Vivante Programming:

ACUITY Toolkit User Guide

Document Revision 0.94

25 March 2022

Compatible with Vivante ACUITY Toolkit Version 6.6.x

VERISILICON

LEVEL B: CONFIDENTIAL – DISTRIBUTION RESTRICTED

Legal Notices

For our current distributors, sales offices, design resource centers, and product information, visit our web page located at http://www.verisilicon.com.

For technical support, please email vivante-support@verisilicon.com.

Preface

The ACUITY Toolkit User Guide describes the offered tools and their usages in detail.

Abbreviations

Abbreviation	Full Name
API	Application Programming Interface
Blob	binary large object
CPU	central processing unit
GB	gigabyte
GPU	graphics processing unit
GRU	Gated Recurrent Unit
IDE	integrated development environment
KL divergence	Kullback-Laiber divergence
LSTM	Long Short Term Memory
NBG	network binary graph
NN	neural network
NPU	neural processing unit
ONNX	Open Neural Network Exchange
OS	operating system
OVX	OpenVX
PPU	parallel processing unit
protobuf	protocol buffer
RAM	random access memory
TFLite	TensorFlow Lite
VIP	Vision Image Processing

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: `Verisilicon_Tool_Acuity_Toolkit_<version>.tgz`
- Replaceable argument values in command lines and APIs 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.

References

The relevant documents for you to learn the mapping between NN frameworks and the Vivante OpenVX drivers and other useful websites are provided herein.

Vivante Documents

- *Vivante Programming: ACUITY Operation Mapping and Support*
- *Vivante Programming: Neural Network OVXLIB Operation Support with GPU*
- *Vivante Programming: Neural Network OVXLIB Operation Support with NPU*
- *Vivante Programming: Neural Network OVXLIB Operation Support with NPU V9.1.0*
- *Vivante Programming: Neural Network Runtime Overview*

Useful Websites

- www.ubuntu.com
- www.khronos.org
- www.tensorflow.org
- caffe.berkeleyvision.org

Table of Contents

1 Introduction

The Vivante ACUITY Toolkit provides both command line tools and APIs for you to easily deploy machine learning models onto devices with Vivante neural processing units (Vivante NPUs). They serve to translate model formats, optimize networks, perform model training, quantization, and inference until final integrations with devices.

This user guide describes the toolkit working principle, features, and the provided tools in detail.

1.1 Toolkit Overview

The ACUITY Toolkit provides both ready-made command lines and advanced APIs for you to choose to perform a fast model deployment.

These tools are:

- pegasus Command Line

  Ready-made command line tool, available in both Python and binary versions.
  - Python syntax example:
    ```
    python3 pegasus.py quantize -h
    ```
  - Binary syntax example:
    ```
    ./pegasus quantize -h
    ```
- VSInn API

  Enables you to create your own Python programs to deploy models.

See Also

2.1, Installing ACUITY Binary Version

2.2, Installing ACUITY Python Version

3, pegasus Command Line

4, VSInn API

1.2 Working Principle

You can use either the ACUITY command line tools or the ACUITY advanced VSInn APIs to deploy a machine learning model through the following stages:

1. Import and translate NN models into ACUITY formats.
2. (Optional) Optimize the generated ACUITY models until high-performance inference is achieved.

   You can use either command line tools or APIs to optimize the models by performing network pruning, model training, dumping, quantization, and inference.
3. Export the optimized ACUITY models for deployment use.
   - To deploy onto edge devices: Export as an OpenVX application and integrate it with the device OpenVX Vivante drivers.
   - To deploy onto embedded devices: Export as TensorFlow Lite (TFLite) models for use with Vivante Neutral Network Runtime.

     For details about Vivante Neutral Network Runtime, see *Vivante Programming: Neural Network Runtime Overview*, provided in the release package.

The following figure illustrates the model deployment workflow.

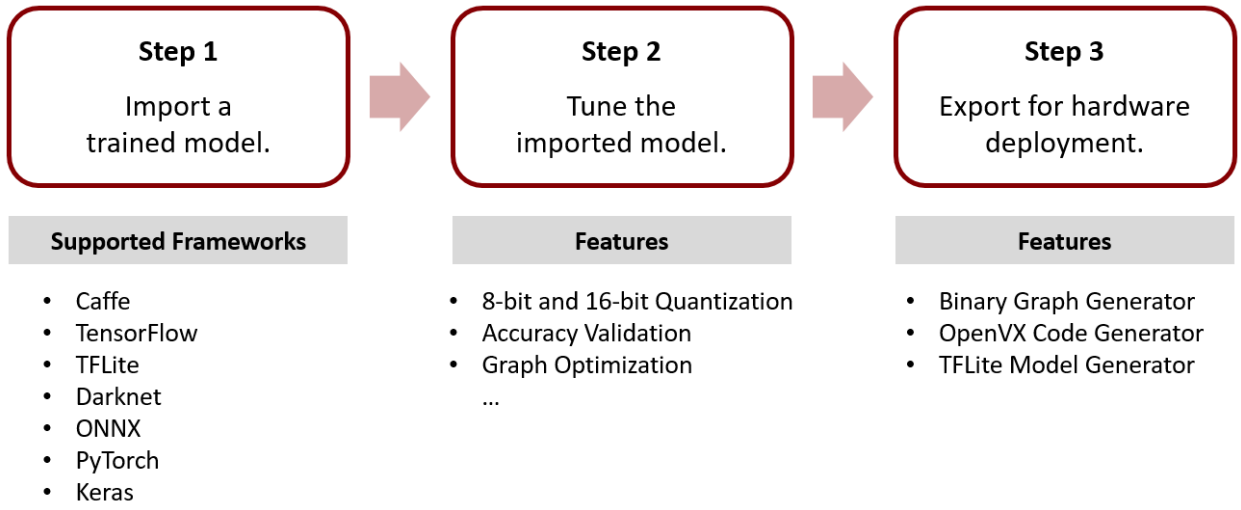

Figure 1-1 ACUITY Workflow

See Also

1.1, Toolkit Overview

1.3, Features

1.3 Features

The ACUITY Toolkit is built on TensorFlow™ and supports a wide range of NN framework format translations. Table 1-1 through Table 1-3 list the ACUITY features for input, NN compute, and output.

Table 1-1 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, and 2.6.x.
	TensorFlow Lite	Supports the TFLite schema from TensorFlow 2.3.0 (hash c4b29518884e16f8476ad92c75c3da48140a9eab). **Note:** TFLite models of other schemas may not be imported successfully for compatibility reasons.
	ONNX	Supports ONNX 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, and 2.6.x.
	Darknet	Supports Darknet models listed on https://pjreddie.com/darknet/.
Dataset format	Text	Supported by both command line tools and VSInn APIs.
	SQLite	Supported by command line tools.
	NumPy	Supported by the pegasus command line tool and VSInn APIs. For information about the pegasus, see Section *1.1, Toolkit Overview*.

Table 1-2 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 Signed int8 with the dynamic fixed point 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 1-3 Output Features

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

1.4 System Requirements

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

For more detailed requirements, see the README file provided with the release package.

Table 1-4 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. A compatible version of CUDA and cuDNN is installed for TensorFlow.
RAM	8 GB at least
Disk	160 GB
OS	Ubuntu 20.04 LTS 64-bit with Python 3.8 (recommended), Ubuntu 18.04 LTS 64-bit with Python 3.6, or Ubuntu 16.04 LTS 64-bit with Python 3.5 **Note:** Other Ubuntu versions are not recommended. **Important:** The support for Ubuntu 16.04 will be ended. Use the 20.04 version instead.

2 Working with the ACUITY Toolkit

This chapter describes the toolkit installation and usages of different tools in detail.

2.1 Installing ACUITY Binary Version

This task installs the binary package of the ACUITY Toolkit.

About This Task

The ACUITY Toolkit provides both binary and Python versions. Use the binary version only in a binary working environment. For development and Python releases, use the Python version instead.

Prerequisites

Make sure:

- Ubuntu 20.04 with Python 3.8, Ubuntu 18.04 with Python 3.6, or Ubuntu 16.04 with Python 3.5 is set up.
- The host meets the requirements as described in Section *1.4, System Requirements*.
- The VivanteIDE is installed correctly if you want to generate NBG cases from the ACUITY Toolkit.
  For details about VivanteIDE, see *VivanteIDE User Guide*, provided in the release package.

Procedure

1. Extract the ACUITY package, `Verisilicon_Tool_Acuity_Toolkit_<version>.tgz`, to a destination directory.
2. From this directory, locate and unpack the binary package `acuity-toolkit-binary-<version>`.
   When the unpacking completes, the installation finishes.

What to Do Next

1. Switch to the `./acuity-toolkit-binary-<version>/bin` directory.
2. Set the environment variable ACUITY_PATH as follows:

```
export ACUITY_PATH=<directory of acuity-toolkit-binary-version>/bin
```

3. Execute the ACUITY binary tools to perform model translations, optimizations, and deployments.
   For more details, see Section *2.3, Using Command Line Tools*.

See Also

2.2, Installing ACUITY Python Version

3, pegasus Command Line

2.2 Installing ACUITY Python Version

This task installs the TensorFlow and the ACUITY Toolkit Wheel package.

About This Task

The ACUITY Toolkit provides both Python and binary versions. For development and releases, use the ACUITY Python version. Use the binary version only in a binary working environment.

Prerequisites

Make sure:

- Python environment with a pip version from 20.0 to 20.3.4 is set up.
- The host meets the requirements as described in Section *1.4, System Requirements*.
- The VivanteIDE is installed correctly if you want to generate NBG cases from the ACUITY Toolkit.
  For details about VivanteIDE, see *VivanteIDE User Guide*, provided in the release package.
- TensorFlow is installed as described in Section *2.2.1, Installing TensorFlow*.
  For the supported TensorFlow versions, see Section *1.4, System Requirements*.

Procedure

1. Extract the ACUITY package, `Verisilicon_Tool_Acuity_Toolkit_<version>.tgz`, to a destination directory.
2. From this directory, locate and unpack the ACUITY wheel package `acuity-toolkit-whl-<version>`.
3. From the `./acuity-toolkit-whl-<version>/bin` directory, select either a CPU Wheel package or a GPU Wheel package to install.

   Note 1: You can only install either CPU or GPU Wheel package. Installation of both packages is not supported.

   Note 2: In the following `pip3` commands, if TensorFlow and other dependent libraries have been correctly installed, add the `--no-deps` command option to skip the dependence installation for an acceleration.

   - To use the host CPU for computing, install the CPU wheel package with the following commands:
     ```
     pip3 install ./acuity_bin_cpu-*.whl
     ```
   - To use the host GPU for computing, install the GPU wheel package with the following commands:
     ```
     pip3 install ./acuity_bin_gpu-*.whl
     ```
     For the supported GPUs, see Section *1.4, System Requirements*.

What to Do Next

From the `./acuity-toolkit-whl-<version>/bin` directory, execute the ACUITY Python commands to perform model translations, optimizations, and deployments onto the Vivante NPU drivers.

For more details about the ACUITY Python tools, see Section *2.3, Using Command Line Tools* and Chapter *4, VSInn API*.

See Also

1.2, Working Principle

2.1, Installing ACUITY Binary Version

3, pegasus Command Line

2.2.1 Installing TensorFlow

This task uses a series of commands to install the used TensorFlow version.

For the supported TensorFlow versions, see Section *1.4, System Requirements*.

About This Task

The ACUITY Toolkit Python version requires a TensorFlow installation. For more details, see Section *2.2, Installing ACUITY Python Version*.

Procedure

Use the following commands to install the used TensorFlow version.

```
pip3 install tensorflow==2.6.0
pip3 install networkx>=1.11
pip3 install lmdb==0.93
pip3 install onnx==1.10.2
pip3 install dill==0.2.8.2
pip3 install ruamel.yaml==0.15.81
pip3 install ply==3.11
pip3 install torch==1.5.1
```

What to Do Next

Install the ACUITY Wheel package as described in Section *2.2, Installing ACUITY Python Version*.

2.3 Using Command Line Tools

You can use the ACUITY offered command line tools to easily deploy an NN model onto devices with Vivante NPUs. They help you in model translations, optimizations, and integrations with the Vivante NPU drivers.

The available command line tool is pegasus, which supports multiple input layers and flexible input tensor shapes.

About This Task

ACUITY command line tools support both binary and Python commands. Use either of them according to your working environment. Choose Python especially during development.

Alternatively, you can also use the ACUITY advanced VSInn APIs to create custom Python scripts for a model deployment. For more details, see Section *2.4, Programming with VSInn APIs*.

Before You Begin

Make sure:

- The ACUITY Toolkit is installed successfully.
- Your machine learning NN model and datasets for training and inference are prepared.

  For the supported NN frameworks, see Section *1.3, Features*.
- For recurrent neutral network LSTM and GRU models, you must set up the models as described in Section *2.3.1, Training LSTM Models* and Section *2.3.2, Training GRU Models*.

Procedure

1. Switch to the commands execution directory:
   - For binary commands: `<directory of acuity-toolkit-binary-version>/bin`
   - For Python commands: `<directory of acuity-toolkit-whl-version>/bin`
2. For binary, skip to the last step to execute commands directly.

   For pegasus Python commands, enable the intelligent code completion as follows, which automatically populates the supported commands or arguments in the current command line after you press the **Tab** key:

   Note: After the following setup, the code completion is valid only if you enter commands in the syntax of `pegasus.py <command> <arguments>`. It is invalid when entering `python3 pegasus.py <command> <arguments>`.

   a. Enter the following command to generate a `pegasus_completion` file:

   ```
   python3 ./pegasus.py completion
   ```

   b. Use the generated `pegasus_completion` file to enable the intelligent code completion as follows:
      – To enable it only for the current console, enter the `source pegasus_completion` command.
      – To enable it globally, copy the `pegasus_completion` file to the `/etc/bash_completion.d` directory.
3. Enter pegasus commands to perform a model deployment in the following sequence:

   a. Import and translate the input model to ACUITY formats.

   The generated files are:

   `.data`: Contains the model network coefficients.

   `.json`: Contains the model network connections.

   For the mapping between the supported NN frameworks and ACUITY, see *Vivante Programming: ACUITY Operation Mapping and Support,* provided in the release package.

b. Set up inputmeta and post-processing configuration files.

   Inputmeta is required by train, dump, quantize, inference, and export actions.

   Post-processing is required by the inference and export actions.

c. (Optional) Optimize the model through model training, quantizing, until high performance inferring achieved.

   Note: For quantized TensorFlow and TensorFlow Lite models, do not quantize them again in ACUITY. Use their provided quantization files directly for network dumping, inference, and export.

d. Export the ACUITY model into the following formats to support the needed deployment:
   - For edge device deployments: Export to an OpenVX application integrated with the Vivante NPU drivers.

     The output file types are:

     `.c`, `.h`, and `.export.data`.

     Or,

     `.c`, `.h`, and `.nbg`, if packing binary graphs is enabled.
   - For embedded device deployments: Export to TFLite formats for use by the Vivante NN Runtime.

     The output file type is `.tflite`.

Note:

- Example to execute binary commands: `./pegasus import caffe -h`
- Example to execute Python commands: `python3 pegasus.py import caffe -h`
- Example to execute pegasus Python commands with the code completion enabled: `pegasus.py import caffe -h`

What to Do Next

- For the exported OpenVX applications, deploy them onto the devices.
- For the exported TFLite models, deploy them onto the embedded devices with the Vivante NN Runtime support.

  For details about Vivante Neutral Network Runtime, see *Vivante Programming: Neural Network Runtime Overview*, provided in the release package.

Example

You can use the offered sample models to practice model deployments with the ACUITY command line tools:

- Single input model

  Resides in the `<directory of acuity-toolkit-binary/whl-version>/lenet` directory.
- Multi-input model

  Resides in the `<directory of acuity-toolkit-binary/whl-version>/pegasus_samples` directory.

See Also

2.1, Installing ACUITY Binary Version

2.2, Installing ACUITY Python Version

3, pegasus Command Line

2.3.1 Training LSTM Models

If LSTM models are used, use TensorFlow to train the LSTM models first. After TensorFlow training, two types of LSTM models are available:

- LSTM models with all timesteps output
- LSTM models with only the last timestep output

Note: The current release of ACUITY Toolkit can only train classification networks. It supports the LSTM models that are trained by TensorFlow 1.13.2.

