Model Transcoding and Running User Guide

Revision: 1.0

Release Date: 2022-02-10

Contact Information

- Website: www.amlogic.com
- Pre-sales consultation: contact@amlogic.com
- Technical support: support@amlogic.com

Revision History

Issue 1.0 (2022-02-10)

This is the tenth official release.

1. Remove a set of conversion model commands in the previous version.
2. Add the unified conversion command Pegasus, and add the model conversion process corresponding to Pegasus.

Issue 0.9 (2021-08-10)

This is the ninth official release.

1. Added quantitative guidance for new ID
2. Added per-channel quantitative corresponding guidance and restrictions

Issue 0.8 (2021-01-15)

This is the eighth official release.

Compared to last version, the following changes are made:

1. Added environmental installation notes
2. Added guidance for using multiple pictures when quantifying

Issue 0.7 (2020-07-18)

This is the seventh official release.

Compared to last version, the following changes are made:

1. Added 3.6 to describe transformation instruction using multi-input model.
2. Deleted IDE tool related description.

Issue 0.6 (2020-05-18)

This is the sixth official release.

Compared to last version, chapter 3.5 are added.

Issue 0.5 (2019-12-20)

This is the fifth official release.

Compared to last version, the following changes are made:

1. Added a guide for using acuity tool to inference.
2. Added detailed guidance on quantization parameters during model conversion
3. Removed the FAQ chapter and moved the corresponding chapter to the Amlogic-NN-FAQ document.
4. Added darknet and Keras framework model imports instructions.

Issue 0.4 (2019-07-18)

This is the fourth official release.

Compared to last version, the following changes are made:

1. Optimized the environment installation procedure and provided the one-click installation method.
2. Added introduction to model quantification and guidance to quantitative parameter selection for model transcoding.
3. Deleted the project code compilation chapter. For details about project code compilation, see *Android&Linux Development Guide.docx*.
4. Deleted the general knowledge chapter.
5. Added quantitative/anti-quantization guidance in the FAQ chapter to help customers solve the problem of exceedingly long processing time.

Issue 0.3 (2019-07-08)

This is the third official release.

Compared to last version, the following changes are made:

1. Added ONNX model transcoding commands.
2. Added whl package installation instructions for the model transcoding tool.
3. Changed the type of DDK_6.3.3.4 offline model to nbg and modified the corresponding module description.
4. Deleted the code summary chapter because the case code directory is directly generated after the case code is transcoded through the DDK_6.3.3.4 model.
5. Added the case code introduction chapter.

Issue 0.2 (2018-12-20)

This is the second official release.

Compared to last version, the following changes are made:

1. Modified errors of and missing information about the default environment installation procedure.
2. Deleted the chapter describing how to use IDE to transcode case code and adjust content in some chapters.
3. Simplified the transcoding process with some of the operations deleted and move the FAQ part to Chapter 7.
4. Added the IDE tool instructions chapter, introducing how to import case code, add the jpeg library, and compile and run the code.
5. Added the method for external personnel to obtain version information in the project code chapter and the introduction to case code compilation based on internal and external code.
6. Added the FAQ chapter.

Issue 0.1 (2018-11-12)

This is the Initial release.

Contents

1. Introduction

This document introduces how you can use the Acuity_tool to transcode case code from a model for the convenience of model transcoding for both internal and external developers.

Currently, NN chips only support models of tensorflow, caffe, tflite, darknet, onnx,pytorch and keras types. In this case, if you use another type of model, you need to convert the model type to one of the preceding types at first.

2. Tools

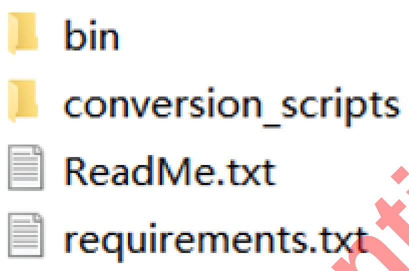
2.1 Overview

Currently, you can use the following tools to transcode case code.

1. The acuity-toolkit model tool: You can use it to quantize models and transcode case code, which can be run after compilation.

2.2 Model Transcoding Tool

2.2.1 Introduction to the Tool Directory



bin
conversion_scripts
ReadMe.txt
requirements.txt

The **bin** folder includes executable files required for model transcoding and the configuration file directory.

The **conversion_scripts** folder includes model transcoding demos (including transcoding scripts and the mobilenet_v1.pb model). You only need to run the three sh scripts in sequence after the environment is ready. Before you transcode your own model, be sure to modify the configuration parameters of the three scripts.

The **requirements.txt** file lists all the software packages required by the environment.

The **ReadMe.txt** file simply introduces how the tool works.

2.2.2 Environmental Dependencies

Operating System	**Ubuntu16.04 (x64)**
Python	Python3
Dependent library	tensorflow==1.13.2 numpy==1.16.4 scipy==1.1.0 Pillow==5.3.0 protobuf==3.9.0 networkx==1.11 image==1.5.5 lmdb==0.93 onnx==1.4.1 h5py==2.9.0 flatbuffers==1.10 matplotlib==2.1.0 dill==0.2.8.2 ruamel.yaml==0.15.81 onnx_tf==1.2.1 ply==3.11

	torch==1.2.0

Note: The above only lists some libraries that are currently needed, and the specific dependent libraries are based on the requirements.txt in the acuity_toolkit-binary-xxx inside the tool.

2.2.3 Installing the Tool

Step 1 Prepare a computer with the 64-bit Ubuntu 16.04 system. (If it is a virtual machine, it's RAM memory space must be larger than 4 GB.),Python requires version 3.5.2.

Step 2 Run the following command to install python3 and pip:

```
sudo apt-get install python3 python3-pip python3-virtualenv
```

Step 3 Do as follows to install the related dependencies:

```
Obtain the acuity-toolkit-binar toolkit and go to the root directory.

Execute:   for req in $(cat requirements.txt); do pip3 install $req; done
```

Step 4 Check the environment

```
Execute: python3 bin/checkenv.py

Note: if check success, will have Env Pass: Env check SUCCESS!!! print.
```

3. Model Transcoding

3.1 Introduction

During model transcoding, models are first quantized to data in the int8, int16, or uint8 format, and then are converted to nbg files that can be run in our platform. Then, case code can be exported.

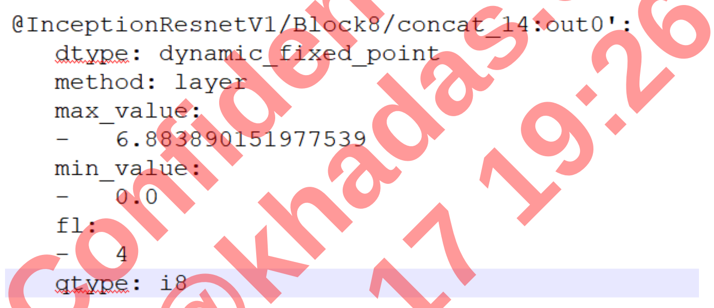
3.2 Quantization

Quantization is the process that 32- or 64-bit floating point numbers are stored as 2-bit data.

3.2.1 Introduction

Int8: indicates that 32-bit floating point numbers are quantized to int8 data. Int8 data includes eight bits, among which one is the sign bit and the other seven are valid numbers. You need to calculate the fl value that indicates the number of bits in the decimal. (The mechanism for int16 is similar to that of int8.)

You can refer to xxxx.quantize that is generated after Step 2 in section 3.4 is executed. The following figure shown the analysis process:



```
@InceptionResnetV1/Block8/concat_14:out0':
   dtype: dynamic_fixed_point
   method: layer
   max_value:
   -    6.883890151977539
   min_value:
   -    0.0
   fl:
   -    4
   qtype: i8
```

Analysis: qtype indicates that the quantization mode is int8. The value range of the calculation result is (0.0, 6.88).

