Qualcomm
Qualcomm Technologies, Inc.

Qualcomm®
AI Engine Direct SDK
Reference Guide

⌂

v2.22.6.240515184619_92920

Next >

Introduction

Purpose

This document provides a reference guide for Qualcomm® AI Engine Direct software development kit (SDK).

Note

Qualcomm® AI Engine Direct is also referred to as Qualcomm Neural Network (QNN) in the source code and documentation.

Conventions

Function declarations, function names, type declarations, filenames, directory names and library names are shown in a different font. For example: `#include`

Commands and code samples appear in a specially formatted code section. For example:

```
output = (input - offset) * scale.
```

Mathematical expressions appear in a specially formatted math section. For example:

$$y = x + 1$$

Environment variables appear preceded by $, for example $QNN_SDK_ROOT.

Platform Differences

Qualcomm® AI Engine Direct supports both Windows and Linux platforms. There are several differences between Linux and Windows systems:

Library and Executable Program Naming Convention

On Linux, shared library names will be prefixed with `lib` , and have the `.so` extension. On Windows, shared library do not have a prefix and will have the `.dll` extension.

On Linux, library names have the `.a` extension. On Windows, static use the `.lib` extension. The executables on Windows have the `.exe` extension, while Linux executables do not.

As an example, the QnnCpu backend library would be `libQnnCpu.so` on Linux and `QnnCpu.dll` on Windows.

Release Folder for Different Platforms

Users for Qualcomm® AI Engine Direct software development kit (SDK) can find the corresponding binaries and executables through relative folders. For example, for Linux and Windows platforms, users can find the specific path under the root of the SDK that indicates the platform:

- (For Linux-X86): /bin/x86_64-linux-clang.
- (For Windows-X86): /bin/x86_64-windows-msvc.
- (For Windows-aarch64): /bin/aarch64-windows-msvc.

Environment setup for Linux and Windows

Operation for model tools should follow the environment setup by platform dependencies as a prerequisite. See Setup for more information.

Release Notes

Release notes are available as QNN_ReleaseNotes.txt in the SDK.

Previous

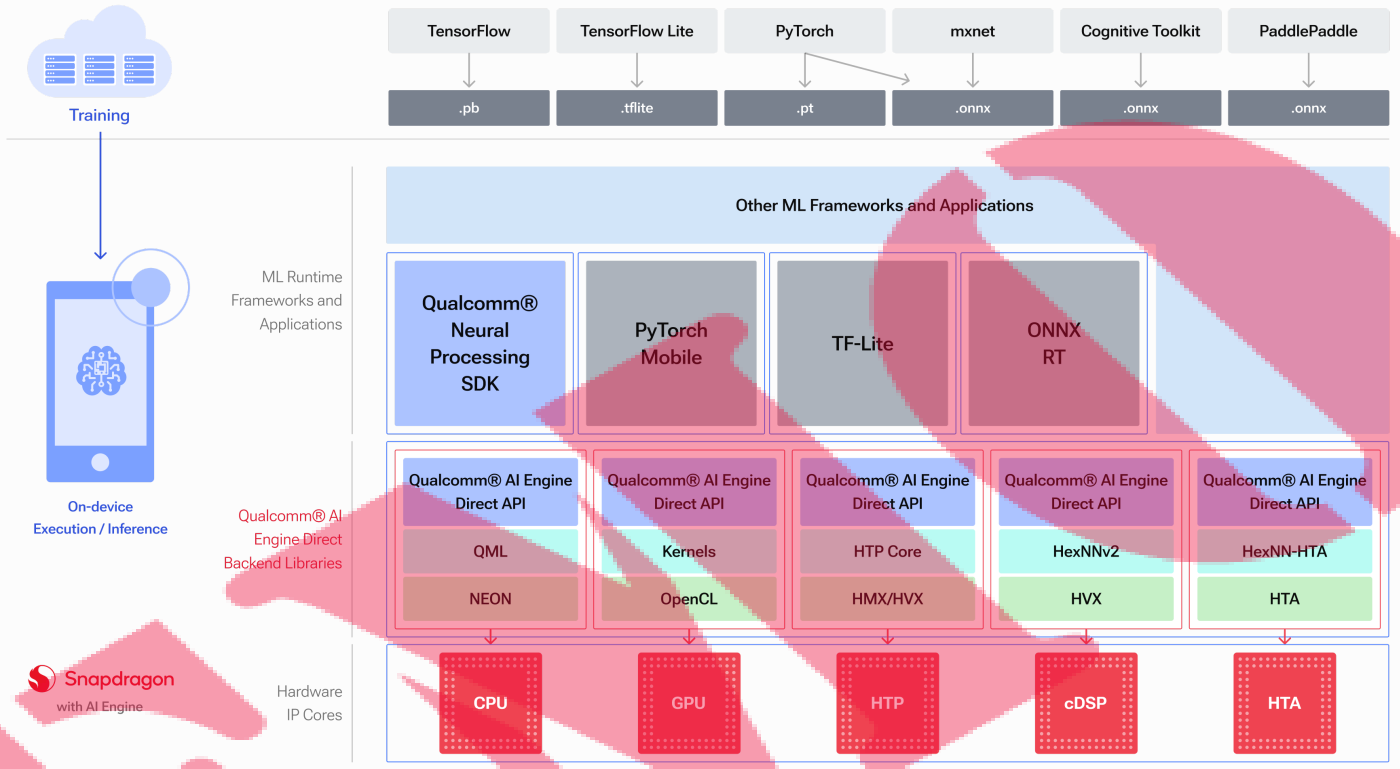
Next

Overview

Qualcomm® AI Engine Direct is Qualcomm Technologies Inc. (QTI) software architecture for AI/ML use cases on QTIs chipsets and AI acceleration cores.

Qualcomm® AI Engine Direct architecture is designed to provide an unified API and modular and extensible per-accelerator libraries which form a reusable basis for full stack AI solutions, both QTI's own and third party frameworks (as illustrated with AI Software Stack with Qualcomm AI Engine Direct diagram).

AI Software Stack with Qualcomm AI Engine Direct



Features

Modularity based on hardware accelerators

The Qualcomm® AI Engine Direct architecture is designed to be modular and allows for clean separation in the software for different hardware cores/accelerators such as the CPU, GPU and DSP that are designated as *backends*. Learn more about Qualcomm® AI Engine Direct backends here.

The Qualcomm® AI Engine Direct backends for different hardware cores/accelerators are compiled into individual core-specific libraries that come packaged with the SDK.

Unified API across IP Cores

One of the key highlights of Qualcomm® AI Engine Direct is that it provides a unified API to delegate operations such as graph creation and execution across all hardware accelerator backends. This allows users to treat Qualcomm® AI Engine Direct as a hardware abstraction API and port applications easily to different cores.