2.3.1.1 LSTM Training with All Timesteps Output

To train an LSTM model with all timesteps output, refer to the following example:

```
import os
import tensorflow
import tensorflow.examples.tutorials.mnist as mnist
main_version = tensorflow.__version__.split('.')[0]
if int(main_version) == 2:
    import tensorflow.compat.v1 as tf
    tf.disable_eager_execution()
else:
    import tensorflow as tf

data_set_path = os.path.join('..', 'DATASET', 'MNIST', 'gz')
mnist_data = mnist.input_data.read_data_sets(data_set_path, one_hot=True)

time_steps = 28
num_units = 128
n_input = 28
learning_rate = 0.001
n_classes = 10
batch_size = 128

out_weights = tf.Variable(tf.random_normal([num_units, n_classes]))
out_bias = tf.Variable(tf.random_normal([n_classes]))

x = tf.placeholder(tf.float32, [None, time_steps, n_input, 1], name = 'x')
y = tf.placeholder(tf.float32, [None, n_classes])

input = tf.reshape(x, [tf.shape(x)[0], time_steps, n_input])

fw_cell = tf.nn.rnn_cell.LSTMCell(num_units, forget_bias=1.0)
outputs, _ = tf.nn.dynamic_rnn(fw_cell, input, dtype=tf.float32)

outputs = tf.slice(outputs, begin=[0, tf.shape(outputs)[1]-1, 0],
size=[tf.shape(outputs)[0], 1, 128])
outputs = tf.reshape(outputs, [tf.shape(outputs)[0], -1])

outputs = tf.matmul(outputs, out_weights)
prediction = tf.nn.bias_add(outputs, out_bias, name='logits')
```

```
#loss function
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=prediction,
labels=y))

#optimization
opt = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
saver = tf.train.Saver(max_to_keep=4)

#model evaluation
correct_prediction = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

#initialize variables
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    iter = 1
    while iter < 1800:
        batch_x, batch_y = mnist_data.train.next_batch(batch_size=batch_size)
        print(batch_x.shape, batch_y.shape)
        batch_x = batch_x.reshape((batch_size, time_steps, n_input, 1))

        sess.run(opt, feed_dict={x:batch_x, y:batch_y})

        if iter % 10 == 0:
            acc = sess.run(accuracy, feed_dict={x:batch_x, y:batch_y})
            los = sess.run(loss, feed_dict={x:batch_x, y:batch_y})
            print("For iter ", iter)
            print("Accuracy", acc)
            print("Loss", los)
            print("__________________________")

        if iter % 100 == 0:
            saver.save(sess, os.path.join('.', 'lstm_mnist'), global_step=iter)
            tf.train.write_graph(sess.graph_def, '.', 'lstm_mnist_dynamic.pbtxt')
```

2.3.1.2 LSTM Training with Last Timestep Output

To train an LSTM model with the last timestep output, refer to the following example:

```
out_weights = tf.Variable(tf.random_normal([num_units, n_classes]))
out_bias = tf.Variable(tf.random_normal([n_classes]))

x = tf.placeholder(tf.float32, [None, time_steps, n_input, 1], name = 'x')
y = tf.placeholder(tf.float32, [None, n_classes])

input = tf.reshape(x, [tf.shape(x)[0], time_steps, n_input])

fw_cell = tf.nn.rnn_cell.LSTMCell(num_units, forget_bias=1.0)
outputs, last = tf.nn.dynamic_rnn(fw_cell, input, dtype=tf.float32)

outputs = tf.matmul(last[-1], out_weights)
prediction = tf.nn.bias_add(outputs, out_bias, name='logits')

#loss function
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=prediction,
labels=y))
#optimization
opt = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
saver = tf.train.Saver(max_to_keep=4)

#model evaluation
correct_prediction = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

2.3.2 Training GRU Models

If GRU models are used, use TensorFlow to train the GRU models first. After TensorFlow training, three types of GRU models are available:

- GRU models with the last timestep HSTATE
- GRU models with outputs
- GRU models with outputs and the last timestep HSTATE

Note: The current release of ACUITY Toolkit can only train classification networks. It supports the G models that are trained by TensorFlow 1.13.2.

2.3.2.1 GRU Training with Last Timestep HSTATE

To train a GRU model with the last timestep HSTATE, refer to the following example:

```
import os
import tensorflow
import tensorflow.examples.tutorials.mnist as mnist

main_version = tensorflow.__version__.split('.')[0]
if int(main_version) == 2:
    import tensorflow.compat.v1 as tf
    tf.disable_eager_execution()
else:
    import tensorflow as tf

print("********tensorflow {} version *********".format(tensorflow.__version__))

data_set_path = os.path.join('..', 'DATASET', 'MNIST', 'gz')
mnist_data = mnist.input_data.read_data_sets(data_set_path, one_hot=True)

print(type(mnist_data))


time_steps = 28
num_units = 128
n_input = 28
learning_rate = 0.001
n_classes = 10
batch_size = 128

out_weights = tf.Variable(tf.random_normal([num_units, n_classes]))
out_bias = tf.Variable(tf.random_normal([n_classes]))

x = tf.placeholder(tf.float32, [None, time_steps, n_input, 1], name = 'x')
y = tf.placeholder(tf.float32, [None, n_classes])

input = tf.reshape(x, [tf.shape(x)[0], time_steps, n_input])

cell = tf.nn.rnn_cell.GRUCell(num_units, dtype=tf.float32)
outputs, hstate= tf.nn.dynamic_rnn(cell, input, dtype=tf.float32)

outputs = tf.matmul(hstate, out_weights)
prediction = tf.nn.bias_add(outputs, out_bias, name='logits')

#loss function
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=prediction,
labels=y))
#optimization
opt = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
saver = tf.train.Saver(max_to_keep=4)

#model evaluation
correct_prediction = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
#initialize variables
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    iter = 1
    while iter < 1800:
        batch_x, batch_y = mnist_data.train.next_batch(batch_size=batch_size)
        print(batch_x.shape, batch_y.shape)
        batch_x = batch_x.reshape((batch_size, time_steps, n_input, 1))

        sess.run(opt, feed_dict={x:batch_x, y:batch_y})

        if iter % 10 == 0:
            acc = sess.run(accuracy, feed_dict={x:batch_x, y:batch_y})
            los = sess.run(loss, feed_dict={x:batch_x, y:batch_y})
            print("For iter ", iter)
            print("Accuracy", acc)
            print("Loss", los)
            print("__________________________")

        if iter % 100 == 0:
            saver.save(sess, os.path.join('.', 'gru_mnist'), global_step=iter)
            tf.train.write_graph(sess.graph_def, '.', 'gru_mnist_dynamic.pbtxt')

        iter = iter + 1
```

2.3.2.2 GRU Training with Outputs

To train a GRU model with outputs, refer to the following example:

```
import os
import tensorflow
import tensorflow.examples.tutorials.mnist as mnist
main_version = tensorflow.__version__.split('.')[0]
if int(main_version) == 2:
    import tensorflow.compat.v1 as tf
    tf.disable_eager_execution()
else:
    import tensorflow as tf

print("********tensorflow {} version *********".format(tensorflow.__version__))

data_set_path = os.path.join('..', 'DATASET', 'MNIST', 'gz')
mnist_data = mnist.input_data.read_data_sets(data_set_path, one_hot=True)

print(type(mnist_data))

time_steps = 28
num_units = 128
n_input = 28
learning_rate = 0.001
n_classes = 10
batch_size = 128

out_weights = tf.Variable(tf.random_normal([num_units, n_classes]))
out_bias = tf.Variable(tf.random_normal([n_classes]))

x = tf.placeholder(tf.float32, [None, time_steps, n_input, 1], name = 'x')
y = tf.placeholder(tf.float32, [None, n_classes])

input = tf.reshape(x, [tf.shape(x)[0], time_steps, n_input])

cell = tf.nn.rnn_cell.GRUCell(num_units, dtype=tf.float32)
outputs, _ = tf.nn.dynamic_rnn(cell, input, dtype=tf.float32)

outputs = tf.slice(outputs, begin=[0, tf.shape(outputs)[1]-1, 0],
size=[tf.shape(outputs)[0], 1, 128])
outputs = tf.reshape(outputs, [tf.shape(outputs)[0], -1])

outputs = tf.matmul(outputs, out_weights)
prediction = tf.nn.bias_add(outputs, out_bias, name='logits')
```

2.3.2.3 GRU Training with Outputs and Last Timestep HSTATE

To train a GRU model with outputs and the last timestep HSTATE, refer to the following example:

```
import os
import tensorflow
import tensorflow.examples.tutorials.mnist as mnist
main_version = tensorflow.__version__.split('.')[0]
if int(main_version) == 2:
    import tensorflow.compat.v1 as tf
    tf.disable_eager_execution()
else:
    import tensorflow as tf

print("********tensorflow {} version *********".format(tensorflow.__version__))

data_set_path = os.path.join('..', 'DATASET', 'MNIST', 'gz')
mnist_data = mnist.input_data.read_data_sets(data_set_path, one_hot=True)

print(type(mnist_data))

time_steps = 28
num_units = 128
n_input = 28
learning_rate = 0.001
n_classes = 10
batch_size = 128

out_weights = tf.Variable(tf.random_normal([num_units, n_classes]))
out_bias = tf.Variable(tf.random_normal([n_classes]))

x = tf.placeholder(tf.float32, [None, time_steps, n_input, 1], name = 'x')
y = tf.placeholder(tf.float32, [None, n_classes])

input = tf.reshape(x, [tf.shape(x)[0], time_steps, n_input])

cell = tf.nn.rnn_cell.GRUCell(num_units, dtype=tf.float32)
outputs, hstate = tf.nn.dynamic_rnn(cell, input, dtype=tf.float32)

outputs = tf.slice(outputs, begin=[0, tf.shape(outputs)[1]-1, 0],
size=[tf.shape(outputs)[0], 1, 128])
outputs = tf.reshape(outputs, [tf.shape(outputs)[0], -1])

outputs = tf.matmul(outputs, out_weights)
prediction = tf.nn.bias_add(outputs, out_bias, name='logits')
hstate = tf.identity(hstate, name='hstate')
```

2.3.3 Saving Frozen PB and H5 Models

You can use TensorFlow to save frozen PB and H5 models.

To convert a model into frozen H5 and save the frozen H5 model by using TensorFlow 2.3.0, refer to the following example:

```
# Imports
import tensorflow as tf
import datetime
import numpy as np
from tensorflow.python.framework.convert_to_constants import
convert_variables_to_constants_v2

# Define model
def create_model ():
  return tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1), name='input'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax', name='output')
  ])

# Load MNIST dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data ()

# Add a new axis
x_train = x_train[:, :, :, np.newaxis]
x_test = x_test[:, :, :, np.newaxis]

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print(x_train[0].shape, 'image shape')

# Normalize
x_train = x_train / 255.0
x_test = x_test / 255.0

# Create model
model = create_model ()

# Compile model
model.compile (
              optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy']
             )

# Tensorboard logging
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

```
# Model summary
model.summary ()

# Fit model
model.fit (
               x = x_train,
               y = y_train,
               epochs = 5,
               validation_data = (x_test, y_test),
               callbacks = [tensorboard_callback]
          )

# Save model
model.save ('saved_model')

# Convert to Frozen model and save
full_model = tf.function(lambda x: model(x))
full_model = full_model.get_concrete_function(tf.TensorSpec(model.inputs[0].shape,
model.inputs[0].dtype, name="input"))
frozen_model = convert_variables_to_constants_v2(full_model)
frozen_model.graph.as_graph_def()
tf.io.write_graph(graph_or_graph_def=frozen_model.graph, logdir="./frozen_models",
name="frozen_model.pb",as_text=False)

model.save ('my_model.h5')
```

2.4 Programming with VSInn APIs

This section walks you through using the VSInn APIs to create a Python script for a model deployment onto devices with the Vivante NPUs.

About This Task

This advanced method offers more flexibility, productivity, and reusability across different models on various devices.

However, if you are not familiar with programming, ACUITY also offers ready-made command line tools for your easy model deployment. For more details, see Section *2.3, Using Command Line Tools*.

Before You Begin

Make sure:

- The ACUITY Toolkit Wheel package is installed successfully.
  For details, see Section *2.2, Installing ACUITY Python Version*.
- A Python script is prepared.
- For recurrent neutral network LSTM and GRU models, you must follow the LSTM and GRU examples provided in Section *3.5, pegasus Use Cases* to set up the model.

Procedure

1. Import the `VSInn` class as follows:
   ```
   from acuitylib import VSInn
   ```
2. Create a `VSInn` object with the `VSInn()` API as described in Section *4.1, VSInn Object Creation*.
3. Create an AcuityNet object by either translated from a high-level machine learning model or setting up an ACUITY model.
   For more details, see the model conversion APIs in Section *4.2, ACUITY Model Conversions*, and the model creation API in Section *4.3, ACUITY Network Creation*.
4. Configure inputmeta and post-processing tasks with the APIs in Section *2.4, Programming with VSInn APIs*.
   Inputmeta prepares datasets and defines pre-processing tasks for dump, quantize, inference, and export actions.
   Post-processing defines post-processing tasks for inference and export actions.
5. Set up quantization, inference, dump, and network pruning as needed.
   For the associated APIs, see Section *4.8, Inference and Export*.
   Note: For quantized TensorFlow and TensorFlow Lite models, do not quantize them again in ACUITY. Use their provided quantization files directly.
6. Prepare export for a model deployment.
   - To export the AcuityNet object to an OpenVX application for edge device deployment, use the API described in Section *4.8.3, Export OVXLIB*.
   - To export to a TFLite model for embedded device deployment, use the API described in Section *4.8.4, Export TFLite*.

What to Do Next

Execute the Python script for a model deployment in the following sequence:

1. Execute your script as follows:
   ```
   python3 <script name>.py
   ```
2. From the destination folder, examine the generated files for your model deployment either onto edge devices or onto embedded devices.
   - For exported OpenVX applications, the output files contain:
     - `.c`, `.h`, and `.export.data`.

       Or,
     - `.c`, `.h`, and `.nbg`, if packing binary graphs is enabled.
   - For exported TFLite models, the output file is `.tflite`.

Example

For a complete script example, see Section *4.9, Example of VSInn Programming*.

See Also

2.2, Installing ACUITY Python Version

4, VSInn API

3.5, pegasus Use Cases

3 pegasus Command Line

The pegasus command line tool provides a set of commands for you to perform model translations, optimizations, inferences, and final deployments. This chapter describes the command syntax, arguments, and usage examples.

For how to use the commands to perform tasks, see Section *2.3, Using Command Line Tools*.

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 For details about the supported frameworks, see Section *1.3, Features*.
Network pre-processing and post-processing	pegasus.py generate inputmeta pegasus.py generate postprocess-file pegasus.py generate fakedata
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 tflit

3.1 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.

3.2 pegasus Workflow

For float models, the pegasus typical use sequence is:

1. Import and translate an NN model to ACUITY formats.
2. Generate inputmeta files.
3. Quantize this generated ACUITY model.
4. Inference the ACUITY model with a float data type or with a quantization data type.
5. Export the float application or the quantized application for device deployment.

For quantized models, the typical pegasus sequence is:

1. Import and translate an NN model to ACUITY formats.
2. Generate inputmeta files.
3. Inference the ACUITY model with the quantization data type.
4. Export the quantized application for device deployment.

In quantization, inference, validation, and test stages, you may need to modify the generated `inputmeta.yml` file according to the pre-processing datasets for each input layer. You can also create a `postprocess.yml` file to define the post-processing tasks for each output layer during inference and for export.

3.3 Dataset Formats

pegasus supports three dataset formats, which are text files, NumPy binary files, and SQLite database files. This section describes the definitions of these formats in detail.

3.3.1 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 constitutes the dataset of an input port. The order of the columns must follow the input port order specified in the `ports` parameter in the `inputmeta.yml` file.

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

Examples

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

```
./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 ports 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 ports.

Elements in each row are separated by spaces. The order of the columns must follow the input port order specified in the `ports` parameter in the `inputmeta.yml` file, as specified in Section *3.5.1, Example: Pre-processing Setups*.