The maximum absolute value of the result is 6.88, which only requires 3 bits for the integer part.

Therefore, the number of bits used to represent the decimal is fl=8-1-3=4.

u8: indicates that 32-bit floating point numbers are quantized to unsigned int8 data. All eight bits indicate numbers, and there is no sign bit. That is, the value range is 0 to 255. In this case, you need to calculate two parameters, scale and zero_point based on the following formula:

scale = (max_value - min_value)/255

zero_point = max_value - max_value/scale

You can refer to xxxx.quantize that is generated after Step 2 in section 3.4 is executed.

```
'@FeatureExtractor/MobilenetV1/MobilenetV1/Conv2d_0_1:weight':
    dtype: asymmetric_quantized
    method: layer
    max_value:
    -   2.7051823139190674
    min_value:
    -   -2.880463123321533
    zero_point:
    -   132
    scale:
    -   0.021990729495882988
    qtype: u8
```

Analysis: qtype indicates that the quantization mode is u8. The value range of calculation results is (–2.88, 2.7).

Therefore, the parameters are calculated as follows:

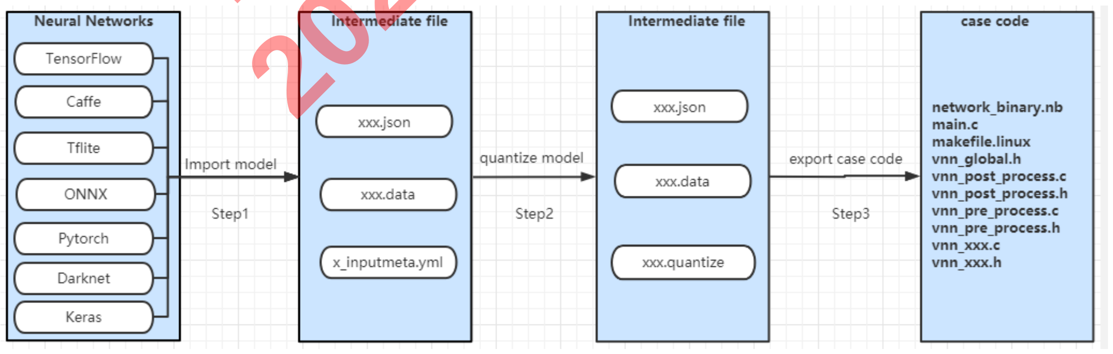
scale = (2.705 -(-2.88))/255= 0.0219

zero_point = 255 - 2.705/scale = 132

3.2.2 Rules of Selecting a Proper Quantitation Method

1. In normal cases, u8 and int8 are applicable to all models. However, Google recommends u8.
2. In normal cases, int16 is not recommended because it requires models with size that is twice the models used for 8-bit methods. Additionally, int16 has no advantage in accuracy.
3. Take the value ranges for pre- and post-processing into consideration. For example, if the value range for pre-processing is (0, 255), u8 is recommended.
4. Randomly select a quantization method. After quantization is completed, check the generated xxxx.quantize file and check the max_value and min_value of related statistics. If there is any value with its absolute value greater than 256, int16 is recommended. If there is any value with its absolute value greater than 128, u8 is recommended. In other cases, both int8 and u8 are applicable.

3.3 Model Conversion Flowchart



3.4 Model Transcoding Procedure

Model transcoding is performed in the **acuity-toolkit** directory. The executable files are all in bin directory.

Step 1 : Import the model and generate a quantitative configuration file.
Commands:

```
Step A:

    Caffe:
        ./bin/pegasus   import caffe   --model xxxx.prototxt --weights xxx.caffemodel \
         --output-data   xxx.data   --output-model xxx.json

    Darknet:
        ./bin/pegasus   import   darknet    --model xxx.cfg --weights xxx.weights \
                     --output-data   xxx.data   --output-model xxx.json

    Tensorflow:
                ./bin/pegasus import    tensorflow   --model xxx.pb   \
                 --inputs input   --outputs InceptionV1/Logits/Predictions/Reshape_1 \
    --input-size-list '224,224,3'   \
    --output-data xxx.data --output-model xxx.json

    Tflite:
                ./bin/pegasus import   tflite   --model   xxxx.tflite   \
                 --output-data xxx.data --output-model xxx.json

    Onnx:
                ./bin/pegasus import   onnx --model   xxxx.onnx   \
                 --output-data xxx.data --output-model xxx.json   \
    # --inputs "0 1 2"   --input-size-list "288,288,3#288,288,3#288,288,3" \
    # --outputs "327 328"

    Keras:
                ./bin/pegasus import keras --model   xxxx..hdf5 \
                  --output-data xxx.data --output-model xxx.json

    Pytorch:
                ./bin/pegasus import pytorch --model   xxxx...pt   \
                 --output-data xxx.data --output-model xxx.json   --input-size-list '3,224,224'

Step B:
    Generate a configuration file:
                ./bin/pegasus   generate inputmeta --model ${NAME}.json \
                 --channel-mean-value "128 128 128 0.0078125"   --source-file dataset.txt \
                --input-meta-output ${NAME}_inputmeta.yml   \
                #   --separated-database
```

Result: Three intermediate files, **xxx.json** , **xxx.data** and `xxx_inputmeta.yml` are generated.

Note

1. *For tensorflow models, names of the input and output nodes and the size of the input HWC are required. The information can be obtained through summarize_graph or tensorboard.*
2. ***--inputs/--outputs** indicates names of input/output nodes. If there are multiple input or output nodes, separate their names with spaces, for example, **output1 output2 output3**.*
3. ***--input-size-list** If there are multiple input nodes, separate their names with number signs (#), for example, **224,224,3#299,299,3**.*
4. *Currently, only Caffe/Tensorflow/Tflite/Darknet/Onnx models are supported. If you use models of other types, convert them to one of the preceding types first.*
5. If there are some logic control inputs for the tensorflow model, please refer to the 3.17 instructions in the Amlogic-NN-FAQ document for processing.
6. ***--channel-mean-value** indicates that you need to set the pre-processing command-line parameters based on the pre-processing methods used for model training. It includes four values, m1, m2, m3, and scale. The first three are mean-value parameters and the last one is the scale parameter. The following uses the three-channel input data (data1, data2, data3) to show how pre-processing is performed:*

   Out1 = (data1-m1)*scale

   Out2 = (data2-m2)*scale

   Out3 = (data3-m3)*scale

   For example: if the pre-processing requires that data is normalized to the range [–1, 1], set the parameter value range to (128 128 128 0. 0078125).

   If the pre-processing requires that data is normalized to the range [0, 1], set the parameter value range to (0 0 0 0. 00392156).

   For single-channel parameters, set the parameter value range to (m1, 0, 0, scale).

   The last number scale is a decimal, such as 0.0078125 (1/128), 0.00392156 (1/256).
7. ***dataset.txt** specifies the quantized input image path, for example, **/data/test/cat.jpg**. You are advised to provide about 200 images showing the use and running conditions for quantization to ensure that the maximum and minimum values of the statistics are the same as those when the model is running for better quantization effects.*
8. Notes on multi-input model conversion

   1. ***--separated-database** If the model is a multi-input model, the yml file generated by adding the subparameters will have a separate dataset.txt for each input. if this parameter is not set, there will be a single dataset.txt for all the inputs, so the dataset.txt should have multiple inputs on one line separated by blank spaces, e.g. a set of inputs: . /1.jpg . . /2.jpg . /3.jpg. If you want to quantize more than one image, you can set multiple image paths.*

   2. ***--channel-mean-value** Set the channel-mean value corresponding to the input, you can also directly modify the generated xxx_inputmeta.yml file, and separate the channel-mean of multiple input pairs by #, e.g. a set of channel-mean-value of input pairs: "128 128 128 128 0.00715#106 115 67 0.00715#121 99 112 0.00715".*

   3. ***--source-file** Set the input source, or you can modify the generated xxx_inputmeta.yml file directly. If --separated-database is set, the corresponding setting for input at this point is: "dataset1.txt#dataset2.txt#dataset3.txt"*

Step 2 Quantify the model:

Commands:

```
../bin/pegasus   quantize   --quantizer asymmetric_affine\
                  --qtype uint8   --with-input-meta   ${NAME}_inputmeta.yml \
                  --model   ${NAME}.json --model-data   ${NAME}.data \
                  --rebuild   \

     # --batch-size   16   --iterations 10
```

Result: According to the input image in dataset.txt, the maximum and minimum values of each layer and those of the model weight can be calculated through forward reasoning. You can calculate the quantization parameters of each layer and save them as quantized result file **xxx.quantize**, which will be used when you export case code.

