NETINT AI Toolkit

User Guide V4.5

08/2023

NETINT Technologies Inc.

Legal Notices

Information in this document is provided in connection with NETINT products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in NETINT's terms and conditions of sale for such products, NETINT assumes no liability whatsoever and NETINT disclaims any express or implied warranty, relating to sale and/or use of NETINT products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right.

A "Mission Critical Application" is any application in which failure of the NETINT Product could result, directly or indirectly, in personal injury or death. Should you purchase or use NETINT's products for any such mission critical application, you shall indemnify and hold NETINT and its subsidiaries, subcontractors and affiliates, and the directors, officers, and employees of each, harmless against all claims costs, damages, and expenses and reasonable attorney's fees arising out of, directly or indirectly, any claim of product liability, personal injury, or death arising in any way out of such mission critical application, whether or not NETINT or its subcontractor was negligent in the design, manufacture, or warning of the NETINT product or any of its parts.

NETINT may make changes to specifications, technical documentation, and product descriptions at any time, without notice. The information here is subject to change without notice. Do not finalize a design with this information. The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications.

NETINT™, Codensity™, Quadra™ and the NETINT Logo are trademarks of NETINT Technologies Inc. All other trademarks or registered trademarks are the property of their respective owners.

Conventions

- Hexadecimal numbers are indicated by the prefix "0x" —for example, 0x32CF.
- Binary numbers are indicated by the prefix "0b" —for example, 0b0011.0010.1100.1111
- Code snippets and file paths are given in `Consolas` font.
- Replaceable terms in directory and file paths are surrounded with <>.

  Example: `<NETINT_toolkit>.tar.gz`
- Replaceable argument values in command lines use italics all caps.

  Example: ***SEPARATED_DATABASE***
- In command lines:
  - **[x]** denotes optional items.
  - **{a, b, c}** denotes only one of the options must be chosen.

1. Introduction

The NETINT AI Toolkit offers an Artificial Intelligence hardware development environment for you to translate, quantize, inference and export deep learning models onto Quadra AI Inference Engine. The NETINT AI Toolkit includes several required tools, such as an IDE, NBInfo Parser Tool and the aiperf tool etc. This document provides a specific user guide to the types of tools included in the Toolkit. Additional context is provided on how they can be used or adapted, and the use cases for each.

The following chapters describe the NETINT AI Development Toolkit working principle, features, and the provided tools in detail.

1.1. NETINT AI Toolkit Overview

The NETINT AI Toolkit provides command line tools for you to translate, quantize, inference and export deep learning models onto devices with Quadra AI Inference Engine.

For example:

```
./convert.sh -c
```

Above command will convert user model into Toolkit internal format.

1.2. IDE Overview

The NETINT AI Development Toolkit provides an IDE which is an advanced Integrated Development Environment for VIP software development. The IDE includes a set of features for compiling, debugging, and profiling your OpenVX projects in a customized, automated manner.

The IDE provides many powerful features for every stage of OpenVX project development, and enables programmers to efficiently accomplish editing, building, and debugging a project with a NETINT simulator. The installation guide can be found in 2. Setup the NETINT AI Development Environment.

1.3. NBInfo Parse Tool Overview

The NBInfo is a parse tool for inspecting network binary graph. It can interpret NBG files to display detailed information, including input, output, and all the layers and operations. This tool is packed in the NETINT AI Toolkit package. The installation guide can be found in 2. Setup the NETINT AI Development Environment.

1.4. Working Principle

The NETINT AI Toolkit is a powerful and flexible platform to translate deep learning models to the unified formats through the following steps:

1. Use the provided NETINT tools to create a workspace containing all scripts, and necessary sub-folders. Refer to 3.1. Setup workspace.
2. Setup the generated shell script, and then use the modified script to import and translate NN models into unified formats. Refer to 3.3. Model Import and Format Translation.
3. Evaluate the imported model. Refer to 3.4. Inference with the Imported Model.
4. Quantize and optimize the model. Refer to 3.5 Model Optimization and Quantization.
5. Evaluate the quantized model. Refer to 3.6. Inference with the Quantized Model.
6. Export the OpenVX model into formats to support the Quadra AI Inference Engine deployment. Refer to 3.7. Export the OpenVX Model.
7. Evaluate the OpenVX models. Refer to 3.9. Inference with the OpenVX Model.
8. Post-process the output tensor files. Refer to 3.10. Model Post-Processing.
9. Analysis of the post-processing results.
10. Export the Network Binary Graph file (NBG). Refer to 3.11. Export the Network Binary Graph.
11. Analyze the exported NBG file. Refer to 3.12 Analysis the Network Binary Graph
12. Evaluate the NBG with Quadra. Refer to 3.13. Inference with Network Binary Graph on the chip.

More details of working procedures are presented in the following chapters.

1.5. Features

The NETINT AI Toolkit is built on TensorFlow™ and supports a wide range of neural network framework format conversion.

Tables 1, 2, and 3 list the NETINT tool features for input, Neural Network compute, and output.

Table 1. Input Features

Input support		Description
NN model framework	Caffe	Supports Caffe formats of both standard and non-standard protocols.
	TensorFlow	Supports inference graphs saved by `tf.io.write_graph()` in TensorFlow versions 1.4.x, 2.0.x, 2.3.x, 2.6.x, and 2.8.x.
	TensorFlow Lite	Supports the TFLite schema from TensorFlow 2.8.0 (hash c4b29518884e16f8476ad92c75c3da48140a9eab). Note: TFLite models of other schemas may not be imported successfully for compatibility reasons.
	ONNX	Supports ONNX 1.8.0 (default), 1.10.2 (operator sets 1-15).
	PyTorch	Supports PyTorch 1.5.1.
	Keras	Supports Keras models generated by TensorFlow 2.0.x, 2.3.x, 2.6.x, and 2.8.x.
	Darknet	Supports Darknet models listed on https://pjreddie.com/darknet/.

Table 2. NN Compute Features

NN compute support		Description
Quantization	Data type	Unsigned int8 with the asymmetric affine quantizer Signed int16 with the dynamic fixed point quantizer Signed int8 with the per-channel symmetric affine quantizer Bfloat16 with the bfloat16 quantizer
	Algorithm	Normal Kullback-Laiber divergence Moving average Automatic hybrid quantization
Inference	Data type	Quantized Float32
Host compute engine	CPU	Default use
	GPU	Valid on NVIDIA® GPU cards with CUDA® architectures. For more details, visit https://www.tensorflow.org/install/gpu.

Table 3. Output Features

Output support		Description
Format	OpenVX	File types are: `.c`, `.h`, and `.export.data`. If packing binary graphs is enabled, `.c`, `.h`, and `.nbg` files are generated.
	TensorFlow Lite	File type is `.tflite`.
Data type		Float point of 16 bits Float point of 32 bits Quantized

2. Setup the NETINT AI Development Environment

This chapter describes the toolkit installation, and the usage of the different tools in more detail. Note that you may use a python virtual environment to setup pip and install this toolkit. Such a virtual environment can be used to perform all the features described in this user guide as well.

2.1. System Requirements

The following table lists the basic system requirements to run the NETINT AI Toolkit.

Table 4. Host System Requirements

Resource	Note
CPU	Intel® Core™ i5-6500 CPU @ 3.2 GHz x4 with support of the Intel® Advanced Vector Extensions.
GPU	(Optional) NVIDIA® GPU cards with CUDA® architectures.
RAM	4 GB at least
Disk	160 GB
OS	Ubuntu 20.04 LTS 64-bit with Python 3.8 **Note:** Other Ubuntu versions are not recommended.

2.2. Setup Python

Prerequisites

- The host meets the requirements as described in 2.1 System Requirements.

Procedure

2. Check the system has Python 3 installed and ensure the Python version is at least Python 3.8.X.

```
python3 --version
```

   If the Python version meets the requirements, you can skip next step.

3. Update Python to the recommended version (python3.8). The command is as follows:

```
sudo apt install python3.8.10
```

2.3. Setup pip

Prerequisites

- The host meets the requirements as described in 2.1 System Requirements.

Procedure

1. Install pip.

```
sudo apt install python3-pip
```

2. Check the pip version is 21.0.1.

```
pip3 --version
```

If the pip version meets the requirement, you can skip the next step.

3. Update pip to the specified version.

```
python3 -m pip install --user --upgrade pip==21.0.1
```

2.4. Setup NETINT AI Toolkit

Setup the provided *NETINT_toolkit* package, it contains the necessary tools, such as the *IDE*, *NBInfo Parser Tool* and the *toolkit wheel package*.

Prerequisites

- The host meets the requirements as described in 2.1 System Requirements, 2.2 Setup Python, 2.3 Setup pip.

Procedure

1. Unpack the *<NETINT_toolkit>.tar.gz* package to the home directory. The command is as follows:

   ```
   tar -zxvf <NETINT_toolkit>.tar.gz -C ~
   ```

2. The folder has 9 subfolders and 2 text files: *examples*, *installation_packages*, *NB_Info_v1.2.10*, *toolkit-whl-6.12.7*, *tools, templates, IDE, aiperf, network_wrapper, README.txt, requirements.txt*. The structure of workspace as shown below:

Files & Folders	Description
examples	Example of deep learning models
NB_Info_v1.2.10	App to read network binary graph file info
toolkit-whl-6.12.7	Toolkit wheel files
tools	Deployment scripts and tool files
templates	Templates to test OpenVX c-model and perform post-processing procedure.
IDE	NETINT IDE for building models on Quadra AI Inference Engine
network_wrapper	AI Python Inference API
aiperf	App to evaluate network binary graph performance
installation_packages	CPU and GPU Wheel installation packages.
README.txt	README file describing AI tarball contents
requirements.txt	List of dependencies for NETINT AI Toolkit

3. Install the toolkit wheel package, from the `~/NETINT_toolkit/installation_packages/`, select either a CPU Wheel package or a GPU Wheel package to install.

   - **(Recommend)** To use the host CPU for computing, install the CPU wheel package with the following command:

     ```
     pip3 install ~/NETINT_toolkit/installation_packages/acuity_bin_cpu-*.whl
     ```

   - To use the host GPU for computing, install the GPU wheel package with the following command:

     ```
     pip3 install ~/NETINT_toolkit/installation_packages/acuity_bin_gpu-*.whl
     ```

   The GPU version requires the installation of correct CUDA version for TensorFlow. For the supported GPUs, see 2.1 System Requirements.

4. Install other dependencies with the following command:

   ```
   pip3 install -r ~/NETINT_toolkit/requirements.txt
   ```

5. Install the patch for the IDE with the following command:

   ```
   sudo apt install libcanberra-gtk-module libcanberra-gtk3-module
   ```

6. Build NBInfo tool, from the `~/NETINT_toolkit/NB_Info_v1.2.10/`, with the following command:

   ```
   ./build_nbinfo.sh
   ```

2.5. Setup Environment Variables

This task shows how to set up the environment variables. The NETINT AI Toolkit has the tools to setup the required environment variables.

Prerequisites

- The NETINT AI Toolkit is installed correctly as described in 2.4 Setup NETINT AI Toolkit.

Procedure

1. From the ~/NETINT_toolkit/tools folder, use the *setup_bashrc.py* script to set the required environment variables in the ~/*.bashrc* file. The required environment variables are:

Variables	Description
LD_LIBRARY_PATH	The system LD Library path
ACUITY_PATH	The toolkit binary file path
VIVANTE_BASE	The IDE path
VIVANTE_VIP_HOME	The IDE home path
VIVANTE_SDK_DIR	The IDE software development kit path
NETINT_TOOLS	The NETINT AI Toolkits path
LMX_LICENSE_PATH	The IDE license path
VSIMULATOR_CONFIG	The simulation type of the official AI hardware
NBINFO	The NBinfo Parser Tool path
USE_IDE_LIB	The value to define the IDE LIB
TF_CPP_MIN_LOG_LEVEL	The variable of TensorFlow log level
PIP_DISABLE_PIP_VERSION_CHECK	The variable to avoid pip version check

2. The commands are as follows:

```
./setup_bashrc.py
source ~/.bashrc
```

3. After completing the above, all required environment variables will now be set in the *~/.bashrc* file as follows:

```
export TF_CPP_MIN_LOG_LEVEL=3
export PIP_DISABLE_PIP_VERSION_CHECK=1
export ACUITY_PATH=/home/[user_name]/NETINT_toolkit/toolkit-whl-6.12.7/bin
export VIVANTE_BASE=/home/[user_name]/NETINT_toolkit/VivanteIDE5.7.2
export VIVANTE_VIP_HOME=$VIVANTE_BASE
export VIVANTE_SDK_DIR=${VIVANTE_VIP_HOME}/cmdtools/vsimulator
export NETINT_TOOLS=/home/[user_name]/NETINT_toolkit/tools
export LMX_LICENSE_PATH=/home/[user_name]/NETINT_toolkit/VivanteIDE5.7.2/
       license/[license_name]
export VSIMULATOR_CONFIG=VIP8000NANOQI_PLUS_PID0XB1
export LD_LIBRARY_PATH=${VIVANTE_VIP_HOME}/cmdtools/vsimulator/lib
export USE_IDE_LIB=1
export NBINFO=/home/[user_name]/NETINT_toolkit/NB_Info_v1.2.10
export
       PATH=$PATH:$ACUITY_PATH:${VIVANTE_VIP_HOME}/cmdtools/common/bin:$NETINT_TOOLS:$V
       IVANTE_BASE:${VIVANTE_BASE}/ide:$NBINFO
```

2.6. Verify the Development Environment

This task verifies the required environment variables are correct on machine. The NETINT AI Toolkit has the tools to check the completed installation.