```
./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
```

3.3.2 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 port. Each port 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 port requires Shape A while the array in a NumPy binary file follows Shape B. To assign such file to this input port, 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.

    Or,
  - 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 port needs a separate NumPy database. The databases must not be shared across ports, which means that an `.npy` file must be exclusively used by only one port ID.

Examples

Example 1: An input port that requires four-dimension 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: An input port that requires three-dimension 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)
```

3.3.3 SQLite Database File (.dsx)

Refer to *www.sqlite.org* for SQLite storage formats.

3.4 Command Reference

This section explicates pegasus commands in functions, syntax, and argument descriptions.

3.4.1 Import Caffe

Description

Translates a Caffe model to ACUITY 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, a `.prototxt` file.
[--weights]	Requires a string value to denote the file path of the imported weights file, a `.caffemodel` file. For example, `'mobilenet.caffemodel'`. If the weights file does not exist, the system generates a `.data` file which contains fake data.
[-h]	Displays help information.
[--output-model]	Requires a string value to denote the file path of the generated ACUITY 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 ACUITY 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 Caffe model>/<model name>.data`.

Argument	Description
[--proto]	Requires a string value to represent the protocol used by the imported Caffe model. Set this argument to one of the following values: • `'caffe'`: (Default) Represents the standard Caffe format protocol. • `'lstm_caffe'`: Represents the LSTM layer protocol. • `'<other protocol name>'`: For another protocol with its file `<xxx>_pb2.py`, specify the absolute path of the file as the value of `--proto`. When this argument is omitted, the system uses the default value.

Example

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

3.4.2 Import TensorFlow

Description

Translates a TensorFlow model to ACUITY 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 ACUITY.

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

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, a `.pb` file. **Note:** Models trained and frozen by the following TensorFlow versions have been proven compatible with ACUITY: 1.4.x, 2.0.x, and 2.3.x.
--inputs (Required)	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'`.
--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'`.

Argument	Description
[-h]	Displays help information.
[--output-model]	Requires a string value to denote the file path of the generated ACUITY 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 ACUITY 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. `True` or `False` for different input points are separated with commas and enclosed by ''. For example, `'True,False,True'`. When this argument is omitted, the system uses `False` for all input points listed. **Example 1:** The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the `--input-size-list` argument are `'243,243,3'`, set `--size-with-batch` to `'False'`. **Example 2:** Three input ports with only one input point at each port. The tensor shape sizes of each input point are (?, 243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the `--input-size-list` argument are `'243,243,3#88#10,7'`, set `--size-with-batch` to `'False,False,True'`.
[--mean-values]	Requires a string to specify the mean value of each input point listed in the `--inputs` argument. Multiple mean values are separated with commas. For example, `'128.0,128.0'`. **Note:** Specify this argument only for quantized TensorFlow models.
[--std-values]	Requires a string to specify the standard value of each input point listed in the `--inputs` argument. Multiple standard values are separated with commas. For example, `'128.0,128.0'`. **Note:** Specify this argument only for quantized TensorFlow models.

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

3.4.3 Import TFLite

Description

Translates a TensorFlow Lite model to ACUITY 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 in ACUITY.

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

Syntax

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

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, a `.tflite` file. The supported TFLite schema is 0c4f5dfea4ceb3d7c0b46fc04828420a344f7598, committed on https://github.com/tensorflow/tensorflow/commits/master/tensorflow/lite/schema/schema.fbs. **Note:** Import failures may occur if the TFLite model uses an unsupported TFLite schema.
[-h]	Displays help information.
[--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.

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. `True` or `False` for different input points are separated with commas and enclosed by ''. For example, `'True,False,True'`. When this argument is omitted, the system uses `False` for all input points listed. **Example 1:** The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the `--input-size-list` argument are `'243,243,3'`, set `--size-with-batch` to `'False'`. **Example 2:** Three input ports with only one input point at each port. The tensor shape sizes of each input point are (?, 243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the `--input-size-list` argument are `'243,243,3#88#10,7'`, set `--size-with-batch` to `'False,False,True'`.
[--output-model]	Requires a string value to denote the file path of the generated ACUITY 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.
[--output-data]	Requires a string value to denote the file path of the generated ACUITY 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.

3.4.4 Import Darknet

Description

Translates a Darknet model to ACUITY 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 imported weights file, a `.weights` file. For example, `'mobilenet.weights'`.
[-h]	Displays help information.
[--output-model]	Requires a string value to denote the file path of the generated ACUITY 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 ACUITY 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`.

3.4.5 Import ONNX

Description

Translates an ONNX model to ACUITY 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 15 are supported.
[-h]	Displays help information.
[--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. `True` or `False` for different input points are separated with commas and enclosed by ''. For example, `'True,False,True'`. When this argument is omitted, the system uses `False` for all input points listed. **Example 1:** The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the `--input-size-list` argument are `'243,243,3'`, set `--size-with-batch` to `'False'`. **Example 2:** Three input ports with only one input point at each port. The tensor shape sizes of each input point are (?, 243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the `--input-size-list` argument are `'243,243,3#88#10,7'`, set `--size-with-batch` to `'False,False,True'`.
[--input-dtype-list]	Requires a string to denote data types of the input tensors on the input points. The allowed strings for the supported data types are `'float'`, `'int8'`, `'uint8'`, `'int16'`, and `'uint16'`. String values for the data types of different input tensors are separated with hashtags. For example, `'float#int8#uint16'`. When this argument is omitted, the system uses the default value `'float'` for all input tensors.
[--output-model]	Requires a string value to denote the file path of the generated ACUITY 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 ACUITY 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.

3.4.6 Import PyTorch

Description

Translates a PyTorch model to ACUITY 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.
[-h]	Displays help information.
[--inputs]	Requires a string to list the input points of the PyTorch model. Multiple points are separated with 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. `True` or `False` for different input points are separated with commas and enclosed by ''. For example, `'True,False,True'`. When this argument is omitted, the system uses `False` for all input points listed. **Example 1:** The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the `--input-size-list` argument are `'243,243,3'`, set `--size-with-batch` to `'False'`. **Example 2:** Three input ports with one input point at each port. The tensor shape sizes of each input point are (?,243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the `--input-size-list` argument are `'243,243,3#88#10,7'`, set `--size-with-batch` to `'False,False,True'`.
[--output-model]	Requires a string value to denote the file path of the generated ACUITY 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 ACUITY 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 ACUITY formats. With this argument, you do not need to use any other configuration arguments listed above. To learn configuration file details, see *Example 2*.

Example

Example 1: Translate a PyTorch model, `alexnet.pt`, with configuration arguments.

```
python3 pegasus.py import pytorch    --model 'alexnet.pt'  --inputs 'input_1.1' \
--outputs '194'       --input-size-list '3,224,224'
```

Example 2: Translate a PyTorch model, `alexnet.pt`, by using a configuration file `config_pytorch.json`.

```
python3 pegasus.py import pytorch    --config 'config_pytorch.json'
```

The code snippet of `config_pytorch.json` is as follows.

```
{
  "//": "The source describe",
  "source": {
    "//": "The model source platform",
    "platform": "pytorch",
    "//": "The model file",
    "model_file": "alexnet.pt",
    "//": "The model data file. For some platform, the model data is a standalone file",
    "model_data": "",
    "//": "The source platform optimizer option list",
    "optim_options": [],
    "//": "A list of structures to describe input tensor details",
    "model_input_tensors":[
      {
        "//": "The tensor name, it's a must provide value",
        "tensor_name": "input_1.1",
        "//": "The tensor data type, default is float",
        "element_type": "float",
        "//": "The tensor's shape, default value is an empty list",
        "shape": [1, 3, 224, 224],
        "//": "does the provide shape already have batch info.",
        "//": "If set it false, the final model will insert a batch info before the
       first dimension value",
        "shape_with_batch": true,
        "//": "The value of this tensor",
        "//": "Set this value, at an internal inference the file is loaded as input
       value",
        "//": "Set to null, an internal inference will use fake data as input value",
        "predef_value": null,
        "//": "as_input default value is True,",
        "//": "Set True, this tensor will set as input tensor in final export model",
        "//": "Set False, this tensor will set as Const node",
        "as_input": true
      }
    ],
    "//": "A List of tensor name list",
    "model_output_tensors":["194"]
  },
  "//": "The Target platform list",
  "//": "Which support more than one target platforms",
  "target": [{
    "//": "Target Platform",
```

```
    "//": "Target Platform include two platforms: acuity and onnx",
    "//": "Set Acuity, will generate Acuity model and data file",
    "//": "Set ONNX, will generate ONNX model file",
    "platform": "acuity",
    "//": "Before generating a Target Platform, set the optimizer option list",
    "optim_options": [],
    "check": true,
    "target_model_file": "alexnet_import_from_pytorch.json",
    "target_data_file": "alexnet_import_from_pytorch.data"
  },
  {
    "//": "Target Platform",
    "//": "Target Platform include two platforms: acuity and onnx",
    "//": "Set Acuity, will generate an Acuity model and data file",
    "//": "Set ONNX, will generate an ONNX model file",
    "platform": "onnx",
    "check" : false,
    "//": "Before generating the Target Platform, set the optimizer option list",
    "optim_options": [],
    "target_model_file": "alexnet_import_from_pytorch.onnx"
  }]}
```

3.4.7 Import Keras

Description

Translates a Keras model to ACUITY formats. The supported Keras backend engine is TensorFlow 1.13.2.

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.
[-h]	Displays help information.
[--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.
[--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.

Argument	Description
[--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 ACUITY 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 ACUITY 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`.

3.4.8 Generate Inputmeta

Description

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

For details about the generated inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*.

Note: The inference and export actions need both inputmeta and post-processing files. For the post-processing file generation, see Section *3.4.9, Generate Postprocess File*.

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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
[-h]	Displays help information.
[--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 ACUITY 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 ACUITY 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.

Example

This example generates an inputmeta file for the ACUITY model `mobilenet.json`.

```
python3 pegasus.py generate inputmeta \
--model 'mobilenet.json'     --input-meta-output 'mobilenet_inputmeta.yml'
```

See Also

3.4.9, Generate Postprocess File

3.4.11, Quantize

3.4.12, Inference

3.4.13, Export OVXLIB

3.4.14, Export TFLite

3.4.15, Dump

3.4.17, Train

3.5.1, Example: Pre-processing Setups

3.4.9 Generate Postprocess File

Description

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

Note: The inference and export actions need both inputmeta and post-processing files. For the inputmeta file generation, see Section *3.4.8, Generate Inputmeta* and Section *3.4.12, Inference*.

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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
[-h]	Displays help information.
[--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 ACUITY model>/<model name>_prostprocess_file.yml`.

Example

This example generates a post-processing file for the ACUITY model `mobilenet.json`.

```
python3 pegasus.py generate postprocess-file \
--model 'mobilenet.json'     --postprocess-file 'mobilenet_postprocess_file.yml'
```

See Also

3.4.8, Generate Inputmeta
3.4.12, Inference
3.4.13, Export OVXLIB
3.4.14, Export TFLite
3.5.2, Example: Post-processing Setups

3.4.10 Generate Fakedata

Description

Generates fake data for an ACUITY model.

Syntax

```
python3 pegasus.py generate fakedata
          [-h]
          [--model-data MODEL_DATA]
          [--model-data-output MODEL_DATA_OUTPUT]
          [--override-const-tensors]
          --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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
[-h]	Displays help information.
[--model-data]	Requires a string value to denote the file path of an ACUITY model coefficient data file, a `.data` file.
[--model-data-output]	Requires a string value to denote the file path of the generated fake data file, a `.data` file.
[--override-const-tensors]	Overrides constant tensors in the ACUITY model coefficient data file (`.data`) with the generated fake data. When this argument is omitted, the system overrides all constant tensors except for the model coefficient data file. **Note:** This argument does not require values.

Example

This example generates a fake data file for the ACUITY model `mobilenet.json`.

```
python3 pegasus.py generate fakedata        --model 'mobilenet.json' \
       --model-data 'mobilenet.data' \      --model-data-output 'mobilenetFdata.data' \
       --override-const-tensors
```

3.4.11 Quantize

Description

Quantizes network data. Do not perform quantization on ACUITY networks converted from TensorFlow and TensorFlow Lite models that have been quantized.

Note: Make sure that the quality of each sample image in the input dataset meets the quantization and inference criteria.

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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
--model-data (Required)	Requires a string value to denote the file path of an ACUITY model coefficient data file, a `.data` file. For example, `'mobilenet.data'`.
[-h]	Displays help information.
[--model-quantize]	Requires a string value to denote the file path of the generated `.quantize` file when the `--rebuild` argument is specified, or the file path of the `.quantize` file to use when the `--hybrid` argument is specified. If this argument is omitted when `--rebuild` is specified, the `.quantize` file is generated in the same directory as the model file.

Argument	Description
[--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. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*. **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. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*.
[--quantizer]	Requires one of the following values to specify the quantizer for quantizing network tensors: • `'asymmetric_affine'`: Default • `'dynamic_fixed_point'` • `'perchannel_symmetric_affine'` • `'bfloat16'` • `'symmetric_affine'`: for experimental use only When this argument is omitted, the system uses the default quantizer. **Note:** This argument is used together with `--qtype`.

Argument	Description
[--qtype]	Requires one of the following values to specify the quantization data type: • `'int8'` • `'int16'` • `'uint8'`: Default • `'bfloat16'` • `'int4'`: for experimental use only If the `--qtype` argument is set to `'int4'`, the precision cannot be guaranteed. To revert to 8-bit quantization, ACUITY performs automatic hybrid quantization for some layers. Exported applications that are int4 quantized may fail to run on the software stack. When this argument is omitted, the system uses the default data type. **Note**: This argument is used together with `--quantizer`.
[--hybrid]	Performs hybrid quantization on the network model. This argument is mutually exclusive with the `--rebuild` and `--rebuild-all` arguments.
[--rebuild]	Performs quantization with the quantization file generated. To quantize the network with a quantizer other than `bfloat16`, specify this argument. For details about quantizer types, see the description of `--quantizer`. This argument is mutually exclusive with the `--hybrid` and `--rebuild-all` arguments.
[--rebuild-all]	(Experimental use only) Rebuilds a quantization table with the default quantization rules and quantizes all activations. To quantize the network with the `bfloat16` quantizer, specify this argument. For details about quantizer types, see the description of `--quantizer`. This argument is mutually exclusive with the `--hybrid` and `--rebuild` arguments.
[--algorithm]	Requires one of the following values to specify the quantization algorithm: • `'normal'`: The default algorithm. • `'kl_divergence'`: The KL divergence algorithm. If this value is chosen, specify either `--divergence-nbins` or `--divergence-first-quantize-bits`. • `'moving_average'`: The moving-average algorithm. If this value is chosen, specify the `--moving-average-weight` argument. • `'auto'`: The KL-divergence-based hybrid quantization algorithm with quantized layers automatically determined. If this value is chosen, do not specify the `--MLE` argument. When this argument is omitted, the system uses the default algorithm.

Argument	Description
[--moving-average-weight]	Requires a positive floating-point value to specify the coefficient for the moving average model. This argument is valid only if the `--algorithm` argument is set to `'moving_average'`. When this argument is omitted, the system uses the default coefficient 0.01.
[--divergence-nbins]	Requires an integer to specify the number of bins in the Kullback-Laiber divergence (KL divergence) histogram. The integer must equal 2^N where N is a positive integer. Specify this argument only if `--algorithm` is set to `'kl_divergence'`. When this argument is used, do not specify the `--divergence-first-quantize-bits` argument. When this argument is omitted, the system uses the value 2^{11}. **Note:** `--divergence-nbins` will be deprecated. Use `--divergence-first-quantize-bits` instead.
[--divergence-first-quantize-bits]	Requires a positive integer to calculate the number of bins in the KL divergence histogram. Given an integer m, the calculated KL bin count is 2^m. When this argument is omitted, the system uses the default value 11. **Note:** Specify this argument only if `--algorithm` is set to `'kl_divergence'`. When this argument is used, do not specify the `--divergence-nbins` argument.
[--compute-entropy]	Measures the precision of the current quantization by calculating the entropy of each layer in the range of 0 to 1. The system stores these values in the `entropy.txt` file in the current workspace. If the entropy is low, the precision is high. If the entropy is high, the precision is low. When this argument is omitted, the system does not perform such measurement. **Note:** This argument does not require values. It is valid only if the `--batch-size` argument is set to 1.
[--MLE]	Minimizes per-layer quantization errors of the network. 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-comsuming.

Example

```
python3 pegasus.py quantize          --model 'mobilenet.json' \
--model-data 'mobilenet.data'        --with-input-meta 'mobilenet_inputmeta.yml' \
--quantizer 'asymmetric_affine'      --qtype 'uint8' \
--rebuild
```

See Also

3.4.8, Generate Inputmeta

3.4.9, Generate Postprocess File

3.4.12, Inference

3.4.15, Dump

3.4.12 Inference

Description

Performs inference for an ACUITY model.

Note: Make sure that the quality of each sample image in the input dataset meets the quantization and inference criteria.

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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
--model-data (Required)	Requires a string value to denote the file path of an ACUITY model coefficient data file, a `.data` file. For example, `'mobilenet.data'`.
[-h]	Displays help information.
[--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 sample images per batch. When this argument is omitted, the system uses the value of `shape[0]` in the inputmeta file. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*. 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 sample 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. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*.
[--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. For command details, see Section *3.4.9, Generate Postprocess 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.

Examples

Example 1: Perform inference on a model, `mobilenet.json`, with post-processing configuration arguments.