Right level of abstraction

The Qualcomm® AI Engine Direct API is designed to support an efficient execution model with capabilities such as graph optimizations to be taken care of internally. At the same time however, it leaves out broader functionality such as model parsing and network partitioning to higher level frameworks.

Flexibility in composition

With Qualcomm® AI Engine Direct, users can choose appropriate tradeoffs between capabilities provided by the backends and the footprint in terms of library size and memory utilization. This offers the ability to compose a Qualcomm® AI Engine Direct operation package with only operations required to serve a set of models targeted by a use-case [1]. With this, users can create nimble applications with low memory footprint that fits a wide variety of hardware products.

Extensible Operation Support

Qualcomm® AI Engine Direct also provides support for clients to integrate custom operations to work seamlessly alongside the built-in operations.

Improved Execution Performance

With optimized network loading and asynchronous execution support Qualcomm® AI Engine Direct serves to provide a highly efficient interface for ML frameworks and applications to load and execute network graphs on their desired hardware accelerator.

Supported Snapdragon Devices

Snapdragon Device/Chip	Supported Toolchains	SOC Model [2]	Hexagon Arch
Snapdragon 8cx Gen 4 (SC8380XP)	aarch64-windows-msvc arm64x-windows-msvc	60	V73
Snapdragon 8cx Gen 3 (SC8280X)	aarch64-windows-msvc	37	V68
Snapdragon 7c Gen 2 (SC7280X)	aarch64-windows-msvc	44	V68
SD 8 Gen 3 (SM8650)	aarch64-android	57	V75
SD 8 Gen 2 (SM8550)	aarch64-android	43	V73
SD 8+ Gen 1 (SM8475)	aarch64-android	42	V69
SD 8 Gen 1 (SM8450)	aarch64-android	36	V69
888+ (SM8350P) 888 (SM8350)	aarch64-android	30	V68
7 Gen 1 (SM7450)	aarch64-android	41	V69
778G (SM7325)	aarch64-android	35	V68
QCM6490	aarch64-android aarch64-ubuntu-gcc9.4 aarch64-oe-linux-gcc11.2	35	V68
865 (SM8250)	aarch64-android	21	V66
765 (SM7250)	aarch64-android	25	V66
750G (SM7225) 690 (SM6350)	aarch64-android	29	V66 V66
QRB5165	aarch64-ubuntu-gcc7.5	21	V66

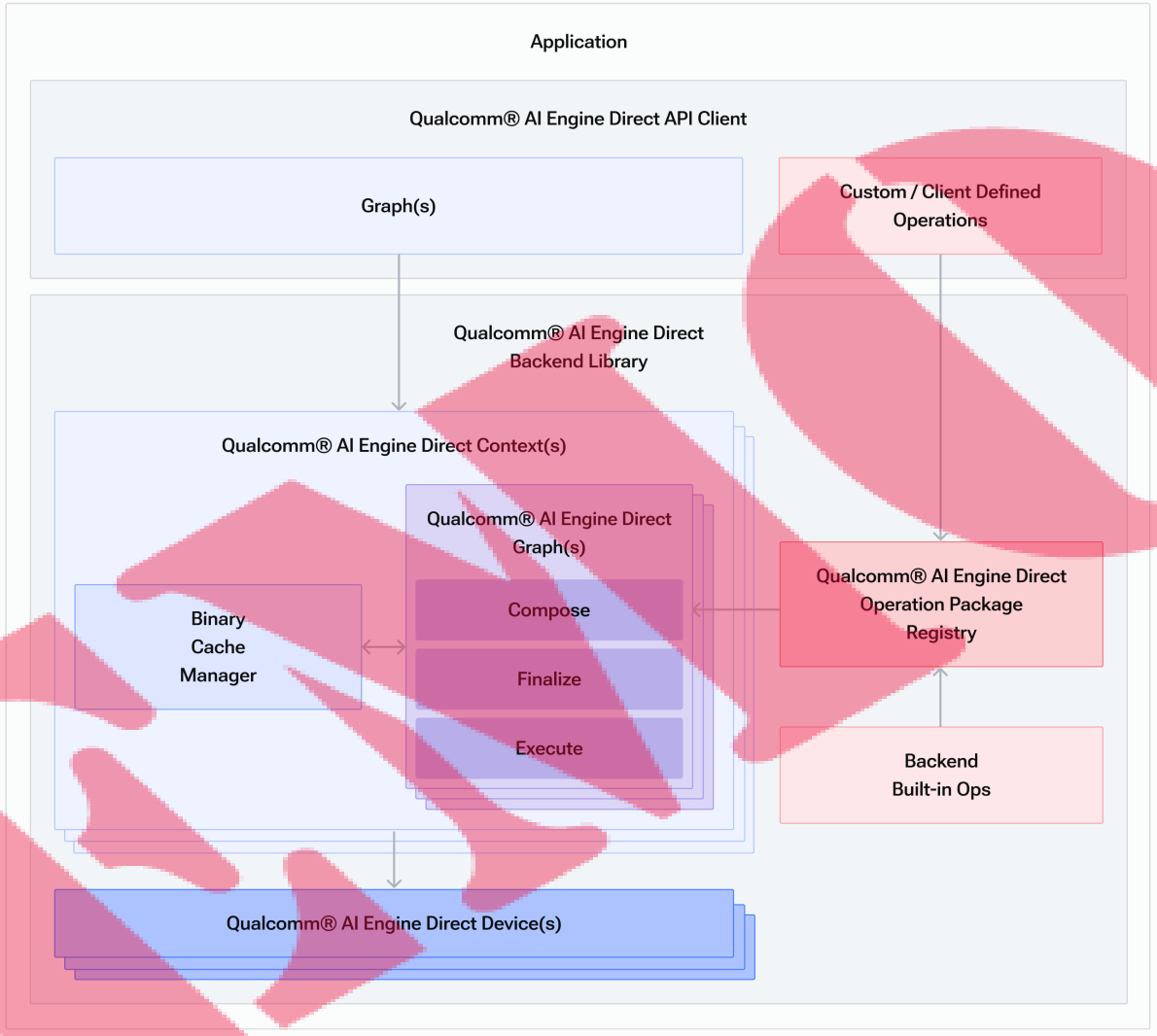
	aarch64-oe-linux-gcc9.3		
QCS7230	aarch64-android aarch64-oe-linux-gcc9.3	51	V66
680 (SM6225)	aarch64-android	40	V66
480 (SM4350) 695 (SM6375)	aarch64-android	31	V66 V66
460 (SM4250) 662 (SM6115) QCM4290	aarch64-android	28	V66 V66 V66
QCS610	aarch64-android aarch64-oe-linux-gcc9.3	16	V66
QCS410	aarch64-android aarch64-oe-linux-gcc9.3	33	V66
QCM6125	aarch64-android	19	V66
QRB4210	aarch64-oe-linux-gcc9.3	49	V66
QCM4490	aarch64-android	59	N/A
780G (SM7350)	aarch64-android	32	V68
SM8325	aarch64-android	34	V68
SM7315	aarch64-android	38	V68
6 Gen 1 (SM6450)	aarch64-android	50	V73
7+ Gen 2 (SM7475)	aarch64-android	54	V69
4 Gen 2 (SM4450)	aarch64-android	59	N/A
8s Gen 3 (SM8635)	aarch64-android	68	V73