Prerequisites

- The host meets the requirements as described in 2.1 System Requirements.
- The NETINT AI Toolkit is installed correctly as described in 2. Setup the NETINT AI Development Environment.

Procedure

1. Execute the provided *envcheck.py* with the following command:

   ```
   ./NETINT_toolkit/tools/envcheck.py
   ```

2. If message shows “Pass: Env Check SUCCESS!!”, this means that you have installed all the necessary packages and setup the correct environment variables. Please go to the next chapter.

3. Development Workflow

This chapter describes the complete workflow, features, and the provided tools in more detail.

3.1. Setup workspace

The *create_workspace.py* tool is able to create the project workspace, containing the generated scripts and the necessary work folders.

Prerequisites

- The NETINT AI Toolkit is installed correctly as described in 2. Setup the NETINT AI Development Environment.
- The pre-trained deep learning model and calibration/testing data are prepared.

3.1.1. Create Workspace

Description

Create the project workspace containing all scripts and sub-folders.

Procedure

1. Check script arguments.

```
create_workspace.py
          --model_string MODEL_FILE
          --target_path PATH
```

Arguments	Description
-m, --model_string	Requires a string value to denote the file path of the prepared Neural Network (NN) model files. If the model has configuration and weight files, such as Darknet and Caffe based models, these need to be separated by commas.
-t, --target_path	Requires a string value to denote the file path of the created workspace.

3.1.2. Workspace Structure

Description

Review the generated scripts and sub-folders, as shown in the following tables.

Table 5. Generated Scripts

Script	Description
convert.sh	Contains functions for the following model import, quantize, validation, export OpenVX and NBG model.
dataset.txt	Save the path of testing images.
Model files	The copied model files

Table 6. Generated Sub-folders

Sub-folders	Description
data	Save the prepared testing images.
templates	Contains templates, such as run_c_model.py, and postprocess_tensor.py
raw_data_f	Save the validation results of floating model
raw_data_i	Save the validation results of quantized model
raw_data_c	Save the validation results of OpenVX model
results	Save the final results after post-processing.

3.2. Setup Testing Data

Prerequisites

- The workspace structure is generated correctly as described in 3.1.2 Workspace Structure.

Procedure

1. Copy the sample data to the `./data` folder.
2. Save the data path into dataset.txt file. For example:

   ```
   ls -R ./data/*.* > dataset.txt
   ```

   All testing data listed in dataset.txt will be used in the following procedures.

3.3. Model Import and Format Conversion

Prerequisites

- The workspace structure is generated correctly as described in 3.1.2 Workspace Structure.

Procedure

1. Based on model's type, you need to modify the `convert()` function, such as model inputs name, input size and model pre-processing configurations in *convert.sh*.
2. The neural network viewer, Netron, can help you to review the structure of model, such as the name of inputs, outputs, and input size.
3. Perform model conversion.

```
./convert.sh -c
```

4. Check the converted model files.

Output Files	Description
[model_name].data	Contains imported floating model weights
[model_name].json	Contains imported model structure
[model_name]_inputmeta.yml	Contains the setting of inputs, such as dataset, layout, channel order, mean and scale, etc. Please review those parameters and change them if necessary

Note: The explanation of various arguments can be referred to Appendix. A.2 Pegasus Command Line.

3.4. Inference with the Imported Model

Prerequisites

- The model is converted successfully as described in 3.3 Model Import and Format Translation

Procedure

1. Review preset parameters in validate() function and change them if necessary.
2. Inference the floating model.

```
./convert.sh -v
```

3. Check the results at *./raw_data_f* folder.

Output Files	Description
input tensor files	The input tensor files after the pre-processing
output tensor files	The inference output tensor file for imported floating model. These tensor files will be used to perform model post-processing (See 3.12 Model Post-processing)

Note: The explanation of various arguments can be referred to Appendix. A.2.3.11 Inference.

3.5. Model Optimization and Quantization

Perquisites

- The calibration data (defined in *dataset.txt*) is prepared as described in 3.2 Setup Testing Data.

Procedure

1. Depending on the validation results, you may need to modify the quantize() function in the convert.sh file, such as quantization method, data type etc.
2. Based on the quantize parameter setting defined in quantize() function as well as the calibration data defined in *dataset.txt*, perform model quantization:

```
./convert.sh -q
```

3. Check the quantization file at current path.

Output File	Description
[model_name].quantize	Contains quantization parameters

Note: Depends on the quantization method used, there are additional steps required to perform more complicated quantization such as hybrid, MLE. The explanation of various arguments can be referred to Appendix. A.2.3.10 Quantize.

3.6. Inference with the Quantized Model

Prerequisites

- The quantization file is generated as described in 3.5 Model Optimization and Quantization.

Procedure

1. Review the validate_q() function.

2. Inference the quantized model.

```
./convert.sh -vq
```

3. Check the results at *./raw_data_i* folder.

Output Files	Description
input tensor file	The input tensor files after the pre-processing
output tensor files	The output tensor files are inferenced with the quantized model. These tensor files will be used to perform model post-processing (See 3.12 Model Post-processing)

Note: The explanation of various arguments can be referred to Appendix. A.2.3.11 Inference

3.7. Export the OpenVX Model

Prerequisites

- The quantization file is generated as described in 3.5 Model Optimization and Quantization.

Procedure

1. Export the OpenVX model.

```
./convert.sh -o
```

2. Check the results at *./ovx* folder

Output Files	Description
[model_name].export.data	The weights of the OpenVX model
vnn_[model_name].c, vnn_[model_name].h	The source code of the OpenVX model structure
vnn_pre_process.c, vnn_pre_process.h	The source code for pre_processing
vnn_post_process.c, vnn_post_process.h	The source code for post_processing
main.c	The main source file
vnn_global.h	The global header file

Note: The explanation of various arguments can be referred to Appendix. A.2.3.12 Export OVXLIB

3.8. Compile the OpenVX Model

Prerequisites

- The OpenVX model is exported successfully as described in 3.7. Export the OpenVX model

Procedure

1. Compile OpenVX model under *./ovx* folder:

```
make -f makefile.linux
```

Output File	Description
[model_name]	The executable binary file

2. NETINT AI Toolkit provides two ways to compile OpenVX model, command line (recommend) and IDE. For IDE compile steps, see A.5.1 Import and Compile Models.

3.9. Inference with the OpenVX Model

Prerequisites

- The input tensor files after preprocessing such as input tensor generated by the quantized model as described in 3.6. Inference with the Quantized Model.
- The OpenVX model is compiled successfully as described in 3.8 Compile the OpenVX Model

Procedure

1. Run the OpenVX model manually to find out the correct input/output filenames.

```
./ovx/[model_name] ./ovx/[model_name].export.data ./raw_data_i/iter_x_inputxxx.tensor
```

it will generate *input_x.txt* and *output_x.txt*.

Output Files	Description
input_x.txt	The OpenVX model output filename for input data
outputx_x.txt	The output filenames for output data.

2. You also can use *run_c_model.py* from templates to run inference on all inputs from *dataset.txt*. There are few arguments need to be modified:

Arguments	Description
model_name	Requires a string value to denote the file name of a model binary file
image_list	Requires a string value to denote the file name of the validation data list
input_ext	Requires a string value to denote the input tensor data in raw_data_i without iter_x prefix.
output_exts	Requires a string value to denote the output tensor data in raw_data_i without iter_x prefix.
result_input	Requires a string value to denote the input filename produced by OpenVX model for input data (input_x.txt).
result_outputs	Requires a string value to denote the output filename produced by OpenVX model for output data (outputx_x.txt).

3. Fill in the generated file name in the corresponding position of the template file. For example:

```
model_name = "lenet_tf"    # model binary name; required for test run
image_list = 'dataset.txt'  # test data list
input_ext = "_attach_input_x-input_out0_1_out0_1_28_28_1.tensor"  #input tensor data without iter_{i}
output_exts = ["_attach_output_out0_0_out0_1_10.tensor"] #output tensor data without iter_{i}
result_input = "input_0.txt"    # cmodel output filename for input data
result_outputs = ["output0_10_1.txt"]   # cmodel output filenames for output data
```

4. Inference the OpenVX model through template file, *run_c_model.py*

```
./run_c_model.py
```

5. Check the results at *./raw_data_c.*

Output Files	Description
input tensor file	The input tensor files after the pre-processing
output tensor files	The output tensor files are inferenced with OpenVX model. These tensor files will be used to perform model post-processing (See 3.12 Model Post-processing)

3.10. Model Post-Processing

About This Task

The post-processing always depends on the model and its application. We recommend modifying the existing postprocessing template, *postprocess_tensor.py*, based on your requirements.

Prerequisites

One of the inferences results are available:

- The inference results produced by the imported floating model as described in 3.4 Inference with the Imported Model
- The inference results produced by quantized model as described in 3.6 Inference with the Quantized Model
- The inference results produced by OpenVX model as described in 3.9 Inference with the OpenVX model

Procedure

1. Set the respective arguments in *postprocess_tensor.py*.

Arguments	Values
output_exts	The output tensor file without prefix (iter_X)
output_type	txt/ png/ jpg

In addition to these arguments, you may need to modify and write postprocessing snippet in the postprocess_and_save() function. The example below shows a simple post-processing for classification task. The complete source code can be found in the examples folder.

```
# postprocess results with original data and outputfile name
def postprocess_and_save(data_list, orig_image, outputfile_path):
    print("Number of output: " + str(len(data_list)))
    #print("Orignial imagefile shape: " + str(orig_image.shape))
    print("Output file: " + outputfile_path)
    print("========================================")
    print("Please add code here to perform postprocessing!")
    print("postprocessing ......")
    print("Plesae save the results into " + outputfile_path)
    with open(outputfile_path,'w') as f:
        result = data_list[0]
```

```
        a = np.argsort(result)[::-1]
        b = np.sort(result)[::-1]
        for i in range(10):
              f.write(' Index: '+str(a[i])+ ' Prob: '+str(b[i]) + '\n')
```

2. Post-process model inference results.

```
# F: floating model
# I: quantized model
# C: OpenVX model
./postprocess_tensor.py -F # inference floating model
```

3. Review results at *./results* folder.

Output Files	**Description**
test_X_f.[output_type]	Post-processed results generated by the floating model.
test_X_i.[output_type]	Post-processed results generated by the quantized model.
test_X_c.[output_type]	Post-processed results generated by the OpenVX model.

3.11. Export the Network Binary Graph

Prerequisites

- The quantization file is generated as described in 3.5 Model Optimization and Quantization.

Procedure

1. Export the NBG file.

```
./convert.sh -e
```

2. Check the generated NBG file at *./ovx_network_nbg_unify/* folder

Output File	Description
network_binary.nb	The Network Binary Graph file

3. Optionally, in this case of embedding preprocessing within NBG, you need to turn on the preprocessing node. The below setting shows preprocessing turned on with BGR input and normalization with [0,1] and the expected input layout will be NCHW which is equivalent to *IMAGE_RGB888_PLANAR*.

```
preprocess:
   reverse_channel: true
   mean:
   - 0.0
   - 0.0
   - 0.0
   scale: 0.00392157
   preproc_node_params:
     add_preproc_node: true
     preproc_type: IMAGE_RGB888_PLANAR
     preproc_image_size:
     - 416
     - 416
     preproc_crop:
       enable_preproc_crop: false
       crop_rect:
        - 0
```

```
        - 0
        - 416
        - 416
    preproc_perm:
        - 0
        - 1
        - 2
        - 3
redirect_to_output: false
```

3.12. Analysis the Network Binary Graph

Prerequisites

- The NBG file is exported as described in 3.11 Export the Network Binary Graph

Procedure

1. Analysis the Network Binary Graph (NBG)

```
nbinfo -a network_binary.nb > nbinfo_result.txt
```

2. Check the generated NBInfo file.

Output File	Description
nbinfo_result.txt	Model structure and other configuration.

3. Review the NBG information described in the result file.

Types	Description
Overall_Info	Overall OpenVX model information
Layer Table	Used layer name and operation count
Operation Table	Involved operation type, such as Tensor Processing (TP), Parallel Processing Unit (PPU), Neural Network (NN)
Input Table	Information of the model input
Output Table	Information of the model output
Memory Info	Memory information such as total read only memory which will be read only by VIP and total command buffer which is the command buffer size of the network graph.
Input and Memory_pool overlap Info	Memory pool size

3.13. Inference with the Network Binary Graph on Quadra

Prerequisites

- The host has Quadra card installed correctly.
- The aiperf is installed correctly.
- The Xcoder Coder Library is installed correctly.

Procedure

1. Install the dependencies.

```
sudo apt install libopencv-dev
sudo apt install libjson-c-dev
```

2. Compile the aiperf tools.

```
cd ~/NETINT_toolkit/aiperf/
make
```

3. Initialize the NETINT transcoder resource pool.

```
sudo init_rsrc -r
```

4. Inference with Quadra AI Inference Engine.

```
sudo ./aiperf --network_binary ./NBG.nb --input_dir ./dataset.txt --output_dir ./output/ --file_type batch --file_format nhwc --channel_order rgb --golden_dir ./golden_dir --loop 1
```

5. Check the results in the output folder.

Output Files	Description
input.tensor	The raw input sent to hardware in tensor format
output.tensor	The inferenced output tensor file
dat file	The inferenced output results in binary format

6. Performance Testing

   Setup configuration in config.json as below:

```
{
        “configure”:[
        {
```

```
            "nb": "/path/to/NBG.nb",
            "dataset": "/path/to/dataset.txt",
            "outdir": "/path/to/outdir",
            "golddir": "/path/to/golddir",
            "format": "nchw",
            "order": "bgr",
            "devid": "0",
            "loop": "1"
        }]
}
```

Use the following command to test model performance, it will print the performance of the model by FPS.

```
sudo ./aiperf --conf_file config_example.json --skip_result
```

Note: Performance depends on the number of threads used for the evaluation. We recommend using 8 threads for saturating AI load.