```
python3 pegasus.py inference        --model 'mobilenet.json' \
--model-data 'mobilenet.data'       --with-input-meta 'mobilenet_inputmeta.yml' \
-postprocess 'classification_classic'
```

Example 2: Perform inference on a model, `mobilenet.json`, by using a post-processing configuration file.

```
pegasus.py inference                --model 'mobilenet.json' \
--model-data 'mobilenet.data'       --with-input-meta 'mobilenet_inputmeta.yml' \
--postprocess-file 'mobilenet_postprocess.yml'                  --iteration 1000 \
--device 'CPU'
```

See Also

3.4.8, Generate Inputmeta

3.4.9, Generate Postprocess File

3.4.11, Quantize

3.4.13 Export OVXLIB

Description

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

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

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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
--model-data (Required)	Requires a string value to denote the file path of an ACUITY 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. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*.
[-h]	Displays help information.
[--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
[--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. For command details, see Section *3.4.9, Generate Postprocess 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.
[--save-fused-graph]	(Experimental use only) Saves a fused model to a `.json` file for debugging use. It is mutually exclusive with the `--pack-nbg-unify` argument. 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.

Argument	Description
[--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 the `--pack-nbg-unify` argument. When this argument is omitted, permutation layers are kept and the tensor shape is in the NHWC sequence. When this argument is specified, the tensor shape may be in the NCHW sequence. Make sure that the data sequence of the tensor shape is NHWC before application deployment onto devices with Vivante NPUs. For example, for a TensorFlow model with input of (1,224,224,3) in the NHWC sequence: • If this argument is not specified, the tensor with the shape (1,224,224,3) in the NHWC sequence is used. • If this argument is specified, the tensor shape may be (1,3,224,224) in the NCHW sequence. When the data sequence is NCHW, convert it into NHWC before application deployment. **Note:** This argument does not require values.
[--pack-nbg-unify]	Packs binary graphs for the unified driver and generates 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 `'/home/xxx/Verisilicon/VivanteIDEx.x.x/*cmdtools'` if VivanteIDE is installed. Specify this argument if `--pack-nbg-unify` is specified.
[--build-platform]	Builds a compiling tool to generate NBG applications. The available value is `'make'`. When this argument is omitted, the system uses the default value `'make'`.
[--target-ide-project]	Requires one of the following values to specify the environment of VivanteIDE, which is used to run the exported unify application code. • `'linux64'`: Default • `'win32'` When this argument is omitted, the system uses the default value.

Argument	Description
[--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. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*.
[--customer-lids]	Reserved for future use.
[--customer-ops]	Reserved for future use.

Example

This example exports an OVXLIB C project with pre-processing nodes. When pre-processing tasks need to be added in an OVXLIB C project application, the `--add_preproc_node` argument in the `inputmeta.yml` file must be set to `true` and the corresponding node parameters must be modified.

```
python3 pegasus.py export ovxlib           --model 'mobilenet.json' \
--model-data 'mobilenet.data'       --with-input-meta 'mobilenet_inputmeta.yml'
```

See Also

3.4.8, Generate Inputmeta

3.4.9, Generate Postprocess File

3.4.14 Export TFLite

Description

Exports an ACUITY model to a TensorFlow Lite model.

Syntax

```
python3 pegasus.py export tflite
           [-h]
           --model MODEL
           --model-data MODEL_DATA
           [--model-quantize MODEL_QUANTIZE]
           [--batch-size BATCH_SIZE]
           [--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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
--model-data (Required)	Requires a string value to denote the file path of an ACUITY model coefficient data file, a `.data` file. For example, `'mobilenet.data'`.
[-h]	Displays help information.
[--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 a positive integer to specify the batch size that the exported model supports. When this argument is omitted, the system uses the value of `shape[0]` in the inputmeta file. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*.
[--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>'`, where: • 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.

See Also

3.4.8, Generate Inputmeta

3.4.9, Generate Postprocess File

3.4.15 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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
--model-data (Required)	Requires a string value to denote the file path of an ACUITY model coefficient data file, a `.data` file. For example, `'mobilenet.data'`.
[-h]	Displays help information.
[--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. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*. Set this argument to a small value if the RAM or GPU does not support a large batch size. **Note:** This argument is used together with `--iterations`.
[--iterations]	Requires an integer to specify the number of image batches. When this argument is omitted, the system uses the default value 1. **Note:** This argument is used together with `--batch-size`.

Argument	Description
[--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. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*.
[--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 data sequence 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.
[--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.

Example

```
python3 pegasus.py dump              --model 'mobilenet.json' \
--model-data 'mobilenet.data'        --with-input-meta 'mobilenet_inputmeta.yml'
```

See Also

3.4.8, Generate Inputmeta

3.4.9, Generate Postprocess File

3.4.11, Quantize

3.4.12, Inference

3.4.16 Prune

Description

Prunes a network.

Syntax

```
python3 pegasus.py prune
          [-h]
          --model MODEL
          --model-data MODEL_DATA
          [--output-data OUTPUT_DATA]
          --config-file CONFIG_FILE
          [--prune-percent PRUNE_PERCENT]
          [--prune-level PRUNE_LEVEL]
```

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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
--model-data (Required)	Requires a string value to denote the file path of an ACUITY model coefficient data file, a `.data` file. For example, `'mobilenet.data'`.
--config-file (Required)	Requires a string value to denote the file path of the pruning configuration file, which sets the pruning percentage of each layer. If the file does not exist, the system generates a configuration file template.
[-h]	Displays help information.
[--output-data]	Requires a string value to denote the file path of the pruned `.data` file. When this argument is omitted, the pruned `.data` file overwrites the original `.data` file.
[--prune-percent]	Requires a value to denote the pruning percentage for the coefficient data of all layers. The value is a floating-point value in the range [0.0, 100.0]. When this argument is omitted, 0 is assigned to this argument. Note that: • When the percentage specified by this argument is greater than 0, the system uses this specified percentage to prune all layers. The percentage in the specified file is ignored and updated accordingly. • When the percentage specified by this argument is equal to 0, the system uses the pruning percentage in the configuration file.

Argument	Description
[--prune-level]	Requires one of the following values to denote the pruning granularity level: • 'element' • 'vector' • 'kernel': Default • 'filter' When this argument is omitted, the system uses the default pruning granularity level.

Example

This example prunes an ACUITY model. The model file is `mobilenet.json`. The data file is `mobilenet.data`.

```
python3 pegasus.py prune     --model 'mobilenet.json' \
--model-data 'mobilenet.data'       --config-file 'output.txt' \
--prune-percent 50
```

3.4.17 Train

Description

Trains a network.

Note: ACUITY can only train classification networks in the current release.

Syntax

```
python3 pegasus.py train
          [-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]
          [--lr LR]
          [--optimizer OPTIMIZER]
          [--decay-steps DECAY_STEPS]
          [--iterations-to-save-checkpoint ITERATIONS_TO_SAVE_CHECKPOINT]
          [--checkpoint-path CHECKPOINT_PATH]
          [--max-checkpoint-num MAX_CHECKPOINT_NUM]
```

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 an ACUITY network model file, a `.json` file. For example, `'mobilenet.json'`.
[-h]	Displays help information.
[--model-data]	Requires a string value to denote the file path of an ACUITY 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:** This argument is required if the `--dtype` argument is set to `'quantized'`.
[--batch-size]	Requires an integer to specify the number of sample images per batch. When this argument is omitted, the system uses the value of `shape[0]` in the inputmeta file. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*. 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 sample 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. For details about the inputmeta file, see Section *3.5.1, Example: Pre-processing Setups*.
[--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.
[--lr]	Requires a floating-point value to specify the learning rate. When this argument is omitted, the system uses the default learning rate 0.01.
[--optimizer]	Requires one of the following values to specify the training gradient optimizer: • `'default'`: The default optimizer. • `'adam'`: The optimizer that implements the Adam algorithm. • `'momentum'`: The Gradient descent optimizer with momentum. When this argument is omitted, the system uses the default optimizer.
[--decay-steps]	Requires an integer to specify the learning rate decay step. When this argument is omitted, the system uses the default decay step 1.
[--iterations-to-save-checkpoint]	Requires a positive integer to specify the interval in iterations to save checkpoints. When this argument is omitted, the system uses the default value 0.
[--checkpoint-path]	Requires a string value to denote the directory to store the checkpoints.
[--max-checkpoint-num]	Requires a positive integer to specify the maximum number of checkpoints. When checkpoints reach the maximum limit, the system keeps only the latest checkpoints. When this argument is omitted, the system uses the default value 5.

Example

```
python3 pegasus.py train                    --model 'mobilenet.json' \
--model-data 'mobilenet.data'               --batch-size 32 \
--iterations 10000                          --device 'CPU' \
--with-input-meta mobilenet_inputmeta.yml   --lr 0.0001 \
--optimizer 'default'                       --iterations-to-save-checkpoint 10000
```

See Also

3.4.8, Generate Inputmeta

3.4.12, Inference

3.5 pegasus Use Cases

This section provides typical use case examples to use the pegasus commands.

3.5.1 Example: Pre-processing Setups

To simplify pre-processing in ACUITY, you can manually create an `inputmeta.yml` file or generate one with the `generate inputmeta` command. For details about this command, see Section *3.4.8, Generate Inputmeta*.

The following inputmeta file snippet shows the parameters to be configured. In most cases, the generated inputmeta can be used directly without modification, or require only slight modification if the database type is changed. For details about the configuration, see Table *3-5 Inputmeta Parameters*.

```
%YAML 1.2
---
# !!!This file disallow TABs!!!
# "category" allowed values: "image, undefined"
# "database" allowed types: "H5FS, SQLITE, TEXT, LMDB, NPY, GENERATOR"
# "tensor_name" only support in H5FS database
# "preproc_type" allowed types:"IMAGE_RGB, IMAGE_RGB888_PLANAR, IMAGE_RGB888_PLANAR_SEP,
       IMAGE_I420,
# IMAGE_NV12, IMAGE_YUV444, IMAGE_GRAY, IMAGE_BGRA, TENSOR"
input_meta:
  databases:
  - path: dataset.txt
    type: TEXT
    ports:
    - lid: data_0
      category: image
      dtype: float32
      sparse: false
      tensor_name:
      layout: nhwc
      shape:
      - 50
      - 224
      - 224
      - 3
      preprocess:
        reverse_channel: false
        mean:
        - 103.94
        - 116.78
        - 123.67
        scale: 0.017
        preproc_node_params:
          preproc_type: IMAGE_RGB
          add_preproc_node: false
          preproc_perm:
          - 0
          - 1
```

```
        - 2
        - 3
  - lid: label_0
    redirect_to_output: true
    category: undefined
    tensor_name:
    dtype: float32
    shape:
    - 1
    - 1
```

Table 3-5 Inputmeta Parameters

Parameter	Value Description
input_meta	The top keyword that requires no value and indicates the inputmeta to be configured with the subsequent parameters.
databases	The keyword that requires no value and indicates the database configurations to be set with the subsequent parameters. The configuration set of a database includes the `path`, `type`, and `port` parameters. Each database configuration set is identified by the `path` parameter. A database can be shared by multiple ports. A port can use data from only one database.
path	The file path of a dataset file, either an absolute path or a relative path to the execute directory. You can specify one or more dataset files. Each dataset requires to set the path and the following parameters.
type	The type of the dataset file. The supported types are `H5FS`, `SQLITE`, `TEXT`, `LMDB`, and `NPY`. If `type` is set to `H5FS`, set the `tensor_name` parameter for the port of the label input to `'label'`. If `type` is set to `LMDB`, the dataset must be in the Caffe blob format. For details about the dataset formats, see Section *3.3, Dataset Formats*.
ports	The keyword that requires no value and indicates the network input configurations to be set with the subsequent parameters.
lid	The unique ID of an input port in the dataset file. If the dataset file contains multiple input ports, configure each port one by one in the same order of the dataset file. You need to set the `lid` and the following parameters for each port. For unused ports, set their `lid` to `'undefined'`. **Note**: If the dataset file is a TEXT file, one column represents one input port. The most left column is the first port to be configured. The most right column is the last port be configured. For more details, see Section *3.3, Dataset Formats*.
category	The category of the inputs. Set this parameter to one of the following values: • `image`: Image inputs • `undefined`: Other types of inputs

Parameter	Value Description
dtype	The data type for the input tensor, which serves to send the data at the input port into the ACUITY network. The supported data types are `float32` and `quantized`.
sparse	Specifies whether the network model is in the sparse format. Set this parameter to one of the following values: • `true`: The network model is in the sparse format. • `false`: The network model is in a compressed format.
tensor_name	(Valid only for datasets of the H5FS type) The name for this input tensor. Leave this parameter vacant for other types of datasets.
layout	The format of the input tensor. Use `nchw` for Caffe, Darknet, ONNX, and PyTorch models. Use `nhwc` for TensorFlow, TensorFlow Lite, and Keras models.
shape	The shape for this tensor. The first dimension, `shape[0]`, indicates the number of inputs per batch, which allows multiple inputs to be sent into the network before an NN operation. Set `shape[0]` to a positive integer. If the `--batch-size` argument is also specified in a pegasus command, the argument value is used and `shape[0]` is ignored. For details about this argument, see Section *3.4, Command Reference*. **Note**: When the dataset is a TEXT file, one row represents one input. For a batch input, the number of the inputs in a batch depends on the number of rows you want to input into the ACUITY network in one execution. Each row contains one or more columns to represents the contained input ports. For more details, see Section *3.4, Command Reference*.
fitting	Reserved.
preprocess	The keyword that requires no value and indicates the input tensors are preprocessed with the subsequent parameters. Specify the parameters in the sequence of the pre-processing tasks you want to perform. Pre-processing includes data normalization and adding pre-processing layers to the exported applications.
reverse_channel	Specifies whether to reserve the channel order. Set this parameter to one of the following values: • `true`: Reserves the channel order. • `false`: Does not reserve the channel order. Use `false` only for models of TensorFlow and TensorFlow Lite frameworks.
mean	The mean value for each channel.

