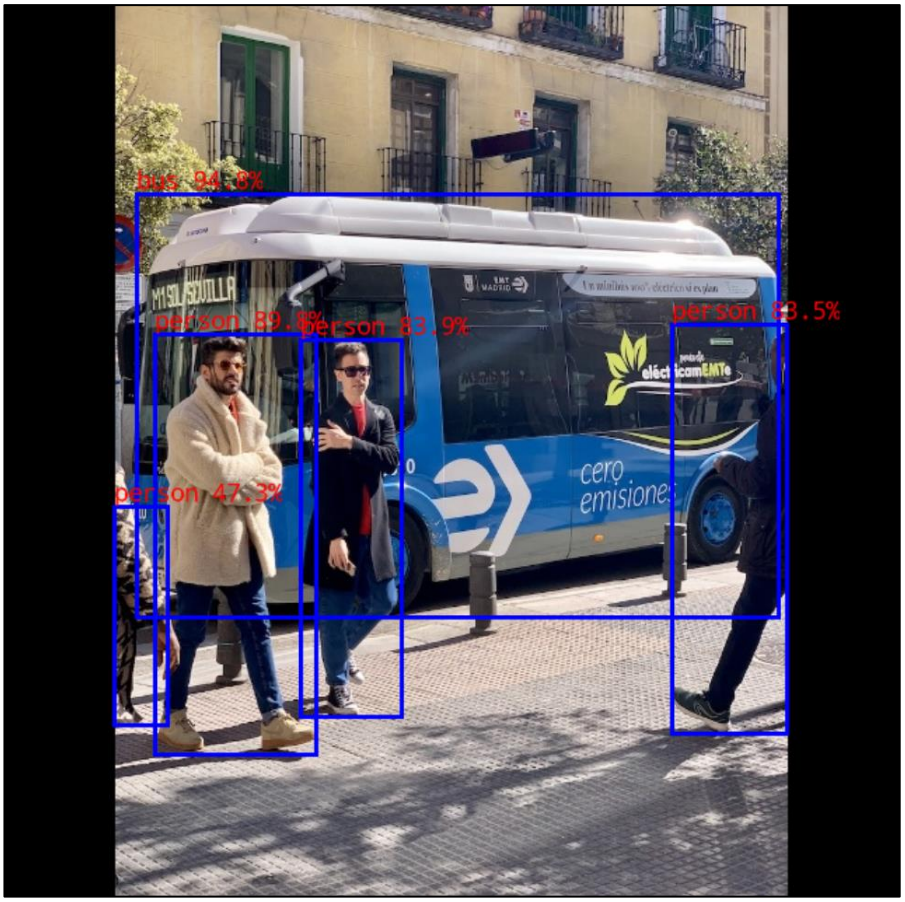


Figure 4.5-7 Output image out.png

4.6 Test the yolov8_pose model

First, the model needs to be downloaded. This can be done by executing the model download script. Before execution, it is necessary to check if the Ubuntu network is normal. In the rknn_model_zoo directory after the Ubuntu decompression, execute the following command to download the onnx format model of yolov8_pose.

```
cd examples/yolov8_pose/model
sh download_model.sh
```

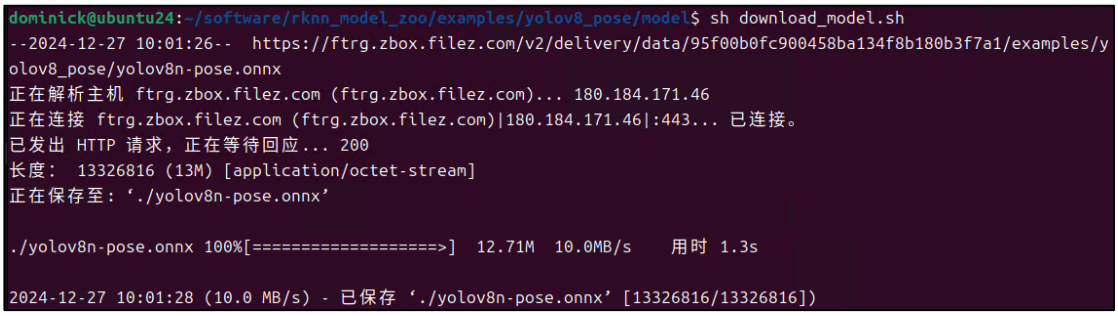


Figure 4.6-1 Download the yolov8_pose model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the yolov8_pose routine directory under the modelzoo.

```
cd ~/software/rknn_model_zoo/examples/yolov8_pose/python
```

If you are using the rk3588 development board, execute the following command to convert the yolov8n_pose model.

```
python3 convert.py ../model/yolov8n-pose.onnx rk3588 i8
```

The last parameter i8 refers to the quantization precision. It can also be changed to fp as per requirements (note: if it is changed to another quantization precision, the quantization precision of the subsequent inference must be consistent to obtain the correct result).

Note: i8 refers to the int8 type.

fp refers to the float type. It generally does not have a quantization precision and the inference speed will be slower.

```
W hybrid_quantization_step2: The default input dtype of 'images' is changed from 'float32' to 'int8' in rknn mode
l for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W hybrid_quantization_step2: The default output dtype of '/model.22/Concat_1_output_0' is changed from 'float32'
to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W hybrid_quantization_step2: The default output dtype of '/model.22/Concat_2_output_0' is changed from 'float32'
to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W hybrid_quantization_step2: The default output dtype of '/model.22/Concat_3_output_0' is changed from 'float32'
to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W hybrid_quantization_step2: The default output dtype of '/model.22/Concat_6_output_0' is changed from 'float32'
to 'float16' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
I rknn building ...
I rknn building done.
done
--> Export rknn model
output_path: ../model/yolov8_pose.rknn
done
```

Figure 4.6-2 Convert the yolo8n_pose model to a rknn model

Perform model testing to see if it can perform normal inference. Pay attention to specifying the device ID of the rk3588 development board to the corresponding device through the "device_id" parameter. If there are multiple rk3588 devices, you can also specify the rk3588 devices for inference through the "--devices_id" parameter. After connecting the development board, execute "adb devices" under Ubuntu to obtain the device ID.

```
adb devices
```

```
(python3.12-tk2-2.3) dominick@dominick-VMware-Virtual-Platform:~/software/rknn_model_zoo/examples/yolov8_obb/python$ adb devices
List of devices attached
11a48dc6b976db80        device
```

Figure 4.6-3 Obtain the device ID

```
python3 yolov8_pose.py --model_path ../model/yolov8_pose.rknn --target rk3588 --device_id Your device ID
```

For example, if the device number here is 11a48dc6b976db80, then the command for the connection board is as follows.

```
python3 yolov8_pose.py --model_path ../model/yolov8_pose.rknn --target rk3588 --device_id 11a48dc6b976db80
```

The result of the reasoning process is saved in result.jpg. Based on the previous tests, once it is confirmed that the model has been successfully converted, the routines in the CPP directory need to be compiled first, and then the compiled folder should be pushed to the development board for deployment and inference on the board. Enter the initial directory of modelzoo and open the terminal. First, enable the compiler, and then execute the following command to compile the routines.

```
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolov8_pose
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d yolov8_pose
./build-linux.sh -t rk3588 -a aarch64 -d yolov8_pose
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=yolov8_pose
BUILD_DEMO_PATH=examples/yolov8_pose/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_yolov8_pose_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_yolov8_pose_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
```

Figure 4.6-4 Compile the yolov8_pose program.