4. User Case Studies

This chapter briefly introduces examples in the NETINT AI Toolkit and provides a complete workflow demonstration using mobilenet.

Model	Category	Framework	Input shape	Comments
deeplabv3	Segmentation	TFLite	257x257x3	
lenet	Classification	Caffe	1x28x28	
lenet	Classification	TensorFlow	28x28x1	Specify inputs, outputs, and input-size-list for model convert.
mobilenet	Classification	Keras	224x224x3	Use *--MLE* argument for quantize to improve accuracy. (Details showed in the following section)
mobilenetv2	Classification	ONNX	3x224x224	Specify inputs, outputs, input-size-list, size-with-batch for model convert.
resnet18	Classification	Torch	3x224x224	Specify input-size-list for model convert.
yolov4_tiny	Object Detection	Darknet	3x416x416	

Note: Comments in the above table indicate the changes that need to be made in *convert.sh*.

4.1. Complete Workflow for Mobilenet

The task uses the mobilenet classification model as an example to show the key procedures.

Prerequisites

- The NETINT AI Toolkits are installed correctly as described in 2.6 Verify the Development Environment.
- The mobilenet Keras model and calibration/testing data are prepared.

Procedure

The details of the work procedures will be explained below.

4.1.1. Create Workspace

```
create_workspace.py --model_string mobilenet.h5 --target_path ./
```

The structure of the generated work folders refers to 3.1 Setup workspace

4.1.2. Prepare Model Conversion Script

Based on model's requirements, you need to modify convert() function in the generated convert.sh.

Arguments	Values
--inputs	input_2
--outputs	last_softmax
--input-size-list	224,224,3
--mean	127.0,127.0,127.0
--scale	0.00787402

4.1.3. Convert Model

```
./convert.sh -c
```

Check the output files:

Output Files	Description
mobilenet.data	Contains mobilenet model weights
mobilenet.json	Contains mobilenet model structure
mobilenet_inputmeta.yml	Contains several preprocessing parameters, such as, reverse_channel, mean and scale, etc. Please review those parameters and change them if necessary

4.1.4. Setup Validation Data

```
ls -R ./data/*.* > dataset.txt
```

All the data listed in dataset.txt will be used in the following steps.

4.1.5. Inference with the Imported Model

```
./convert.sh -v
```

Check the output files:

Output Files	Description
iter_X_input_2_93_out0_1_224_224_3.tensor	The input tensor file after the pre-processing
iter_X_attach_last_softmax_Softmax_out0_0_out0_1_3.tensor	The inference output tensor file for imported model. This tensor file will be used to perform model post-processing (See 3.10 Model Post-processing)

4.1.6.Quantize the Imported Model

Based on the quantize parameter setting defined in convert.sh as well as the calibration data defined in dataset.txt, perform model quantization:

```
./convert.sh -q
```

The result of quantization is saved in *mobilenet.quantize.*

Output File	Description
mobilenet.quantize	Contains quantization parameters

4.1.7. Validate the Quantized Model

```
./convert.sh -vq
```

Output Files	Description
iter_X_input_2_93_out0_1_224_224_3.tensor	The input tensor file after the pre-processing
iter_X_attach_last_softmax_Softmax_out0_0_out0_1_3.tensor	The inference output tensor file for quantized model. This tensor file will be used to perform model post-processing (See 3.10 Model Post-processing)

4.1.8.Export an OpenVX Model

```
./convert.sh -o
```

The outputs of the model export are saved in *./ovx* folder:

Output Files	Description
mobilenet.export.data	The weights of the mobilenet OpenVX model
vnn_mobilenet.c, vnn_mobilenet.h	The structure of the mobilenet OpenVX model
vnn_pre_process.c, vnn_pre_process.h	The source code for pre_processing
vnn_post_process.c, vnn_post_process.h	The source code for post_processing
main.c	The main source file
vnn_global.h	The global header file

4.1.9.Compile the OpenVX Model

```
make -f makefile.linux
```

Output File	Description
mobilenet	The executable binary file

4.1.10.Evaluate the OpenVX Model

The generated binary model file can be used for inference, such as:

```
./ovx/mobilenet ./ovx/mobilenet.export.data ./raw_data_i/iter_0_input_2_93_out0_1_224_224_
       3.tensor
```

Check the generated files:

Output Files	Description
input_0.txt	The input data inferenced by OpenVX model
output0_3_1.txt	The output data inferenced by OpenVX model

To run multiple inferences in batch, *run_c_model.py* must be modified based on the input/output names:

Arguments	Description
model_name	mobilenet
image_list	dataset.txt
input_ext	_input_2_93_out0_1_224_224_3.tensor
output_exts	_attach_last_softmax_Softmax_out0_0_out0_1_3.tensor
result_input	input_0.txt
result_outputs	output0_3_1.txt

After setup all required arguments, you can execute *run_c_model.py* with the following commands:

```
./run_c_model.py
```

Check the output tensor files:

Output Files	Description
iter_X_input_2_93_out0_1_224_224_3.tensor	The input tensor file after the pre-processing
iter_7_attach_last_softmax_Softmax_out0_0_out0_1_3.tensor	The inference output tensor file for OpenVX model. This tensor file will be used to perform model post-processing (See 3.10 Model Post-processing)

4.1.11.Model Post-processing

Following the steps in 3.10 Model Post-Processing, set required arguments in the *postprocess_tensor.py*

```
cp ./templates/postprocess_tensor.py .
```

Variables	Values
output_exts	_attach_last_softmax_Softmax_out0_0_out0_1_3.tensor.
output_type	txt

In addition to these arguments, you need to modify postprocess_and_save() function. All demo codes are included in the toolkit examples folder.

After setup all required arguments and functions, you can run *postprocess_tensor.py*

```
./postprocess_tensor.py -F # inference floating model
./postprocess_tensor.py -I # inference quantized model
./postprocess_tensor.py -C # inference OpenVX model
```

Output Files	Description
test_X_f.txt	The results of floating model.
test_X_i.txt	The results of quantized model.
test_X_c.txt	The results of OpenVX model.

4.1.12. Export a Network Binary Graph

```
./convert.sh -e
```

Check the generated network binary file at *./ovx_network_nbg_unify*.

Output File	Description
network_binary.nb	The Network Binary Graph file of the mobilenet model

4.1.13. Inference with the Network Binary Graph on the Chip

Prerequisites

- The host has Quadra card installed correctly.
- The aiperf is installed correctly.
- The Xcoder Coder Library is installed correctly.

```
sudo ./aiperf -network_binary ./mobilenet.nb -input_dir ./dataset.txt -output_dir ./output
       -file_type batch -file_format nhwc -channel_order rgb -golden_dir ./golden_dir -
       loop 1
```

The inference outputs are:

Output Files	Description
input.tensor	The raw input sent to hardware in tensor format
output.tensor	The inferenced output tensor file
dat file	The inferenced output results in binary format

4.2. Quantization Accuracy Improvement

In previous example, the default quantization method (QUANTIZER="--quantizer asymmetric_affine --qtype uint8”) are used, and the top1 classification accuracy is 73.33% comparing to the original model with accuracy of 96.67%. In this section, we demonstrate how to improve quantization accuracy by adjusting quantization parameters.

4.2.1. Modify Quantization Method

You can manually set up different quantization method to gain the better accuracy. In this example, we demonstrate how to use ‘MLE’ argument with uint8 qtype to quantize model to obtained better accuracy.

Example

```
function quantize()
{
        if [ -f ${NAME}.quantize ]; then
                rm ${NAME}.quantize
        fi
        QUANTIZER="--quantizer asymmetric_affine --qtype uint8"
        ${pegasus} quantize --model ${NAME}.json --model-data ${NAME}.data \
                --with-input-meta ${NAME}_inputmeta.yml --algorithm kl_divergence \
                --iterations ${DATASET_FILE_NUM} --MLE --device CPU \
                ${QUANTIZER}
}
```

4.2.2.Quantize the Imported Model

Based on the quantize parameter setting defined in convert.sh as well as the calibration data defined in dataset.txt, perform model quantization:

```
./convert.sh -q
```

Check the output quantization file:

Output File	Description
mobilenet.quantize	Contains quantization parameters
mobilenet.mle.data	New data file generated by performing MLE with quantization.

4.2.3.Validate the Quantized Model

You need to modify validate_q() function to apply the new DATA file in order to inference with the MLE quantized model as below:

```
function validate_q()
{
        ${pegasus} inference --model ${NAME}.json --model-data ${NAME}.mle.data \
                --with-input-meta ${NAME}_inputmeta.yml \
                --postprocess 'dump_results' --output-dir raw_data_i \
                --dtype quantized --model-quantize ${NAME}.quantize \
                --batch-size 1 --iterations ${DATASET_FILE_NUM} --device CPU
}
```

Then perform regular workflow.

4.2.4.Export an OpenVX Model

You need to modify export_ovx() function to apply the new DATA file in order to export the correct OpenVX model as below:

```
function export_ovx()
{
        OUTPUT_OPT="ovx/${NAME}"
        ${pegasus} export ovxlib --model ${NAME}.json --model-data ${NAME}.mle.data \
                --with-input-meta ${NAME}_inputmeta.yml --output-path ${OUTPUT_OPT}   \
                --target-ide-project 'linux64' --save-fused-graph  \
                --dtype quantized --model-quantize ${NAME}.quantize
}
```

Then perform regular workflow.

4.2.5.Export the Network Binary Graph

You need to modify export_net() function to apply the new DATA file in order to export the correct OpenVX NBG as below:

```
function export_net()
{
        DRIVER_OPT=--pack-nbg-unify
        OUTPUT_OPT="ovx_network/${NAME}"
        TARGET="VIP8000NANOQI_PLUS_PID0XB1"
        ${pegasus} export ovxlib --model ${NAME}.json --model-data ${NAME}.mle.data \
                --with-input-meta ${NAME}_inputmeta.yml \
                --output-path ${OUTPUT_OPT} ${DRIVER_OPT}  \
                --optimize ${TARGET} --viv-sdk ${VIVANTE_SDK_DIR} \
                --batch-size 1 --dtype quantized --model-quantize ${NAME}.quantize
}
```

Then perform regular workflow.

4.2.6.The Comparison of Two Different Quantization Methods

The mobilenet_keras example contains two versions with different quantization methods for a toy classification problem (cat, dog, and panda) as below:

Top1	Original Model (Keras)	Converted Model (Floating point)	Quantized Model (Default)	Quantized Model (Default + MLE)
Accuracy (%)	96.67%	96.67%	73.33%	96.67%

By applying MLE (Minimizes per-layer quantization errors), the quantized model reaches the same accuracy as the original model in this example. Note that MLE often to be time consuming and the quantization accuracy depends on many factors such as data type, calibration data, and quantization algorithms.

A. Appendix

The Appendix chapter demonstrates the additional tools, including the explanation of command syntax, arguments, and usage examples.

A.1. NETINT AI Toolkit Explanation

This section describes high level tools and scripts provided by NETINT AI toolkit.

A.1.1. Create Workspace

Description

The *create_workspace.py* creates a workspace and necessary scripts based on given model types.

Syntax

```
create_workspace.py -m [model_files] -t [target_dir]
```

Arguments

Argument	Description
-m, --model_string	Requires a string value to denote the file path of the prepared Neural Network (NN) model files. If the model has configuration and weight files, such as Darknet and Caffe based models, these need to be separated by commas.
-t, --target_path	Requires a string value to denote the file path of the created workspace

The generated scripts and work folders.

Script	Description
convert.sh	Contains functions for the following model import, model quantize, validation of imported model and quantized model, export OpenVX project and NBG files.
dataset.txt	Save the path of testing images in data folder.
model files	The copied model files
data	Save the prepared testing images.
templates	Contains several templates, such as run_c_model.py, postprocess_tensor.py, which may use in the working procedure
raw_data_f	Saves the validation results of imported model

Script	Description
raw_data_i	Saves the validation results of quantized model
raw_data_c	Save the validation results of OpenVX model
results	Save the final results.

A.1.2. Setup the Environment Variables

Description

The *setup_bashrc.py* sets the required environment variables into *.bashrc* file.

Syntax

```
./setup_bashrc.py
```

When this command is executed, it will set up the required environment values into *.bashrc* file. These values are:

Values	Description
LD_LIBRARY_PATH	The system LD Library path
ACUITY_PATH	The toolkit binary file directory path
VIVANTE_BASE	The IDE directory path
VIVANTE_VIP_HOME	The IDE home path
VIVANTE_SDK_DIR	The IDE software development kit directory
NETINT_TOOLS	The NETINT AI high level tool's path
LMX_LICENSE_PATH	The IDE license path
VSIMULATOR_CONFIG	The Quadra AI hardware configuration type
NBINFO	The NBinfo Parser Tool directory
USE_IDE_LIB	The switch for turn on IDE LIB during compilation
TF_CPP_MIN_LOG_LEVEL	The value of TensorFlow log level
PIP_DISABLE_PIP_VERSION_CHECK	The value for avoiding pip version check

A.1.3. Review the Environment Variables

Description

The *envcheck.py* checks whether the required packages and environment variables are set correctly.

Syntax

```
./envcheck.py
```

If you follow the above installation steps, you will get a pass success signal, which means that you have installed all the necessary packages and setup environment.

If you get a failure signal, please follow the prompts to reinstall and return to the installation chapter to verify the point of error.