Parameter	Value Description
scale	The scaling value for input tensors. Mean and scale values serve normalization of the input tensors according to the formula (inputTensor - mean) × scale.
preproc_node_params	The keyword that requires no value and indicates to add a pre-processing layer into the exported applications by configuring the following parameters. This parameter is valid only if the `add_preproc_node` parameter is set to true.
add_preproc_node	Specifies whether to enable the pre-processing layer added to the exported applications. • `true`: Enables the pre-processing layer. The `preproc_node_params` and `preproc_type` parameters are valid only if this parameter is set to true. • `false`: Disables the pre-processing layer.
preproc_type	The type of the inputs on the pre-processing layer. The available types are `IMAGE_RGB`, `IMAGE_RGB888_PLANAR`, `IMAGE_RGB888_PLANAR_SEP`, `IMAGE_I420`, `IMAGE_NV12`, `IMAGE_YUV444`, `IMAGE_GRAY`, `IMAGE_BGRA`, and `TENSOR`. For details about these types, see Section *3.1, Input Types for Pre-processing in OpenVX Applications* in the *Vivante Programming FAQ: ACUITY pegasus Customized Scripts*. This parameter is valid only if the `add_preproc_node` parameter is set to true.
preproc_perm	The parameter used to permute the inputs of the pre-processing layer.
redirect_to_output	Specifies whether to send data of the input port, which is represented by tensors, directly to the network output. • `true`: Sends the data directly to the network output. The lid of this port specified in the corresponding `labels_tensor` of the `postprocess.yml` file must be the same as the lid given here, so that the data can be further processed before final output. • `false`: Does not send the data directly to the network output. This parameter is valid only for classification networks with TEXT-type dataset files.

3.5.2 Example: Post-processing Setups

By default, ACUITY dumps output tensors for input layers and output layers and prints the top 5 output tensors of the output layers. To enable more post-processing tasks in ACUITY, use the `postprocess.yml` file. You can either manually create a `postprocess.yml` file or generate one with the `generate postprocess-file` command. For command details, see Section *3.4.9, Generate Postprocess File*.

Note: You can also use the `--postprocess` argument in the `inference` command to modify the configurations of the post-processing tasks, `print_topn` and `dump_results`. For details, see Section *3.4.12, Inference*.

postprocess.yml File

The following post-processing file snippet shows the parameters to be configured. For details about the configuration, see Table *3-6 Post-processing Parameters*.

```
%YAML 1.2
---
# "acuity_postprocs" allowed types: "dump_results, print_topn, classification_validate,
       mute_built_in_actions"
postprocess:
  acuity_postprocs:
    print_topn:
      topn: 5
    dump_results:
      file_type: TENSOR
    # mute_built_in_actions: true
    # python:
    # -file_path: postprocess.py
    #  output_tensors:
    #  - '@xxxx:out0'
  app_postprocs:
  - lid: output_0
    postproc_params:
      add_postproc_node: false
      perm:
      - 0
      - 1
      force_float32: true
```

Table 3-6 Post-processing Parameters

Parameter	Value Description
postprocess	The top keyword that indicates the post-processing configurations to be set with the subsequent parameters.
acuity_postprocs	The keyword that indicates the post-processing tasks to be performed during inference in ACUITY. The supported post-processing tasks include `print_topn`, `dump_results`, `classification_validate`, `mute_built_in_actions`, and `python`. Specify the tasks in the sequence you want to perform them. For details about the task parameters, see the subsections of Section *3.5.2, Example: Post-processing Setups*.
app_postprocs	The keyword that indicates the post-processing tasks for export of unify applications only.
lid	The unique ID of an output layer.
postproc_params	The keyword that indicates to add a post-processing layer into the exported applications by configuring the following parameters.
add_postproc_node	Specifies whether to enable the post-processing layer added to the exported applications. • `true`: Enables the post-processing layer. • `false`: Disables the post-processing layer.
perm	The parameter used to permute the outputs of the post-processing layer.
force_float32	The parameter used to forcibly set the output data type of the post-processing layer to float32.

3.5.2.1 Post-processing Task Configurations

With the `postprocess.yml` file, you can configure the following post-processing tasks in ACUITY. For details about the `postprocess.yml` file, see Section *3.5.2, Example: Post-processing Setups*.

Post-processing Task	Description	Parameters
print_topn	Prints a specified number of top significant output tensors. **Note:** This feature is useful for classification networks.	• `topn`: Requires a positive integer value to specify the number of top significant network output tensors to be shown on screen.
dump_results	Dumps output tensors for input layers and output layers. The data sequence of the tensors in the dump result is the same as that specified by the source model.	• `file_type`: Requires `TENSOR`, `BIN`, or `NPY` to specify the file type of the dump result.
classification_validate	Maps an output tensor to a classification label.	• `predictions_tensor`: Requires the URL to an output tensor in the graph. For example, `@output_0:out0`, where output_0 is a layer in the graph and out0 is an output port. • `labels_tensor`: Requires the URL to the classification label, an output tensor in the graph or a tensor defined in `inputmeta.yml`. For example, `@label_0:out0`, where label_0 does not exist in the graph, but is defined in `inputmeta.yml` and the input is sent directly to network output for post-processing. For more information about the inputmeta.yml file, see Section *3.5.1, Example: Pre-processing Setups*.
mute_built_in_actions	A switch to enable or disable all ACUITY built-in actions. If `mute_built_in_actions` is set to `true`, the built-in tasks `dump_results`, `print_topn`, and `classification_validate` are disabled. If `mute_built_in_actions` is omitted or set to `false`, the built-in tasks are enabled.	None.

Post-processing Task	Description	Parameters

python	Defines custom post-processing tasks with `.py` files.
	Note: You can specify multiple groups of the following parameters in the `postprocess.yml` file to define multiple custom post-processing tasks.
		• `file_path`: Requires the absolute file path to the customized post-processing task script, a `.py` file.
		To customize post-processing task script, create a class with the same name as the filename and a function named `process()`. The following provides an example of the `postprocess.py` file:
		`class postprocess:`
		`  def __init__(self, **kwargs):`
		`  self.tensor_url = kwargs.get('output_tensors')`
		`  def process(self, **kwargs):`
		`  tensors = dict()`
		`  for url in self.tensor_url:`
		`  tensors[url] = kwargs.get(url)`
		As shown in the example, enable the `__init__()` function to get the required tensor URLs. Then, in the `process()` function, obtain the tensor data dictionary from the URL dictionary before dealing with the tensor data.
		• `output_tensors`: Requires a list to specify the URLs of the output tensors to be used. For example, `@output_0:out0`, where output_0 is a layer in the graph and out0 is an output port.

3.5.3 Example: Quantizing in Hybrid

To perform a hybrid quantization, use the `python3 pegasus.py quantize` command in the following sequence:

1. Use the `--rebuild` and `--compute-entropy` arguments to quantize the network (`.data` and `.json` files) with a supported quantization type.

   This generates a `.quantize` file and an `entropy.txt` file:

   - In the `.quantize` file, the `customized_quantize_layers` section provides suggested layers for a further quantization with the `dynamic_fixed_point-i16` quantization type.
   - For more details about the `entropy.txt` file, see the description of the `--compute-entropy` argument in Section *3.4.11, Quantize*.

2. Update the `customized_quantize_layers` section to add, modify, or remove layers for a further quantization.

   To determine new quantization types for these layers, reference the entropies in `entropy.txt`. For high entropy layers, you can use the `float32` type to improve the precision.

   Note: The supported data types in `customized_quantize_layers` are `dynamic_fixed_point-i16` and `float32`.

3. Use `--hybrid` and `--model-quantize` arguments with the updated `.quantize` file to quantize the network with the same quantization type as in Step 1.

   This performs a hybrid quantization and generates `.quantize.json` and `.quantize` files.
   The `.quantize.json` file also contains the newly added dtype_converter layers.

4. Export such hybrid-quantized model with the `.data` and the generated `.quantize.json` files.

   In the exported model, the dtype_converter layers are inserted.

Example

The following example shows a typical hybrid quantization in pegasus commands, in which the layer called `conv2_3` is changed from the `asymmetric_affine-u8` data type to the `float32` data type.

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

```
python3 pegasus.py quantize \
--model '~/lenet/lenet.json' \
--model-data '~/lenet/lenet.data' \
--quantizer 'asymmetric_affine' \
--qtype 'uint8' \
--with-input-meta '~/lenet_inputmeta.yml' \
--rebuild \
--compute-entropy \
```

2. In the `lenet.quantize` file, modify the `customized_quantize_layers` section by adding the layer named `conv2_3` with the `float32` data type as follows:

```
customized_quantize_layers:
    conv2_3: float32
```

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

```
python3 pegasus.py quantize \
--model '~/lenet/lenet.json' \
--model-data '~/lenet/lenet.data' \
--model-quantize '~/lenet/lenet.quantize' \
--quantizer 'asymmetric_affine' \
--qtype 'uint8' \
--with-input-meta '~/lenet/lenet_inputmeta.yml' \
--hybrid \
```

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

4. Export the hybrid-quantized model as follows:

```
python3 pegasus.py export ovxlib \
--model '~/lenet/lenet.quantize.json' \
--model-data '~/lenet/lenet.data' \
--model-quantize '~/lenet/lenet.quantize' \
--with-input-meta '~/lenet/lenet_inputmeta.yml' \
--dtype 'quantized' \
```

   The exported case also contains the dtype_converter layers.

4 VSInn API

The advanced ACUITY VSInn API set allows you to create your own Python programs to flexibly deploy machine learning models onto devices with Vivante NPUs.

These APIs support:

- Import and translate models to ACUITY models.
- Set up inputmeta and post-processing files.
- Quantize networks.
- Perform inference.
- Export ACUITY models to for various kinds of devices deployments.

This chapter describes functions and arguments of each API in detail.

4.1 VSInn Object Creation

Description

Creates a VSInn object.

Syntax

```
from acuitylib.vsi_nn import VSInn
nn = VSInn(project_dir = os.getcwd())
```

Arguments

Argument	Required	Data Type	Description
project_dir	No	String	The directory of the generated files. The default directory is the current working path.

Returns

A VSInn object

4.2 ACUITY Model Conversions

ACUITY provides the following APIs to convert various other models into ACUITY formats:

- *4.2.1, Load Caffe*
- *4.2.2, Load TensorFlow*
- *4.2.3, Load TFLite*
- *4.2.4, Load Darknet*
- *4.2.5, Load ONNX*
- *4.2.6, Load PyTorch*
- *4.2.7, Load Keras*

For information about how to create an empty ACUITY network and load files to the network, see Section *4.3, ACUITY Network Creation* and Section *4.4, Model File Adding*.

4.2.1 Load Caffe

Description

Translates a Caffe model to ACUITY formats.

Syntax

```
nn.load_caffe(
    model,
    weights=None,
    proto='caffe'
)
```

Arguments

Argument	Required	Data Type	Description
model	Yes	String	The file path of the imported Caffe model file, a `.prototxt` file.
weights	No	Sting	The file path of the imported weights file, a `.caffemodel` file. For example, `'mobilenet.caffemodel'`. If the weights file is not specified or does not exist, the system generates a `.data` file which contains fake data.
proto	No	String	The protocol used by the imported Caffe model. Set this argument to one of the following values: • `'caffe'`: (Default) Represents the standard Caffe format protocol. • `'lstm_caffe'`: Represents the LSTM layer protocol. • `'<other protocol name>'`: For another protocol with its file `<xxx>_pb2.py`, set the `proto` argument to the absolute path of the file.

Returns

An AcuityNet object.

4.2.2 Load TensorFlow

Description

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

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

Syntax

```
nn.load_tensorflow(
    model, inputs,
    input_size_list,
    outputs,
    size_with_batch=None,
    **kwargs
)
```

Arguments

Argument	Required	Data Type	Description
model	Yes	String	The file path of the TensorFlow frozen protocol buffer (protobuf) file, a `.pb` file. **Note:** Models trained and frozen by the following TensorFlow versions have been proven compatible with ACUITY: 1.4.x, 2.0.x, and 2.3.x.
inputs	Yes	String	The input points of the TensorFlow graph. Multiple points are separated with spaces. For example, `'input_point_1 input_point_2'`.
input_size_list	Yes	String	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	Yes	String	The output points of the TensorFlow graph. Multiple points are separated with spaces. For example, `'output_point_1 output_point_2'`.

Argument	Required	Data Type	Description
size_with_batch	No	String	Specifies 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. Set the value for the size of each input point to one of the following values: • `True`: Contains. • `False`: Does not contain. `True` or `False` for different input points are separated with commas and enclosed by ''. For example, `'True,False,True'`. When this argument is omitted, the system uses `False` for all input points listed. **Example 1:** The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the `input_size_list` argument are `'243,243,3'`, set `size_with_batch` to `'False'`. **Example 2:** Three input ports with only one input point at each port. The tensor shape sizes of each input point are (?, 243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the `input_size_list` argument are `'243,243,3#88#10,7'`, set `size_with_batch` to `'False,False,True'`.
mean_values	No	String	The mean values of each input point listed in the `inputs` argument. Multiple mean values are separated with commas. For example, `'128.0,128.0'`. When this argument is omitted, the default value None is assigned to this argument. **Note:** Specify this argument only for quantized TensorFlow models.
std_values	No	String	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'`. When this argument is omitted, the default value None is assigned to this argument. **Note:** Specify this argument only for quantized TensorFlow models.

Argument	Required	Data Type	Description
predef_file	No	String	The file path of a predef file, an `.npz` file. Specify this argument to import complex models and enable custom control logics. When this argument is omitted, the default value None is assigned to this argument. **Note:** To generate a predef file, you can use the NumPy function `np.savez(\'prd.npz', <path name>=<predefined value>)` where `<path name>` refers to the name of a network path for a specific compute stage. For example, `np.savez(\'prd.npz', train_stage=False)`. If a placeholder name contains unsupported characters, map the placeholder name to a supported alias. For example, NumPy does not support forward slashes. If the placeholder name is 'inference/train_stage', then use the following function to generate the predef file: `np.savez(\'prd.npz', stage=False, map={'stage':'inference/train_stage'})`. For more information about the `np.savez()` function, visit *NumPy documentation*.
subgraphs	No	String	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'`. When this argument is omitted, the default value None is assigned to this argument.

Returns

An AcuityNet object.

4.2.3 Load TFLite

Description

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

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

Syntax

```
nn.load_tflite(
    model,
    inputs=None,
    input_size_list=None,
    size_with_batch=None,
    outputs=None
)
```

Arguments

Argument	Required	Data Type	Description
model	Yes	String	The file path of the imported TensorFlow Lite (TFLite) model, in an optimized FlatBuffer format identified by the `.tflite` file extension. The supported TFLite schema is 0c4f5dfea4ceb3d7c0b46fc04828420a344f7598, committed on *https://github.com/tensorflow/tensorflow/commits/master/tensorflow/lite/schema/schema.fbs*. **Note:** Import failures may occur if the TFLite model uses an unsupported TFLite schema.
inputs	No	String	The input points of the TFLite graph. Multiple points are separated with spaces. For example, `'output_point_1 output_point_2'`. When this argument is omitted, the system uses all header points of this model.
input_size_list	No	String	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.

Argument	Required	Data Type	Description
size_with_batch	No	String	Specifies 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. Set the value for the size of each input point to one of the following values: • `True`: Contains. • `False`: Does not contain. `True` or `False` for different input points are separated with commas and enclosed by ''. For example, `'True,False,True'`. When this argument is omitted, the system uses `False` for all input points listed. **Example 1:** The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the `input_size_list` argument are `'243,243,3'`, set `size_with_batch` to `'False'`. **Example 2:** Three input ports with only one input point at each port. The tensor shape sizes of each input point are (?, 243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the `input_size_list` argument are `'243,243,3#88#10,7'`, set `size_with_batch` to `'False,False,True'`.
outputs	No	String	The output points of the TFLite graph. 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.

Returns

An AcuityNet object.

4.2.4 Load Darknet

Description

Translates a Darknet model to ACUITY formats.

Syntax

```
nn.load_darknet(
    model,
    weights
)
```

Arguments

Argument	Required	Data Type	Description
model	Yes	String	The file path of the imported Darknet model, a `.cfg` file.
weights	Yes	String	The file path of the imported weights file, a `.weights` file.

Returns

An AcuityNet object.

4.2.5 Load ONNX

Description

Translates an ONNX model to ACUITY formats.

Syntax

```
nn.load_onnx(
    model,
    inputs=None,
    outputs=None,
    input_size_list=None,
    size_with_batch=None,
    input_dtype_list=None
)
```

Arguments

Argument	Required	Data Type	Description
model	Yes	String	The file path of the imported ONNX model, an `.onnx` file.
inputs	No	String	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.
outputs	No	String	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.
input_size_list	No	String	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	Required	Data Type	Description
size_with_batch	No	String	Specifies 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. Set the value for the size of each input point to one of the following values: • `True`: Contains. • `False`: Does not contain. `True` or `False` for different input points are separated with commas and enclosed by ''. For example, `'True,False,True'`. When this argument is omitted, the system uses `False` for all input points listed. **Example 1:** The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the `input_size_list` argument are `'243,243,3'`, set `size_with_batch` to `'False'`. **Example 2:** Three input ports with only one input point at each port. The tensor shape sizes of each input point are (?, 243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the `input_size_list` argument are `'243,243,3#88#10,7'`, set `size_with_batch` to `'False,False,True'`.
input_dtype_list	No	String	The 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 `'float'` for all input tensors.