Note

1. ***--quantizer and -dtype*** *are together used to select the quantization type, supported types are:*

   asymmetric_affine ---> corresponds to the qtype option to set uint8

   dynamic_fixed_point ---> corresponds to the qytpe option to set int8 or int16

   perchannel_symmetric_affine (perchannel quantization, corresponding to qytpe to set int8, currently only E8 chip support, refer to generate case code in Step3---optimize parameter setting) dynamic_fixed_point-i8

2. ***--quantized-rebuild*** *is the default parameter. You are advised to set it because if you do not set it and you use different quantization methods to quantize a model, the second quantization process does not take effect on the condition that you do not delete the generated* ***xxx.quantize*** *file.*

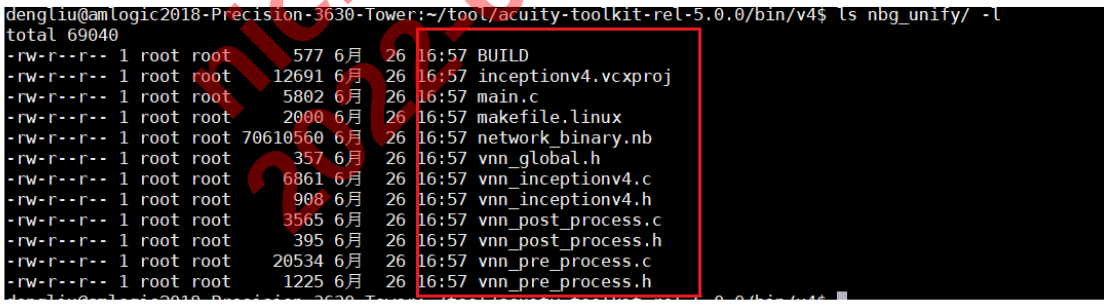
3. ***--batch-size and —iterations*** *To set parameters for quantizing , if the number of images is more than 1, add the quantization parameters --batch-size (1 as default) and --iterations (1 as default), and iterations *batch_size = number of images. For example, for 5000 images, set the parameters as: --batch-size 100 --iterations 50.*

   If the computer memory is small, you can set the value of batch_size to a smaller value.

Step 3 Code to generate model case:

```
./bin/pegasus     export ovxlib   \
          --model xxx.json --model-data xxx.data \
          -model-quantize xxxx.quantize   --with-input-meta   xxxx_inputmeta.yml \\
            --dtype quantized    \
          --optimize VIPNANOQI_PID0X88   \
          --viv-sdk    ./bin/vcmdtools \
          --pack-nbg-unify
```

Result: Case code is generated, as shown in the following figure:



```
dengliu@amlogic2018-Precision-3630-Tower:~/tool/acuity-toolkit-rel-5.0.0/bin/v4$ ls nbg_unify/ -l
total 69040
-rw-r--r-- 1 root root      577 6月  26 16:57 BUILD
-rw-r--r-- 1 root root    12691 6月  26 16:57 inceptionv4.vcxproj
-rw-r--r-- 1 root root     5802 6月  26 16:57 main.c
-rw-r--r-- 1 root root     2000 6月  26 16:57 makefile.linux
-rw-r--r-- 1 root root 70610560 6月  26 16:57 network_binary.nb
-rw-r--r-- 1 root root      357 6月  26 16:57 vnn_global.h
-rw-r--r-- 1 root root     6861 6月  26 16:57 vnn_inceptionv4.c
-rw-r--r-- 1 root root      908 6月  26 16:57 vnn_inceptionv4.h
-rw-r--r-- 1 root root     3565 6月  26 16:57 vnn_post_process.c
-rw-r--r-- 1 root root      395 6月  26 16:57 vnn_post_process.h
-rw-r--r-- 1 root root    20534 6月  26 16:57 vnn_pre_process.c
-rw-r--r-- 1 root root     1225 6月  26 16:57 vnn_pre_process.h
```

Note

1. *Currently,* ***--optimize*** *is set to* ***VIPNANOQI_PID0X88****. Its value range is as follows:*

   VIPNANOQI_PID0X7D, VIPNANOQI_PID0X88

   VIPNANOQI_PID0X99, VIPNANOQI_PID0XA1

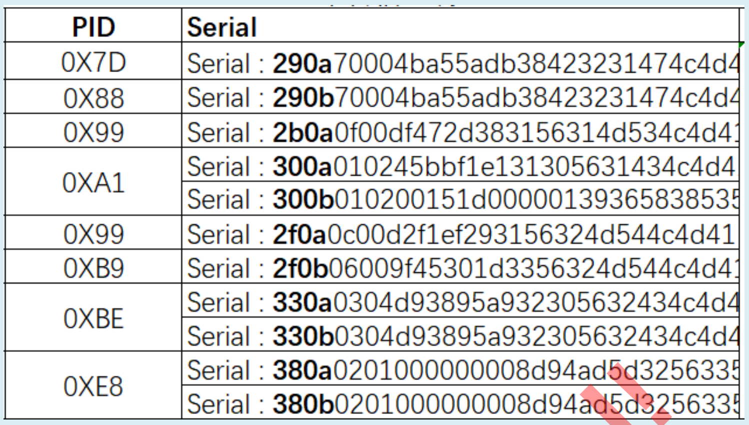
   VIPNANOQI_PID0XB9, VIPNANOQI_PID0XBE

   VIPNANOQI_PID0XE8, VIPNANOQI_PID0X1E

   You can use the following command to confirm the values that should be set, *Execute: cat /proc/cpuinfo*

*Check **the first four characters** of the serial code corresponding to Serial in the penultimate line (**the part in bold after Serial**)*

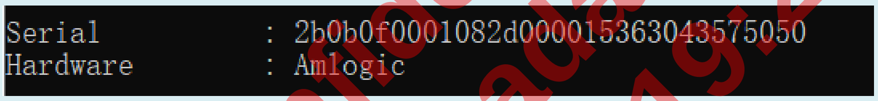
The mapping between Serial parameter `VIPNANOQI_PIDOX? ?` *and the corresponding relationship are shown in the figure below.*



PID	Serial
0X7D	Serial : **290a**70004ba55adb38423231474c4d4
0X88	Serial : **290b**70004ba55adb38423231474c4d4
0X99	Serial : **2b0a**0f00df472d383156314d534c4d41
0XA1	Serial : **300a**010245bbf1e131305631434c4d4
	Serial : **300b**010200151d00000139365838535
0X99	Serial : **2f0a**0c00d2f1ef293156324d544c4d41
0XB9	Serial : **2f0b**06009f45301d3356324d544c4d41
0XBE	Serial : **330a**0304d93895a932305632434c4d4
	Serial : **330b**0304d93895a932305632434c4d4
0XE8	Serial : **380a**0201000000008d94ad5d3256335
	Serial : **380b**0201000000008d94ad5d3256335

For example:

After executing the command cat / proc / cpuinfo on the serial port or the CMD window, the serial is shown in the figure below.



```
Serial          : 2b0b0f0001082d000015363043575050
Hardware        : Amlogic
```