After the compilation is completed, executable files and packaged models and other files will be generated in the modelzoo directory under the install/rk3588_linux_aarch64/rknn_yolov8_pose_demo directory. Execute the following command to copy the model and executable files and other contents to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolov8_pose_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_yolov8_pose_demo/ /u
serdata/aidemo
install/rk3588_linux_aarch64/rknn_yolov8_pose_demo/: 6 files pushed, 0 skipped. 1.0 MB/s (13853492 bytes in 12.985s)
```

Figure 4.6-5 Push the yolov8_pose program and model to the development board.

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_yolov8_pose_demo that was pushed to the development board, and execute the following commands in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_yolov8_pose_demo
./rknn_yolov8_pose_demo model/yolov8_pose.rknn model/bus.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yolov8_pose_demo# ./rknn_yolov8_pose_demo model/yolov8_pose.rknn model/bus.jpg
load lable ./model/yolov8_pose_labels_list.txt
model input num: 1, output num: 4
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128,
scale=0.003922
output tensors:
  index=0, name=/model.22/Concat_1_output_0, n_dims=4, dims=[1, 65, 80, 80], n_elems=416000, size=416000, fmt=NCHW, type=INT8, qnt_type
=AFFINE, zp=31, scale=0.141366
  index=1, name=/model.22/Concat_2_output_0, n_dims=4, dims=[1, 65, 40, 40], n_elems=104000, size=104000, fmt=NCHW, type=INT8, qnt_type
=AFFINE, zp=67, scale=0.235531
  index=2, name=/model.22/Concat_3_output_0, n_dims=4, dims=[1, 65, 20, 20], n_elems=26000, size=26000, fmt=NCHW, type=INT8, qnt_type=A
FFINE, zp=50, scale=0.159995
  index=3, name=/model.22/Concat_6_output_0, n_dims=4, dims=[1, 17, 3, 8400], n_elems=428400, size=856800, fmt=NCHW, type=FP16, qnt_typ
e=AFFINE, zp=0, scale=1.000000
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
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]
rknn_run
rknn_run time=36.88ms, FPS = 27.12
post_process time=0.23ms, FPS = 4310.34
person @ (108 234 223 536) 0.889
person @ (211 241 284 508) 0.872
person @ (477 235 560 518) 0.853
write_image path: out.png width=640 height=640 channel=3 data=0x7fb7a2c010
```

Figure 4.6-6 Execute program reasoning at the board end

From the result shown in the above figure, it can be seen that the corresponding detection target, the corresponding position information, and the corresponding confidence level will be printed out. The final output result image is saved as out.png and can be transmitted to the PC end for viewing.

Figure 4.6-7 Output image out.png

4.7 Test the mobilesam model

First, you need to download the model. This can be done by executing the model download script. Before execution, it is necessary to check if the Ubuntu network is normal. In the rknn_model_zoo directory after decompressing Ubuntu, execute the following command to download the onnx format model of mobilesam. This will download the encoder and decoder models.

```
cd examples/mobilesam/model
sh download_model.sh
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/mobilesam/model$ sh download_model.sh
--2024-12-27 10:51:42--  https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/mobilesam/mobilesam_encoder_tiny.onnx
正在解析主机 ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
正在连接 ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度： 27931848 (27M) [application/octet-stream]
正在保存至：‘./mobilesam_encoder.onnx’

./mobilesam_encoder.onnx         100%[=====================================================>]  26.64M  7.65MB/s    用时 3.5s

2024-12-27 10:51:46 (7.65 MB/s) - 已保存 ‘./mobilesam_encoder.onnx’ [27931848/27931848])

--2024-12-27 10:51:46--  https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/mobilesam/mobilesam_decoder.onnx
正在解析主机 ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
正在连接 ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度： 16483151 (16M) [application/octet-stream]
正在保存至：‘./mobilesam_decoder.onnx’

./mobilesam_decoder.onnx         100%[=====================================================>]  15.72M  10.1MB/s    用时 1.5s

2024-12-27 10:51:48 (10.1 MB/s) - 已保存 ‘./mobilesam_decoder.onnx’ [16483151/16483151])
```

Figure 4.7-1 Download the mobilesam model

Open the Ubuntu terminal and execute the following command to enter the conda environment we created earlier.

```
conda activate python3.12-tk2-2.3
```

Enter the mobilesam routine directory under the modelzoo, and execute the following command to convert the mobilesam model.

```
cd ~/software/rknn_model_zoo/examples/mobilesam/python/decoder
python3 convert.py ../../model/mobilesam_decoder.onnx rk3588
cd ~/software/rknn_model_zoo/examples/mobilesam/python/encoder
python3 convert.py ../../model/mobilesam_decoder.onnx rk3588
```

```
D RKNN: [11:08:46.501] 161 Conv              mask_decoder.output_hypernetworks_mlps.2.layers.2.bias                  FLOAT     (32)
        | 0x000fea80 0x000feb00 0x00000080
D RKNN: [11:08:46.501] 162 ConvRelu          mask_decoder.output_hypernetworks_mlps.3.layers.1.weight                FLOAT16   (256,256,1,1)
        | 0x0011ef00 0x0013ef00 0x00020000
D RKNN: [11:08:46.501] 162 ConvRelu          mask_decoder.output_hypernetworks_mlps.3.layers.1.bias                  FLOAT     (256)
        | 0x0013ef00 0x0013f300 0x00000400
D RKNN: [11:08:46.501] 163 Conv              mask_decoder.output_hypernetworks_mlps.3.layers.2.weight                FLOAT16   (32,256,1,1)
        | 0x0013f300 0x00143300 0x00004000
D RKNN: [11:08:46.501] 163 Conv              mask_decoder.output_hypernetworks_mlps.3.layers.2.bias                  FLOAT     (32)
        | 0x00143300 0x00143380 0x00000080
D RKNN: [11:08:46.501] 166 Reshape           iou_predictions_mm_tp_rs_i1                                             INT64     (2)
        | 0x0084b180 0x0084b190 0x00000010
D RKNN: [11:08:46.501] 169 Reshape           /Concat_12_output_0                                                     INT64     (4)
        | 0x0084b000 0x0084b020 0x00000020
D RKNN: [11:08:46.501] ------------------------------------------------------------------------------------------------------------------------
--------+--------------------------------
D RKNN: [11:08:46.504] ----------------------------------------
D RKNN: [11:08:46.504] Total Internal Memory Size: 4716KB
D RKNN: [11:08:46.504] Total Weight Memory Size: 9279.31KB
D RKNN: [11:08:46.504] ----------------------------------------
D RKNN: [11:08:46.504] <<<<<<<< end: rknn::RKNNMemStatisticsPass
I rknn building done.
done
--> Export rknn model
done
```

Figure 4.7-2 Convert the mobilesam_decoder model to a rknn model

```
D RKNN: [11:13:51.102] 199 Conv           neck.0.weight                                  FLOAT16   (256,320,1,1)  | 0x00015c00 0x0003d
c00 0x00028000
D RKNN: [11:13:51.102] 200 exNorm         onnx::Mul_2672                                 FLOAT16   (1,256,1,1)    | 0x00afb440 0x00afb
640 0x00000200
D RKNN: [11:13:51.102] 200 exNorm         onnx::Add_2674                                 FLOAT     (1,256,1,1)    | 0x00afb640 0x00afb
a40 0x00000400
D RKNN: [11:13:51.102] 201 Reshape        /Constant_output_0_sw                          INT64     (4)            | 0x018a3000 0x018a3
020 0x00000020
D RKNN: [11:13:51.102] 202 Conv           neck.2.weight                                  FLOAT16   (256,256,3,3)  | 0x0003dc00 0x0015d
c00 0x00120000
D RKNN: [11:13:51.102] 203 exNorm         onnx::Mul_2676                                 FLOAT16   (1,256,1,1)    | 0x00afba40 0x00afb
c40 0x00000200
D RKNN: [11:13:51.102] 203 exNorm         onnx::Add_2678                                 FLOAT     (1,256,1,1)    | 0x00afbc40 0x00afc
040 0x00000400
D RKNN: [11:13:51.102] ------------------------------------------------------------------------------------------------+------------------
--------------
D RKNN: [11:13:51.106] ----------------------------------------
D RKNN: [11:13:51.106] Total Internal Memory Size: 49784KB
D RKNN: [11:13:51.106] Total Weight Memory Size: 25230.5KB
D RKNN: [11:13:51.106] ----------------------------------------
D RKNN: [11:13:51.107] <<<<<<<< end: rknn::RKNNMemStatisticsPass
I rknn building done.
done
--> Export rknn model
done
```

Figure 4.7-3 Convert the mobilesam_encoder model to a rknn model.