7+ Gen 3 (SM7675)	aarch64-android	70	V73

Software Architecture

Qualcomm® AI Engine Direct API and the associated software stack provides all the constructs required by an application to construct, optimize and execute network models on the desired hardware accelerator core. Key constructs are illustrated by the Qualcomm AI Engine Direct Components - High Level View diagram.

Qualcomm AI Engine Direct Components - High Level View



Device

Software abstraction of a hardware accelerator platform. Provides all constructs required to associate desired hardware accelerator resources for execution of user composed graphs. A platform is broken down into potentially multiple devices. Devices may have multiple cores.

Backend

The backend is a top level API component which hosts and manages most of the backend resources required for graph composition and execution, including an operation registry that stores all available operations. Learn more about Qualcomm® AI Engine Direct backends here.

Context

A construct that represents all Qualcomm® AI Engine Direct components required to sustain a user application. Hosts networks provided by the user and allows constructed entities to be cached into serialized objects for future use. It enables interoperability between multiple graphs by providing a shareable memory space in which tensors can be exchanged between graphs.

Graph

The Qualcomm® AI Engine Direct way of representing a loadable network model. Consists of nodes that represent operations and tensors that interconnect them to compose a directed acyclic graph. The Qualcomm® AI Engine Direct graph construct supports APIs that perform initialization, optimization and execution of network models.

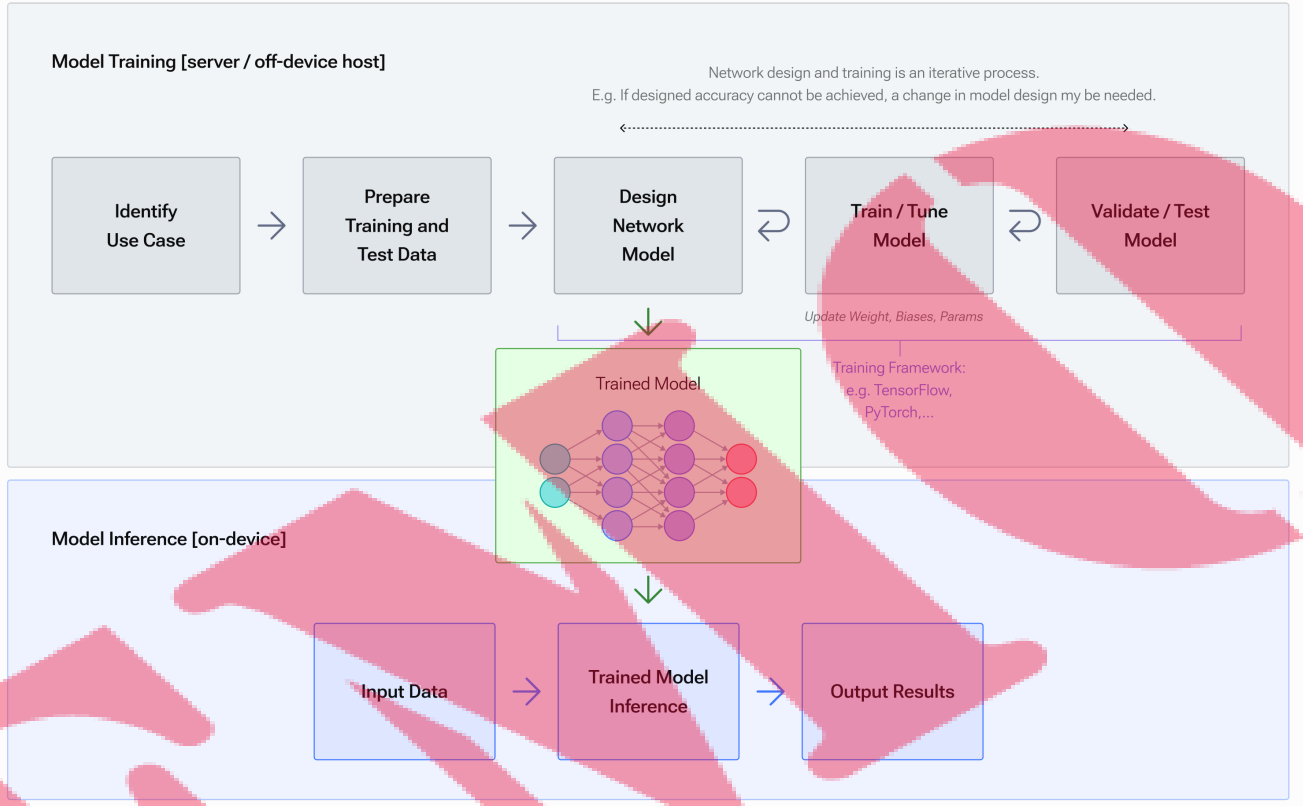
Operation Package Registry

A registry that maintains record of all operations available to execute a model. These operations can be built-in or supplied by the user as Custom Operations. Learn more about operation packages here.