At this point, the parameter should be set to: `--optimize VIPNANOQI_PIDOX99`

2. ***--viv-sdk** depends on the sdk package, which is stored in the* `acuity_tool_xxx/`***bin\vcmdtools** directory. You can enter its relative path.*
3. ***--pack-nbg-unify** is used to generate a nbg file.*
4. *After the parameters described in 1, 2, and 3 are set, case demo code for the nbg-type model is generated. We usually use the nbg case demo. At this time, the normal case demo will also be generated in the model directory. Execute the following two lines of commands to organize the normal case into one directory:*

```
mkdir normal_case_demo
mv   *.h *.c .project .cproject *.vcxproj *.lib BUILD *.linux *.export.data
normal_case_demo
```

The difference between the two commands:

***Normal_case**: When loading the model, online compilation may takes a long time.*
***Android platform**: supports running normal case directly*
***Linux platform**: does not *support running normal case directly*. If Linux platform is used, 1.you need to push acuity_tool_xx/bin/vcmdtoos directory to the board data directory, and then set the environment variable:export VIVANTE_SDK_DIR=/data/vcmdtools. 2.You also need to push the path buildroot_sdk/build/so/drivers_64_exportdata/ corresponding so file in the buildroot_sdk_64xx directory to the board, and then set the environment variable:export LD_LIBRARY_PATH=/xxx/xxx/drivers_64_exportdata

***NBG case:** This step of on-line compilation has been completed on the PC. Nb file can be loaded directly on the board, and the model loading speed is fast.*

5. After the nbg-type model is successfully transcoded, there will be a **network_binary.nb** file in the case code directory.

6. Demo is a model conversion directory, and demo_nbg_unify will be generated in the same level directory. This is the nbg case demo directory.

Introduction to the generated case code

The generated code is stored in the **nbg_unify** directory. Details are as follows:

1. **network_binary.nb** is the generated nbg file, which stores the weight and graph of the model. You can change the file name as needed.
2. **vnn_inceptionv4.c** is the code file used for model creation and release.
3. **vnn_pre_process.c** is the pre-processing model file, where read images are quantized to 8-bit data. Data about interfaces can be duplicated, or the model can be quantized itself.
4. **vnn_post_process.c** is the post-processing model file. The current transcoded case code only experiences top5 processing, and you can add post-processing to the model as needed.

Note

1. *Check the logs for Step 3 to confirm which files are generated. For example, a* ***mbox_priorbox_185.bin*** *file is generated for caffe ssd networks.*
2. *The demo for model transcoding of demo is provided in the tool package.*
   The execution sequence for the scripts is as follows:
   0_import_model.sh ->
   1_quantize_model.sh ->
   2_export_case_code.sh.
3. *After the above 3 steps excuted, you can transcode the mobilenet_v1 model in the demo to case code.*
4. *For the models that are transcoded themselves, place the models in the directory and modify the parameters in the sh scripts.*
5. *With extractoutput.py in Demo, execute the command: python extractoutput.py xxx.json to generate outnamelist.txt, which records mapping between* ***Blob name*** *and* ***tensor ID****. You can use this as references when obtaining transcoding results.*

Step 4 Inference on the PC side

```
./bin/pegasus inference \
                    --dtype quantized   --model xxx.json \
                    --model-data xxx.data    --with-input-meta xxxx_inputmeta.yml \
```

Note

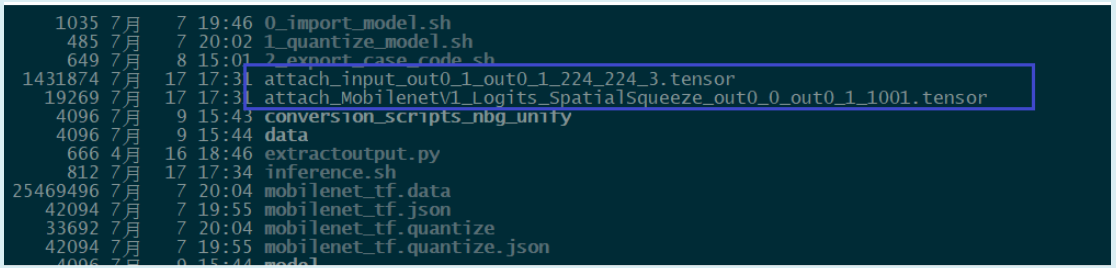
1. *--dtype: sets the PC-side inference method, with parameters* **quantized**, **float32**

2. *The image used in Inference is from the source file used for quantification set in xxx_imputmeta.xml. Generally, only one image is simulated for inference, if multiple images are used for quantification, please modify the corresponding sourefile in the yml file when inferring*

3. *After execution of inference, the tensor file corresponding to input and output will be saved in the current directory. The input file is the float data after preprocessing. The output file is the data hat has been inversed quantization to float. In general, it is recommended to use the tensor file saved by inference as the input file when you confirm the accuracy problem of board running time.*

For example:

After executing the inference of the Mobilenet model, the tensor file of input and output is saved. You can use the input corresponding to tensor file `attach_input_out0_1_out0_1_224_224_3.tensor` *as input file of demo execution on board.*



```
    1035 7月    7 19:46 0_import_model.sh
     485 7月    7 20:02 1_quantize_model.sh
     649 7月    8 15:01 2_export_case_code.sh
 1431874 7月   17 17:31 attach_input_out0_1_out0_1_224_224_3.tensor
   19269 7月   17 17:31 attach_MobilenetV1_Logits_SpatialSqueeze_out0_0_out0_1_1001.tensor
    4096 7月    9 15:43 conversion_scripts_nbg_unify
    4096 7月    9 15:44 data
     666 4月   16 18:46 extractoutput.py
     812 7月   17 17:34 inference.sh
25469496 7月    7 20:04 mobilenet_tf.data
   42094 7月    7 19:55 mobilenet_tf.json
   33692 7月    7 20:04 mobilenet_tf.quantize
   42094 7月    7 19:55 mobilenet_tf.quantize.json
    4096 7月    9 15:44 model
```

Step 5 Quantitative model

For quantitative models (e.g. tflite quantitative models), the acuity tool also supports conversions. The current execution of step1/step3 will complete the conversion, which may require the use of --mean-values, --std-values, see section 3.6 on Pegasus parameters, in which the quantization parameters that come with the quantization model itself are directly used. The xxx_imputmeta.xml generated in step1 has no effect on the quantization parameters, but is only needed to generate the nbg case in step3. It is also possible to execute step1/step2/step3 sequentially, which will re-quantize the model and the quantization parameters of the quantization model itself will not be used.

3.5 Hybrid Quantization

Hybrid quantization is to use different quantization methods for different layers of the same model according to accuracy. For example, the 8 bit quantization effect is poor, and resulting in a large error in the finale result of the model, we can use a hybrid quantization method to set the corresponding layer to a different quantization method to improve accuracy.

3.5.1 Process Summary

1、Use the **--rebuild** and **--compute-entropy** arguments to quantize the network(.data and .json files) with a support quantization type. This generates a .quantize file and an entropy.txt files. In the xxx.quantize file, the **customized_quantize_layers** section provides suggested layer for a futher quantization with the **dynamic_fixed_point-i16** quantization type.

2、Update the customized_quantize_layers section to add, modify, or remove layers for a further quantization..To determine new quantization types for these layers, reference the entropies in entropy.txt. For high entropy layers, you can use the float32 type to improve the precision.. Note: he supported data types in customized_quantize_layers are dynamic_fixed_point-i16 and float32.

3、Use --hybrid and --model-quantize arguments with the updated .quantize file to quantize the network with the same quantization type as in step1.This performs a hybrid quantization and generates .quantize .json and .quantize files.

4、Export such hybrid-quantized model with the .data and the generated .quantize .json files. In the exported model, the dbype_converter layers are inserted.