Test whether the connected board inference test model can perform normal inference.

```
cd ~/software/rknn_model_zoo/examples/mobilesam/python
python3 mobilesam.py --encoder ../model/mobilesam_encoder.rknn --decoder ../model/mobilesam_decoder.rknn --target rk3588
```

The result of the reasoning is saved in the file "result.jpg".

```
D RKNNAPI: ==========================================
D RKNNAPI: RKNN VERSION:
D RKNNAPI:   API: 2.3.0 (e80ac5c build@2024-11-07T12:57:35)
D RKNNAPI:   DRV: rknn_server: 2.3.0 (e80ac5c build@2024-11-07T12:52:53)
D RKNNAPI:   DRV: rknnrt: 2.3.0 (c949ad889d@2024-11-07T11:35:33)
D RKNNAPI: ==========================================
D RKNNAPI: Input tensors:
D RKNNAPI:   index=0, name=image_embeddings, n_dims=4, dims=[1, 28, 28, 256], n_elems=200704, size=401408, w_stride = 0, size_with_stride =
0, fmt=NHWC, type=FP16, qnt_type=NONE, zp=0, scale=1.000000
D RKNNAPI:   index=1, name=point_coords, n_dims=3, dims=[1, 2, 2], n_elems=4, size=8, w_stride = 0, size_with_stride = 0, fmt=UNDEFINED, typ
e=FP16, qnt_type=NONE, zp=0, scale=1.000000
D RKNNAPI:   index=2, name=point_labels, n_dims=2, dims=[1, 2], n_elems=2, size=4, w_stride = 0, size_with_stride = 0, fmt=UNDEFINED, type=F
P16, qnt_type=NONE, zp=0, scale=1.000000
D RKNNAPI:   index=3, name=mask_input, n_dims=4, dims=[1, 112, 112, 1], n_elems=12544, size=25088, w_stride = 0, size_with_stride = 0, fmt=N
HWC, type=FP16, qnt_type=NONE, zp=0, scale=1.000000
D RKNNAPI:   index=4, name=has_mask_input, n_dims=1, dims=[1], n_elems=1, size=2, w_stride = 0, size_with_stride = 0, fmt=UNDEFINED, type=FP
16, qnt_type=NONE, zp=0, scale=1.000000
D RKNNAPI: Output tensors:
D RKNNAPI:   index=0, name=iou_predictions, n_dims=2, dims=[1, 4], n_elems=4, size=8, w_stride = 0, size_with_stride = 0, fmt=UNDEFINED, typ
e=FP16, qnt_type=NONE, zp=0, scale=1.000000
D RKNNAPI:   index=1, name=low_res_masks, n_dims=4, dims=[1, 4, 112, 112], n_elems=50176, size=100352, w_stride = 0, size_with_stride = 0, f
mt=NCHW, type=FP16, qnt_type=NONE, zp=0, scale=1.000000
result save to result.jpg
```

Figure 4.7-4 Board chaining inference succeeded

After the previous tests, it was confirmed that the model conversion was completed normally. Before proceeding, the routines in the CPP directory need to be compiled first, and then the compiled folder should be pushed to the development board for board-side inference deployment. Enter the initial directory of modelzoo and open the terminal. First, enable the compiler, and then execute the following command to compile the routines.

```
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d mobilesam
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d mobilesam
./build-linux.sh -t rk3588 -a aarch64 -d mobilesam
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=mobilesam
BUILD_DEMO_PATH=examples/mobilesam/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_mobilesam_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_mobilesam_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
```

Figure 4.7-5 Compile the mobilesam program

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_mobilesam_demo directory under the modelzoo folder. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_mobilesam_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_mobilesam_demo/ /userdata/aidem
o
install/rk3588_linux_aarch64/rknn_mobilesam_demo/: 8 files pushed, 0 skipped. 1.0 MB/s (61632461 bytes in 57.508s)
```

Figure 4.7-6 Push the mobilesam program and model to the development board.

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_mobilesam_demo under the directory pushed to the development board, and execute the following commands in the development board terminal. Use bus.jpg for inference, and the execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_mobilesam_demo
./rknn_mobilesam_demo model/mobilesam_encoder.rknn model/picture.jpg model/mobilesam_decoder.rknn model/coords.txt model/labels.txt
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_mobilesam_demo# ./rknn_mobilesam_demo model/mobilesam_encoder.rknn model/picture.jpg model/mobi
lesam_decoder.rknn model/coords.txt model/labels.txt
--> init mobilesam encoder model
model input num: 1, output num: 1
input tensors:
  index=0, name=input.1, n_dims=4, dims=[1, 448, 448, 3], n_elems=602112, size=1204224, fmt=NHWC, type=FP16, qnt_type=AFFINE, zp=0, sca
le=1.000000
output tensors:
  index=0, name=2044, n_dims=4, dims=[1, 256, 28, 28], n_elems=200704, size=401408, fmt=NCHW, type=FP16, qnt_type=AFFINE, zp=0, scale=1
.000000
model is NHWC input fmt
input image height=448, input image width=448, input image channel=3
--> init mobilesam decoder model
model input num: 5, output num: 2
input tensors:
  index=0, name=image_embeddings, n_dims=4, dims=[1, 28, 28, 256], n_elems=200704, size=401408, fmt=NHWC, type=FP16, qnt_type=AFFINE, z
p=0, scale=1.000000
  index=1, name=point_coords, n_dims=3, dims=[1, 2, 2], n_elems=4, size=8, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000
000
  index=2, name=point_labels, n_dims=2, dims=[1, 2], n_elems=2, size=4, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  index=3, name=mask_input, n_dims=4, dims=[1, 112, 112, 1], n_elems=12544, size=25088, fmt=NHWC, type=FP16, qnt_type=AFFINE, zp=0, sca
le=1.000000
  index=4, name=has_mask_input, n_dims=1, dims=[1], n_elems=1, size=2, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
output tensors:
  index=0, name=iou_predictions, n_dims=2, dims=[1, 4], n_elems=4, size=8, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000
000
  index=1, name=low_res_masks, n_dims=4, dims=[1, 4, 112, 112], n_elems=50176, size=100352, fmt=NCHW, type=FP16, qnt_type=AFFINE, zp=0,
 scale=1.000000
model is NHWC input fmt
input image height=28, input image width=28, input image channel=256
origin size=769x770 crop size=768x768
input image: 769 x 770, subsampling: 4:2:0, colorspace: YCbCr, orientation: 1
num_lines=2
num_lines=2
--> inference mobilesam encoder model
scale=0.581818 dst_box=(2 0 445 447) allow_slight_change=1 _left_offset=2 _top_offset=0 padding_w=4 padding_h=0
src width is not 4/16-aligned, convert image use cpu
finish
rknn_run
--> inference mobilesam decoder model
rknn_run
write image path: out.png width=769 height=770 channel=3 data=0x7f9763c010
```

Figure 4.7-7 The program reasoning is executed at the board end.

The final output result image is saved as "out.png", which can be transmitted to the PC end for viewing. You can see that the violin on the billboard will be segmented and enclosed by a rectangular frame in the end.

Figure 4.7-8 Output image out.png

4.8 Test clip model

First, the model needs to be downloaded. This can be done by executing the model download script. Before execution, it is necessary to check if the Ubuntu network is normal. In the rknn_model_zoo directory after the Ubuntu decompression, execute the following command to download the ONNX format model of clip. This will download the clip_image and clip_text two models.

```
cd examples/clip/model
sh download_model.sh
```

Figure 4.8-1 Download the clip model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the clip routine directory under modelzoo, and then enter the corresponding directory to execute the following commands for converting the clip model.

```
cd ~/software/rknn_model_zoo/examples/clip/python/images
python3 convert.py ../../model/clip_images.onnx rk3588
cd ~/software/rknn_model_zoo/examples/clip/python/text
python3 convert.py ../../model/clip_text.onnx rk3588
```