Returns

An AcuityNet object.

4.2.6 Load PyTorch

Description

Translates a PyTorch model to an AcuityNet object by using the ONNX backend. PyTorch 1.5.1 is supported.

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

Syntax

```
nn.load_pytorch_by_onnx_backend(
    model,
    input_size_list,
    inputs=None,
    size_with_batch=None,
    outputs=None,
    target_onnx_file=None
)
```

Arguments

Argument	Required	Data Type	Description
model	Yes	String	The file path of the imported PyTorch model, a `.pt` file.
input_size_list	Yes	String	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.
inputs	No	String	The input points of the PyTorch model. Multiple points are separated with space. For example, `'input_point_1 input_point_2'`. Each input point can only have one output tensor. When this argument is omitted, the system uses all head points of the imported model.

Argument	Required	Data Type	Description
size_with_batch	No	String	Specifies 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. Set the value for the size of each input point to one of the following values: • `True`: Contains. • `False`: Does not contain. `True` or `False` for different input points are separated with commas and enclosed by ''. For example, `'True,False,True'`. When this argument is omitted, the system uses `False` for all input points listed. **Example 1:** The sizes of the input tensor shape are (100, 243, 243, 3). Given the shape sizes listed in the `input_size_list` argument are `'243,243,3'`, set `size_with_batch` to `'False'`. **Example 2:** Three input ports with only one input point at each port. The tensor shape sizes of each input point are (?, 243, 243, 3), (11, 88), and (10, 7). If the sizes listed in the `input_size_list` argument are `'243,243,3#88#10,7'`, set `size_with_batch` to `'False,False,True'`.
outputs	No	String	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 this model.
target_onnx_file	No	String	The file path of the generated backend ONNX model file, which is converted from the PyTorch model.

Returns

An AcuityNet object

4.2.7 Load Keras

Description

Translates a Keras model to ACUITY formats. The supported Keras backend engine is TensorFlow 1.13.2.

Syntax

```
nn.load_keras(
    model,
    inputs=None,
    input_size_list=None,
    outputs=None,
    convert_engine='Keras'
)
```

Arguments

Argument	Required	Data Type	Description
model	Yes	String	The file path of the imported Keras model, an `.h5` file.
inputs	No	String	The input points of the Keras graph. 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.
input_size_list	No	String	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	No	String	The output points of the Keras graph. Multiple points are separated with spaces. For example, `'output_point_1 output_point_2'`.
convert_engine	No	String	Reserved for future use.

Returns

An AcuityNet object with Keras model.

4.3 ACUITY Network Creation

Description

Creates an empty ACUITY network with no model information.

To load model information to the ACUITY network, use the APIs described in Section *4.4, Model File Adding*.

Note: For information about how to convert other models into ACUITY formats, see Section *4.2, ACUITY Model Conversions*.

Syntax

```
nn.create_net()
```

Returns

An empty AcuityNet object.

4.4 Model File Adding

After creating an empty ACUITY network with `nn.create_net()`, you can use the following APIs to load files to the network:

- *4.4.1, Load Model Network File*
- *4.4.2, Load Model Data File*
- *4.4.3, Load Model Quantization File*
- *4.4.4, Load Model Inputmeta*
- *4.4.5, Load Model Postprocess*

For information about how to convert other models into ACUITY formats, see Section *4.2, ACUITY Model Conversions*.

4.4.1 Load Model Network File

Description

Loads a `.json` model file to an empty ACUITY network created by `nn.create_net()`.

For details about the `nn.create_net()` API, see Section *4.3, ACUITY Network Creation*.

Syntax

```
nn.load_model(
    net,
    model
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
model	Yes	String	The file path of the `.json` model file. **Note:** If the specified file already exists, ACUITY overwrites the content of the file.

4.4.2 Load Model Data File

Description

Loads a `.data` data file to an empty ACUITY network created by `nn.create_net()`.

For details about the `nn.create_net()` API, see Section *4.3, ACUITY Network Creation*.

Syntax

```
nn.load_model_data(
    net,
    model_data
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
model	Yes	String	The file path of the `.data` data file. **Note:** If the specified file already exists, ACUITY overwrites the content of the file.

4.4.3 Load Model Quantization File

Description

Loads a `.quantize` quantization file to an empty ACUITY network created by `nn.create_net()`.

For details about the `nn.create_net()` API, see Section *4.3, ACUITY Network Creation*.

Syntax

```
nn.load_model_quantize(
    net,
    model_quantize
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
model_quantize	Yes	String	The file path of the `.quantize` quantization file. **Note:** If the specified file already exists, ACUITY overwrites the content of the file.

4.4.4 Load Model Inputmeta

Description

Configures inputmeta with an existing inputmeta file, a `.yml` file.

For information about how to create `.yml` inputmeta files, see Section *4.5, Pre-processing and Post-processing*.

Syntax

```
nn.load_model_inputmeta(
    net,
    model_inputmeta
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
model_inputmeta	Yes	String	The file path of the `.yml` inputmeta file. **Note:** If the specified file already exists, ACUITY overwrites the content of the file.

4.4.5 Load Model Postprocess

Description

Configures post-processing with an existing post-processing file, a `.yml` file.

For information about how to create `.yml` post-processing files, see Section *4.5, Pre-processing and Post-processing*.

Syntax

```
nn.load_model_postprocess_file(
    net,
    postprocess_file
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
postprocess_file	Yes	String	The file path of the `.yml` post-processing file. **Note:** If the specified file already exists, ACUITY overwrites the content of the file.

4.5 Pre-processing and Post-processing

Pre-processing involves dataset preparation and pre-processing task configuration required by dump, quantize, inference, and export actions. Post-processing defines post-processing tasks for inference and export actions.

ACUITY provides the following APIs to configure pre-processing:

- *4.5.1, Set Database* (Recommended)
- *4.5.2, Set Preprocess* (Recommended)
- *4.5.5, Generate Inputmeta*

The following APIs are also supported for post-processing configuration:

- *4.5.3, Set ACUITY Postprocess* (Recommended)
- *4.5.4, Set APP Postprocess* (Recommended)
- *4.5.6, Generate Outputmeta*

4.5.1 Set Database

Description

Sets datasets for an ACUITY network with a specified dataset file.

Syntax

```
set_database(
    net,
    dataset_files,
    dataset_type
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
dataset_files	Yes	String	The file path of the dataset file.
dataset_type	Yes	String	The type of the dataset file. Set this argument to `'TEXT'` or `'NPY'`.

See Also

4.5.2, Set Preprocess
4.5.3, Set ACUITY Postprocess
4.5.4, Set APP Postprocess
4.8.1, Quantize
4.8.2, Inference
4.8.3, Export OVXLIB
4.8.4, Export TFLite
4.8.5, Dump

4.5.2 Set Preprocess

Description

Works together with the `set_database()` API to set pre-processing tasks for an ACUITY network. To save pre-processing setups into an inputmeta `.yml` file, use the `save_model_inputmeta()` API. For details about these APIs, see Section *4.5.1, Set Database* and Section *4.7.4, Save Model Inputmeta File*.

Syntax

```
set_preprocess(
    net,
    preprocess_dict,
    set_by_lid=False
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
preprocess_dict	Yes	Dictionary	The shape, mean, and scale of the input points. For example, `{'input':{'shape':[1,1,28,28], 'mean':[0], 'scale':1/256}}`.
set_by_lid	No	Boolean	Specifies whether to use ACUITY layer IDs or original layer IDs in the `preprocess_dict` argument. Set this argument to one of the following values: • `True`: The ACUITY layer IDs are used. • `False`: The output layer IDs from the source model are used.

See Also

4.5.1, Set Database
4.5.3, Set ACUITY Postprocess
4.5.4, Set APP Postprocess
4.8.1, Quantize
4.8.2, Inference
4.8.3, Export OVXLIB
4.8.4, Export TFLite
4.8.5, Dump

4.5.3 Set ACUITY Postprocess

Description

Sets post-processing tasks for an ACUITY network, which are performed during inference.

Syntax

```
set_acuity_postprocess(
    net,
    postprocess_list,
    set_by_lid=False
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
postprocess_list	Yes	List of strings	The post-processing tasks. For example, `['dump_results', ('print_topn',{'topn':7}), ('classification_classic'), ('python',({'python_file': 'postprocess/postprocess.py', 'output_tensors':('output')}))]`. Supported tasks include `'classification_classic'`, `'print_topn'`, `'dump_results'`, `'classification_validate'`, and `'python'`.
set_by_lid	No	Boolean	Specifies whether to use ACUITY layer IDs or original layer IDs in the `postprocess_list` argument. • `True`: The ACUITY layer IDs are used. • `False`: The output layer IDs from the source model are used.

See Also

4.5.1, Set Database
4.5.2, Set Preprocess
4.5.4, Set APP Postprocess
4.8.2, Inference
4.8.3, Export OVXLIB
4.8.4, Export TFLite

4.5.4 Set APP Postprocess

Description

Sets post-processing tasks for export of unify applications only.

Syntax

```
set_app_postprocess(
    net,
    postprocess_list,
    set_by_lid=False
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
postprocess_list	Yes	List	A list of post-processing tasks. For example, `[['output1',[{'add_postproc_node': True,'perm':[0,1],'force_float32':True}]], ['output2',[{'add_postproc_node': True,'perm':[0,3,1,2]}]]]`. Only permutation tasks are supported.
set_by_lid	No	Boolean	Specifies whether to use ACUITY layer IDs or original layer IDs in the `postprocess_list` argument. • `True`: The ACUITY layer IDs are used. • `False`: The output layer IDs from the source model are used.

See Also

4.5.1, Set Database
4.5.2, Set Preprocess
4.5.3, Set ACUITY Postprocess
4.8.2, Inference
4.8.3, Export OVXLIB
4.8.4, Export TFLite

4.5.5 Generate Inputmeta

Description

Generates an inputmeta file in the `.yml` format for an ACUITY network with a specified dataset file. You need to customize the generated `.yml` file before you can use it to set pre-processing tasks for this AcuityNet object with the `nn.load_model_inputmeta()` API. For details about `nn.load_model_inputmeta()`, see Section *4.4.4, Load Model Inputmeta*.

Note: A fast method to configure pre-processing is to use the `set_database()` and `set_preprocess()` APIs. For details about these APIs, see Section *4.5.1, Set Database* and Section *4.5.2, Set Preprocess*.

Syntax

```
generate_inputmeta(
    net,
    inputmeta_output=None,
    dataset_file='dataset.txt',
    dataset_type='TEXT',
    separated_database=False
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
inputmeta_output	No	String	The file path of the generated inputmeta file. When this argument is omitted, the system uses the default file path `<directory of the ACUITY model>/<model name>_inputmeta.yml`.
dataset_file	No	String	The file path of the dataset.
dataset_type	No	String	The type of the dataset file. Set this argument to `'TEXT'` or `'NPY'`.
separated_database	No	Boolean	Specifies whether to separate the dataset into multiple ones in the generated inputmeta file for a multi-input ACUITY network. • `True`: Separates the dataset into multiple ones. • `False`: Does not separate the dataset. When this argument is omitted, the system uses the default value `False`.

See Also

4.5.1, Set Database
4.5.2, Set Preprocess
4.5.3, Set ACUITY Postprocess
4.5.4, Set APP Postprocess
4.8.1, Quantize
4.8.2, Inference
4.8.3, Export OVXLIB
4.8.4, Export TFLite
4.8.5, Dump

4.5.6 Generate Outputmeta

Description

Generates a `.yml` post-processing file after the post-processing tasks are set by the `set_acuity_postprocess()` or `set_app_postprocess()` APIs. For details about these APIs, see Section *4.5.3,* Set ACUITY Postprocess and Section *4.5.4, Set APP Postprocess*.

Syntax

```
generate_outputmeta(
    net,
    outputmeta_output=None
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
outputmeta_output	No	String	The file path of the generated `.yml` post-processing file.

See Also

4.5.1, Set Database
4.5.2, Set Preprocess
4.5.3, Set ACUITY Postprocess
4.5.4, Set APP Postprocess
4.8.2, Inference
4.8.3, Export OVXLIB
4.8.4, Export TFLite

4.6 Fake Data Generation

Description

Generates fake data for an ACUITY network.

Syntax

```
generate_fakedata(
    net,
    output_path=None
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
output_path	No	String	The file path of the generated fake data file, a `.data` file.

Returns

A new AcuityNet object with fake data.

4.7 Model File Saving

You can use the following APIs to save the data or structure of an ACUITY network into files:

- *4.7.1, Save Model Network File*
- *4.7.2, Save Model Data File*
- *4.7.3, Save Model Quantization File*
- *4.7.4, Save Model Inputmeta File*

4.7.1 Save Model Network File

Description

Saves the model of an ACUITY network to a `.json` file.

Syntax

```
save_model(
    net,
    output_model
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
output_model	Yes	String	The file path of the generated `.json` model file.

4.7.2 Save Model Data File

Description

Saves the data of an ACUITY network to a `.data` file.

Syntax

```
save_model_data(
    net,
    output_data
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
output_data	Yes	String	The file path of the generated `.data` data file.

4.7.3 Save Model Quantization File

Description

Saves the quantized tensors of an ACUITY model to a `.quantize` file.

Syntax

```
save_model_quantize(
    net,
    output_quantize
)
```

Argument

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
output_quantize	Yes	String	The file path of the generated `.quantize` file.

4.7.4 Save Model Inputmeta File

Description

Saves the inputmeta of an ACUITY network to a `.yml` file.

Syntax

```
save_model_inputmeta(
    net,
    output_inputmeta
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
output_inputmeta	Yes	String	The file path of the generated `.yml` inputmeta file.

4.8 Inference and Export

ACUITY provides you with the following APIs to adapt and export network models for deployment onto devices with Vivante NPUs:

4.8.1 Quantize

Description

Quantizes an ACUITY network. Do not perform quantization on ACUITY networks converted from TensorFlow and TensorFlow Lite models that have been quantized.

Note: Make sure that the quality of each sample image in the input dataset meets the quantization and inference criteria.

Syntax

```
quantize(
    net,
    quantizer='asymmetric_affine',
    qtype='uint8',
    hybrid=False,
    rebuild=False,
    rebuild_all=False,
    batch_size=None,
    iterations=1,
    device='CPU',
    algorithm='normal',
    moving_average_weight=0.01,
    divergence_nbins=0,
    divergence_first_quantize_bits=11,
    compute_entropy=False,
    **kwargs
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
quantizer	No	String	The quantizer to quantize network tensors. Set this argument to `'asymmetric_affine'`, `'dynamic_fixed_point'`, `'perchannel_symmetric_affine'`, `'symmetric_affine'`, `'asymmetric_quantized'`, or `'bfloat16'`. When this argument is omitted, the system uses the default quantizer `asymmetric_affine`. Among these values: • Quantizers with the prefix perchannel_ are per-layer quantizers. • The quantizer type `'symmetric_affine'` is for experimental use only. • `'asymmetric_quantized'` will be deprecated. Use `'asymmetric_affine'` instead. **Note:** This argument is used together with `qtype`.
qtype	No	String	The quantization data type. Set this argument to `'int8'`, `'int16'`, `'uint8'`, `'bfloat16'`, `'uint4'`, or `'int4'`. When this argument is omitted, the system uses the default data type `'uint8'`. The data types `'uint4'` and `'int4'` are for experimental use only. **Note**: This argument is used together with `quantizer`.
hybrid	No	Boolean	Specifies whether to perform hybrid quantization on the network model. Set this argument to one of the following values: • `True`: Performs. • `False`: Does not perform. When this argument is omitted, the system does not perform hybrid quantization. **Note:** Set one of the `hybrid`, `rebuild_all`, and `rebuild` arguments to `True`. Otherwise, set all these arguments to `False`.