3.5.2 Operation Procedure

Use lenet model as an example to show how to change a layer from quantized dtype to a float dtype:

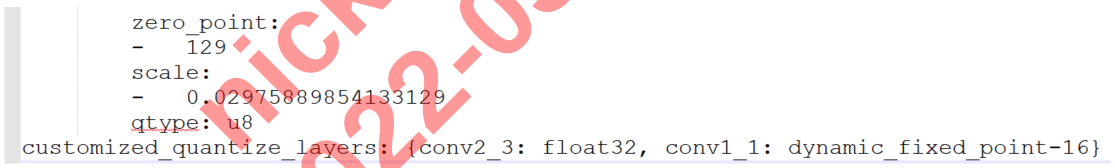
1、Use --rebuild and --compute-entropy to quantize an ACUITY model called lenet to generate a lenet.quantize file as flows:

```
$ ./bin/pegasus quantize \
        --quantizer asymmetric_affine \
        --qtype uint8 \
        --rebuild \
        --with-input-meta   lenet_inputmeta.yml \
        --model   lenet.json \
        --model-data   lenet.data
        --compute-entropy
```

2、Add layer name ‘conv2_3’ and corresponding quantized_dtype ‘float32’ to **customized_quantize_layers** of lenet.quantize.

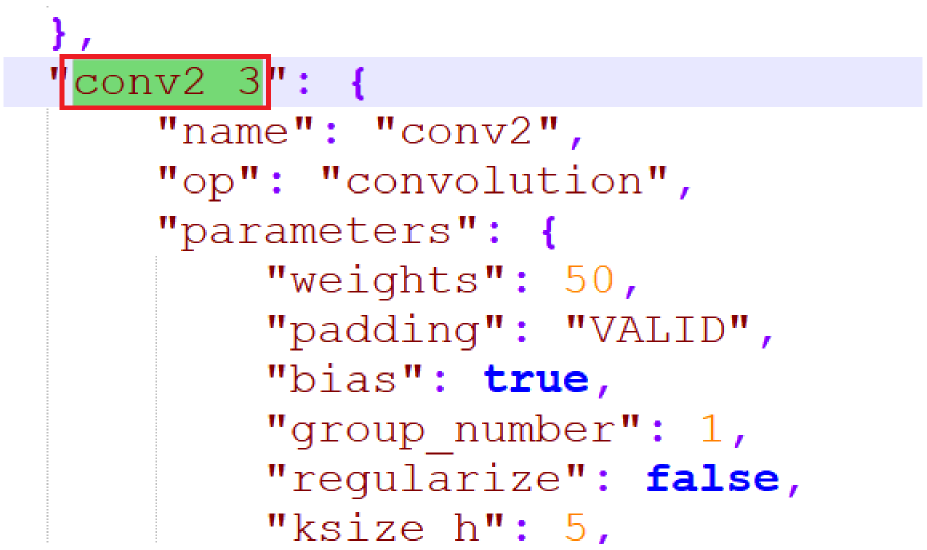
```
customized_quantize_layers: {conv2_3: float32}

Or customized_quantize_layers: {conv2_3: float32, conv1_1: dynamic_fixed_point-16}

CAUTION: if you want to change a connected subgraph of the network to non-quantized layers,
set all the layers in this subgraph to ‘float32’, and you can get the layer name from the *.json file.
```

For example:



```
        zero_point:
        -   129
        scale:
        -   0.02975889854133129
        qtype: u8
customized_quantize_layers: {conv2_3: float32, conv1_1: dynamic_fixed_point-16}
```

The layer name from xxx.json file , Specific as shown below:



```
},
"conv2_3": {
    "name": "conv2",
    "op": "convolution",
    "parameters": {
        "weights": 50,
        "padding": "VALID",
        "bias": true,
        "group_number": 1,
        "regularize": false,
        "ksize_h": 5,
```

3、 Use the --hybird argument and the updated lenet.quantize file to quantize the network into the asymmetric_affine-u8 data type again as follows:

```
$   ./bin/pegasus quantize \
        --model 'lenet.json' \
        --model-data 'lenet.data' \
        --model-quantize 'lenet.quantize' \
        --quantizer 'asymmetric_affine' \
        --qtype 'uint8' \
        --with-input-meta 'lenet_inputmeta.yml' \
        --hybrid
```

This generates hybrid network files lenet.quantize.json and lenet.quantize. The lenet.quantize.json file contains the newly added dtype_converter layers.

4、Use the lenet.data, lenet.quantize.json, lenet.quantize files generated by Step 3 to export application code (refer to Section 0) which wil result in some DATACONVERT layers inserted into the graph in the exported case.

```
$ ./bin/pegasus     export ovxlib
        --model lenet.quantize.json \
        --model-data lenet.data \
        --model-quantize lenet.quantize \
        --with-input-meta ${NAME}_inputmeta.yml
        --dtype quantized \
        --optimize VIPNANOQI_PID0X88    \
        --viv-sdk ../bin/vcmdtools \
        --pack-nbg-unify
```

Note: --optimize VIPNANOQI_PID0X88 Refer to step3 in chapter 3.4

3.6 PASUS extension parameters

3.6.1 Import Caffe

```
./bin/pegasus import caffe
    [-h] --model MODEL [--weights WEIGHTS]
    [--proto PROTO] [--output-model OUTPUT_MODEL]
    [--output-data OUTPUT_DATA]
```

Argument	Description
--model(required)	Requires a string value to denote the file path of the imported Caffe model, a .prototxt file
--weights	Requires a string value to denote the file path of the imported weights file, a .caffemodel file. If the weights file does not exist, the system generates a .data file which contains fake data
--output-model	Requires a string value to denote the file path of the generated ACUITY network model json file, When this argument is omitted, the system uses the default file path
--output-data	Requires a string value to denote the file path of the generated ACUITY network coefficient data data file, When this argument is omitted, the system uses the default file path
--proto	Requires a string value to represent the protocol used by the imported Caffe model. • 'caffe': (Default) Represents the standard Caffe format protocol. • 'lstm_caffe': Represents the LSTM layer protocol.

3.6.2 Import Tensorflow

```
./bin/pegasus import tensorflow
    [-h] --model MODEL
    --inputs INPUTS
    --input-size-list INPUT_SIZE_LIST

    -–outputs OUTPUTS

    [--size-with-batch SIZE_WITH_BATCH]
    [--mean-values MEAN_VALUES]
    [--std-values STD_VALUES]
    [--predef-file PREDEF_FILE]
    [--subgraphs SUBGRAPHS]
```

```
        [--output-model OUTPUT_MODEL]
        [--output-data OUTPUT_DATA]
```

--model(required)	Requires a string value to denote the file path of the TensorFlow frozen protocol buffer (protobuf) file, a `.pb` file. **Note:** Models trained and frozen by the following TensorFlow versions have been proven compatible with ACUITY: 1.4.x, 2.0.x, and 2.3.x.
--inputs (required)	Multiple points are separated with spaces. For example, 'input_point_1 input_point_2'.
--input-size-list(required)	Tensor shape sizes of the same input point are separated with commas. For example, '3,224,224', which represents the tensor shape sizes of one input point. Sizes between different input points are separated with hashtags. For example, '3,224,224#3,299,299#12,1', tensor shape sizes of three input points.
--outputs (required)	Multiple points are separated with spaces. For example, 'output_1 output_2'.
--outputmodel	a string value to denote the file path of the generated ACUITY network model file
--output-data	a string value to denote the file path of the generated ACUITY network data file
--size-with-batch	Requires one of the following values to specify whether the sizes listed in the **--input-size-list** argument contain batch dimension sizes of the input tensors. The batch dimension is the first dimension of an input tensor. • **True**: Contains. • **False**: Does not contain. **Note: True** or **False** for different input points are separated with commas and enclosed by ''. For example, 'True,False,True'. When this argument is omitted, the system uses **False** for all input points listed. Example 1: The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the **--input-size-list** argument are '243,243,3', set **--size-with-batch** to '**False**'. Example 2: Three input ports with only one input point at each port. The tensor shape sizes of each input point are (?, 243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the **--input-size-list** argument are '243,243,3#88#10,7', set **--size-with-batch** to 'False,False,True'.