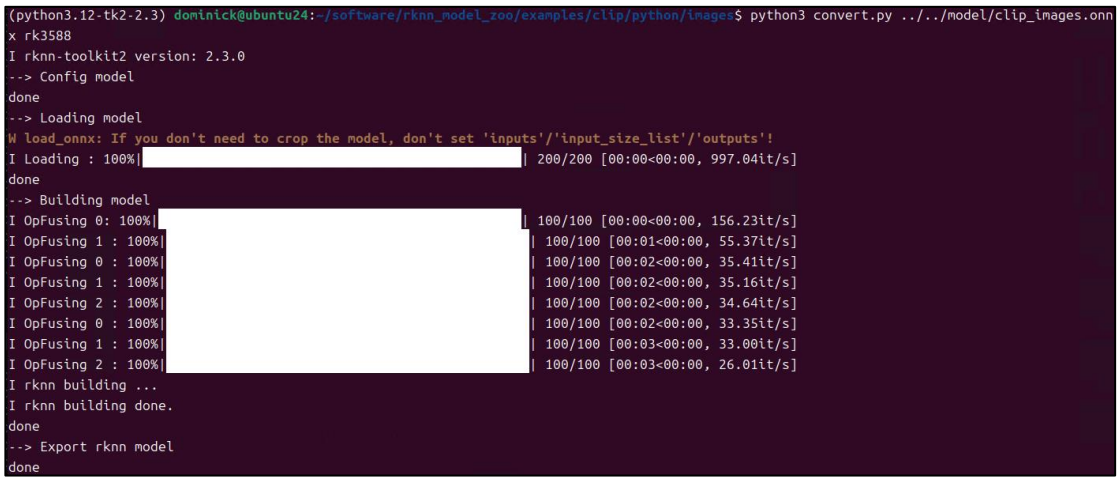


Figure 4.8-2 Convert the clip_image model to a rknn model

```
W build: For tensor ['/text_model/Expand_output_0_6'], the value smaller than -3e+38 has been corrected to -10000. Set opt_level to 2 or low
er to disable this correction.
W build: For tensor ['/text_model/Expand_output_0_7'], the value smaller than -3e+38 has been corrected to -10000. Set opt_level to 2 or low
er to disable this correction.
W build: For tensor ['/text_model/Expand_output_0_8'], the value smaller than -3e+38 has been corrected to -10000. Set opt_level to 2 or low
er to disable this correction.
W build: For tensor ['/text_model/Expand_output_0_9'], the value smaller than -3e+38 has been corrected to -10000. Set opt_level to 2 or low
er to disable this correction.
W build: For tensor ['/text_model/Expand_output_0_10'], the value smaller than -3e+38 has been corrected to -10000. Set opt_level to 2 or lo
wer to disable this correction.
W build: For tensor ['/text_model/Expand_output_0_11'], the value smaller than -3e+38 has been corrected to -10000. Set opt_level to 2 or lo
wer to disable this correction.
I OpFusing 0: 100%|                                | 100/100 [00:00<00:00, 192.17it/s]
I OpFusing 1 : 100%|                               | 100/100 [00:01<00:00, 81.23it/s]
I OpFusing 0 : 100%|                               | 100/100 [00:02<00:00, 41.91it/s]
I OpFusing 1 : 100%|                               | 100/100 [00:02<00:00, 41.11it/s]
I OpFusing 0 : 100%|                               | 100/100 [00:02<00:00, 39.53it/s]
I OpFusing 1 : 100%|                               | 100/100 [00:02<00:00, 39.30it/s]
I OpFusing 2 : 100%|                               | 100/100 [00:02<00:00, 38.65it/s]
I OpFusing 0 : 100%|                               | 100/100 [00:02<00:00, 37.18it/s]
I OpFusing 1 : 100%|                               | 100/100 [00:02<00:00, 36.81it/s]
I OpFusing 2 : 100%|                               | 100/100 [00:03<00:00, 32.67it/s]
I rknn building ...
I rknn building done.
done
--> Export rknn model
done
```

Figure 4.8-3 Convert the clip_text model to a rknn model.

Next, deploy the model to the development board. First, compile the routines in the CPP directory, then push the compiled folder to the development board for on-board deployment and inference. Enter the initial directory of modelzoo, open the terminal, and first enable the compiler, then execute the following command to compile the routines.

```
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d clip
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d clip
./build-linux.sh -t rk3588 -a aarch64 -d clip
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=clip
BUILD_DEMO_PATH=examples/clip/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_clip_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_clip_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ - skipped
```

Figure 4.8-4 Compile the clip program

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_clip_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_clip_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_clip_demo/ /userdata/aidemo
install/rk3588_linux_aarch64/rknn_clip_demo/: 7 files pushed, 0 skipped. 1.1 MB/s (326117961 bytes in 290.561s)
```

Figure 4.8-5 Push the clip program and model to the development board

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_clip_demo under the directory pushed to the development board, and execute the following commands in the development board terminal. Use bus.jpg for inference, and the execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_clip_demo
./rknn_clip_demo model/clip_images.rknn model/dog_224x224.jpg model/clip_text.rknn model/text.txt
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_clip_demo# ./rknn_clip_demo model/clip_images.rknn model/dog_224x224.jpg model/clip_text.rknn model/
text.txt
--> init clip image model
model input num: 1, output num: 1
input tensors:
  index=0, name=pixel_values, n_dims=4, dims=[1, 224, 224, 3], n_elems=150528, size=301056, fmt=NHWC, type=FP16, qnt_type=AFFINE, zp=0, scal
e=1.000000
output tensors:
  index=0, name=image_embeds, n_dims=2, dims=[1, 512], n_elems=512, size=1024, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.0000
00
model is NHWC input fmt
input image height=224, input image width=224, input image channel=3
--> init clip text model
model input num: 1, output num: 1
input tensors:
  index=0, name=input_ids, n_dims=2, dims=[1, 20], n_elems=20, size=160, fmt=UNDEFINED, type=INT64, qnt_type=AFFINE, zp=0, scale=1.000000
output tensors:
  index=0, name=text_embeds, n_dims=2, dims=[1, 512], n_elems=512, size=1024, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.00000
0
model is UNDEFINED input fmt
input text batch size=1, input sequence length=20
origin size=224x224 crop size=224x224
input image: 224 x 224, subsampling: 4:2:0, colorspace: YCbCr, orientation: 1
num_lines=2
--> inference clip image model
rga_api version 1.10.1_[0]
rknn_run
--> inference clip text model
rknn_run
rknn_run
--> rknn clip demo result
images: model/dog_224x224.jpg
text  : a photo of a dog
score : 0.989
```

Figure 4.8-6 The program reasoning is executed at the board end.

As can be seen from the above figure, the final matching result for the image "dog_224x224.jpg" is "a photo of a dog" with a score of 0.989.

Figure 4.8-7 Output image dog_224x224.jpg

4.9 Testing the wav2vec2 model

Note: This section uses the compiler provided by Rockchip Microelectronics; otherwise, errors may occur.

First, you need to download the model. Download the model by executing the model download script. Before execution, it is necessary to check whether the network of Ubuntu is normal. In the rknn_model_zoo directory after decompressing Ubuntu, execute the following command to download the wav2vec2 ONNX format model.

```
cd examples/wav2vec2/model
sh download_model.sh
```

Figure 4.9-1 Download the wav2vec2 model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the directory of the wav2vec2 routine under modelzoo, and execute the following command to convert the wav2vec2 model.

```
cd ~/software/rknn_model_zoo/examples/wav2vec2/python
python3 convert.py ../model/wav2vec2_base_960h_20s.onnx rk3588
```