Integration Workflow

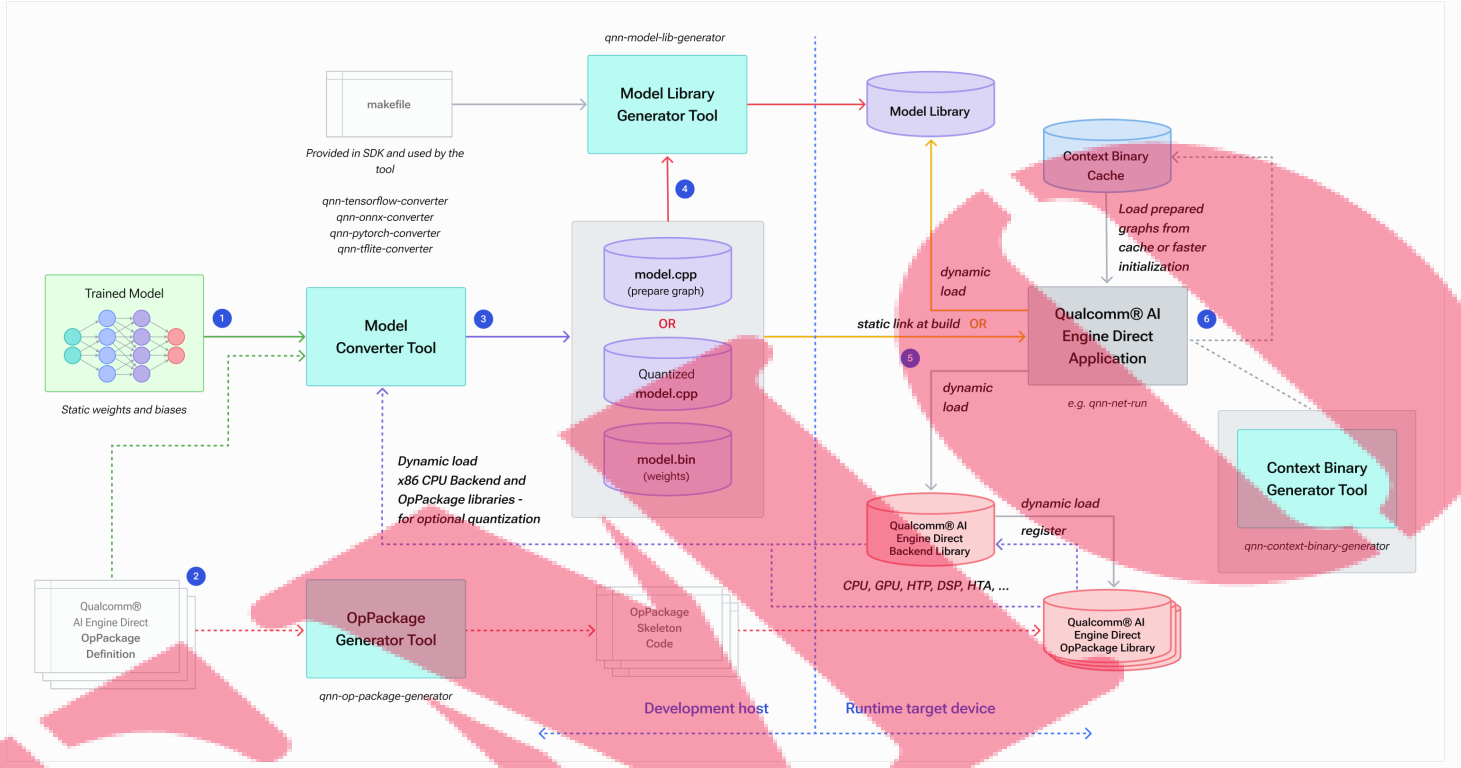
The Qualcomm® AI Engine Direct SDK provides tools and extensible per-accelerator libraries with uniform API enabling flexible integration and efficient execution of ML/DL neural networks on QTI chipsets. The Qualcomm® AI Engine Direct API is designed to support inference of trained neural networks and as such clients are responsible for training a ML/DL network in a training framework of their choice. The training process is normally done on server hosts, off-device. Once network is trained clients can use Qualcomm® AI Engine Direct to get it ready to deploy and run on-device. This workflow is illustrated with the Training vs. Inference Workflow diagram.

Training vs. Inference Workflow



The Qualcomm® AI Engine Direct SDK includes tools to aid clients in integrating trained DL networks into their applications. The basic integration workflow is illustrated with the Qualcomm AI Engine Direct Integration Workflow diagram.

Qualcomm AI Engine Direct Integration Workflow



1. Clients call Qualcomm® AI Engine Direct converter tool by providing their trained network model file as input. The network must be trained in a framework supported by Qualcomm® AI Engine Direct converter tools. See Tools section for more details on Qualcomm® AI Engine Direct converters.
2. When source models contain operations that are not supported natively by Qualcomm® AI Engine Direct backend, clients need to provide OpPackage definition files to the converter, expressing custom / client defined operations. Optionally, they can use the OpPackage generator tool in order to generate skeleton code to implement and compile their custom operations into OpPackage libraries. See qnn-op-package-generator for usage details.
3. Qualcomm® AI Engine Direct model converter is a tool aiding clients in writing a sequence of Qualcomm® AI Engine Direct API calls to construct Qualcomm® AI Engine Direct graph representation of a trained network which was provided as input to the tool. The converter outputs the following files:

- `.cpp` source file (e.g. `model.cpp`) containing required Qualcomm® AI Engine Direct API calls to construct a network graph
- `.bin` binary file (e.g. `model.bin`) containing network weights and biases as float32 data

Clients can optionally direct converter to output a quantized model instead of the default one, as indicated in the diagram as *quantized model.cpp*. In this case `model.bin` file will contain quantized data, and `model.cpp` will reference quantized tensor data types and include quantization encodings. Quantized models may be required by some Qualcomm® AI Engine Direct backend libraries, e.g. HTP or DSP (see general/api:Backend Supplements for information on supported data types). For details on converter quantization function and options see Quantization Support.

4. Clients optionally can use Qualcomm® AI Engine Direct model library generator tool to produce a model library. See qnn-model-lib-generator for usage details.
5. Clients integrate Qualcomm® AI Engine Direct model into their application by either dynamically loading model library or compiling and statically linking `model.cpp` and `model.bin`. In order to prepare and execute model (i.e. run inference), clients also need to load required Qualcomm® AI Engine Direct backend accelerator and OpPackage libraries. Qualcomm® AI Engine Direct OpPackage libraries are registered with and loaded by the backend.
6. Clients can optionally save context binary cache with prepared and finalized graphs. See Context Caching for reference. Such graphs can be repeatedly loaded from the cache without the need for model .cpp / library any further. Loading model graph from the cache is significantly faster than preparing through a sequence of graph composition API calls provided in model .cpp / library. Cached graphs cannot be further modified; they are meant for deployment of prepared graphs, enabling faster initialization of client applications.
7. Clients can optionally utilize Deep Learning Containers (DLCs) produced from Qualcomm Neural Processing SDK in conjunction with the provided `libQnnModelDlc.so` library to produce QNN graph handles from DLC paths in their application. This provides a single format for use across products and support for large models that cannot be compiled into a shared model library. Details on usage can be found in Utilizing DLCs.

Developers on Linux

Executables and libraries can be found under the target folder of the SDK with “linux”, “ubuntu” or “android” in the name. See Release Folder for Different Platforms for reference. Operation mentioned above can be run under Linux OS such as Ubuntu system.

Integration Workflow on Windows

Developers on Windows

The Qualcomm SDK provide three different platforms for Windows host. For users familiar to Linux operation, we suggest using WSL (Windows Subsystem for Linux) on Windows. For developers who want to use tools on Windows-PC directly through the PowerShell environment, Qualcomm® AI Engine Direct provides tools based on x86_64-windows. Please check the following prerequisites.