A.1.4. Generate Inputmeta

Description

The *generate_inputmeta.py* generates the *[model_name]_inputmeta.yml* file which saves the preprocessing setting.

Syntax

```
./generate_inputmeta.py      --model [model_name].json
                                    --mean MEAN
                                    --scale SCALE
                                    --reverse_channel true
```

There are some required arguments need set up:

Arguments

Arguments	Description
--model	Requires a string value to denote the file path of the model.json file.
--mean	Mean value for each channel after reverse channel (if reverse_channel is true), such as '128,128,128'
--scale	Scale value for the tensor. It will multiply a float value to the tensor. Such as '0.00392157'
--reverse_channel	Define whether the model's channel is reverse or not, true/false.

The generated *[model_name]_inputmeta.yml*

Output File	Description
[model_name]_inputmeta.yml	This file contains several preprocessing parameters, such as, reverse_channel, mean and scale, etc., Please review those parameters and change them if necessary.

A.1.5. Convert the Tensor to Image

Description

The *tensor2image.py* converts the input tensor file to an image. Based on the preprocessing setting defined in the *[model_name]_inputmeta.yml* file, the converted image should be consistent with the original image.

Syntax

```
python3 tensor2image.py --inputmeta [model_name]_inputmeta.yml \
                        --tensorfile INPUT \
                        --output OUTPUT
```

Argument	Description
--input_meta	Requires a string value to denote the file path of the network meta file, a .yml file.
--tensorfile	Requires a path to denote the input tensor file. For example, "./input.tensor"
--output	Requires a name to denote the output image files. For example, "./img.jpg"

When the execution of the above procedure is successful, it will generate an image file. For example:

Output Files	Description
[file_name].jpg/.png	The generated image file

A.1.6. Convert the Image to Tensor

Description

The *image2tensor.py* tool is able to convert the image file to tensor according to the preprocessing setting defined in the *[model_name]_inputmeta.yml* file. This tool will resize the image and the ratio are not preserved.

Syntax

```
python3 image2tensor.py --input_meta [model_name]_inputmeta.yml \
                        --input INPUT \
                        --output OUTPUT
```

Arguments	Description
--input_meta	Requires a string value to denote the file path of the network meta file, a .yml file.
--input	Requires a path to denote the input tensor file. For example, “./input.jpg"
--output	Requires a name to denote the output tensor files. For example, “./output.tensor"

When the execution of the above procedure is successful, it will generate a tensor file. For example:

Files	Description
Output.tensor	The generated tensor file

A.1.7. Resize the Image

Description

The *image_resize.py* is able to resize the input image to the target size with/without keep aspect ratio of the original image.

Syntax

```
python3 image_resize.py      --width WIDTH \
                             --height HEIGHT \
                             --input_path INPUT_PATH \
                             --output_path OUTPUT_PATH  \
                             --keep_ratio True  \
                             --format FORMAT  \
```

Arguments

Arguments	Description
--width	The target width of the input image
--height	The target height of the input image
--input_path	Requires a string value to denote the file path of input image files
--output_path	Requires a string value to denote the file path of output resized image files
--keep_ratio	Keep the ratio of the image. If set to false, it may impact model accuracy for classification or detection task. The default value is 'True'
--format	The format for the output resized image, jpg or png

A.2. Pegasus Command Line

The Pegasus command line tool provides a set of commands for the user to perform model translations, optimizations, inferences, and export on the deep learning models. This chapter describes the command syntax, arguments, and usage examples.

The following table categorizes the pegasus commands by functions.

Table 3-1 pegasus Commands Summary

Function	Command
Model import and format translation	pegasus.py import caffe pegasus.py import tensorflow pegasus.py import tflite pegasus.py import darknet pegasus.py import onnx pegasus.py import pytorch pegasus.py import keras
Network preprocessing and post processing	pegasus.py generate inputmeta pegasus.py generate postprocess-file
Model optimization, training, and inference	pegasus.py prune pegasus.py train pegasus.py dump pegasus.py quantize pegasus.py inference
Model export	pegasus.py export ovxlib pegasus.py export tflite

A.2.1. Syntax Conventions

Pegasus command lines use the following syntax conventions:

- Arguments and their values are case sensitive.
- Arguments use lower cases.
- Arguments, except for -h, start with two dashes and no space.
  Example: --separated-database
- Replaceable argument values use all caps and italics.
  Example: *SEPARATED_DATABASE*
- String input values are surrounded by quotation marks.
- The arguments input order is flexible.
- [x] denotes optional items.
- {a, b, c} denotes only one of the options must be chosen.

A.2.2. Dataset Formats

Pegasus supports text, npy dataset formats. This section describes the definitions of this formats in detail.

A.2.2.1. Text File (.txt)

A text file (`.txt`) contains one or more rows of image information for network inference. Each row contains one or more elements (for example, image paths, NPY file paths, or literal values) separated by spaces. Elements in each column constitute the dataset of an input layer. The order of the columns must follow the input layer order specified in the ports parameter in the `inputmeta.yml` file.

For a network with only one input layer, each row can be assigned a different label. However, labels are not supported in multi-layer scenarios.

Examples

Example 1: `dataset.txt` for a network of only one input layer

```
./image/1.jpg 1                         #label is 1.
./image/2.png 2                         #label is 2.
```

Example 2: `dataset.txt` for a network that has two input layers and requires six rows of data in a batch. The `dataset.txt` file contains at least six rows and two columns, where the columns represent the input layers. Elements in each row are separated by spaces. The order of the columns must follow the input layer order specified in the ports field in the `inputmeta.yml` file.

```
./image1/a.jpg ./image/aa.png
./image1/b.jpg ./image/bb.png
./image1/c.jpg ./image/cc.png
./image1/d.jpg ./image/dd.png
./image1/e.jpg ./image/ee.png
./image1/f.jpg ./image/ff.png
```

A.2.2.2. Numpy Binary File (.npy)

Note: NumPy binary files are supported only by pegasus.

A NumPy binary file (.npy) contains an array of image information in the tensor shape of an input layer. Each layer is assigned a unique NumPy binary file. If the data in a NumPy binary file is less than required, pegasus duplicates the data.

Assume that an input layer requires Shape A while the array in a NumPy binary file follows Shape B. To assign such file to this input layer, Shape B must meet one of the following criteria:

- Shape B is the same as Shape A.
- Shape B differs Shape A only in the first dimension in:
  - The element count is different.
  - Shape B does not have this first dimension. It is a subset of Shape A.

The NumPy (.npy) dataset format is similar to the text (./txt) dataset format, but easier to use. The only difference is that each input layer needs a separate NumPy database. The databases must not be shared across layers, which means that an ./npy file must be exclusively used by only one layer ID.

Examples

Example 1: A four dimensional input layer shape (?,224,224,3) with a varied element count in the first dimension.

Then, in the NumPy binary file, the array would have one of the following shapes:

```
(224,224,3)
(1,224,224,3)
(2,224,224,3)
(100,224,224,3)
```

Example 2: A three-dimensional input layer shape (224,224,3).

Then, in the NumPy binary file, the array would have one of the following shapes:

```
(224,224,3)
(224,3)
(1,224,3)
(100,224,3)
```

A.2.3. Command Reference

A.2.3.1. Import Caffe

Description

Translates a Caffe model to unified formats.

Syntax

```
python3 pegasus.py import caffe
           [-h]
           --model MODEL
           --weights WEIGHTS
           [--proto PROTO]
           --output-model OUTPUT_MODEL
           --output-data OUTPUT_DATA
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with '', such as 'networkFilePath'.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the imported Caffe model.
--weights	Requires a string value to denote the file path of the imported weights file, the `.caffemodel` file. For example, '`mobilenet.caffemodel`'. When this argument is omitted, the system uses the default file path 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 neural network model file, the `.json` file. For example, '`mobilenet.json`'. When this argument is omitted, the system uses the default file path `<directory of the Caffe model>/<model name>.json`.
--output-data	Requires a string value to denote the file path of the generated neural network coefficient data file, the `.data` file. For example, '`mobilenet.data`'.

Argument	Description
	When this argument is omitted, the system uses the default file path `<directory of the Caffe model>/<model name>.data`.
[--proto]	Requires a string value to represent the protocol used by the imported Caffe model. The supported values are: • `'caffe'`: (Default) Represents the standard Caffe format protocol. • `'lstm_caffe'`: Represents the LSTM layer protocol. • `'<other protocol name>'`: For any other type of protocol with the filename *<xxx>*`_pb2.py`, specify the absolute path of the file as the value of --proto.
[-h]	Displays help information.

Example

```
python3 pegasus.py import caffe     --model 'mobilenet.prototxt' \
--weights 'mobilenet.caffemodel'    --output-model 'mobilenet.json' \
--output-data 'mobilenet.data'      --proto 'protoclNew'
```

A.2.3.2. Import TensorFlow

Description

Translates a TensorFlow model to unified formats. If the model is a quantized TensorFlow model, this command also generates a .quantize file, which is used during inference and export actions in NETINT.

Note: For quantized TensorFlow and TensorFlow Lite models, do not quantize them again in NETINT.

Syntax

```
python3 pegasus.py 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
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, 'networkFilePath'.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the TensorFlow frozen protocol buffer (protobuf) file. Note: Models trained and frozen by the following TensorFlow versions have been proven compatible 1.4.x, 2.0.x, 2.3.x, 2.6.x, and 2.8.x.
--inputs (Required)	Requires a string to list the input points of the TensorFlow graph. Multiple points are separated with spaces. For example, `'input_point_1 input_point_2'`.

Argument	Description
--input-size-list (Required)	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.
--outputs (Required)	Requires a string to specify the output points of the TensorFlow graph. Multiple points are separated with spaces. For example, `'output_point_1 output_point_2'`.
--output-model	Requires a string value to denote the file path of the generated network model file, the `.json` file. For example, `'mobilenet.json'`. When this argument is omitted, the system uses the default file path `<directory of the TensorFlow protobuf file>/<file name>.json`. When the model is a quantized TensorFlow model, the system also generates a `.quantize` file in the same directory.
--output-data	Requires a string value to denote the file path of the generated network coefficient data file, the `.data` file. For example, `'mobilenet.data'`. When this argument is omitted, the system uses the default file path `<directory of the TensorFlow protobuf file>/<file name>.data`.
[--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 layers with only one input point at each layer. 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'`.

Argument	Description
[--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's 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'`.
[-h]	Displays help information.

A.2.3.3. Import TFLite

Description

Translates a TensorFlow Lite model to unified formats. If the model is a quantized TensorFlow Lite model, this command also generates a `.quantize` file, which is used during inference and export actions.

Note: For quantized TensorFlow and TensorFlow Lite models, do not quantize them again.

Syntax

```
python3 pegasus.py import tflite
           [-h]
           --model MODEL
           [--inputs INPUTS]
           [--input-size-list INPUT_SIZE_LIST]
           [--outputs OUTPUTS]
           [--size-with-batch SIZE_WITH_BATCH]
           [--output-model OUTPUT_MODEL]
           [--output-data OUTPUT_DATA]
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'`.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the imported TensorFlow Lite (TFLite) model.Note: Import failures may occur if the TFLite model uses an unsupported TFLite schema.
--output-model	Requires a string value to denote the file path of the generated the network model file, the `.json` file. For example, '`mobilenet.json`'. When this argument is omitted, the system uses the default file path `<directory of the TFLite model>/<model name>.json`. When the model is a quantized TFLite model, the system also generates a `.quantize` file in the same directory.

Argument	Description
--output-data	Requires a string value to denote the file path of the generated network coefficient data file, the `.data` file. For example, '`mobilenet.data`'. When this argument is omitted, the system uses the default file path `<directory of the TFLite model>/<model name>.data`.
--outputs	Requires a string to specify the output points of the TFLite model. Multiple points are separated with spaces. For example, '`output_point_1 output_point_2`'. When this argument is omitted, the system uses all tail points of this model.
[--inputs]	Requires a string to list the input points of the graph. Multiple points are separated with spaces. For example, '`input_point_1 input_point_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.
[--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 layer. 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`'.
[-h]	Displays help information.

A.2.3.4. Import Darknet

Description

Translates a darknet model to unified formats.

Syntax

```
python3 pegasus.py import darknet
           [-h]
           --model MODEL
           --weights WEIGHTS
           --output-model OUTPUT_MODEL
           --output-data OUTPUT_DATA
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'`.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the imported Darknet model, a `.cfg` file.
--weights	Requires a string value to denote the file path of the weights file of the imported Darknet model. For example, '`mobilenet.weights`'.
--output-model	Requires a string value to denote the file path of the generated network model file, the `.json` file. For example, '`mobilenet.json`'. When this argument is omitted, the system uses the default file path `<directory of the Darknet model>/<model name>.json`.
--output-data	Requires a string value to denote the file path of the generated network coefficient data file, the `.data` file. For example, '`mobilenet.data`'. When this argument is omitted, the system uses the default file path `<directory of the Darknet model>/<model name>.data`.
[-h]	Displays help information.

A.2.3.5. Import ONNX

Description

Translates an ONNX model to unified formats.