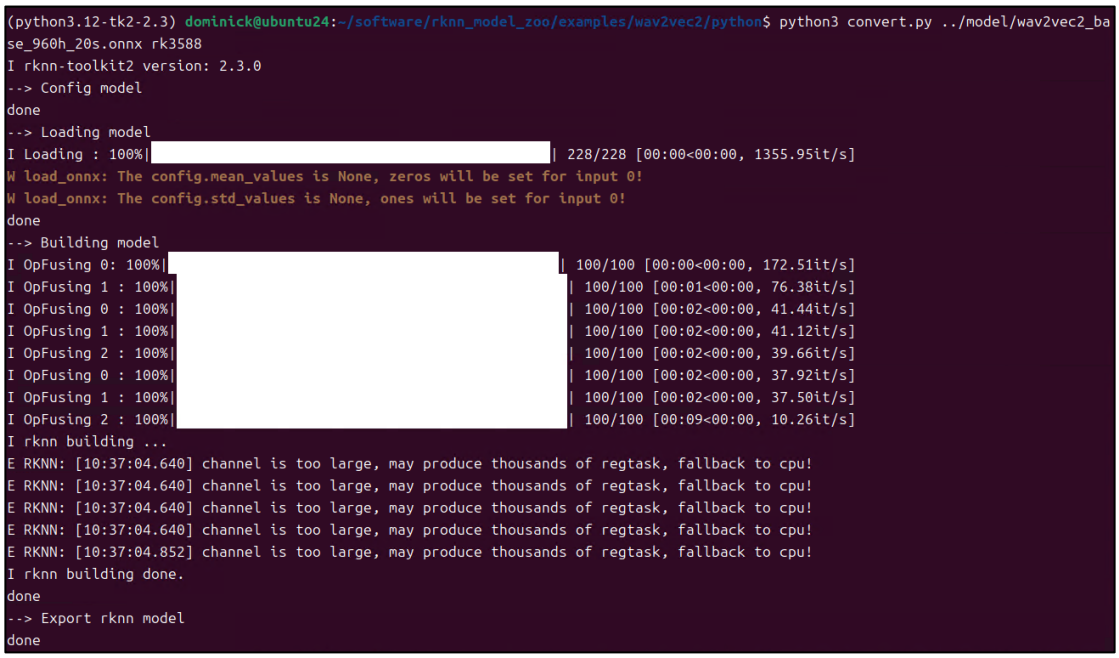


Figure 4.9-2 Convert wav2vec2 to rknn model

After confirming that the model conversion is completed normally, you need to first compile the routines in the CPP directory, and then push the compiled folder to the development board for deployment and inference on the board. Enter the initial directory of modelzoo, open the terminal, and first enable the compiler. Here, you need to use the compiler provided by rk, otherwise, an error will occur. Execute the following command to compile the routines.

```
export GCC_COMPILER=~/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d wav2vec2
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d wav2vec2
./build-linux.sh -t rk3588 -a aarch64 -d wav2vec2
/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
==================================
BUILD_DEMO_NAME=wav2vec2
BUILD_DEMO_PATH=examples/wav2vec2/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_wav2vec2_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_wav2vec2_demo_rk3588_linux_aarch64_Release
CC=/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc
CXX=/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g++
==================================
-- The C compiler identification is GNU 6.3.1
-- The CXX compiler identification is GNU 6.3.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g
cc - skipped
-- Detecting C compile features
```

Figure 4.9-3 Compile the wav2vec2 program

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_wav2vec2_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_wav2vec2_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_wav2vec2_demo/ /userd
ata/aidemo
install/rk3588_linux_aarch64/rknn_wav2vec2_demo/: 5 files pushed, 0 skipped. 1.1 MB/s (220358012 bytes in 184.940s)
```

Figure 4.9-4 Push the wav2vec2 program and model to the development board.

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_wav2vec2_demo that was pushed to the development board, and execute the following commands in the terminal of the development board. Use the test.wav in the model directory for inference. The execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_wav2vec2_demo
./rknn_wav2vec2_demo model/wav2vec2_base_960h_20s.rknn model/test.wav
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_wav2vec2_demo# ./rknn_wav2vec2_demo model/wav2vec2_base_960h_20s.rknn model/test.wav
-- read_audio & convert_channels & resample_audio use: 1.135000 ms
-- audio_preprocess use: 0.453000 ms
model input num: 1, output num: 1
input tensors:
  index=0, name=input, n_dims=2, dims=[1, 320000], n_elems=320000, size=640000, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=
1.000000
output tensors:
  index=0, name=output, n_dims=3, dims=[1, 999, 32], n_elems=31968, size=63936, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=
1.000000
-- init_wav2vec2_model use: 440.355988 ms
-- inference_wav2vec2_model use: 2955.760010 ms

Wav2vec2 output: MISTER QUILTER IS THE APOSTLE OF THE MIDDLE CLASSES AND WE ARE GLAD TO WELCOME HIS GOSPEL

Real Time Factor (RTF): 2.956 / 20.000 = 0.148
```

Figure 4.9-5 Execute program reasoning at the board end

It can be seen that the recognized result is "MISTER QUILTER IS THE APOSTLE OF THE MIDDLE CLASSES AND WE ARE GLAD TO WELCOME HIS GOSPEL". Using gst-launch-1.0 at the development board terminal, the test.wav file can be played for comparison. The command is as follows.

```
gst-launch-1.0 filesrc location=test.wav ! wavparse ! audioconvert ! autoaudiosink
```

The effect of speech recognition is quite good.

4.10 Testing the Whisper model

Note: This section uses the compiler provided by Rockchip Microelectronics. Otherwise, errors may occur.

First, you need to download the model. Download the model by executing the model download script. Before execution, you need to check if the network of Ubuntu is normal. Execute the following command in the rknn_model_zoo directory after decompressing Ubuntu to download the ONNX format model of Whisper.

```
cd examples/whisper/model
sh download_model.sh
```

Figure 4.10-1 Download the Whisper model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the Whisper routine directory under modelzoo, and execute the following command to convert the encoder and decoder models of Whisper.

```
cd ~/software/rknn_model_zoo/examples/whisper/python
python3 convert.py ../model/whisper_encoder_base_20s.onnx rk3588
python3 convert.py ../model/whisper_decoder_base_20s.onnx rk3588
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/whisper/python$ python3 convert.py ../model/whisper_enco
der_base_20s.onnx rk3588
I rknn-toolkit2 version: 2.3.0
--> Config model
done
--> Loading model
I Loading : 100%|                                                | 93/93 [00:00<00:00, 2600.10it/s]
W load_onnx: The config.mean_values is None, zeros will be set for input 0!
W load_onnx: The config.std_values is None, ones will be set for input 0!
done
--> Building model
I OpFusing 0: 100%|                                              | 100/100 [00:01<00:00, 90.36it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:01<00:00, 77.61it/s]
I OpFusing 0 : 100%|                                             | 100/100 [00:01<00:00, 59.05it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:01<00:00, 58.75it/s]
I OpFusing 2 : 100%|                                             | 100/100 [00:01<00:00, 58.19it/s]
I OpFusing 0 : 100%|                                             | 100/100 [00:01<00:00, 57.31it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:01<00:00, 56.90it/s]
I OpFusing 2 : 100%|                                             | 100/100 [00:02<00:00, 33.52it/s]
I rknn building ...
I rknn building done.
done
--> Export rknn model
done
```

Figure 4.10-2 Convert the whisper encoder into a rknn model

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/whisper/python$ python3 convert.py ../model/whisper_deco
der_base_20s.onnx rk3588
I rknn-toolkit2 version: 2.3.0
--> Config model
done
--> Loading model
I Loading : 100%|                                                | 147/147 [00:00<00:00, 554.38it/s]
W load_onnx: The config.mean_values is None, zeros will be set for input 1!
W load_onnx: The config.std_values is None, ones will be set for input 1!
done
--> Building model
W build: For tensor ['/blocks.0/attn/Constant_5_output_0'], the value smaller than -3e+38 has been corrected to -10000. Set opt_le
vel to 2 or lower to disable this correction.
W build: For tensor ['/blocks.0/attn/Constant_5_output_0_1'], the value smaller than -3e+38 has been corrected to -10000. Set opt_
level to 2 or lower to disable this correction.
W build: For tensor ['/blocks.0/attn/Constant_5_output_0_2'], the value smaller than -3e+38 has been corrected to -10000. Set opt_
level to 2 or lower to disable this correction.
W build: For tensor ['/blocks.0/attn/Constant_5_output_0_3'], the value smaller than -3e+38 has been corrected to -10000. Set opt_
level to 2 or lower to disable this correction.
W build: For tensor ['/blocks.0/attn/Constant_5_output_0_4'], the value smaller than -3e+38 has been corrected to -10000. Set opt_
level to 2 or lower to disable this correction.
W build: For tensor ['/blocks.0/attn/Constant_5_output_0_5'], the value smaller than -3e+38 has been corrected to -10000. Set opt_
level to 2 or lower to disable this correction.
I OpFusing 0: 100%|                                              | 100/100 [00:01<00:00, 75.13it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:01<00:00, 59.77it/s]
I OpFusing 0 : 100%|                                             | 100/100 [00:02<00:00, 43.09it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:02<00:00, 42.86it/s]
I OpFusing 2 : 100%|                                             | 100/100 [00:02<00:00, 42.34it/s]
I OpFusing 0 : 100%|                                             | 100/100 [00:02<00:00, 40.75it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:02<00:00, 40.41it/s]
```

Figure 4.10-3 Convert the whisper decoder to an RKNN model

After confirming that the model conversion is completed normally, you need to first compile the routines in the CPP directory, and then push the compiled folder to the development board for deployment and inference on the board. Enter the initial directory of the model zoo, open the terminal, and first enable the compiler. Here, you need to use the compiler provided by RK, otherwise, an error will occur. Execute the following command to compile the routines.