WSL platform

For WSL developers: The workflow on a Windows host is the same as on a Linux host, though some steps will require execution on WSL (x86) and others will be executed natively on Windows as outlined below. Because WSL is run on GNU/Linux environment, model tools and libraries should get from x86_64-linux-clang respectively. To understand more about WSL setup visit Linux Platform Dependency.

Windows Platform

For Windows native/x86_64 PC developers: The tools and libraries are located within the “x86_64-windows-msvc” folder. Tools are highly related to Python version and setting. Therefore, the environment setup for PowerShell is required before operation. Please refer to the setting for the Windows Windows Platform Dependencies

For Windows on Snapdragon developers: The tools and libraries are located within the “aarch64-windows-msvc” folder. The environment setup for PowerShell is required before operation. Please refer to the setting for the Windows Windows Platform Dependencies

1. For OP Customization, the op package skeleton code is generated by running the Linux OpPackage Generator Tool on WSL (x86).

2. For Context Binary Generation, clients can use either Linux Context Binary Generator Tool on WSL (x86) or windows-native PowerShell. The tool executable and libraries used need to be from corresponding folder, as mentioned in Release Folder for Different Platforms.
3. For Model Library Generation, the model library is produced by running the Windows Model Library Generator Tool natively on Windows.
4. Tools mentioned in "Integration Workflow" can be applied through WSL or Windows native x86. The supporting list of the tools by platform is demonstrated in the Tools.
5. The ARM64X package format is supported for CPU and HTP backend on SC8380XP. The tools and libraries are located within the "arm64x-windows-msvc" folder. See ARM64X Tutorial for the usage and details.

Note

When using WSL, the Model Tools must be obtained from the `linux` folder as it is a subsystem for Linux.

Note

When run on Windows natively, the Model Library Generation Tool must be run with python. See Model Build on Windows Host section for an example.

Notes

[1] : Future feature.

[2] : The SOC Model number is intended to be used in QNN API calls, for example when configuring the QNN HTP backend.

Setup

This page will outline the setup required for using the Qualcomm® AI Engine Direct SDK. The page is split into Linux Platform Dependencies , Windows Platform Dependencies and Environment Setup.

Linux Platform Dependencies

Host OS

The Linux host operating system that Qualcomm® AI Engine Direct SDK is verified with is **Ubuntu 22.04 LTS (Jammy)**. Qualcomm® AI Engine Direct SDK is also verified to work in the Windows Subsystem for Linux (WSL2) environment version 1.1.3.0, currently limited to Linux host runnable artifacts such as converters, model generation and run tools (see Tools for more details). If you wish to set up your own WSL2 environment, you can follow the instructions at https://learn.microsoft.com/en-us/windows/wsl/install.

Python

This Qualcomm® AI Engine Direct SDK distribution is supported on python3 only. Download and installation instructions vary based on the host OS. The SDK has been tested with python3.10. If python3.10 is not already installed on your system, you may install it with the following commands:

```
$ sudo apt-get update
$ sudo apt-get install python3.10 python3-distutils libpython3.10
```

Virtual Environment (VENV)

Whether managing multiple python installations or just to keep your system python installation clean we recommend using python virtual environments.

```
$ sudo apt-get install python3.10-venv
$ python3.10 -m venv "<PYTHON3.10_VENV_ROOT>"
$ source <PYTHON3.10_VENV_ROOT>/bin/activate
```

Note

If your environment is in WSL, <PYTHON3.10_VENV_ROOT> must be under $HOME.

Additional Packages

Some additional python packages need to be available in your environment in order to be able to interact with Qualcomm® AI Engine Direct components and tools. This Qualcomm® AI Engine Direct release is verified to work with the versions of the packages specified below:

Package	Version
absl-py	2.1.0
attrs	23.2.0
dash	2.12.1
decorator	4.4.2
invoke	1.7.3
joblib	1.4.0
jsonschema	4.19.0
lxml	5.2.1
Mako	1.1.0

matplotlib	3.3.4
mock	3.0.5
numpy	1.26.4
opencv-python	4.5.4.58
optuna	3.3.0
packaging	24.0
pandas	2.0.1
paramiko	3.4.0
pathlib2	2.3.6
Pillow	10.2.0
plotly	5.20.0
protobuf	3.19.6
psutil	5.6.4
pytest	8.1.1
PyYAML	5.3
scikit-optimize	0.9.0
scipy	1.10.1
six	1.16.0
tabulate	0.9.0
typing-extensions	4.10.0
xlsxwriter	1.2.2

Note

Matplotlib with 3.5.0 and Pandas with 2.0.2 have been verified on Windows

Run the following script to check and install missing dependencies:

```
$ python3 -m pip install --upgrade pip
$ ${QNN_SDK_ROOT}/bin/check-python-dependency
```

Optional packages

Some additional model- and dataset-specific python packages need to be installed to interact with the accuracy evaluator. The following table lists the packages and the recommended version.

Package	Version
pycocotools	2.0.6
transformers	4.31.0
sacrebleu	2.3.1
scikit-learn	1.3.0
OpenNMT-py	2.3.0
sentencepiece	0.1.98

Note

To set `QNN_SDK_ROOT` see Environment Setup for Linux.

Linux

Compiling artifacts for the x86_64 target requires clang++. This Qualcomm® AI Engine Direct SDK release is verified to work with clang-14.

Run the following script to check and install missing linux dependencies:

```
# Note: the following command should be run as administrator/root to be able to install system lib

$ sudo bash ${QNN_SDK_ROOT}/bin/check-linux-dependency.sh
```

Note

To set `QNN_SDK_ROOT` see Environment Setup for Linux.

ML Frameworks

In order to convert ML models trained on different frameworks into intermediate representations consumable by Qualcomm® AI Engine Direct you may need to download and install the corresponding frameworks on your host machine.

This Qualcomm® AI Engine Direct release is verified to work with the following versions of the ML training frameworks.

- **TensorFlow:** tf-2.10.1
- **TFLite:** tflite-2.3.0
- **PyTorch:** torch-1.13.1
- **Torchvision:** torchvision-0.14.1
- **ONNX:** onnx-1.12.0
- **ONNX Runtime:** onnxruntime-1.17.1
- **ONNX Simplifier:** onnxsim-0.4.36

Toolchains

The Qualcomm® AI Engine Direct SDK allows users to compile custom operation packages to be used with the different backends such as the CPU, GPU, HTP, DSP, and others. You may need to install appropriate cross-compilation toolchains in order to compile such packages for a particular backend.