Argument	Required	Data Type	Description
rebuild	No	Boolean	Specifies whether to perform quantization with the quantization file generated. Set this argument to one of the following values: • `True`: Performs. • `False`: Does not perform. To quantize the network with a quantizer other than `bfloat16`, set this argument to `True`. For details about quantizer types, see the description of `quantizer`. When this argument is omitted, the system does not perform quantization with the quantization file generated. **Note:** Set one of the `hybrid`, `rebuild_all`, and `rebuild` arguments to `True`. Otherwise, set all these arguments to `False`.
rebuild_all	No	Boolean	(Experimental use only) Specifies whether to rebuild a quantization table with the default quantization rules and quantizes all activations. Set this argument to one of the following values: • `True`: Rebuild. • `False`: Does not rebuild. To quantize the network with the `bfloat16` quantizer, set this argument to `True`. For details about quantizer types, see the description of `quantizer`. When this argument is omitted, the system does not rebuild the quantization or quantize activations. **Note:** Set one of the `hybrid`, `rebuild_all`, and `rebuild` arguments to `True`. Otherwise, set all these arguments to `False`.
batch_size	No	Integer	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]` specified in the `set_preprocess()` API. For details, see Section *4.5.2, Set Preprocess*. **Note:** This argument is used together with `iterations`.

Argument	Required	Data Type	Description
iterations	No	Integer	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	No	String	The compute device type. Set this argument to `'GPU'` or `'CPU'`. When this argument is omitted, the system uses the default device type `'CPU'`.
algorithm	No	String	The quantization algorithm. Set this argument to one of the following values: • `'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. When this argument is omitted, the system uses the default algorithm.
moving_average_weight	No	Float	The positive coefficient for the moving average model. When this argument is omitted, the system uses the default coefficient 0.01. This argument is valid only if the `algorithm` argument is set to `'moving_average'`.
divergence_nbins	No	Integer	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.

Argument	Required	Data Type	Description
divergence_first_quantize_bits	No	Integer	A positive integer used to calculate the number of bins in the KL divergence histogram. Given an integer m, the calculated KL bin count is 2^m. Specify this argument only if `algorithm` is set to `'kl_divergence'`. When this argument is used, do not specify the `divergence_nbins` argument. When this argument is omitted, the system uses the default value 11.
compute_entropy	No	Boolean	Specifies whether to measure the precision of the current quantization by calculating the entropy of each layer in the range of 0 to 1. Set this argument to one of the following values: • `True`: Measures. • `False`: Does not measure. The system stores the entropy 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 measure the quantization precision. **Note**: This argument is valid only if the `batch_size` argument is set to 1.

Returns

A new AcuityNet object after quantization.

See Also

4.5.1, Set Database
4.5.2, Set Preprocess
4.5.3, Set ACUITY Postprocess
4.5.4, Set APP Postprocess
4.8.2, Inference
4.8.5, Dump

4.8.2 Inference

Description

Performs inference for an ACUITY model.

Note: Make sure that the quality of each sample image in the input dataset meets the quantization and inference criteria.

Syntax

```
inference(
    net,
    output_path=None,
    batch_size=None,
    iterations=1,
    device='CPU',
    **kwargs
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
output_path	No	String	The directory for the generated files.
batch_size	No	Integer	The number of sample images per batch. When this argument is omitted, the system uses the value of `shape[0]` specified in the `set_preprocess()` API. For details, see Section *4.5.2, Set Preprocess*. Set this argument to a small value if the RAM or GPU does not support a large batch size. **Note:** This argument is used together with `iterations`.
iterations	No	Integer	The number of sample image batches. When this argument is omitted, the system uses the default value 1. **Note:** This argument is used together with `batch_size`.
device	No	String	The compute device type. Set this argument to `'GPU'` or `'CPU'`. When this argument is omitted, the system uses the default device type `'CPU'`.

Returns

The list of input and output tensors of inference.

See Also

4.5.1, Set Database
4.5.2, Set Preprocess
4.5.3, Set ACUITY Postprocess
4.5.4, Set APP Postprocess
4.8.1, Quantize

4.8.3 Export OVXLIB

Description

Exports an ACUITY model into an OpenVX application to run with the Vivante OVXLIB C library.

Syntax

```
export_ovxlib(
    net,
    output_path=None,
    optimize='default',
    dtype='float',
    save_fused_graph=False,
    pack_nbg_unify=False,
    viv_sdk=None,
    build_platform='make',
    target_ide_project='linux64',
    batch_size=None,
    force_remove_permute=False,
    **kwargs
)
```

Arguments

Argument	Required	Data Type	Description
Net	Yes	AcuityNet object	The ACUITY network.
output_path	No	String	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	No	String	The optimization method for the export. Set this argument to one of the following values: • `'None'`: Does not optimize. • `'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.

Argument	Required	Data Type	Description
dtype	No	String	The tensor data type of the exported OVXLIB application. Set this argument to one of the following values: • `'float'` or `'float16'`: `'float'` and `'float16'` are equivalents. • `'float32'` • `'quantized'`: A quantization data type specified in the `.quantize` file when the input model is a quantized model. If this value is chosen, the ACUITY network must have been quantized. When this argument is omitted, the system uses the default data type `'float'`.
save_fused_graph	No	Boolean	(Experimental use only) Specifies whether to save a fused model into a `.json` file for debugging use. It is mutually exclusive with the `pack_nbg_unify` argument. Set this argument to one of the following values: • `True`: Saves the fused model. The generated fused model contains network structures only of the exported unify applications. • `False`: Does not save the fused model. When this argument is omitted, no fused model is saved.
pack_nbg_unify	No	Boolean	Specifies whether to pack binary graphs for the unified driver and generate 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. Set the `pack_nbg_unify` argument to one of the following values: • `True`: Packs binary graphs for the unified driver. • `False`: Does not pack binary graphs for the unified driver. **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.

Argument	Required	Data Type	Description
viv_sdk	No	String	The file path of the directory that contains the binary SDK of VSim. During execution, VSim generates NBG files. For example, the file path may be '/home/xxx/Verisilicon/VivanteIDEx.x.x/*cmdtools' if VivanteIDE is installed. Specify this argument if `pack_nbg_unify` is specified.
build_platform	No	String	Specifies whether to build 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	No	String	The environment of VivanteIDE, which is used to run the exported unify application code. Set this argument to `'linux64'` or `'win32'`. When this argument is omitted, the system uses the default value `'linux64'`.
batch_size	No	Integer	The batch size that the exported application supports. When this argument is omitted, the system uses the value of `shape[0]` specified in the `set_preprocess()` API. For details, see Section *4.5.2, Set Preprocess*.

Argument	Required	Data Type	Description
force_remove_permute	No	Boolean	(Experimental use only) Specifies whether to remove the head permutation layers inserted after the input layer and the tail permutated layers inserted before the output layer. Set this argument to one of the following values: • `True`: Removes. • `False`: Does not remove. This argument is used only for export of unify applications from TensorFlow and TensorFlow Lite models. It is mutually exclusive with the `pack_nbg_unify` argument. When this argument is omitted, permutation layers are kept and the tensor shape is in the NHWC sequence. When this argument is specified, the tensor shape may be in the NCHW sequence. Make sure that the data sequence of the tensor shape is NHWC before application deployment onto devices with Vivante NPUs. For example, for a TensorFlow model with input of (1,224,224,3) in the NHWC sequence: • If this argument is set to `True`, the tensor with the shape (1,224,224,3) in the NHWC sequence is used. • If this argument is set to `False`, 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.

See Also

4.5.1, Set Database
4.5.2, Set Preprocess
4.5.3, Set ACUITY Postprocess
4.5.4, Set APP Postprocess

4.8.4 Export TFLite

Description

Exports an ACUITY model to a TensorFlow Lite model.

Syntax

```
export_tflite(
    net,
    output_path=None,
    dtype='float',
    save_fused_graph=False,
    float_io=False,
    batch_size=None,
    **kwargs
)
```

Arguments

Argument	Required	Data Type	Description
Net	Yes	AcuityNet object	The ACUITY network.
output_path	No	String	The directory of the exported TensorFlow Lite (TFLite) model 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.
dtype	No	String	The tensor data type of the exported TFLite model. Set this argument to one of the following values: • `'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.

Argument	Required	Data Type	Description
save_fused_graph	No	Boolean	(Experimental use only) Specifies whether to generate a fused model for debugging use. Set this argument to one of the following values: • `True`: Generates the fused model. The generated fused model contains network structures only of the exported unify applications. • `False`: Does not generate the fused model. When this argument is omitted, no fused model is saved.
float_io	No	Boolean	Specifies whether to set the data type of the network inputs and outputs to float32. Set this argument to one of the following values: • `True`: Sets the data type to float32. • `False`: Does not set the data type to float32. When this argument is omitted, the system does not set the data type to float32.
batch_size	No	Integer	The batch size that the exported TFLite model supports. When this argument is omitted, the system uses the value of `shape[0]` specified in the `set_preprocess()` API. For details, see Section *4.5.2, Set Preprocess*.

See Also

4.5.1, Set Database
4.5.2, Set Preprocess
4.5.3, Set ACUITY Postprocess
4.5.4, Set APP Postprocess

4.8.5 Dump

Description

Dumps tensors from each layer.

Syntax

```
dump(
    net,
    output_path=None,
    format='nchw',
    batch_size=None,
    iterations=1,
    device='CPU',
    save_quantize=False,
    save_file_type=’tensor’,
    **kwargs
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
output_path	No	String	The directory for the generated snapshot files.
format	No	String	The data sequence in which to save snapshot tensor data. Use `'nchw'` for Caffe, Darknet, ONNX, and PyTorch models. Use `'nhwc'` for TensorFlow, TensorFlow Lite, and Keras models. When this argument is omitted, the system uses the default sequence `'nchw'`.
batch_size	No	Integer	The number of images per batch. When this argument is omitted, the system uses the value of `shape[0]` specified in the `set_preprocess()` API. For details, see Section *4.5.2, Set Preprocess*. Set this argument to a small value if the RAM or GPU does not support a large batch size. **Note:** This argument is used together with `iterations`.
iterations	No	Integer	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`.

Argument	Required	Data Type	Description
device	No	String	The compute device type. Set this argument to `'GPU'` or `'CPU'`. When this argument is omitted, the system uses the default device type `'CPU'`.
save_quantize	No	Boolean	Specifies whether to save data in the quantized format. Set this argument to one of the following values: • `True`: Saves data in the quantized format. Quantized tensors are saved in the format specified in the `.quantize` file. • `False`: Save data in the float32 format.
save_file_type	No	String	The format of the saved snapshot files. Set this argument to one of the following values: • `'tensor'`: 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 `'tensor'`. For the data type of the numbers, see the description of the `save_quantize` argument.

See Also

4.5.1, Set Database
4.5.2, Set Preprocess
4.5.3, Set ACUITY Postprocess
4.5.4, Set APP Postprocess
4.8.1, Quantize
4.8.2, Inference

4.8.6 Prune

Description

Prunes an ACUITY network.

Syntax

```
prune(
    net,
    config_file,
    prune_percent=0.0,
    prune_level='kernel'
)
```

Arguments

Argument	Required	Data Type	Description
net	Yes	AcuityNet object	The ACUITY network.
config_file	Yes	String	The file path of the pruning configuration file, which sets the pruning percentage of each layer. If the file does not exist, the system generates a configuration file template.
prune_percent	No	Float	The pruning percentage for the coefficient data of all layers. Set this argument to a value in the range of [0.0, 100.0]. When this argument is omitted, the default value 0 is assigned to this argument. Note that: • When the percentage specified by this argument is greater than 0, the system uses this specified percentage to prune all layers. The percentage in the specified file is ignored and updated accordingly. • When the percentage specified by this argument is equal to 0, the system uses the pruning percentage in the configuration file.
prune_level	No	String	The pruning granularity level. Set this argument to `'element'`, `'vector'`, `'kernel'`, or `'filter'`. When this argument is omitted, the default value `'kernel'` is assigned to this argument.

Returns

A new AcuityNet object after pruning.

4.9 Example of VSInn Programming

This section provides a detailed example of using the VSInn. For detailed description of the programing settings, see Section *2.4, Programming with VSInn APIs*.

```
from acuitylib import VSInn

PROTOTXT = "lenet.prototxt"
WEIGHTS = "lenet.caffemodel"

# create a VSInn object
nn = VSInn()

# load a caffe model
lenet = nn.load_caffe(PROTOTXT, weights=WEIGHTS)

# set database
nn.set_database(lenet, dataset_files='dataset.txt', dataset_type='TEXT')

# set preprocess, 'input' is the key defined by acuity, means the input layer name of
       the original network
preprocess_dict = {'input':{'shape':[1,28,28,1],
                            'mean':[128],
                            'scale':1/128,},
                   }
nn.set_preprocess(lenet, preprocess_dict)

# save .json .data .yml files if needed
nn.save_model(lenet, "lenet.json")
nn.save_model_data(lenet, "lenet.data")
nn.save_model_inputmeta(lenet, "lenet_inputmeta.yml")

# quantization of asymmetric_affine-u8 and dynamic_fixed_point-i16
affine_u8_lenet = nn.quantize(lenet, quantizer="asymmetric_affine", qtype="uint8")
dfp_int16_lenet = nn.quantize(lenet, quantizer="dynamic_fixed_point", qtype="int16")

# save quantize files if needed
nn.save_quantize(affine_u8_lenet, "affine_u8_lenet.quantize")
nn.save_quantize(dfp_int16_lenet, "dfp_int16_lenet.quantize")

# set acuity post-process, the 'python_file' is the python code for processing the
       'output_tensors'. The values of 'output_tensors' are the output point names in
       the original gragh or in a layer of acuity network specified with a layer ID.
acuity_postproccess_list = ['dump_results',
                            ('print_topn', {'topn':7}),
                            ('python', ({'python_file': 'postprocess.py',
                                         'output_tensors':['output',]
                                         },)
                             ),
                           ]
nn.set_acuity_postprocess(lenet, acuity_postproccess_list)
```

```
# inference with float and quantized model
result1 = nn.inference(lenet)
result2 = nn.inference(affine_u8_lenet)
result3 = nn.inference(dfp_int16_lenet)

# dump with float and quantized model
nn.dump(lenet)
nn.dump(affine_u8_lenet)
nn.dump(dfp_int16_lenet)

# export ovxlib case with float and quantized model
nn.export_ovxlib(lenet)
nn.export_ovxlib(affine_u8_lenet)
nn.export_ovxlib(dfp_int16_lenet)
```

Document Revision History

This section describes the differences between different revisions of this document.

Note: This document is not necessarily updated for each patch or minor revision. The information in this document tends to be stable across a revision (nnn) series.

Document Revision	Date	Compatible ACUITY Toolkit	Change History
0.94	2022-03-25	ACUITY Toolkit v6.6.x	• Updated the branding and layout to include VeriSilicon. • Section *1.3, Table 1-1 Input Features*: Changed the supported quantization data type qbfloat16 with the qbfloat16 quantizer to bfloat16 with the bfloat16 quantizer. • Section *2.2.1, Installing TensorFlow*: Removed the dependency on `flatbuffers`. • Section *3.4.11, Quantize*: ▪ Changed the `'qbfloat16'` value to `'bfloat16'` for the `--quantizer` and `--qtype` arguments. ▪ Added the `'symmetric_affine'` value for the `--quantizer` argument for experimental use only. ▪ Added the `'int4'` value for the `--qtype` argument for experimental use only. ▪ Added the constraint that the `--MLE` argument and the `'auto'` value of `--algorithm` cannot be specified at the same time. • Section *3.5.1, Example: Pre-processing Setups*: Added the `IMAGE_RGB888_PLANAR_SEP` value for the `preproc_type` parameter. • Section *3.5.2, Example: Post-processing Setups*: Added the `force_float32` parameter. • Section *4.5.4, Set APP Postprocess*: Updated the example value of `postprocess_list`. • Section *4.8.1, Quantize*: ▪ Removed the `'qbfloat16'` value for the `quantizer` and `qtype` arguments. ▪ Specified that the `'symmetric_affine'` value of the `quantizer` argument is for experimental use only. ▪ Added the values `'int4'` and `'unit4'` for the `qtype` argument for experimental use only.