```
export GCC_COMPILER=~/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d whisper
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d whisper
./build-linux.sh -t rk3588 -a aarch64 -d whisper
/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
==================================
BUILD_DEMO_NAME=whisper
BUILD_DEMO_PATH=examples/whisper/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_whisper_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_whisper_demo_rk3588_linux_aarch64_Release
CC=/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc
CXX=/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g++
==================================
-- The C compiler identification is GNU 6.3.1
-- The CXX compiler identification is GNU 6.3.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g
cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
```

Figure 4.10-4 Compile the whisper program

After the compilation is completed, executable files and packaged models, etc. will be generated in the install/rk3588_linux_aarch64/rknn_whisper_demo directory under the modelzoo folder. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_whisper_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_whisper_demo/ /userda
ta/aidemo
install/rk3588_linux_aarch64/rknn_whisper_demo/: 10 files pushed, 0 skipped. 1.0 MB/s (215148140 bytes in 199.864s)
```

Figure 4.10-5 Push the whisper program and model to the development board

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_whisper_demo under the directory pushed to the development board, and execute the following commands in the development board terminal. Use the test_en.wav in the model directory to conduct inference tests on the recognition effect of English voice. The execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_whisper_demo
./rknn_whisper_demo model/whisper_encoder_base_20s.rknn model/whisper_decoder_base_20s.rknn en model/test_en.wav
```

Execute the following command and then test the recognition effect of Chinese to see how it goes.

```
./rknn_whisper_demo model/whisper_encoder_base_20s.rknn model/whisper_decoder_base_20s.rknn zh model/test_zh.wav
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_whisper_demo# ./rknn_whisper_demo model/whisper_encoder_base_20s.rknn model/whisper_decoder_base_20s.r
knn zh model/test_zh.wav
-- read_audio & convert_channels & resample_audio use: 1.206000 ms
-- read_mel_filters & read_vocab use: 27.594999 ms
model input num: 1, output num: 1
input tensors:
  index=0, name=x, n_dims=3, dims=[1, 80, 2000], n_elems=160000, size=320000, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
output tensors:
  index=0, name=out, n_dims=3, dims=[1, 1000, 512], n_elems=512000, size=1024000, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000
000
-- init_whisper_encoder_model use: 76.311996 ms
model input num: 2, output num: 1
input tensors:
  index=0, name=tokens, n_dims=2, dims=[1, 12], n_elems=12, size=96, fmt=UNDEFINED, type=INT64, qnt_type=AFFINE, zp=0, scale=1.000000
  index=1, name=audio, n_dims=3, dims=[1, 1000, 512], n_elems=512000, size=1024000, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.0
00000
output tensors:
  index=0, name=out, n_dims=3, dims=[1, 12, 51865], n_elems=622380, size=1244760, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000
000
-- init_whisper_decoder_model use: 144.296997 ms
-- inference_whisper_model use: 2054.250000 ms

Whisper output: 对我做了介绍,我想做的是大家如果对我的研究感兴趣

Real Time Factor (RTF): 2.054 / 5.611 = 0.366
```

Figure 4.10-6 The program reasoning is executed at the board end.

It can be seen that the recognized result is "I gave an introduction to you. What I want to do is that if everyone is interested in my research", and using gst-launch-1.0 at the development board terminal, the test_en.wav and test_zh.wav files can be played for comparison. The command is as follows.

```
gst-launch-1.0 filesrc location=model/test_en.wav ! wavparse ! audioconvert ! autoaudiosink
gst-launch-1.0 filesrc location=model/test_zh.wav ! wavparse ! audioconvert ! autoaudiosink
```

By comparison, it can be seen that the English and Chinese recognition effects of the speech recognition are both very good.

4.11 Testing the Zipformer model

Note: This section uses the compiler provided by Rockchip Microelectronics. Otherwise, errors may occur.

First, the model needs to be downloaded. The model can be downloaded by executing the model download script. Before execution, it is necessary to check whether the network of Ubuntu is normal. Execute the following command in the rknn_model_zoo directory after decompressing Ubuntu to download the onnx format model of Zipformer.

```
cd examples/zipformer/model
sh download_model.sh
```

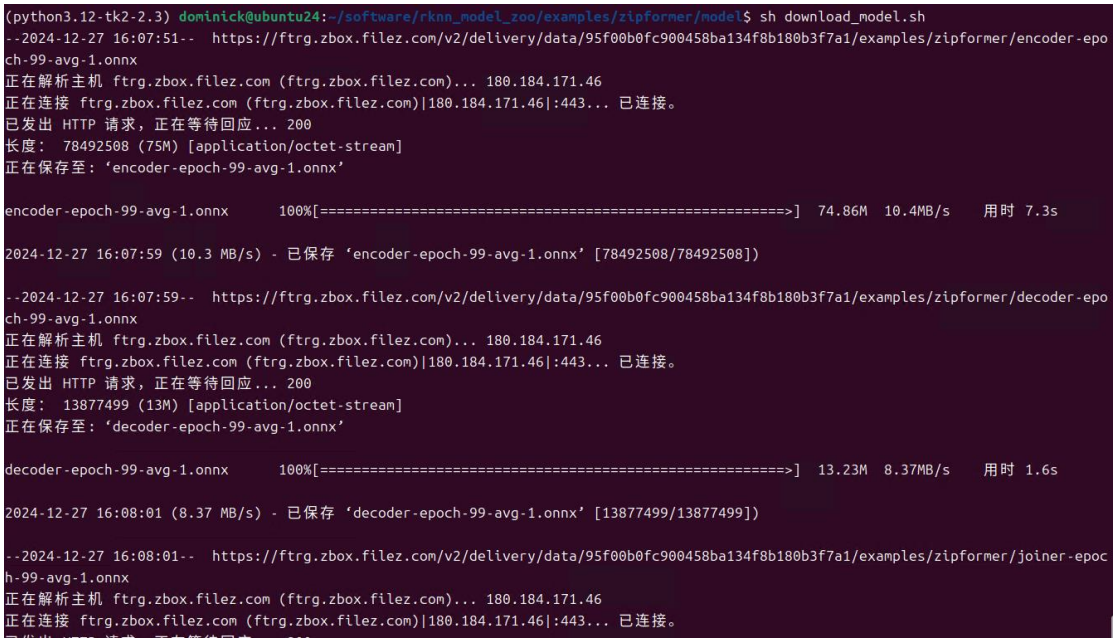


Figure 4.11-1 Download the Zipfomer model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the mobilesam routine directory under modelzoo and execute the following command to convert the zipformer model.

```
cd ~/software/rknn_model_zoo/examples/zipformer/python
python3 convert.py ../model/encoder-epoch-99-avg-1.onnx rk3588
python3 convert.py ../model/decoder-epoch-99-avg-1.onnx rk3588
python3 convert.py ../model/joiner-epoch-99-avg-1.onnx rk3588
```

After confirming that the model conversion is completed normally, you need to first compile the routines in the CPP directory, and then push the compiled folder to the development board for deployment and inference on the board. Enter the initial directory of modelzoo and open the terminal. First, enable the compiler, and then execute the following command to compile the routines.

```
export GCC_COMPILER=~/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d zipformer
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d zipformer
./build-linux.sh -t rk3588 -a aarch64 -d zipformer
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=zipformer
BUILD_DEMO_PATH=examples/zipformer/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_zipformer_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_zipformer_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
```

Figure 4.11-2 Compile the zipformer program

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_zipformer_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_zipformer_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_zipformer_demo/ /user
data/aidemo
install/rk3588_linux_aarch64/rknn_zipformer_demo/: 8 files pushed, 0 skipped. 1.0 MB/s (136611035 bytes in 127.399s)
```

Figure 4.11-3 Push the zipformer program and model to the development board.