--mean-values	Requires a string to specify the mean value of each input point listed in the **--inputs** argument. Multiple mean values are separated with commas. For example, '128.0,128.0'. **Note:** Specify this argument only for quantized TensorFlow models.
--std-values	Requires a string to specify the standard value of each input point listed in the **--inputs** argument. Multiple standard values are separated with commas. For example, '128.0,128.0'. **Note:** Specify this argument only for quantized TensorFlow models.
--predef-file	Requires a string value to denote the file path of a predef file, an `.npz` file. Specify this argument to import complex models and enable custom control logics. To generate a predef file, you can use the NumPy function `np.savez(\'prd.npz', <path name>=<predefined value>)` where <path name> refers to the name of a network path for a specific compute stage. For example, `np.savez(\'prd.npz', train_stage=False)`. If a placeholder name contains unsupported characters, map the placeholder name to a supported alias. For example, NumPy does not support forward slashes. If the placeholder name is 'inference/train_stage', then use the following function to generate the predef file: `np.savez(\'prd.npz', stage=False, map={'stage':'inference/train_stage'})`. For more information about the `np.savez()` function, visit *NumPy documentation*.
--subgraphs	Requires a string to list the input points and output points of subgraphs. Specify this argument to import complex models. Use the following value syntax: • Point lists between different subgraphs are separated with semicolons. • For each subgraph, the input point list is followed by the output point list. The lists are separated with a hashtag. • Multiple points in each list are separated with commas. For example, 'graph1in1,graph1in2#graph1out1,graph1out2;graph2in1#graph2out1'.

3.6.3 Import Tflite

```
./bin/pegasus import tflite
    [-h]   --model MODEL
    [--inputs INPUTS]
    [--input-size-list INPUT_SIZE_LIST]
    [--size-with-batch SIZE_WITH_BATCH]
    [--output-model OUTPUT_MODEL]
    [--output-data OUTPUT_DATA]
```

3.6.4 Import Darknet

```
./bin/pegasus import darknet
    [-h] --model MODEL --weights WEIGHTS
    [--output-model OUTPUT_MODEL]
    [--output-data OUTPUT_DATA]
```

--model(required)	a string value to denote the file path of the imported Darknet model, a `.cfg` file.
--weights	a string value to denote the file path of the imported weights file, a `.weights` file
--output-model	a string value to denote the file path of the generated ACUITY network model file
--output-data	a string value to denote the file path of the generated ACUITY network data file

3.6.5 Import Onnx

```
./bin/pegasus import onnx
    [-h] --model MODEL [--inputs INPUTS] [--outputs OUTPUTS]
    [--input-size-list INPUT_SIZE_LIST] [--size-with-batch SIZE_WITH_BATCH]
    [--input-dtype-list INPUT_DTYPE_LIST]
    [--output-model OUTPUT_MODEL]
    [--output-data OUTPUT_DATA]
```

--model(required)	a string value to denote the file path of the imported ONNX model, an `.onnx` file **Note:** ONNX operator sets 1 through 11 are supported.

--inputs	Requires a string to list the input points of the ONNX model. Multiple points are separated with spaces. For example, 'input_1 input_2'. If this argument is omitted, the system uses all head points of the imported model.
--input-size-list	Requires a string to represent the tensor shape sizes of the input points listed in the --inputs argument. • Tensor shape sizes of the same input point are separated with commas. For example, '3,224,224', which represents the tensor shape sizes of one input point. • Sizes between different input points are separated with hashtags. For example, '3,224,224#3,299,299#12,1', which represents the tensor shape sizes of three input points. When this argument is omitted, the system automatically detects the tensor shape sizes of the input points.
--size-with-batch	Requires one of the following values to specify whether the sizes listed in the **--input-size-list** argument contain batch dimension sizes of the input tensors. The batch dimension is the first dimension of an input tensor. • **True**: Contains. • **False**: Does not contain. **Note: True** or **False** for different input points are separated with commas and enclosed by ''. For example, 'True,False,True'. When this argument is omitted, the system uses **False** for all input points listed. Example 1: The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the **--input-size-list** argument are '243,243,3', set **--size-with-batch** to '**False**'. Example 2: Three input ports with only one input point at each port. The tensor shape sizes of each input point are (?, 243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the **--input-size-list** argument are '243,243,3#88#10,7', set **--size-with-batch** to 'False,False,True'.
--input-dtype-list	Requires a string to denote data types of the input tensors on the input points. The allowed strings for the supported data types are '**float**', '**int8**', '**uint8**', '**int16**', and '**uint16**'. String values for the data types of different input tensors are separated with hashtags. For example, 'float#int8#uint16'. When this argument is omitted, the system uses the default value **'float'** for all input tensors.

--output-model	a string value to denote the file path of the generated ACUITY network model file
--output-data	a string value to denote the file path of the generated ACUITY network data file
--outputs	Requires a string to specify the output points of the ONNX model. Multiple points are separated with spaces. For example, 'output_1 output_2'. When this argument is omitted, the system uses all tail points of this model.

3.6.4 Import Pytorch

```
./bin/pegasus import pytorch
        [-h] [--model MODEL]
        [--inputs INPUTS]
        [--outputs OUTPUTS]
        [--input-size-list INPUT_SIZE_LIST]
        [--size-with-batch SIZE_WITH_BATCH]
        [--output-model OUTPUT_MODEL]
        [--output-data OUTPUT_DATA]
        [--config CONFIG]
```

--model(required)	a string value to denote the file path of the imported PyTorch model, a `.pt` file.
--inputs	Requires a string to list the input points of the PyTorch model. Multiple points are separated with space. For example, 'input_1 input_2'. If this argument is omitted, the system uses all head points of the imported model.
--outputs	Multiple points are separated with spaces. For example, 'output_1 output_2'.
--input-size-list	Requires a string to represent the tensor shape sizes of the input points listed in the **--inputs** argument. • Tensor shape sizes of the same input point are separated with commas. For example, '3,224,224', which represents the tensor shape sizes of one input point.

	• Sizes between different input points are separated with hashtags. For example, '3,224,224#3,299,299#12,1', which represents the tensor shape sizes of three input points. When this argument is omitted, the system automatically detects the tensor shape sizes of the input points.
--size-with-batch	Requires one of the following values to specify whether the sizes listed in the **--input-size-list** argument contain batch dimension sizes of the input tensors. The batch dimension is the first dimension of an input tensor. • **True**: Contains. • **False**: Does not contain. **Note: True** or **False** for different input points are separated with commas and enclosed by ''. For example, 'True,False,True'. When this argument is omitted, the system uses **False** for all input points listed. Example 1: The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the **--input-size-list** argument are '243,243,3', set **--size-with-batch** to '**False**'. Example 2: Three input ports with one input point at each port. The tensor shape sizes of each input point are (?,243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the **--input-size-list** argument are '243,243,3#88#10,7', set **--size-with-batch** to 'False,False,True'.
--output-model	a string value to denote the file path of the generated ACUITY network model file
--output-data	a string value to denote the file path of the generated ACUITY network data file
-config	Requires a string value to denote the file path of the configuration file, a `.json` file. This is an alternative method to translate a PyTorch model to ACUITY formats. With this argument, you do not need to use any other configuration arguments listed above.