Document Revision	Date	Compatible ACUITY Toolkit	Change History
0.93	2021-12-23	ACUITY Toolkit v6.3.x	• Section *1.3, Table 1-1 Input Features*: ▪ Began to support TensorFlow 2.6.x. ▪ Updated the supported ONNX version to 1.10.2 (operator sets 1-15). ▪ Updated the supported PyTorch version to 1.5.1. ▪ Began to support Keras models generated from TensorFlow 2.6.x. • Section *1.4, Table 1-4 Host System Requirements*: ▪ Updated the requirement on the CUDA and cuDNN. ▪ Updated the required RAM size to at least 8 GB. ▪ Began to support Ubuntu 18.04 LTS 64-bit with Python 3.6. • Section *2, Working with the ACUITY Toolkit*: Globally moved the configuration requirement on the ACUITY_PATH environmental variable to the *What to Do Next* sub-section of Section *2.1, Installing ACUITY Binary Version*. • Section *2.2.1, Installing TensorFlow*: ▪ Updated the installed tensorflow and onnx versions. ▪ Removed the dependency on `Pillow` and `matplotlib`. • Section *3.4.1, Import Caffe* and Section *4.2.1, Load Caffe*: Changed the value of `--proto` / `proto` for protocols other than the standard Caffe format protocol and LSTM layer protocol, from the protocol file prefix to the absolute file path. • Section *3.4.13, Export OVXLIB* and Section *4.8.3, Export OVXLIB*: ▪ Updated the output files for the exported unify and nbg_unify applications. ▪ Removed the support for the nbg_unify_ovx application. ▪ Removed the `--pack-nbg-viplite` / `pack_nbg_viplite` argument. • Section *3.4.14, Export TFLite* and Section *4.8.4, Export TFLite*: Added the `--batch-size` argument. • Section *3.5.2, Example: Post-processing Setups*: Removed the `fmt` parameter from the `postprocess.yml` file script. The data sequence of the tensors in the dump result is the same as that specified by the source model. • Globally added support for Ubuntu 18.04 and Python 3.6. • Refined the document content and structure.

Document Revision	Date	Compatible ACUITY Toolkit	Change History
0.92	2021-09-22	ACUITY Toolkit v6.0.0	• Section *1.3, Table 1-2 NN Compute Features*: Removed the signed int8 data type with the symmetric affine quantizer. • Section *1.4, Table 1-4 Host System Requirements*: ▪ Began to support Ubuntu 18.04 LTS 64-bit with Python 3.6. ▪ Specified maintenance stop for Ubuntu 16.04. • Section *3.4.11, Quantize*: Removed the following values from the `--quantizer` argument: ▪ `'symmetric_affine'` ▪ `'asymmetric_quantized'`: Use `'asymmetric_affine'` instead. ▪ `'bfloat16'`: Use `'qbfloat16'` instead. • Section *3.4.13, Export OVXLIB* and Section *4.8.3, Export OVXLIB*: ▪ Specified obsolescence of the nbg_unify_ovx application for the `--pack-nbg-unify` / `pack_nbg_unify` argument. ▪ Specified obsolescence for the `--pack-nbg-viplite` / `pack_nbg_viplite` argument. • Added Section *2.3.3, Saving Frozen PB and H5 Models*. • Globally added support for Ubuntu 18.04 and Python 3.6.
0.91	2021-08-20	ACUITY Toolkit v6.0.0	• Section *1.2, Working Principle*: Updated Figure 1-1 ACUITY Workflow. • Section *1.3, Table 1-2 NN Compute Features:* ▪ Added the bfloat16 data type with the bfloat16 quantizer. ▪ Added the automatic hybrid quantization algorithm. • Section *3.4.11, Quantize*: ▪ Added the `'auto'` value for the `--algorithm` argument. ▪ Added the `--MLE` argument. • Refined the document content and structure.
0.90	2021-06-28	ACUITY Toolkit v5.24.x	• Globally removed the support for the tensorzonex command line tool. • Section *1.3, Table 1-1 Input Features:* Updated the supported ONNX version to 1.8.0. • Section *2.2, Installing ACUITY Python Version*: Added the requirement on the installed pip version. • Section *2.2.1, Installing TensorFlow*: ▪ Updated the installed onnx and torch versions. ▪ Removed the dependency on `numpy`, `protobuf`, `image`, and `h5py`. • Section *3.4.3, Import TFLite*: Added the `--inputs`, `--input-size-list`, and `--size-with-batch` arguments. • Section *3.5.1, Example: Pre-processing Setups*: Added the `layout` parameter. • Section *4.2.3, Load TFLite*: Added the `input_size_list` and `size_with_batch` arguments. • Deleted the `detection_validate` post-processing task. • Deleted the `nn.load_pytorch()` API. • Refined the document content and structure.

Document Revision	Date	Compatible ACUITY Toolkit	Change History
0.80	2021-03-26	ACUITY Toolkit v5.21.x	• Section *1.3, Table 1-1 Input Features*: Began to support TensorFlow 2.3.1 for experimental use only. • Section *2.2.1, Installing TensorFlow*: ▪ Updated the installed numpy version to 1.18.0 or later. ▪ Updated the installed scipy version to 1.1.0 or later. ▪ Updated the installed protobuf version to 3.11.2 or later. • Section *3.4.11*, Section *3.4.12*, Section *3.4.15*, and Section *3.4.17*, the `--with-input-meta` argument: Corrected from optional to required. • Section *4.5.5*, the `separated_database` argument: Corrected the data type from string to Boolean. • Globally updated the supported TensorFlow versions up to 2.3.1. • Updated the maintenance stop notice for the tensorzonex Command Line tool. • Refined the document content and structure.
0.76	2020-12-30	ACUITY Toolkit v5.18.x	• Section *1.3, Table 1-1 Input Features*: ▪ Updated the supported TensorFlow versions up to 2.3.0. ▪ Updated the supported TensorFlow Lite schema. • Section *2.2.1, Installing TensorFlow*: ▪ Updated the installed TensorFlow version to 2.3.0. ▪ Updated the installed astunparse version to 1.6.3. • Section *3.4.7*, the `pegasus.py import keras` command: Reserved the `--convert-engine` argument for future use. • Section *3.4.11*, the `pegasus.py quantize` command: ▪ Updated the `--quantizer` and `--qtype` arguments to specify that the symmetric_affine int8 quantization type is for experimental use only. ▪ Added a new argument `--rebuild-all` for experimental use only. ▪ Added a new argument `--compute-entropy`. • Section *3.4.13*, the `pegasus.py export ovxlib` command: ▪ Updated the `--force-remove-permute` argument for experimental use only. ▪ Added a new argument `--customer-lids` for experimental use only. ▪ Added a new argument `--customer-ops` for experimental use only. • Section *3.4.14*, the `pegasus.py export tflite` command: Updated `--save-fused-graph` argument for experimental use only. • Section *4.8.6*: Added a new VSInn API `export_tflite()`. • Globally updated the supported TensorFlow versions up to 2.3.0. • Specified maintenance stop for the tensonzonex Command Line tool. • Refined the document content and structure.

Document Revision	Date	Compatible ACUITY Toolkit	Change History
0.75	2020-09-23	ACUITY Toolkit v5.16.x	• Chapter *3, Table 2:* ▪ Updated the supported ONNX models from in operator sets 1-7 to ONNX 1.6.0 (operator sets 1-11). ▪ Updated the Tensorflow version for Keras model generation from Tensorflow 1.13.x to Tensorflow 2.0.0. • Section *3.8*: Added the "nbg_file" argument. • Section *3.10*: ▪ Updated the quantization arguments as follows: ▪ Added the "divergence-first-quantize-bits" argument. ▪ Updated the description and the default value of the "quantized-divergence-nbins" argument. • Section *3.12*: ▪ Updated pegasus.py.quantize as follows: Added new data types "bfloat" and "qbfloat" to arguments "quantizer" and "qtype". Added the "divergence-first-quantize-bits" argument. Updated the description and the default value of the "divergence-nbins" argument. ▪ Updated pegasus.py.export.tflite as follows: Added the "float-io" argument. • Section *3.13.11*: ▪ Added new data types "bfloat" and "qbfloat" to arguments "quantizer" and "qtype". ▪ Updated the default value of the "divergence-nbins" argument. ▪ Added the "divergence-first-quantize-bits" argument.
0.74	2020-07-24	ACUITY Toolkit v5.15.x	• Section 2.1.2: Updated table 1. • Section 3.8: Added exportflite function. • Section 3.12: Updated dump parameter "save-file-type". • Section 3.13.6: Updated set_database function. • Section 3.13.8: Updated the generate fakedata function. • Section 3.13.13: Updated dump function.
0.73	2020-06-24	ACUITY Toolkit (as of v5.14.0 of June 2020)	• Section 3.6: Updated convertpytorch parameter. • Section 3.9.1.1: Updated text dataset format. • Section 3.11: Updated the import onnx parameters, update import pytorch parameters. • Section 3.12.3: Added function load_pytorch_by_onnx_backend. • Section 4.3.2: Updated hybrid quantization. • Section 4.6.1: Updated syntax example for import. • Section 4.6.3.1: Updated specify pre-process for inputs. • Section 4.8: Added GRU training.

Document Revision	Date	Compatible ACUITY Toolkit	Change History
0.72	2020-05-27	ACUITY Toolkit (as of v5.13.0 of May 2020)	• Section 2.1.1: Added OVXLIB package. • Section 3.8: Updated parameter description “export-dtype” • Section 3.9: Updated parameter description “quantized_dtype” • Section 3.11: Added parameter “separated_database” for generate input_meta, add parameter “output_dir” for inference, update description of quantize parameter ”qtype”, update description of export ovxlib parameter ”dtype” • Section 3.12.6: Added set_app_postprocess function. • Section 3.12.7: Updated generate_inputmeta parameter. • Section 3.12.12: Updated inference parameter • Section 4.7: Added LSTM training • Section 5: Removed ACUITY Graphical User Interface
0.71	2020-04-27	ACUITY Toolkit (as of v5.12.0 of April 2020)	• Section 2.2.2: update required library • Section 3.11: add parameter ‘save_fused_graph’, ‘dtype’ in export tflite • Section 3.12.3: update load_tflite, load_onnx parameters • Section 3.12.5: add load_model_postprocess_file function • Section 3.12.6: update set_database parameter • Section 3.12.9: update save_model_data parameter • Section 3.12.13: update dump parameter • Section 4.6.3.1: update specify pre-process for inputs • Miscellaneous refinements
0.70	2020-03-23	ACUITY Toolkit (as of v5.11.0 of March 2020)	• Section 2.2.1: remove Qt software • Section 2.2.3: update required libraries • Section 2.2.4: remove install required library • Section 3: update table2 • Section 3.3: add parameter ‘outputs’ • Section 3.5: add parameters ‘input_size_list’, ‘size_with_batch’, ‘input_type_list’ • Section 3.11: add parameter ‘outputs’ in import tflite, add parameters ‘input_size_list’, ‘size_with_batch’, ‘input_type_list in import onnx. • Section 4.6.3.1: update specify pre-process for inputs • Section 4.6.3.2: update specify post-process tasks for output • Section 4.6.3.3: update post-process tasks for output • Section 5.2: remove quick start guide for acuity command line • Section 6: merge into Section 5

Document Revision	Date	Compatible ACUITY Toolkit	Change History
0.69	2019-12-23	ACUITY Toolkit (as of v5.8.0 of December 2019)	• Legal Notices: Distribution Level changed. • Section 2.2.3: update required libraries • Section 3.9.1.1: update text database format • Section 3.11: update parameter 'with_input_meta' in export ovxlib • Section 3.12.3: update load_keras parameters • Section 3.12.13: update dump parameters • Section 4.6.3.1: update specify pre-process for inputs • Section 4.6.3.2: update specify post-process tasks for output
0.68	2019-12-03	ACUITY Toolkit (as of v5.7.0 of November 2019)	• Section 3.1/2/3/4/5/6/7: update the description of parameters 'net_output' and 'data_output' • Section 3.5: add parameters 'inputs' and 'outputs' • Section 3.6: add parameter 'size_with_batch', update the usage of parameters 'pytorch_model' and 'input_size_list' • Section 3.7: update the usage of parameter 'input_size_list' • Section 3.8: update the description of parameter 'pack_nbg_unify' • Section 3.11: add generate postprocess-file function, add parameters 'inputs' and 'outputs' for import onnx, add parameter 'size_with_batch' and update the usage of 'model' and 'input_size_list' for import pytorch, update the usage of parameter 'input_size_list' for import keras, add parameter 'postprocess_file' in inference function, update the description of parameter 'pack_nbg_unify' in export ovxlib, update the description of parameters 'output_model' and 'output_data' in all import functions • Section 3.12: add vsinn • Section 4.6.2: update syntax example for generate • Section 4.6.3: update syntax example for inference • Section 4.6.7: update syntax example for export • Miscellaneous text and format refinements
0.67	2019-10-24	ACUITY Toolkit (as of v5.6.0 of October 2019)	• Section 2.2.3: add torch in required library • Section 2.2.6: remove launch acuity • Section 3.2: update the usage of parameter 'predef_file' in convertensorflow • Section 3.6: add parameters 'inputs' and 'outputs' for convertpytorch • Section 3.8: remove tensorconverter function. update the description of parameter 'batch-size' in export • Section 3.11: remove export ide in Pegasus, update the usage of parameters 'predef_file' in Pegasus, update the description of parameter 'batch-size' in export ovxlib, add parameters 'inputs' and 'outputs' in import pytorch • Section 4.3.2: update the usage of hybrid quantization • Section 5.1.2: update the usage of basic steps • Section 6.8.4: remove export ide function • Miscellaneous refinements

Document Revision	Date	Compatible ACUITY Toolkit	Change History
0.66	2019-08-16	ACUITY Toolkit (as of v5.4.0 of August 2019)	• Section 5: add Table 4, and remove related text from individual tools parameter descriptions • Section 5.12: add symmetric_affine for --quantizer • Section 6.3.2: modify usage of hybrid quantization
0.65	2019-07-26	ACUITY Toolkit (as of v5.3.0 of July 2019)	• Section 2.1.2: update file list • Section 2.2.3 update required libraries • Section 4.8.2/3/4: update section cross references • Section 5. Add paragraph recommending pegasus usage. • Section 5.1, 5.2 Remove –dump-gml from convertcaffe, add causeion for –input-sie-list • Section 5.2 5.12 add –size-with-batch for Tensorflow importer • Section 5.6 add convertpyroch • Section 5.7 add convertkeras • Section 5.10: quantized-dtype, update default value of to asymmetric_affine-u8, modify description • Section 5.10.1 add NumPy datasest format • Section 5.12: misc refinements; add pytorch and keras imports, add –size-with-batch; quantized-dtype, update default value of to asymmetric_affine-u8, modify description • Section 6, various subbsections: undate –quantized-dtype • Section 6.3 and 6.4 Rename • Section 6.6.1 add Pytorch and Keras. • misc refinements and section cross reference updates
0.64	2019-05-29	ACUITY Toolkit (as of v5.1.0 of May 2019)	• Preface: change Acronym ACUITY_PATH to ACUITY_FOLDER to distinguish ACUITY_PATH from other sample script. • Section 2: add onnx_tf requirement, update onnx to 1.4.1, update install wheel/source package. • Section 5: for added clarity, numerous description refinements for parameters for all command line tools. • Refine dataset.txt description. • Section 5.7: Add force-remove-permute • Section 6.3: update hybrid quantization usage. • Section 6.6: switch the sequence of 6.6.1 and 6.6.2. update 6.6.3 to guide user to modify inputmeta.yml and make postprocess.yml.
0.63	2019-03-29	ACUITY Toolkit (as of v5.0.0 of March 2019)	• Section 2.2 refined to update require libraries and package usage • Section 5.6, 5.7, 5.8, 5.9 update some command line tools' parameters tensorconverter, ovxgenerator, tensorzonex, and tensorreduce. • Added Section 5.10 for pegasus command line tool • Section 6.3: added hybrid quantization description • Section 6.5: updated ovxgenerator parameters in examples. • Section 6.6: added syntax for pegasus