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_zipformer_demo on the development board, and execute the following command in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_zipformer_demo
./rknn_zipformer_demo model/encoder-epoch-99-avg-1.rknn model/decoder-epoch-99-avg-1.rknn model/joiner-epoch-99-avg-1.rknn model/test.wav
```

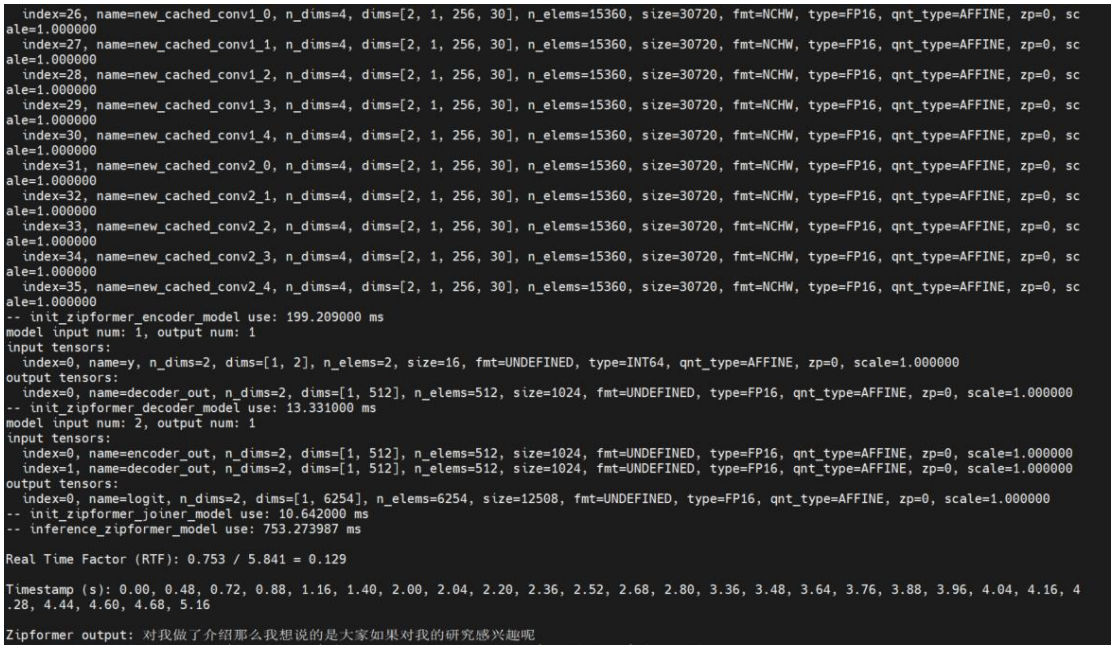


Figure 4.11-4 The program at the board end performs reasoning.

The audio is consistent with the Chinese wav file provided by the previous routine. It can be seen that the recognized result is "I gave an introduction to you. What I want to do is that if everyone is interested in my research, then...". It is more accurate than the whisper recognition, and finally it correctly recognized "then".

4.12 Test the Yamnet model

Note: This section uses the compiler provided by Rockchip Microelectronics. Otherwise, errors may occur.

First, you need to download the model. Download the model by executing the model download script. Before execution, you need to check if the network of Ubuntu is normal. Execute the following command in the rknn_model_zoo directory after decompressing Ubuntu to download the ONNX format model of Yamnet.

```
cd examples/yamnet/model
sh download_model.sh
```

Figure 4.12-1 Download the Yamnet model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the YamNet routine directory under the modelzoo, and execute the following command to convert the onnx model of YamNet.

```
cd ~/software/rknn_model_zoo/examples/yamnet/python
python3 convert.py ../model/yamnet_3s.onnx rk3588
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/yamnet/python$ python3 convert.py ../model/yamnet_3s.onn
x rk3588
I rknn-toolkit2 version: 2.3.0
--> Config model
done
--> Loading model
I Loading : 100%|| 79/79 [00:00<00:00, 16094.33it/s]
W load_onnx: The config.mean_values is None, zeros will be set for input 0!
W load_onnx: The config.std_values is None, ones will be set for input 0!
done
--> Building model
I OpFusing 0: 100%|| 100/100 [00:00<00:00, 3631.56it/s]
I OpFusing 1 : 100%|| 100/100 [00:00<00:00, 1042.88it/s]
I OpFusing 0 : 100%|| 100/100 [00:00<00:00, 287.77it/s]
I OpFusing 1 : 100%|| 100/100 [00:00<00:00, 252.61it/s]
I OpFusing 0 : 100%|| 100/100 [00:00<00:00, 219.85it/s]
I OpFusing 1 : 100%|| 100/100 [00:00<00:00, 209.81it/s]
I OpFusing 2 : 100%|| 100/100 [00:00<00:00, 205.57it/s]
I OpFusing 0 : 100%|| 100/100 [00:00<00:00, 182.93it/s]
I OpFusing 1 : 100%|| 100/100 [00:00<00:00, 175.86it/s]
I OpFusing 2 : 100%|| 100/100 [00:00<00:00, 142.33it/s]
I rknn building ...
I rknn building done.
done
--> Export rknn model
done
```

Figure 4.12-2 Convert Yamnet to RKNN model

After confirming that the model conversion is completed normally, you need to first compile the routines in the CPP directory, and then push the compiled folder to the development board for deployment and inference on the board. Enter the initial directory of modelzoo, open the terminal, and first enable the compiler. Here, you need to use the compiler provided by RK, otherwise, an error will occur. Execute the following command to compile the routines.

```
export GCC_COMPILER=~/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yamnet
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d yamnet
./build-linux.sh -t rk3588 -a aarch64 -d yamnet
/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
==================================
BUILD_DEMO_NAME=yamnet
BUILD_DEMO_PATH=examples/yamnet/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_yamnet_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_yamnet_demo_rk3588_linux_aarch64_Release
CC=/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc
CXX=/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g++
==================================
-- The C compiler identification is GNU 6.3.1
-- The CXX compiler identification is GNU 6.3.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g
cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
```

Figure 4.12-3 Compile the Yamnet program

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_yamnet_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yamnet_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_yamnet_demo/ /userdat
a/aidemo
install/rk3588_linux_aarch64/rknn_yamnet_demo/: 6 files pushed, 0 skipped. 1.0 MB/s (18748438 bytes in 17.217s)
```

Figure 4.12-4 Push the Yamnet program and model to the development board.

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_yamnet_demo under the directory pushed to the development board, and execute the following commands in the development board terminal. Use the test.wav in the model directory to conduct inference tests for the voice classification effect. The execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_yamnet_demo
./rknn_yamnet_demo model/yamnet_3s.rknn model/test.wav
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yamnet_demo# ./rknn_yamnet_demo model/yamnet_3s.rknn model/test.wav
model input num: 1, output num: 3
input tensors:
  index=0, name=new_input, n_dims=2, dims=[1, 48000], n_elems=48000, size=96000, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
output tensors:
  index=0, name=embeddings, n_dims=2, dims=[6, 1024], n_elems=6144, size=12288, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  index=1, name=log_mel_spectrogram, n_dims=2, dims=[336, 64], n_elems=21504, size=43008, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.0000
00
  index=2, name=scores, n_dims=2, dims=[6, 521], n_elems=3126, size=6252, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
-- inference_yamnet_model use: 23.893000 ms

The main sound is: Animal

Real Time Factor (RTF): 0.024 / 3.000 = 0.008
```

Figure 4.12-5 The program reasoning is executed at the board end.

It can be seen that the recognized result is "Animal". On the development board terminal, using gst-launch-1.0, the test.wav file can be played. The command is as follows.

```
gst-launch-1.0 filesrc location=model/test.wav ! wavparse ! audioconvert ! autoaudiosink
```

A voice that sounds like a cat's meow can be heard.

4.13 Test the mms_tts model

Note: This section uses the compiler provided by Rockchip Microelectronics; otherwise, errors may occur.

First, you need to download the model. This can be done by executing the model download script. Before execution, it is necessary to check if the Ubuntu network is functioning properly. Then, in the rknn_model_zoo directory after the Ubuntu installation, execute the following command to download the ONNX format model of mms_tts.