3.6.7 Import Keras

```
    ./bin/pegasus import keras
        [-h] --model MODEL
        [--convert-engine {Keras,TFLite}]
        [--inputs INPUTS]
        [--input-size-list INPUT_SIZE_LIST]
        [--outputs OUTPUTS]
        [--output-model OUTPUT_MODEL]
        [--output-data OUTPUT_DATA]
```

--model(required)	a string value to denote the file path of the imported Keras model, a `.h5` file
--convert-engine	Reserved for future use.
--inputs	Multiple input points are separated with spaces. For example, 'input_1 nput_2'.
--input-size-list	Requires a string to represent the tensor shape sizes of the input points. These sizes must not contain batch sizes, which are the first dimensions of the input tensor shapes. • Tensor shape sizes of the same input point are separated with commas. For example, '3,224,224', which represents the tensor shape sizes of one input point. • Sizes between different input points are separated with hashtags. For example, '3,224,224#3,299,299#12,1', which represents the tensor shape sizes of three input points. When this argument is omitted, the system automatically detects the tensor shape sizes of the input points
--outputs	Multiple output points are separated with spaces. For example, 'output_1 output_2'.
--output-model	a string value to denote the file path of the generated ACUITY network model file
--output-data	a string value to denote the file path of the generated ACUITY network data file

3.6.8 Generate Inputmeta

```
    ./bin/pegasus generate inputmeta
        [-h] [--input-meta-output INPUT_META_OUTPUT]
        [--separated-database SEPARATED-DATABASE]
        --model MODEL --source-file SOURCE_FILE --channel-mean-value
```

--model(required)	a string value to denote the file path of an ACUITY network model file, a `.json` file

--input-meta-output	a string value to denote the file path of the generated inputmeta file. When this argument is omitted, the system uses the default file path .
--separated-database	separate the database into multiple ones in the generated inputmeta file for a multi input ACUITY network. This argument is valid only for multi-input networks. When this argument is added,separate the database into multiple ones
--source-file	a string value to denote the data file for the inputs.if you separate the multi-input, you can set "dataset1.txt#dataset2.txt#dataset3.txt".you can also modify the yml file directly
--channel-mean-value	input channel mean value.if you separate the multi-input, you can set "128 128 128 0.00715#128 128 128 0.00715#128 128 128 0.00715".you can also modify the yml file directly

3.6.9 Quantize

```
    ./bin/pegasus quantize
            [-h] --model MODEL
            --model-data MODEL_DATA
            [--model-quantize MODEL_QUANTIZE]
            [--batch-size BATCH_SIZE] [--iterations ITERATIONS]
            [--device DEVICE] --with-input-meta WITH_INPUT_META
            [--quantizer QUANTIZER] [--qtype QTYPE]
            [--hybrid] [--rebuild] [--rebuild-all]
            [--compute-entropy] [--algorithm ALGORITHM]
            [--moving-average-weight MOVING_AVERAGE_WEIGHT]
            [--divergence-nbins DIVERGENCE_NBINS]
            [--divergence-first-quantize-bits DIVERGENCE_FIRST_QUANTIZE_BITS]
            [--MLE]
```

--model(required)	a string value to denote the file path of an ACUITY network model file, a `.json` file
--model-data(required)	a string value to denote the file path of an ACUITY model coefficient data file

--model-quantize	Requires a string value to denote the file path of the generated `.quantize` file when the **--rebuild** argument is specified, or the file path of the `.quantize` file to use when the **--hybrid** argument is specified. If this argument is omitted when **–rebuild** is specified, the `.quantize` file is generated in the same directory as the model file.
--batch-size	Requires an integer to specify the number of sample images per batch. • If the original network uses a fixed batch size, use the fixed batch size. • If the original network uses a variable batch size, set this argument to 1. When this argument is omitted, the system uses the value of **shape[0]** in the inputmeta file. **Note:** This argument is used together with **--iterations**.
--iterations	Requires an integer to specify the number of sample image batches. For KL divergence, moving-average, and automatic hybrid quantization algorithms, 500 to 1000 iterations are recommended. When this argument is omitted, the system uses the default value 1. **Note:** This argument is used together with **--batch-size**
--device	Requires one of the following values to specify the compute device type. • '**GPU**' • '**CPU**': Default When this argument is omitted, the system uses the default device type.
--with-input-meta(Required)	Requires a string value to denote the file path of the inputmeta file, a `.yml` file
--quantizer	Requires one of the following values to specify the quantizer for quantizing network tensors: • '**asymmetric_affine**': Default • '**dynamic_fixed_point**' • '**perchannel_symmetric_affine**' • '**qbfloat16**' When this argument is omitted, the system uses the default quantizer.

	Note: This argument is used together with **--qtype**.
--qtype	Requires one of the following values to specify the quantization data type: • '**int8**' • '**int16**' • '**uint8**': Default • '**qbfloat16**' When this argument is omitted, the system uses the default data type. **Note**: This argument is used together with **--quantizer**.
--hybrid	Performs hybrid quantization on the network model. This argument is mutually exclusive with the **–rebuild** and **--rebuild-all** arguments.
--rebuild	Performs quantization with the quantization file generated. To quantize the network with a quantizer other than **qbfloat16**, specify this argument. For details about quantizer types, see the description of --quantizer. This argument is mutually exclusive with the **--hybrid** and **--rebuild-all** arguments.
--rebuild-all	Rebuilds a quantization table with the default quantization rules and quantizes all activations. To quantize the network with the **qbfloat16** quantizer, specify this argument. For details about quantizer types, see the description of --quantizer. This argument is mutually exclusive with the **--hybrid** and **--rebuild** arguments.
--algorithm	Requires one of the following values to specify the quantization algorithm: • '**normal**': The default algorithm. • '**kl_divergence**': The KL divergence algorithm. If this value is chosen, specify either **--divergence-nbins** or **--divergence-first-quantize-bits**. • '**moving_average**': The moving-average algorithm. If this value is chosen, specify the **--moving-average-weight** argument. • '**auto**': The KL-divergence-based hybrid quantization algorithm with quantized layers automatically determined. ▪ If **--MLE** is specified, the quantized layers are determined by the Cosine Similarity measure. In this case, the automatic hybrid

	quantization algorithm and the MLE function together ensure a high quantization precision. ▪ If **--MLE** is not specified, the quantized layers are determined based on layer entropy. When this argument is omitted, the system uses the default algorithm.
--moving-average-weight	Requires a positive floating-point value to specify the coefficient for the moving average model. This argument is valid only if the **--algorithm** argument is set to '**moving_average**'. When this argument is omitted, the system uses the default coefficient 0.01.
--divergence-nbins	Requires an integer to specify the number of bins in the Kullback-Laiber divergence (KL divergence) histogram. The integer must equal 2_N where N is a positive integer. Specify this argument only if **--algorithm** is set to '**kl_divergence**'. When this argument is used, do not specify the **--divergence-first-quantize-bits** argument. When this argument is omitted, the system uses the value 2_{11}. **Note: --divergence-nbins** will be deprecated. Use **--divergence-first-quantize-bits** instead.
--divergence-first-quantize-bits	Requires a positive integer to calculate the number of bins in the KL divergence histogram. Given an integer m, the calculated KL bin count is 2_m. When this argument is omitted, the system uses the default value 11. **Note:** Specify this argument only if **--algorithm** is set to '**kl_divergence**'. When this argument is used, do not specify the **--divergence-nbins** argument
--compute-entropy	Measures the precision of the current quantization by calculating the entropy of each layer in the range of 0 to 1. The system stores these values in the `entropy.txt` file in the current workspace. If the entropy is low, the precision is high. If the entropy is high, the precision is low. When this argument is omitted, the system does not perform such measurement. **Note:** This argument does not require values. It is valid only if the **--batch-size** argument is set to 1.
--MLE	Minimizes per-layer quantization errors of the network by automatically adjusting quantization parameters. This function can be applied to all the supported quantization algorithms. Among them, automatic hybrid

	quantization together with the MLE function ensures a high quantization precision. **Note:** If this argument is specified, the quantization process may be time-comsuming