MAKE

Operation packages are compiled with a front-end that is written with Makefiles. If make is not available on your host machine install it using the following command:

```
$ sudo apt-get install make
```

Android NDK

This Qualcomm® AI Engine Direct release is verified to work with Android NDK version r26c. The same can be downloaded from https://dl.google.com/android/repository/android-ndk-r26c-linux.zip. Once the zip file is downloaded and extracted, the extracted location needs to be added to PATH environment variable.

To set the environment to use Android NDK, the following commands can be used to set and check proper configuration:

```
$ export ANDROID_NDK_ROOT=<PATH-TO-NDK>
$ export PATH=${ANDROID_NDK_ROOT}:${PATH}
$ ${QNN_SDK_ROOT}/bin/envcheck -n
```

Note

If your environment is in WSL, Android NDK must be unzipped under $HOME with unzip command of WSL.

clang-14

This Qualcomm® AI Engine Direct release is verified to work with clang-14. Please refer to Linux Dependency section to help with installation of the same.

To check if the environment is setup properly to use clang-14, the following command can be used:

```
$ ${QNN_SDK_ROOT}/bin/envcheck -c
```

CPU

The x86_64 targets are built using clang-14 (see Linux Dependency). The ARM CPU targets are built using Android NDK (see Android NDK).

GPU

The GPU backend kernels are written based on OpenCL. The GPU operations must be implemented based on OpenCL headers with minimum version OpenCL 1.2.

HTP/DSP

Compiling for HTP/DSP requires the use of the Hexagon toolchain available from Qualcomm® Hexagon SDK.

Hexagon SDK installation on Linux

- The Hexagon SDK versions are available at https://developer.qualcomm.com/software/hexagon-dsp-sdk/tools.

Hexagon SDK installation on WSL2 20.04

- Download Hexagon SDKs from https://qpm.qualcomm.com in a Linux PC.
- Copy Hexagon SDKs from Linux PC to Windows PC.

This Qualcomm® AI Engine Direct release is verified to work with:

Backend	Hexagon Architecture	Hexagon SDK Version	Hexagon Tools Version
HTP	V75	5.4.0	8.7.03
HTP	V73	5.4.0	8.6.02
HTP	V69	4.3.0	8.5.03
HTP	V68	4.2.0	8.4.09
DSP	V66	4.1.0	8.4.06
DSP	V65	3.5.2	8.3.07

Additionally, compiling for HTP/DSP requires clang++.

Further setup instructions are available at `$HEXAGON_SDK_ROOT/docs/readme.html`, where `HEXAGON_SDK_ROOT` is the location of the Hexagon SDK installation.

> **Note**
>
> Hexagon SDK tools version 8.4.09/8.4.06/8.3.07 is **not** currently pre-packaged into Hexagon SDK version 4.2.0/4.1.0/3.5.2 respectively. It needs to be downloaded separately and placed at the location $HEXAGON_SDK_ROOT/tools/HEXAGON_Tools/ .

Windows Platform Dependencies

Host OS

Qualcomm® AI Engine Direct SDK is verified with **Windows 10** and **Windows 11** OS on a x86 host platform and with **Windows 11** OS on a Snapdragon platform.

Python

This Qualcomm® AI Engine Direct SDK distribution is supported on python3 only. The SDK has been tested with python3.10. Please download and install it with

https://www.python.org/ftp/python/3.10.11/python-3.10.11-amd64.exe.

After installation, check dependencies by running the following script in `PowerShell` terminal window.

```
$ py -3.10 -m venv "<PYTHON3.10_VENV_ROOT>"
$ & "<PYTHON3.10_VENV_ROOT>\Scripts\Activate.ps1"
$ python -m pip install --upgrade pip
$ python "${QNN_SDK_ROOT}\bin\check-python-dependency"
```

Note

You can set `QNN_SDK_ROOT` environment variable by following Environment Setup for Windows.

Windows

Compiling artifacts for the Windows targets requires **Visual Studio** setup. Qualcomm® AI Engine Direct SDK was verified with the following build environment.

- Visual Studio 2022 17.5.1
- MSVC v143 - VS 2022 C++ x64/x86 build tools - 14.34
- MSVC v143 - VS 2022 C++ ARM64/ARM64EC build tools - 14.34
- Windows SDK 10.0.22621.0
- MSBuild support for LLVM (clang-cl) toolset
- C++ Clang Compiler for Windows (15.0.1)
- C++ CMake tools for Windows

Run the following script in Powershell terminal as **Administrator** to check and install missing Windows dependencies:

```
$ & "${QNN_SDK_ROOT}/bin/check-windows-dependency.ps1"
```

> **Note**
>
> You can set `QNN_SDK_ROOT` environment variable by following Environment Setup for Windows.

To just check if the environment satisfies the requirements, run the following command in your **Developer Powershell terminal**:

```
$ & "${QNN_SDK_ROOT}/bin/envcheck.ps1" -m
```

Environment Setup

Linux

After Linux Platform Dependencies have been satisfied, the user environment can be set with the provided **envsetup.sh** script.

Open a command shell on Linux host and run:

```
$ source ${QNN_SDK_ROOT}/bin/envsetup.sh
```

This will set/update the following environment variables:

- QNN_SDK_ROOT
- PYTHONPATH
- PATH
- LD_LIBRARY_PATH

`${QNN_SDK_ROOT}` represents the full path to Qualcomm® AI Engine Direct SDK root.

The QNN API headers are in `${QNN_SDK_ROOT}/include/QNN`.

The tools are in `${QNN_SDK_ROOT}/bin/x86_64-linux-clang`.

Target specific backend and other libraries are in `${QNN_SDK_ROOT}/lib/*/`.

TensorFlow Setup

Install TensorFlow as a standalone Python module using https://pypi.org/project/tensorflow/2.10.1/ To ensure TensorFlow is in your PYTHONPATH, run the following command.

```
python3 -c "import tensorflow"
```

See ML Frameworks for information on versions Qualcomm® AI Engine Direct SDK was verified with. Note that

installing versions of TensorFlow besides 2.10.1 may update dependencies (e.g. numpy).

ONNX Setup

Install ONNX as a standalone Python module using https://pypi.org/project/onnx/1.12.0/ and ensure ONNX is in your PYTHONPATH by running the following command.

```
python3 -c "import onnx"
```

See ML Frameworks for information on versions Qualcomm® AI Engine Direct SDK was verified with.

ONNX Runtime Setup

Install ONNX Runtime as a standalone Python module using https://pypi.org/project/onnxruntime/1.17.1/ and ensure ONNX Runtime is in your PYTHONPATH by running the following command.

```
python3 -c "import onnxruntime"
```

See ML Frameworks for information on versions Qualcomm® AI Engine Direct SDK was verified with.