Syntax

```
python3 pegasus.py 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
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, 'networkFilePath'.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the imported ONNX model, an `.onnx` file. Note: ONNX operator sets 1 through 16 are supported.
--inputs	Requires a string to list the input points of the ONNX model. Multiple points are separated with spaces. For example, `'input_point_1 input_point_2'`. When 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.

Argument	Description
[--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 layers with only one input point at each layer. 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	Requires a string value to denote the file path of the generated network model file, the `.json` file. For example, '`mobilenet.json`'. When this argument is omitted, the system uses the default file path `<directory of the ONNX model>/<model name>.json`.
[--output-data]	Requires a string value to denote the file path of the generated the network coefficient data file, the `.data` file. For example, '`mobilenet.data`'. When this argument is omitted, the system uses the default file path `<directory of the ONNX model>/<model name>.data`.
[--outputs]	Requires a string to specify the output points of the ONNX model. Multiple points are separated with spaces. For example, `'output_point_1 output_point_2'`. When this argument is omitted, the system uses all tail points of this model.
[-h]	Displays help information.

A.2.3.6. Import PyTorch

Description

Translates a PyTorch model to unified formats either with a series of configuration arguments or with only one configuration file specified by the `--config` argument. PyTorch 1.5.1 is supported.

Note: The PyTorch model must be created with `torch.jit.trace()`.

Syntax

```
python3 pegasus.py 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]
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'`.

Argument	Description
[--model]	Requires 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 by space. For example, `'input_point_1 input_point_2'`. When this argument is omitted, the system uses all head points of the imported model.
[--outputs]	Requires a string to specify the output points of the PyTorch model. Multiple points are separated with spaces. For example, `'output_point_1 output_point_2'`. When this argument is omitted, the system uses all tail points of the imported model.

Argument	Description
[--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 layers with one input point at each layer. 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	Requires a string value to denote the file path of the generated network model file, the `.json` file. For example, 'mobilenet.json'. When this argument is omitted, the system uses the default file path <directory of the PyTorch model>/<model name>.json.
--output-data	Requires a string value to denote the file path of the generated the network coefficient data file, the `.data` file. For example, 'mobilenet.data'. When this argument is omitted, the system uses the default file path <directory of the PyTorch model>/<model name>.data.
[--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 unified formats. With this argument, you do not need to use any other configuration arguments listed above.
[-h]	Displays help information.

A.2.3.7. Import Keras

Description

Translates a Keras model to unified formats.

Syntax

```
python3 pegasus.py import keras
           [-h]
           --model MODEL
           [--convert-engine CONVERT_ENGINE]
           [--inputs INPUTS]
           [--input-size-list INPUT_SIZE_LIST]
           [--outputs OUTPUTS]
           [--output-model OUTPUT_MODEL]
           [--output-data OUTPUT_DATA]
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'`.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the imported Keras model, a `.h5` file.
[--convert-engine]	Reserved for future use.
[--inputs]	Requires a string to list the input points of the Keras model. Multiple input points are separated with spaces. For example, `'input_point_1 input_point_2'`. When this argument is omitted, the system uses the header points of the imported model.

Argument	Description
[--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]	Requires a string to specify the output points of the Keras model. Multiple output points are separated with spaces. For example, `'output_point_1 output_point_2'`. When this argument is omitted, the system uses the tail points of the imported model.
[--output-model]	Requires a string value to denote the file path of the generated network model file, the `.json` file. For example, '`mobilenet.json`'. When this argument is omitted, the system uses the default file path `<directory of the Keras model>/<model name>.json`.
[--output-data]	Requires a string value to denote the file path of the generated network coefficient data file, the `.data` file. For example, '`mobilenet.data`'. When this argument is omitted, the system uses the default file path `<directory of the Keras model>/<model name>.data`.
[-h]	Displays help information.

A.2.3.8. Generate Inputmeta

Description

Generates an inputmeta file in the `.yml` format to configure input datasets for models. The inputmeta file is used in dump, quantize, inference, and export actions.

Syntax

```
python3 pegasus.py generate inputmeta
          [-h]
          [--input-meta-output INPUT_META_OUTPUT]
          [--separated-database {True, False}]
          --model MODEL
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'`.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the network model file, a `.json` file. For example, '`mobilenet.json`'.
[--input-meta-output]	Requires 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 `<directory of the neural network model>/<model name>_inputmeta.yml`.
[--separated-database]	Requires `True` or `False` to separate the database into multiple ones in the generated inputmeta file for a multi-input network. This argument is valid only for multi-input networks. `True` is invalid for single-input networks. When this argument is omitted, the system uses the default value `False` to keep one database in the generated inputmeta file.
[-h]	Displays help information.

A.2.3.9. Generate Postprocess File

Description

Generates a post-processing file for the model. The post-processing file describes the output post-processing tasks.

Note: The inference action needs both inputmeta and post-processing files.

Syntax

```
python3 pegasus.py generate postprocess-file
          [-h]
          [--postprocess-file-output POSTPROCESS_FILE_OUTPUT]
          -model MODEL
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.

Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'`.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the network model file, a `.json` file. For example, `'mobilenet.json'`.
[--postprocess-file-output]	Requires a string value to denote the file path of the generated post-processing file. When this argument is omitted, the system uses the default file path `<directory of the neural network model>/<model name>_prostprocess_file.yml`.
[-h]	Displays help information.

A.2.3.10. Quantize

Description

Quantizes network data.

Note: During the import model procedure, those quantize file has been generated. Please do not perform quantization on TensorFlow and TensorFlow Lite models that have been quantized.

Syntax

```
python3 pegasus.py 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]
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'`.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the network model file, a `.json` file. For example, `'mobilenet.json'`.
--model-data (Required)	Requires a string value to denote the file path of the model coefficient data file, a `.data` file. For example, `'mobilenet.data'`.
[--model-quantize]	Requires a string value to denote the file path of the generarted `.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.

Argument	Description
[--quantizer]	Requires one of the following values to specify the quantizer for quantizing network tensors: • `'asymmetric_affine'`: Default • `'dynamic_fixed_point'` • `'perchannel_symmetric_affine'`: Limited support • `'bfloat16'`: Limited support 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: • `'int16'` • `'uint8'`: Default • `'int8': For experimental use only` • `'bfloat16': For experimental use only` • `'int4': For experimental use only` If the –qtype argument is set to 'int4', the precision cannot be guaranteed. To revert to 8-bit quantization, toolkit performs automatic hybrid quantization for some layers. Exported applications that are int4 quantized may fail to run on the software stack. 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 default quantization rules. If this argument is specified, the previously generated quantization file will be overwritten. To quantize the network with a quantizer other than `bfloat16`, specify this argument. This argument is mutually exclusive with the `--hybrid` and `--rebuild-all` arguments.

Argument	Description
[--rebuild-all]	(Experimental use only) Performs quantization with the default quantization rules and forces to quantize all float activations. If this argument is specified, the previously generated quantization file will be overwritten. To quantize the network with the `bfloat16` quantizer, specify this argument. This argument is mutually exclusive with the `--hybrid` and `--rebuild-all` 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 this value is chosen, do not specify the –MLE arguemnt. 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.

Argument	Description
[--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 except automatic hybrid quantization ('auto'). The generated `*.mle.data` cannot be used to do re-quantization. It can only be used with the matching `.quantize` file for inference, dump and export Note: If this argument is specified, the quantization process may be time-consuming
[-h]	Displays help information.

A.2.3.11. Inference

Description

Performs inference for the model.

Syntax

```
python3 pegasus.py 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]
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, 'networkFilePath'.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the network model file, a `.json` file. For example, '`mobilenet.json`'.
--model-data (Required)	Requires a string value to denote the file path of the model coefficient data file, a `.data` file. For example, '`mobilenet.data`'.
[--model-quantize]	Requires 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 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`.

Argument	Description
[--iterations]	Requires an integer to specify the number of image batches. 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.
[--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.
[-h]	Displays help information.

A.2.3.12. Export OVXLIB

Description

Exports the model to OpenVX applications to run with the OVXLIB C library.

Note: To add pre-processing tasks to the exported OpenVX applications, in the `inputmeta.yml` file, set the `add_preproc_node` field to true and modify other associated fields.

Syntax

```
python3 pegasus.py export ovxlib
           [-h]
           --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]
           [--viv-sdk VIV_SDK]
           [--build-platform 'make']
           [--target-ide-project TARGET_IDE_PROJECT]
           [--batch-size BATCH_SIZE]
           [--force-remove-permute]
           [--customer-lids]
           [--customer-ops]
Arguments
```

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.

Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'`.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the network model file, a `.json` file. For example, '`mobilenet.json`'.

Argument	Description
	When 4-bit quantization is used, set --model to the .quantize.json file generated using the quantize command.
--model-data (Required)	Requires a string value to denote the file path of the model coefficient data file, a `.data` file. For example, '`mobilenet.data`'.
--with-input-meta (Required)	Requires a string value to denote the file path of the inputmeta file, a `.yml` file.
[--model-quantize]	Requires 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'`.
[--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-path]	Requires a string value to denote the directory of the exported applications with the prefix specified. Use the syntax `'<output directory>/<prefix>'`, where: • Output directory: The directory of the generated outputs including subdirectories and files. • Prefix: The prefix of the generated subdirectories and files.
[--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` 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.

Argument	Description
[--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` 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` 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. 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 two applications: • unify application with `.export.data`, `.c`, and `.h` files in the output directory • nbg_unify application with `.nb`, `.c`, and `.h` files in the output directory The output directory is specified by the `--output-path` argument. **Note:** • If `--pack-nbg-unify` is not specified, only the unify application is generated. • Packing is 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 `'~/NETINT_toolkit/VivanteIDEx.x.x/*cmdtools'` if VivanteIDE is installed. Specify this argument if the `--pack-nbg-unify` argument is specified.

Argument	Description
[--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'`.
[--target-ide-project]	Requires one of the following values to specify the environment of IDE, which is used to run the exported unify application code. • `'linux64'`: Default • `'win32'` When this argument is omitted, the system uses the default value.
[--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.
[-h]	Displays help information.

A.2.3.13. Export TFLite

Description

Exports the model to a TensorFlow Lite model.

Syntax

```
python3 pegasus.py export tflite
          [-h]
          --model MODEL
          --model-data MODEL_DATA
          [--model-quantize MODEL_QUANTIZE]
          [--output-path OUTPUT_PATH]
          [--save-fused-graph]
          [--dtype DTYPE]
          [--float-io]
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'`.

Argument	Description
--model (Required)	Requires a string value to denote the file path of the network model file, a `.json` file. For example, '`mobilenet.json`'. When 4-bit quantization is used, set --model to the `.quantize.json` file generated using the quantize command.
--model-data (Required)	Requires a string value to denote the file path of the model coefficient data file, a `.data` file. For example, '`mobilenet.data`'.
[--model-quantize]	Requires 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'`.
[--output-path]	Requires a string value to denote the directory of the exported TensorFlow Lite (TFLite) model with the prefix specified. Use the syntax `'<output directory>/<prefix>'`: • Output directory: The directory of the generated outputs including subdirectories and files. • Prefix: The prefix of the generated subdirectories and files.

Argument	Description
[--save-fused-graph]	(Experimental use only) Generates a fused model for debugging use. The generated fused model contains the network structures of the exported TFLite model. When this argument is omitted, no fused model is saved. Note: This argument does not require values.
[--dtype]	Requires one of the following values to specify the tensor data type of the exported TFLite model. • `'float32'`: The default data type. • `'float16'` • `'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.
[--float-io]	Sets the data type of the network inputs and the outputs to float32.
[-h]	Displays help information.

A.2.3.14. Dump

Description

Dumps tensors from each layer.

Syntax

```
python3 pegasus.py dump
          [-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]
          [--format FORMAT]
          [--save-quantize]
          [--save-file-type SAVE_FILE_TYPE]
          [--output-dir OUT DIR]
```

Arguments

Note 1: Arguments in [] are optional. If they are omitted in a command, default values are used.
Note 2: String argument values must be surrounded with ''. For example, `'networkFilePath'.`

Argument	Description
--model (Required)	Requires a string value to denote the file path of the network model file, a `.json` file. For example, '`mobilenet.json`'.
--model-data (Required)	Requires a string value to denote the file path of the model coefficient data file, a `.data` file. For example, '`mobilenet.data`'.
[--model-quantize]	Requires 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'`.

Argument	Description
[--batch-size]	Requires an integer to specify the number of images per batch. 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 image batches. 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 type of the compute device: • `'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.
[--format]	Requires one of the following values to specify the format in which to save snapshot tensor data: • `'nchw'`: (Default) Use this value for Caffe, Darknet, ONNX, and PyTorch models. • `'nhwc'`: Use this value for TensorFlow, TensorFlow Lite, and Keras models. When this argument is omitted, the system uses the default sequence.
[--save-quantize]	Saves data in the quantized format. When this argument is specified, quantized tensors are saved in the format specified in the `.quantize` file. When this argument is omitted, tensors are saved in the float32 type. **Note:** This argument does not require values.

Argument	Description
[--save-file-type]	Requires one of the following values to specify the format of the saved snapshot files: • `'tensor'`: (Default) Generates a `.tensor` file for each tensor. In `.tensor` files, tensor values are flattened. • `'bin'`: Generates a `.bin` file. • `'npy'`: Generates an `.npy` file for each tensor. When this argument is omitted, the system uses the default format. For the data type of the numbers, see the description of the `--save-quantize` argument.
[--output-dir]	Requires a string value to denote the directory for the generated snapshot files.
[-h]	Displays help information.

A.3. NBInfo Parser Tool

Description

NBInfo is a parsing tool for network binary graph (NBG). It interprets NBG and extracts detailed information, including input, output, all the layers and operations.

Usage

1. To extract the meta information from a Network Binary Graph (NBG):

```
nbinfo OPTIONS network_binary.nb > nbinfo_result.txt
```

Arguments

Argument	Description
-a	Print all information in the NBG.
-b	Print summary of NBG, including version, hardware target, axi_sram_size etc.
-I	Print the layer table in the NBG.
-o	Print the operation table in the NBG.
-in	Print input table in the NBG.
-out	Print output table in the NBG.
-m	Print all memory information in the NBG (user_heap_size and video_memory_size).
-h	Print the usage text.