```
cd examples/mms_tts/model
sh download_model.sh
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/mms_tts/model$ sh download_model.sh
--2024-12-30 16:36:06--  https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/mms_tts/mms_tts_en
g_encoder_200.onnx
正在解析主机 ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
正在连接 ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度： 27947746 (27M) [application/octet-stream]
正在保存至：‘mms_tts_eng_encoder_200.onnx’

mms_tts_eng_encoder_200.onnx     100%[=======================================================>]  26.65M  10.3MB/s    用时 2.6s

2024-12-30 16:36:10 (10.3 MB/s) - 已保存 ‘mms_tts_eng_encoder_200.onnx’ [27947746/27947746])

--2024-12-30 16:36:10--  https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/mms_tts/mms_tts_en
g_decoder_200.onnx
正在解析主机 ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
正在连接 ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度： 97408766 (93M) [application/octet-stream]
正在保存至：‘mms_tts_eng_decoder_200.onnx’

mms_tts_eng_decoder_200.onnx     100%[=======================================================>]  92.90M  10.3MB/s    用时 9.1s

2024-12-30 16:36:19 (10.2 MB/s) - 已保存 ‘mms_tts_eng_decoder_200.onnx’ [97408766/97408766])
```

Figure 4.13-1 Download the mms_tts model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.12-tk2-2.3
```

Enter the mms_tts routine directory under modelzoo, and execute the following command to convert the onnx model of mms_tts.

```
cd ~/software/rknn_model_zoo/examples/mms_tts/python
python3 convert.py ../model/mms_tts_eng_encoder_200.onnx rk3588
python3 convert.py ../model/mms_tts_eng_decoder_200.onnx rk3588
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/mms_tts/python$ python3 convert.py ../model/mms_tts_eng_
encoder_200.onnx rk3588
I rknn-toolkit2 version: 2.3.0
--> Config model
done
--> Loading model
I Loading : 100%|                                                | 851/851 [00:00<00:00, 115704.00it/s]
done
--> Building model
W build: For tensor ['793'], the value smaller than -3e+38 has been corrected to -10000. Set opt_level to 2 or lower to disable th
is correction.
I OpFusing 0: 100%|                                              | 100/100 [00:01<00:00, 57.51it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:04<00:00, 24.23it/s]
I OpFusing 0 : 100%|                                             | 100/100 [00:07<00:00, 14.21it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:07<00:00, 13.38it/s]
I OpFusing 0 : 100%|                                             | 100/100 [00:10<00:00,  9.98it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:10<00:00,  9.89it/s]
I OpFusing 2 : 100%|                                             | 100/100 [00:10<00:00,  9.80it/s]
I OpFusing 0 : 100%|                                             | 100/100 [00:11<00:00,  9.04it/s]
I OpFusing 1 : 100%|                                             | 100/100 [00:11<00:00,  8.96it/s]
I OpFusing 2 : 100%|                                             | 100/100 [00:11<00:00,  8.45it/s]
I rknn building ...
```

Figure 4.13-2 Convert the mms_tts_encoder to a rknn model

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/mms_tts/python$ python3 convert.py ../model/mms_tts_eng_
decoder_200.onnx rk3588
I rknn-toolkit2 version: 2.3.0
--> Config model
done
--> Loading model
I Loading : 100%|| 240/240 [00:00<00:00, 13182.21it/s]
W load_onnx: The config.mean_values is None, zeros will be set for input 0!
W load_onnx: The config.std_values is None, ones will be set for input 0!
W load_onnx: The config.mean_values is None, zeros will be set for input 1!
W load_onnx: The config.std_values is None, ones will be set for input 1!
W load_onnx: The config.mean_values is None, zeros will be set for input 2!
W load_onnx: The config.std_values is None, ones will be set for input 2!
W load_onnx: The config.mean_values is None, zeros will be set for input 3!
W load_onnx: The config.std_values is None, ones will be set for input 3!
done
--> Building model
I OpFusing 0: 100%| 100/100 [00:00<00:00, 208.42it/s]
I OpFusing 1 : 100%| 100/100 [00:01<00:00, 73.74it/s]
I OpFusing 0 : 100%| 100/100 [00:02<00:00, 49.72it/s]
I OpFusing 1 : 100%| 100/100 [00:02<00:00, 48.47it/s]
I OpFusing 2 : 100%| 100/100 [00:05<00:00, 17.84it/s]
I rknn building ...
I rknn building done.
done
--> Export rknn model
done
```

Figure 4.13-3 Convert mms_tts_decoder to rknn model

After confirming that the model conversion is completed normally, you need to first compile the routines in the CPP directory, and then push the compiled folder to the development board for deployment and inference on the board. Enter the initial directory of modelzoo, open the terminal, and first enable the compiler. Here, you need to use the compiler provided by rk, otherwise, an error will occur. Execute the following command to compile the routines.

```
export GCC_COMPILER=~/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d mms_tts
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/mms_tts/python$ export GCC_COMPILER=~/software/gcc-linar
o-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo/examples/mms_tts/python$ cd ../../../
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ ./build-linux.sh -t rk3588 -a aarch64 -d mms_tts
./build-linux.sh -t rk3588 -a aarch64 -d mms_tts
/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
==================================
BUILD_DEMO_NAME=mms_tts
BUILD_DEMO_PATH=examples/mms_tts/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
DISABLE_RGA=OFF
INSTALL_DIR=/home/dominick/software/rknn_model_zoo/install/rk3588_linux_aarch64/rknn_mms_tts_demo
BUILD_DIR=/home/dominick/software/rknn_model_zoo/build/build_rknn_mms_tts_demo_rk3588_linux_aarch64_Release
CC=/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc
CXX=/home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g++
==================================
-- The C compiler identification is GNU 6.3.1
-- The CXX compiler identification is GNU 6.3.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /home/dominick/software/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-g
cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
```

Figure 4.13-4 Compile the mms_tts program

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_mms_tts_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_mms_tts_demo/ /userdata/aidemo
```

```
(python3.12-tk2-2.3) dominick@ubuntu24:~/software/rknn_model_zoo$ adb push install/rk3588_linux_aarch64/rknn_mms_tts_demo/ /userda
ta/aidemo
install/rk3588_linux_aarch64/rknn_mms_tts_demo/: 5 files pushed, 0 skipped. 1.0 MB/s (91030720 bytes in 83.084s)
```

Figure 4.13-5 Push the mms_tts program and model to the development board

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_mms_tts_demo that was pushed to the development board, and execute the following commands in the terminal of the development board. Use the test.wav in the model directory to conduct inference tests on the voice classification effect. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_mms_tts_demo
```

Please note that the following is a single line.

```
./rknn_mms_tts_demo model/mms_tts_eng_encoder_200.rknn model/mms_tts_eng_decoder\
_200.rknn '"Mister quilter is the apostle of the middle classes and we are glad to welc\
ome his gospel."'
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_mms_tts_demo# ./rknn_mms_tts_demo model/mms_tts_eng_encoder_200.rknn model/mms_tts_eng_decoder\
> _200.rknn '"Mister quilter is the apostle of the middle classes and we are glad to welc\
> ome his gospel."'
model input num: 2, output num: 4
input tensors:
  index=0, name=input_ids, n_dims=2, dims=[1, 200], n_elems=200, size=1600, fmt=UNDEFINED, type=INT64, qnt_type=AFFINE, zp=0, scale=1.000000
  index=1, name=attention_mask, n_dims=2, dims=[1, 200], n_elems=200, size=1600, fmt=UNDEFINED, type=INT64, qnt_type=AFFINE, zp=0, scale=1.000000
output tensors:
  index=0, name=log_duration, n_dims=3, dims=[1, 1, 200], n_elems=200, size=400, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  index=1, name=input_padding_mask, n_dims=3, dims=[1, 1, 200], n_elems=200, size=400, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  index=2, name=prior_means, n_dims=3, dims=[1, 200, 192], n_elems=38400, size=76800, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  index=3, name=prior_log_variances, n_dims=3, dims=[1, 200, 192], n_elems=38400, size=76800, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.
000000
-- init_mms_tts_encoder_model use: 61.320999 ms
model input num: 4, output num: 1
input tensors:
  index=0, name=attn, n_dims=4, dims=[1, 400, 200, 1], n_elems=80000, size=160000, fmt=NHWC, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  index=1, name=output_padding_mask, n_dims=3, dims=[1, 1, 400], n_elems=400, size=800, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  index=2, name=prior_means, n_dims=3, dims=[1, 200, 192], n_elems=38400, size=76800, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  index=3, name=prior_log_variances, n_dims=3, dims=[1, 200, 192], n_elems=38400, size=76800, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.
000000
output tensors:
  index=0, name=waveform, n_dims=2, dims=[1, 102400], n_elems=102400, size=204800, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
-- init_mms_tts_decoder_model use: 152.425003 ms
-- read_vocab use: 0.016000 ms
-- inference_mms_tts_model use: 658.310974 ms

Real Time Factor (RTF): 0.658 / 6.400 = 0.103

The output wav file is saved: output.wav
root@ATK-DLRK3588:/userdata/aidemo/rknn_mms_tts_demo#
```

Figure 4.13-6 Execute program reasoning at the board end

Using gst-launch-1.0 on the development board terminal allows you to play the generated output.wav file. The command is as follows.

```
gst-launch-1.0 filesrc location=output.wav ! wavparse ! audioconvert ! autoaudiosink
```

A smooth voice can be heard. "Mister quilter is the apostle of the middle classes and we are glad to welcome his gospel." It is also possible to test by replacing the corresponding words.