ONNX Simplifier Setup

Install ONNX Simplifier as a standalone Python module using https://pypi.org/project/onnxsim/0.4.36/ and ensure ONNX Simplifier is in your PYTHONPATH by running the following command.

```
python3 -c "import onnxsim"
```

See ML Frameworks for information on versions Qualcomm® AI Engine Direct SDK was verified with.

TFLite Setup

Install TFLite as a standalone Python module using https://pypi.org/project/tflite/2.3.0/ and ensure TFLite is in your PYTHONPATH by running the following command.

```
python3 -c "import tflite"
```

See ML Frameworks for information on versions Qualcomm® AI Engine Direct SDK was verified with.

PyTorch Setup

Install PyTorch v1.13.1 as a standalone Python module using https://pytorch.org/get-started/previous-versions/#v1131 and ensure PyTorch is in your PYTHONPATH by running the following command.

```
python3 -c "import torch"
```

See ML Frameworks for information on versions Qualcomm® AI Engine Direct SDK was verified with.

Torchvision Setup

Install Torchvision as a standalone Python module using https://pypi.org/project/torchvision/0.14.1/ and ensure Torchvision is in your PYTHONPATH by running the following command.

```
python3 -c "import torchvision"
```

See ML Frameworks for information on versions Qualcomm® AI Engine Direct SDK was verified with.

Windows

After Windows Platform Dependencies have been satisfied, the user environment can be set with the provided **envsetup.ps1** script.

First, open `Developer PowerShell for VS2022` as Administrator.

```
$ Set-ExecutionPolicy RemoteSigned
```

Then, execute the following script.

```
$ & "<QNN_SDK_ROOT>\bin\envsetup.ps1"
```

This will set/update the following environment variables:

- QNN_SDK_ROOT

`${QNN_SDK_ROOT}` represents the full path to Qualcomm® AI Engine Direct SDK root.

The QNN API headers are in `${QNN_SDK_ROOT}/include/QNN`.

The tools are in `${QNN_SDK_ROOT}/bin/x86_64-linux-clang`.

Target specific backend and other libraries are in `${QNN_SDK_ROOT}/lib/*/`.

Note

`envsetup.ps1` is only required to be executed on Windows host.

Previous Next

Backend

Clients interact with Qualcomm® AI Engine Direct through backends. In general terms, a Qualcomm® AI Engine Direct backend is a software entity that implements QNN API, and is typically compiled in the form of a shared library. The terms *QNN backend* and *QNN backend library* are often used interchangeably.

QNN backend libraries provide integration modularity per hardware accelerator core (see Overview for more details).

The QNN SDK provides several backend libraries. These libraries can be found in `<QNN_SDK_ROOT>/lib/<target-platform>` folders. QNN backend libraries follow naming rules that are platform dependent (see Platform Differences for more details). In the Available Backend Libraries table Library Description, the backend library is referred to by its platform agnostic name.

Available QNN SDK Backend libraries

Backend Name	Backend Description	Target and Library Names	Library Descr
CPU	Backend for Snapdragon™ CPU hardware core(s)	• aarch64-android • x86_64-linux-clang ◦ `libQnnCpu.so` • aarch64-windows-msvc • arm64x-windows-msvc • x86_64-windows-msvc	`QnnCpu` : CPU backe name used across

		◦ QnnCpu.dll	
DSP	Backend for Hexagon™ DSP hardware accelerator	• hexagon-v## ◦ libQnnDspV##Skel.so ◦ libQnnDspV##.so • aarch64-android ◦ libQnnDsp.so ◦ libQnnDspV##Stub.so • aarch64-windows-msvc ◦ QnnDsp.dll ◦ QnnDspV##Stub.dll	**Note: v## indicates libr hexagon architecture available** libQnnDspV##Skel. library which is to be corresponding QnnDspV device CPU side and is executing graphs on the as a proxy for CPU s QnnDspV##Stub : DSI library that commu libQnnDspV##Skel.so RPC chanr QnnDsp : DSP device C library, loads the corresp library based on libQnnDspV##.so backend library that integration on DSP v
GPU	Backend for Adreno™ GPU hardware accelerator	• aarch64-android ◦ libQnnGpu.so	QnnGpu : GPU backe name used across

HTP	Backend for Hexagon™ HTP hardware accelerator	• hexagon-v## ◦ `libQnnHtpV##Skel.so` ◦ `libQnnHtpV##.so` • aarch64-android ◦ `libQnnHtp.so` ◦ `libQnnHtpPrepare.so` ◦ `libQnnHtpV##Stub.so` • x86_64-linux-clang ◦ `libQnnHtp.so` • aarch64-windows-msvc • arm64x-windows-msvc ◦ `QnnHtp.dll` ◦ `QnnHtpPrepare.dll` ◦ `QnnHtpV##Stub.dll`	**Note: v## indicates libr hexagon architecture available** `libQnnHtpV##Skel.` library which is to be corresponding QnnHtpV device CPU side and is executing graphs on the as a proxy for CPU s `QnnHtpV##Stub` : HTI library that commu libQnnHtpV##Skel.so via RPC cha `QnnHtp` : HTP backer are two variants: (aar Android device CPU-side loads the corresponding based on the SoC; (x8 Linux host backend functional simulator fo accelerator and sup preparation on x86 `libQnnHtpV##.so` backend library that integration on HTP `QnnHtpPrepare` : H provides functionality t finalize graphs on the de side. This library is auto by HTP backend library the following operations QnnBackend_validat QnnGraph_addI QnnGraph_final QnnBackend_registerO "CPU" target. If these op requested, this library c be on the de
HTA	Backend for Snapdragon™ HTA hardware accelerator	• aarch64-android ◦ `libQnnHta.so`	`QnnHta` : HTA backer name used across all su
LPAI	Backend for Snapdragon™	• hexagon-v##	**Note:**

LPAI hardware accelerator	◦ `libQnnLpaiV##Skel_v##.so` • aarch64-android ◦ `libQnnLpai.so` ◦ `libQnnLpaiV##Stub.so` • x86_64-linux-clang ◦ `libQnnLpai.so` ◦ `libQnnLpaiSim_v##.so` ◦ `libQnnLpaiPrepare_v##.so`	• `LpaiV##` indi for different hex architecture ver • `Skel_v##` ind libraries for diff architecture ver • `Sim_v##` indi Simulator librar eNPU architect • `Prepare_v##` compiler librari eNPU architect • `libQnnLpaiV##` : LPAI native library loaded using corre QnnLpaiV##Stub. device CPU side ar responsible for exe on the LPAI accele for CPU side backe • `QnnLpaiV##Stu` CPU-side proxy libr communicates wit libQnnLpaiV##Ske the LPAI side via RI • `libQnnLpai.so` backend library. Th variants: (aarch64- Android device CPI library, loads the c LPAI stub library ba SoC; (x86_64-linux host backend serv functional simulato hardware accelera supports graph pre x86_64 hosts.