A.4. IDE Debug and Profile

Integrated Development Environment (IDE) provides powerful features for every stage of software development and enables programmers to efficiently accomplish tasks of compiling, building, debugging, and profiling software on a Quadra hardware. These sections provide a guide to effective use of this powerful development tool.

A.4.1. Import and Compile Models

Description

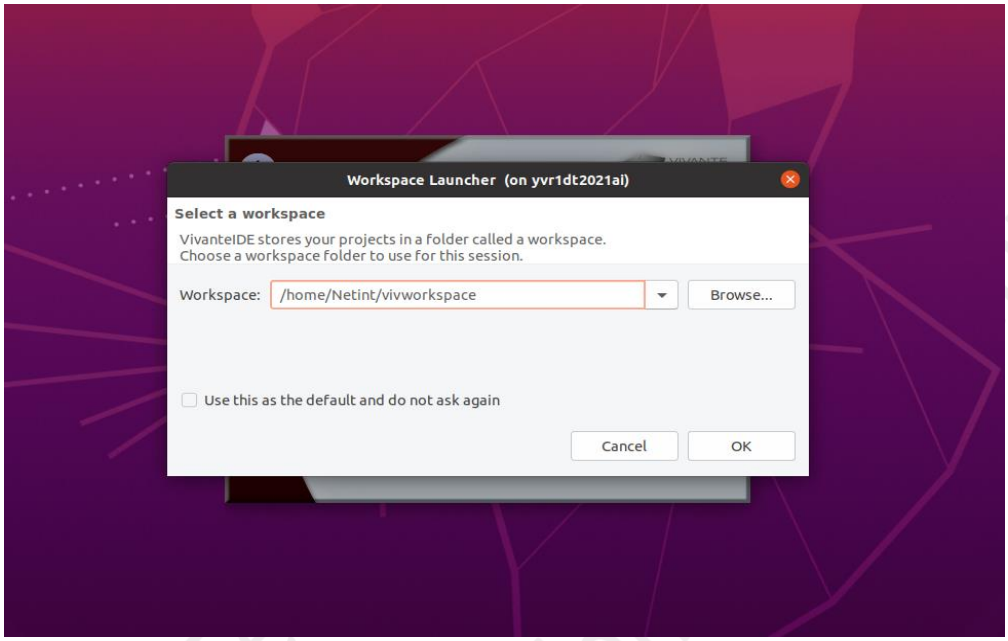
Select the projects that need to be imported and compiled.

Prerequisites

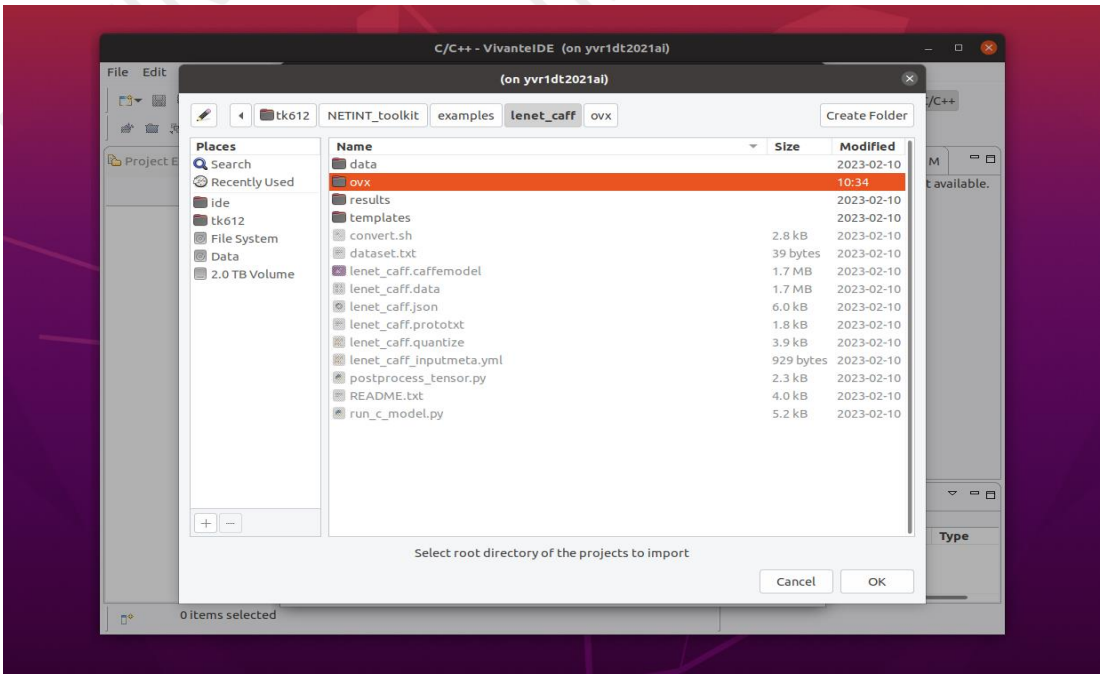
- The host meets the requirements as described in 2.1 System Requirements.
- The model is imported and compiled successfully.
- The NETINT AI Toolkits are installed correctly as described in 2.6 Verify the Development Environment.

Usage

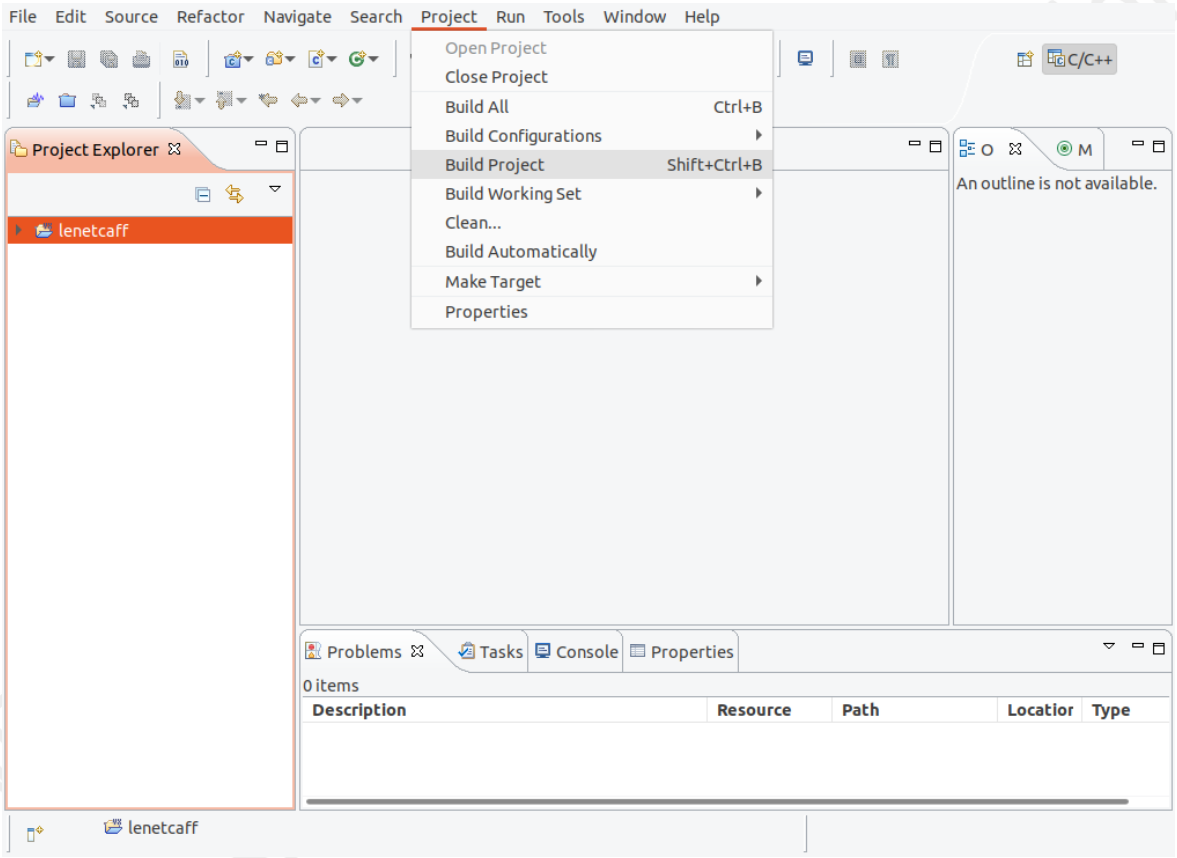
1. To open the IDE, select and setup the workspace:



2. select the `'import'` features in the File menu and enter in the `'Existing Projects into Workspace'` features under the General directory. Browse and set up the workspace which is the *./ovx* folder in the project workspace



2. Select ‘`Build project`’ to compile the model.



A.4.2. Debug Models

Description

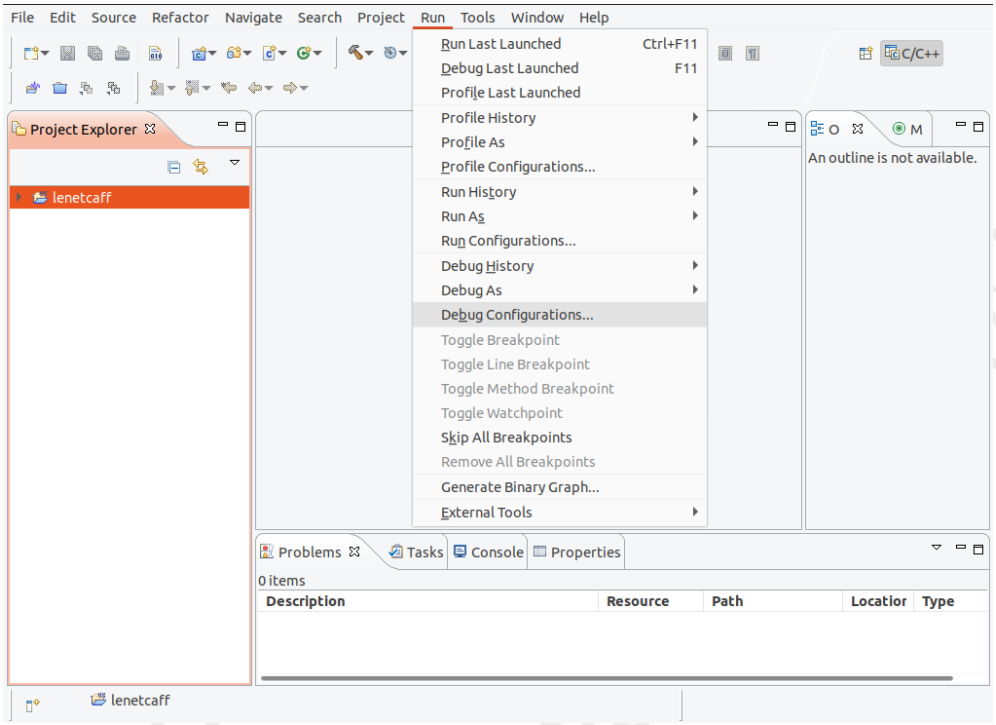
For each debug session, the program counter, associated assembler instructions, and defined breakpoints are visible. In the Debug Configurations dialog, click the Debug button to launch the program. Upon completion, the workbench automatically switches to the Debug Perspective.

Prerequisites

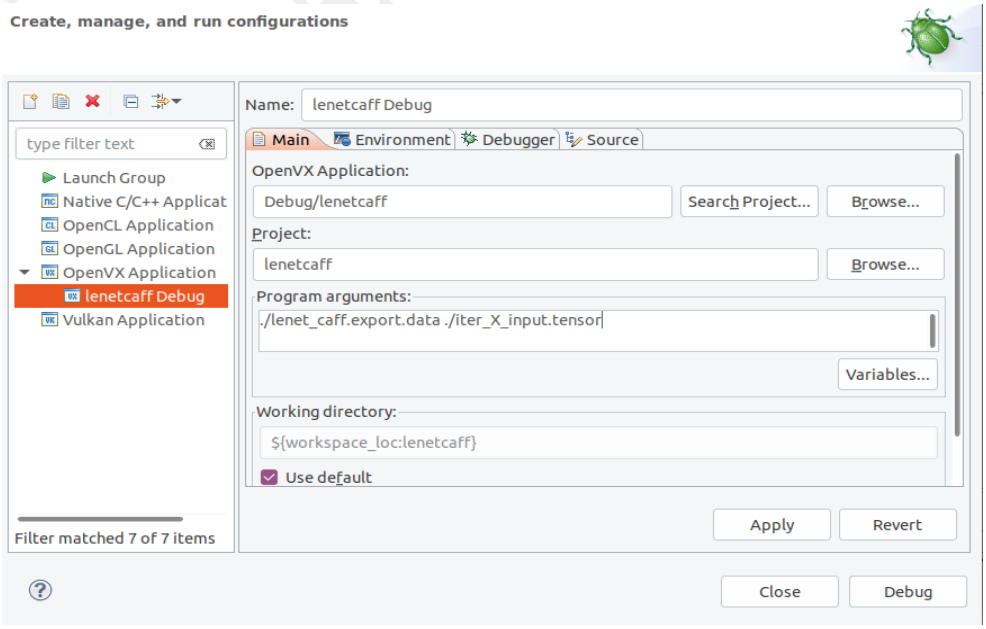
- The host meets the requirements as described in 2.1 System Requirements.
- The model is imported and compiled successfully.
- The NETINT AI Toolkits are installed correctly as described in 2.6 Verify the Development Environment.