3.6.10 Export Ovxlib

```
./bin/pegasus export ovxlib
        --model   --model-data MODEL_DATA
        [--model-quantize MODEL_QUANTIZE]
        [--postprocess-file POSTPROCESS_FILE]
        [--output-path OUTPUT_PATH]
        --with-input-meta WITH_INPUT_META
        [--optimize OPTIMIZE] [--dtype DTYPE]
        [--save-fused-graph] [--pack-nbg-unify]
        [--pack-nbg-viplite] [--viv-sdk VIV_SDK]
        [--build-platform 'make']
        [--batch-size BATCH_SIZE] [--force-remove-permute]
        [--customer-lids] [--customer-ops]
```

--model(required)	a string value to denote the file path of an ACUITY network model file
--model-data(required)	a string value to denote the file path of an ACUITY model coefficient data file
--with-input-meta(required)	Requires a string value to denote the file path of the inputmeta file, a `.yml` file.
--model-quantize	a string value to denote the file path of a quantized tensor description file
--postprocess-file	a string value to denote the file path of the post-processing configuration file, a `.yml` file. Multiple tasks can be set in one configuration file. You can either create a post-processing file or use the generation command to generate a post-processing file
--output-path	a string value to denote the directory of the exported applications with the prefix specified
--optimize	Requires one of the following values to specify the optimization method during the export. • '**None**': Does not optimize. • '**default**': (Default) Optimizes the model based on the default rules.

	• '<configuration file path or configuration name>': Optimizes the model based on the specified configuration file. Specify a configuration file or a configuration name if the **--pack-nbg-unify** or **--pack-nbg-viplite** argument is specified
--dtype	Requires one of the following values to specify the tensor data type of the exported OVXLIB application: • '**float**' or '**float16**': '**float**' and '**float16**' are equivalents. '**float**' is the default data type. • '**float32**' • '**quantized**': A quantization data type specified in the `.quantize` file when the input model is a quantized model. If this value is chosen, specify the **--model-quantize** argument. When this argument is omitted, the system uses the default data type
--save-fused-graph	(Experimental use only) Saves a fused model to a `.json` file for debugging use. It is mutually exclusive with **-- pack-nbg-unify** and **--pack-nbg-viplite** arguments. The generated fused model contains network structures only of the exported unify applications. When this argument is omitted, no fused model is saved. **Note:** This argument does not require values.
--force-remove-permute	(Experimental use only) Removes the head permutation layers inserted after the input layer and the tail permutated layers inserted before the output layer. This argument is used only for export of unify applications from TensorFlow, TensorFlow Lite, and Keras models. It is mutually exclusive with **--pack-nbg-unify** and **--pack-nbg-viplite** arguments. When this argument is omitted, permutation layers are kept and the tensor shape is in the NHWC sequence. When this argument is specified, the tensor shape may be in the NCHW sequence. Make sure that the data sequence of the tensor shape is NHWC before application deployment onto devices with Vivante NPUs. For example, for a TensorFlow model with input of (1,224,224,3) in the NHWC sequence: • If this argument is not specified, the tensor with the shape (1,224,224,3) in the NHWC sequence is used. • If this argument is specified, the tensor shape may be (1,3,224,224) in the NCHW sequence.

	When the data sequence is NCHW, convert it into NHWC before application deployment. **Note:** This argument does not require values
--pack-nbg-unify	Packs binary graphs for the unified driver and generates three applications: • unify application: An `.nb` file in the output directory. • nbg_unify application: An `.nb` file in the unify subdirectory of the output directory. • nbg_unify_ovx application: An `.nb` file in the unify subdirectory of the output directory. **Important:** nbg_unify_ovx applications will be obsoleted at the end of 2021. The output directory is specified by the **--output-path** argument. **Note:** • If neither **--pack-nbg-unify** nor **--pack-nbg-viplite** is specified, only the unify application is generated. • This argument is mutually exclusive with the **--pack-nbg-viplite** argument. • Packing is not supported for edge devices with CPU nodes. If CPU nodes exist, use PPU nodes instead
--pack-nbg-viplite	Packs binary graphs for the VIPLite driver and generates two applications: • unify application: An `.nb` file in the output path. • nbg_unify application: An `.nb` file in the output path. The output directory is specified by the **output_path** argument. **Note:** • If neither **--pack-nbg-unify** nor **--pack-nbg-viplite** is specified, only the unify application is generated. • This argument is mutually exclusive with the **--pack-nbg-unify** argument. • Packages are not supported for edge devices with CPU nodes. If CPU nodes exist, use PPU nodes instead.
--viv-sdk	Requires a string value to denote the file path of the directory that contains the binary SDK of VSim. During execution, VSim generates NBG files. For example, the file path may be '`/home/xxx/Verisilicon/VivanteIDEx.x.x/*cmdtools`' if VivanteIDE is installed. Specify this argument if the **--pack-nbg-unify** or **--pack-nbg-viplite** argument is specified

--build-platform	Builds a compiling tool to generate NBG applications. The available value is '**make**'. When this argument is omitted, the system uses the default value '**make**'.
--batch-size	Requires a positive integer to specify the batch size that the exported application supports. When this argument is omitted, the system uses the value of **shape[0]** in the inputmeta file.
--customer-lids	Reserved for future use.
--customer-ops	Reserved for future use.

3.6.11 Inference

```
    ./bin/pegasus inference
        [-h] --model MODEL --model-data MODEL_DATA
        [--model-quantize MODEL_QUANTIZE]
        [--batch-size BATCH_SIZE] [--iterations ITERATIONS]
        [--device DEVICE] [--with-input-meta WITH_INPUT_META]
        [--dtype DTYPE] [--postprocess POSTPROCESS]
        [--postprocess-file POSTPROCESS_FILE]
        [--output-dir OUTPUT_DIR]
```

--model(required)	a string value to denote the file path of an ACUITY network model file, a `.json` file
--model-data(required)	a string value to denote the file path of an ACUITY model coefficient data file
--model_quantize	a string value to denote the file path of a quantized tensor description file, a `.quantize` file. **Note:** Specify this argument if the **--dtype** argument is set to '**quantized**'.
--batch-size	Requires an integer to specify the number of sample images per batch. When this argument is omitted, the system uses the value of **shape[0]** in the inputmeta file. Set this argument to a small value if the RAM or GPU does not support a large batch size. **Note:** This argument is used together with **--iterations**
--iterations	Requires an integer to specify the number of sample image batches.

	When this argument is omitted, the system uses the default value 1. **Note:** This argument is used tog
--devices	Requires one of the following values to specify the compute device type: • '**GPU**' • '**CPU**': Default When this argument is omitted, the system uses the default device type.
--with-input-meta(required)	Requires a string value to denote the file path of the inputmeta file, a `.yml` file.
--dtype	Requires one of the following values to specify the tensor data type of the input model: • '**float32**': The default data type. • '**quantized**': A quantization data type specified in the `.quantize` file when the input model is a quantized model. If this value is chosen, specify the **--model-quantize** argument. When this argument is omitted, the system uses the default data type
--postprocess	Requires one of the following values to specify the post-processing task: • '**print_topn**': Prints the top 5 output tensors of the output layers. • '**dump_result**': Dumps output tensors for input layers and output layers. • '**classification_classic**': (Default) Performs both of the actions. When this argument is omitted, the system uses the default value. **Note:** This argument is mutually exclusive with the **--postprocess-file** argument
--postprocess-file	Requires a string value to denote the file path of the post-processing configuration file, a `.yml` file. Multiple tasks can be set in one configuration file. You can either create a post-processing file or use the generation command to generate a post-processing file. **Note:** This argument is mutually exclusive with the **--postprocess** argument
--output-dir	Requires a string value to denote the directory for the generated files.