Usage

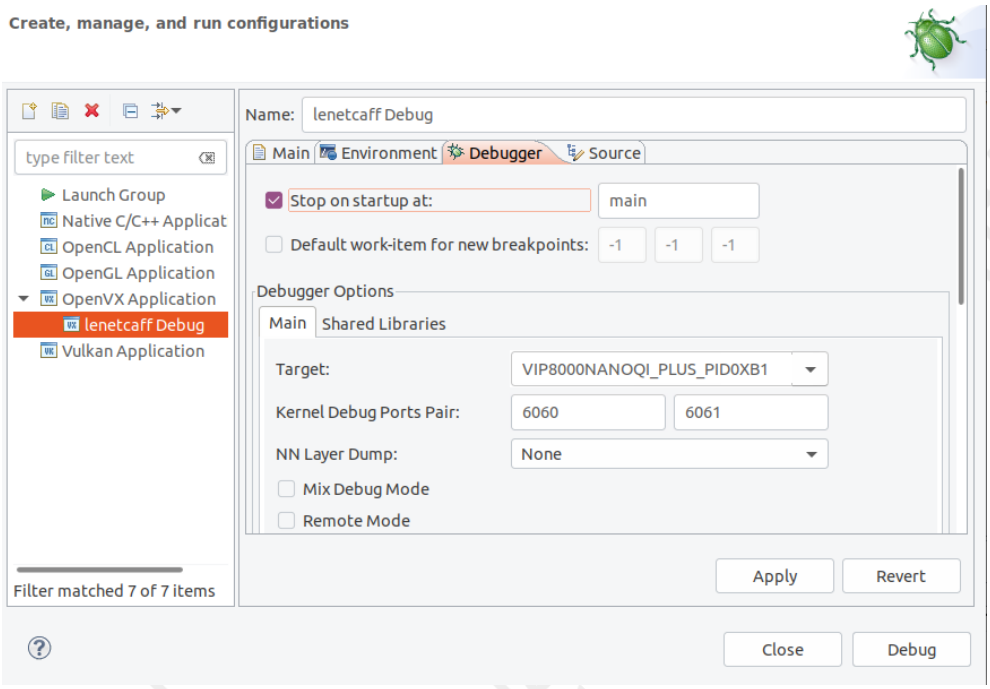
1. To open the Debug Configurations dialog, select from the main menu Run->Debug Configurations…



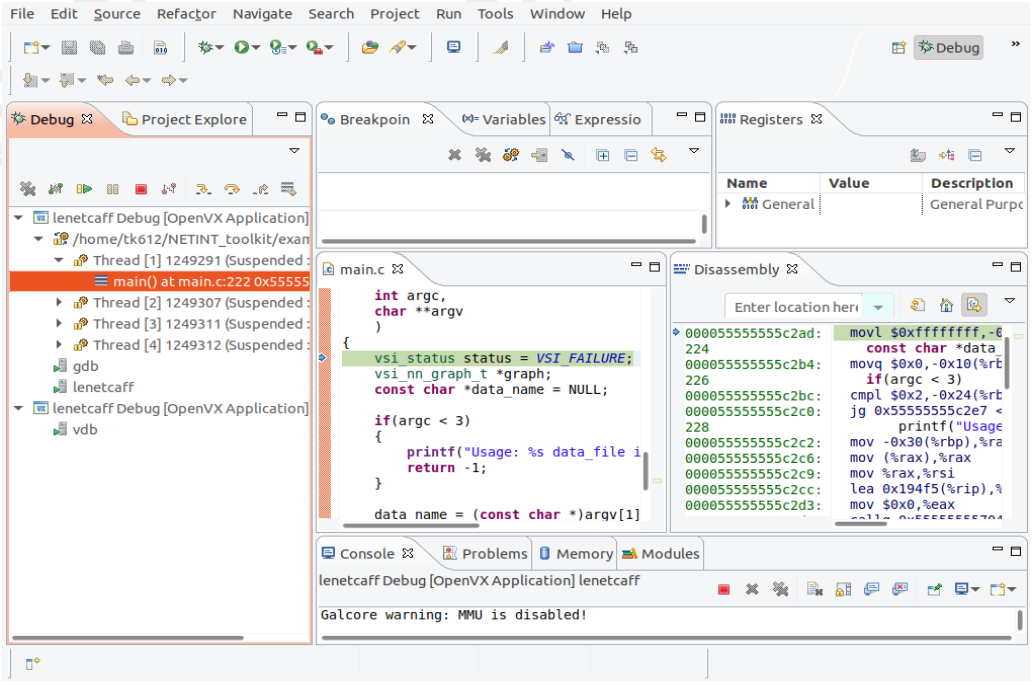
2. First, enter in Debug Configurations to setup basic configs. The OpenVX Application should be consistent with the Debug path. The Program arguments should setup model's data file, e.g., MODEL_uint8.export.data, and the input tensor file generated by the OpenVX model.



3. Second, enter in Debugger, the hardware Target should be setup properly, e.g., VIP8000NANOQI_PLUS_PID0XB1



4. Apply these configs and perform Debug.



5. After finishing the whole Debug procedure, the final output will be displayed in the Console.

A.4.3. Run Models

Description

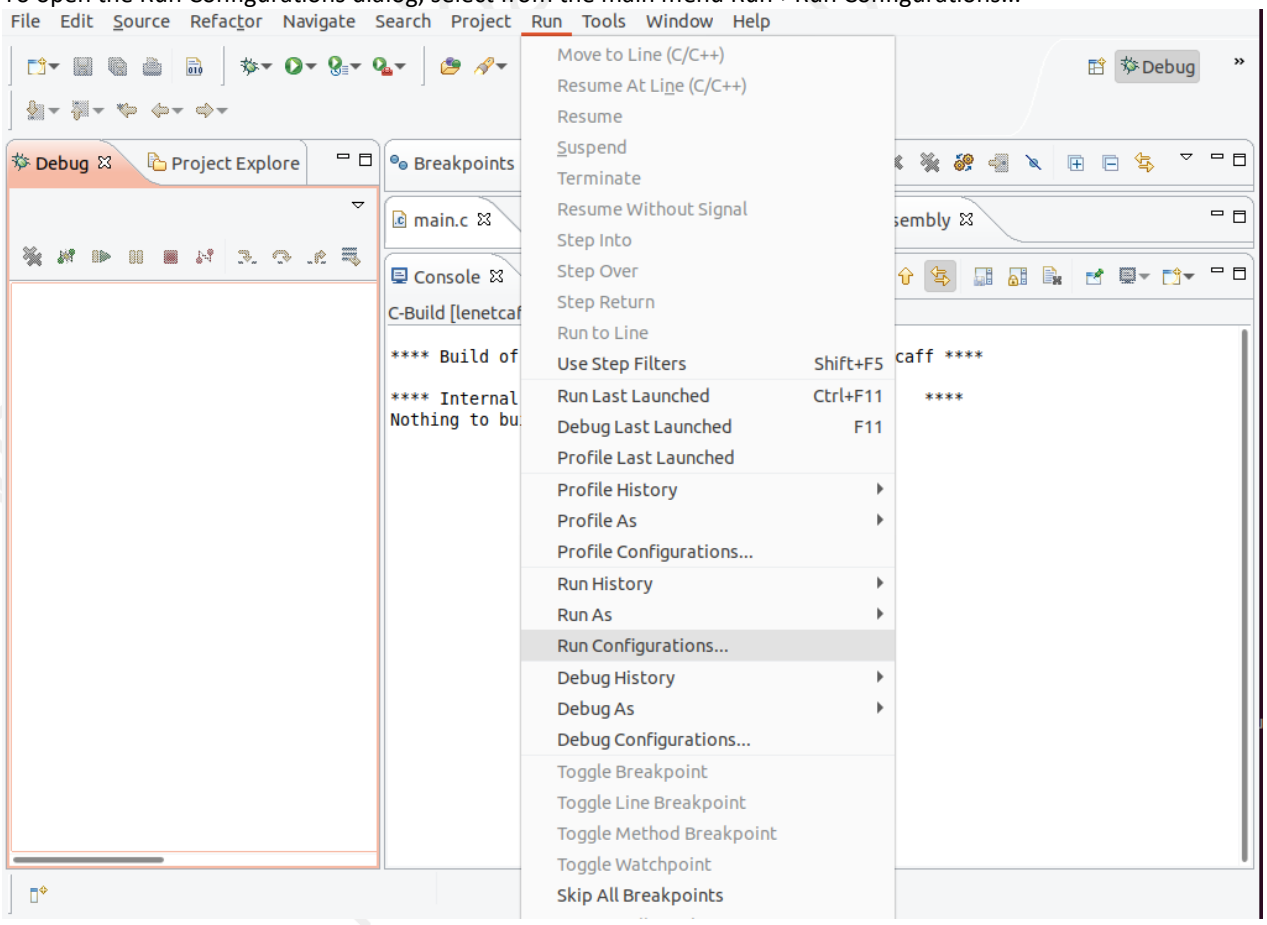
The IDE provides Run model feature for user.

Prerequisites

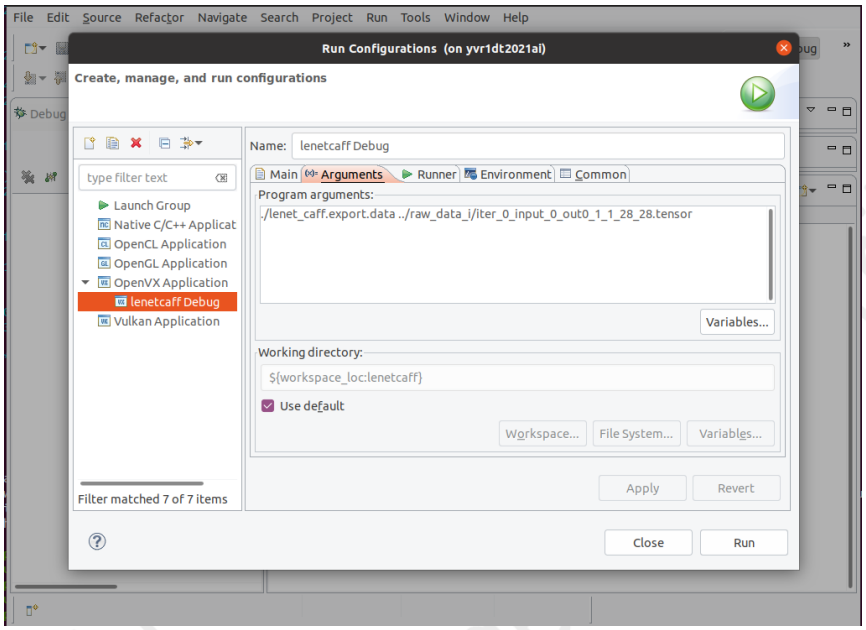
- The host meets the requirements as described in Section *1.4, System Requirements*.
- The NETINT AI Toolkits are installed correctly as described in 2.6 Verify the Development Environment.
- The model is imported and compiled successfully.

Usage

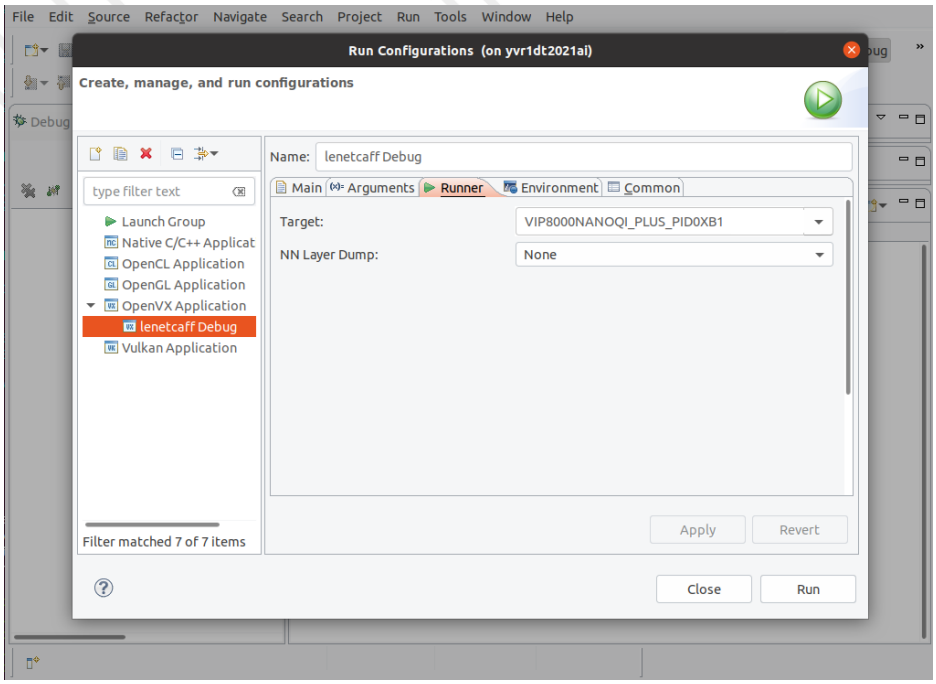
1. To open the Run Configurations dialog, select from the main menu Run->Run Configurations...



2. Enter in the Run Configurations. Same as Debug, the arguments should setup model's data file, e.g., MODEL_uint8.export.data, and the input tensor file.



3. In the Runner catalogue, the hardware Target should be same as Debug configs.



4. After finishing the run procedure, the final output will be displayed in the Console.

A.4.4. Profile Models

Description

The IDE provides Profile feature for user to review the bottleneck during inference.

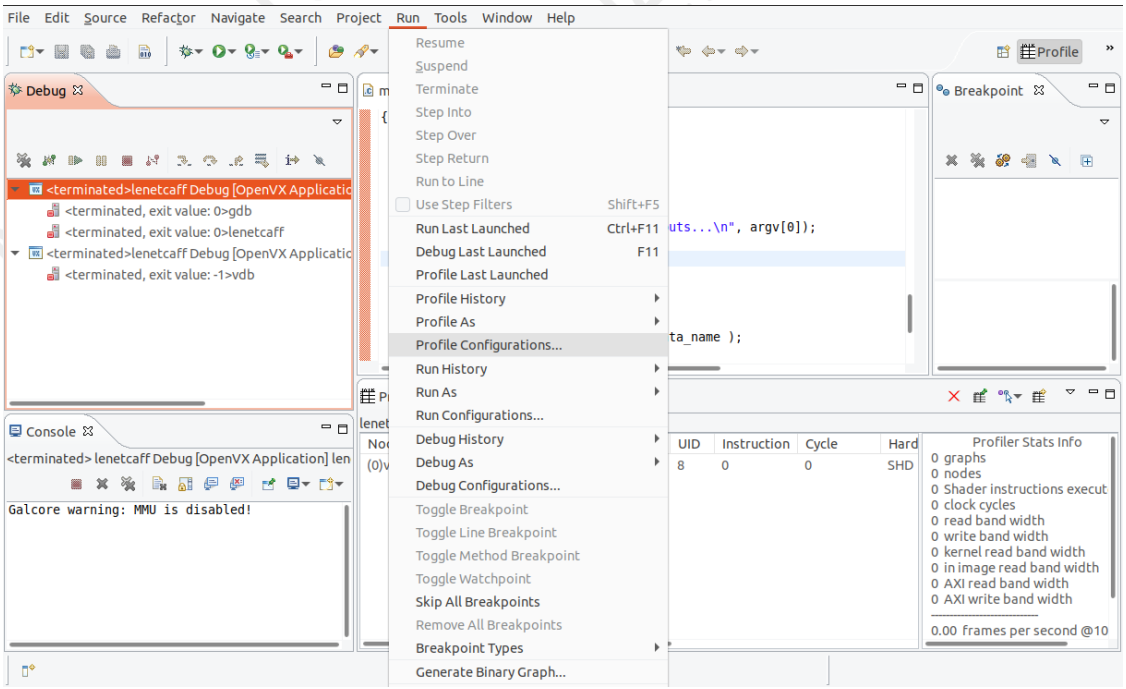
Note: The purpose of profiling is to identify the bottleneck of the inference. The final performance result may be different to the real performance on Quadra chip.

Prerequisites

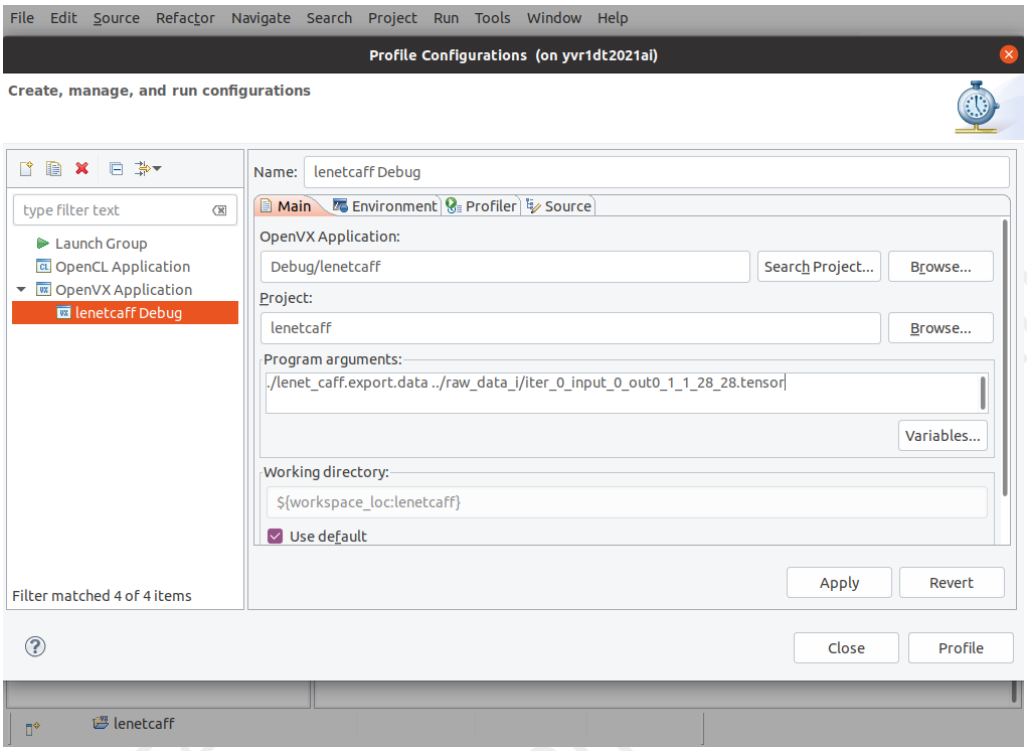
- The host meets the requirements as described in 2.1 System Requirements.
- The NETINT AI Toolkits are installed correctly as described in 2.6 Verify the Development Environment.

Usage

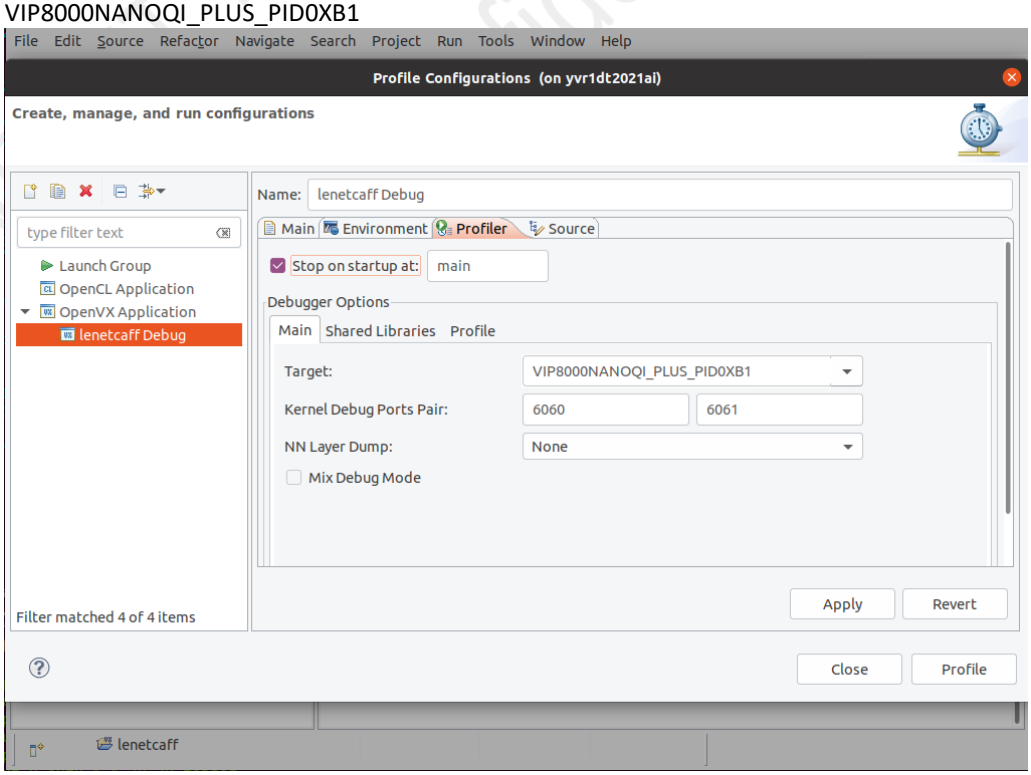
1. To open the Profile Configurations dialog, select from the main menu Run->Profile Configurations...



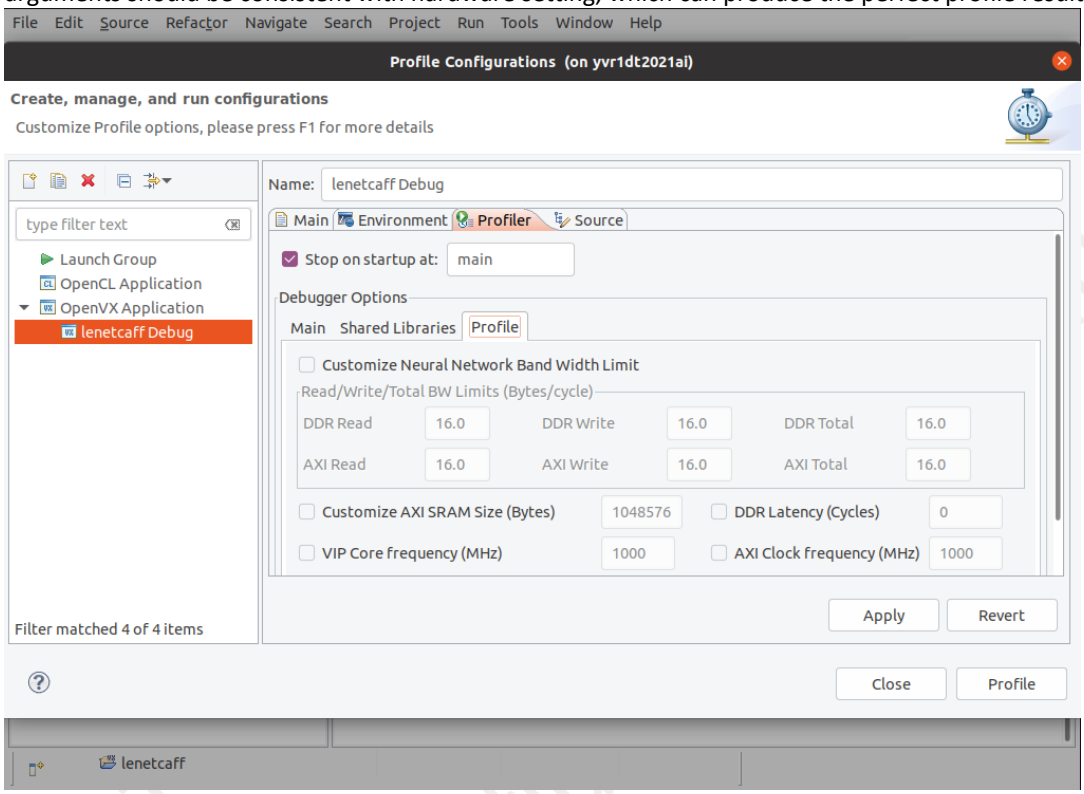
2. Enter in the Profile Configurations, same as Debug, the arguments should setup model's data file, e.g., MODEL.export.data, and the input tensor file generated by the OpenVX model.



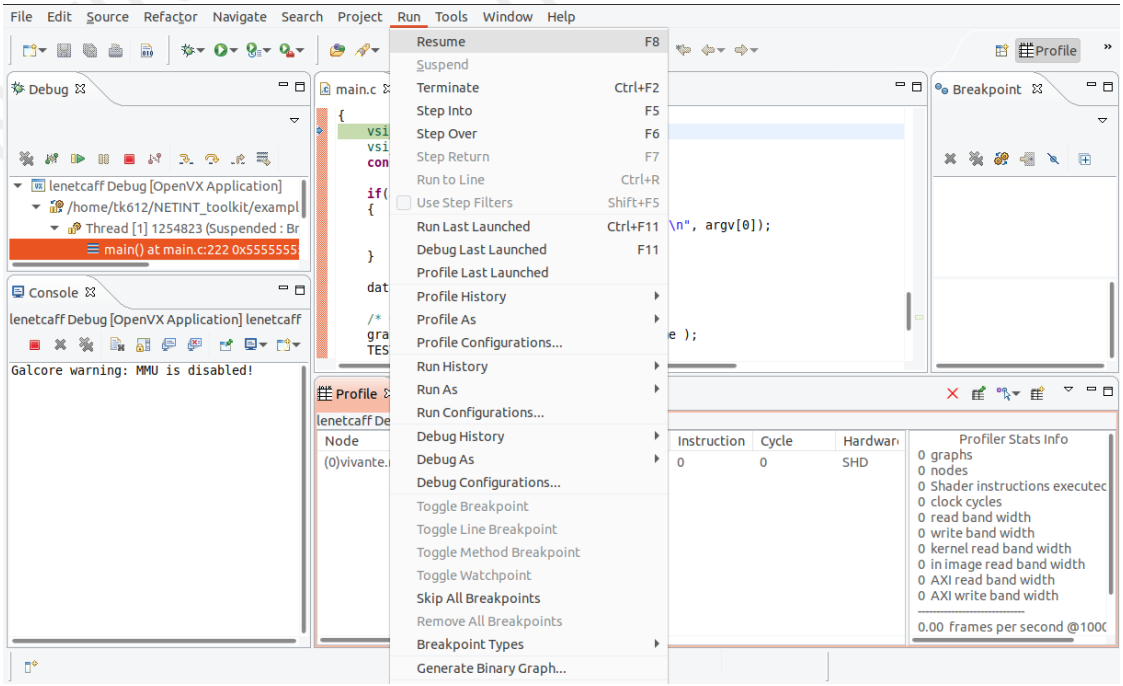
3. Enter in the Profiler catalogue, the hardware Target should be same as Debug and Run configs, e.g,. VIP8000NANOQI_PLUS_PID0XB1



4. In the Profile setting, there are some arguments related to hardware that can be modified. The profile arguments should be consistent with hardware setting, which can produce the perfect profile results.



5. Run profile. In this step, resume need to be performed when profile reaches a breakpoint.



6. Finishing the profile procedure, there are some results displayed in the Console as well as performance information summary displayed in 'Profiler Stats Info'. Also, per-layer performance information will be displayed in Profile category in the bottom.