			• libQnnLpaiSim x86_86 host simul specific eNPU arch library is automatic LPAI backend libra the following opera requested: 1) QnnContext_create QnnContext_create • libQnnLpaiPre : x86_86 host mod library for specific architecture. This l automatically load backend library wh following operatio 1) QnnBackend_valid 2) QnnGraph_addN QnnGraph_finalize
Saver	Backend which captures QNN API calls and saves them to a file as they are issued by a client. This can later be replayed for support and troubleshooting.	• aarch64-android • x86_64-linux-clang • hexagon-v68 ◦ libQnnSaver.so • aarch64-windows-msvc • x86_64-windows-msvc ◦ QnnSaver.dll	QnnSaver : Saver b

Backend Specific Pages

- DSP
- HTP
- HTA
- LPAI

- CPU
- GPU
- Saver

Previous

Next

Op Packages

Operation Packages in Qualcomm® AI Engine Direct, also called 'Op Packages' refer to a collection of operations that are made available to a backend in order to be utilized in creating and executing Qualcomm® AI Engine Direct graphs representing network models.

Qualcomm® AI Engine Direct software architecture is designed to allow the memory footprint of applications that use QNN to be highly customized, depending on the required backends and operations to run their models with. Operation packages are separated from the backends by virtue of being compiled into shared libraries distinct from the core backend libraries. This allows users to compile lean applications by packaging any number of operations they need for their use-cases into Op packages. This is especially desirable in deeply embedded IOT-type use-cases which operate within strict memory limits and target just a handful of network models tailored for their needs.

A Qualcomm® AI Engine Direct backend registers Op packages supplied by users with the API QnnBackend_registerOpPackage().

Multiple Op packages can be registered with a backend, and are shared between contexts and graphs created within the backend as individual, distinct instances. Their lifetime and scope matches the scope of the backend they are registered in, and hence they can outlast any graphs that need them. Crucially, this also means that external parties can easily create and integrate their custom Op packages with Qualcomm® AI Engine Direct using the same mechanism used by Qualcomm® AI Engine Direct Op packages. All Op packages are required to implement the interfaces defined in QnnOpPackage.h.

See Tutorials for a simple example Op package that demonstrates how external parties can create and use custom ops within the Qualcomm® AI Engine Direct framework.

See the following section for info on generating Op Package skeleton code, which can then be implemented as a full-fledged Qualcomm® AI Engine Direct Op Package.

- Generating Op Packages

Previous

Next

Tools

This page describes the various SDK tools and feature for Linux/Android and Windows developers. For the integration flow of different developers, please refer to Overview page for further information.

Category	Tool	Developer					
		Linux/Android			Windows		
		Ubuntu	WSL x86	Device	WSL x86	Windows x86_64	Windows on Snapdragon
Model Conversion	qnn-tensorflow-converter	YES	YES		YES	YES	YES**
	qnn-tflite-converter	YES	YES		YES		
	qnn-pytorch-converter	YES	YES		YES		
	qnn-onnx-converter	YES	YES		YES	YES	YES**
Model Preparation	Quantization Support	YES	YES		YES	YES	YES
	qnn-model-lib-generator	YES	YES			YES	YES
	qnn-op-package-generator	YES	YES		YES		

	qnn-context-binary-generator	YES	YES	YES	YES	YES	YES
Execution	qnn-net-run	YES	YES	YES		YES	YES
	qnn-throughput-net-run	YES	YES	YES			YES
Analysis	qnn-quantization-checker (Beta)	YES			YES	YES	
	qnn-accuracy-evaluator (Beta)	YES					
	qnn-architecture-checker (Beta)	YES	YES		YES	YES	YES**
	qnn-accuracy-debugger (Beta)	YES	YES			YES	
	qnn-platform-validator	YES		YES			YES
	qnn-profile-viewer	YES	YES	YES		YES*	YES*
	Benchmarking	YES					
	qnn-netron (Beta)	YES					
	qnn-context-binary-utility	YES					

Note

The **Beta designation** indicates **pre-production quality**. This means that the component is currently undergoing more rigorous testing and may not fully satisfy compatibility requirements as expected in the production version. In other words, **incompatible changes** (such as alterations in behavior or interface) between releases are allowed without prior notice, although every effort is made to minimize such changes.

Note

* When using converter tools in Windows PowerShell, make sure a virtual environment with the required python packages (see Setup for more details) is activated and converters are executed via **python**, as shown in the following example.

    (venv-3.10) > python qnn-onnx-converter <options>

Note

- Extension naming of library: For Windows developers, please replace all '.so' files with the analogous '.dll' file in the following sections. Please refer to Platform Differences for more details.
- For more detailed information on converters please refer to Converters.
- [*] libQnnGpuProfilingReader.dll is not supported on Windows platform for qnn-profile-viewer.
- [**] Requires the python scripts and the executables from the Windows x86_64 binary folder(bin\x86_64-windows-msvc).

Model Conversion

qnn-tensorflow-converter

The **qnn-tensorflow-converter** tool converts a model from the TensorFlow framework to a CPP file representing the model as a series of QNN API calls. Additionally, a binary file containing static weights of the model is produced.

```
usage: qnn-tensorflow-converter -d INPUT_NAME INPUT_DIM --out_node OUT_NAMES
                                [--input_type INPUT_NAME INPUT_TYPE]
                                [--input_dtype INPUT_NAME INPUT_DTYPE] [--input_encoding INPUT_ENC
                                [--input_layout INPUT_NAME INPUT_LAYOUT] [--custom_io CUSTOM_IO]
                                [--show_unconsumed_nodes] [--saved_model_tag SAVED_MODEL_TAG]
                                [--saved_model_signature_key SAVED_MODEL_SIGNATURE_KEY]
                                [--quantization_overrides QUANTIZATION_OVERRIDES]
                                [--keep_quant_nodes] [--disable_batchnorm_folding]
                                [--keep_disconnected_nodes] [--input_list INPUT_LIST]
                                [--param_quantizer PARAM_QUANTIZER] [--act_quantizer ACT_QUANTIZER
                                [--algorithms ALGORITHMS [ALGORITHMS ...]] [--bias_bw BIAS_BW]
                                [--act_bw ACT_BW] [--weight_bw WEIGHT_BW] [--ignore_encodings]
                                [--use_per_row_quantization]
                                [--use_per_channel_quantization [USE_PER_CHANNEL_QUANTIZATION [USE
                                [--use_native_input_files] [--use_native_dtype]
                                [--use_native_output_files] --input_network INPUT_NETWORK
                                [--debug [DEBUG]] [-o OUTPUT_PATH] [--copyright_file COPYRIGHT_FIL
                                [--float_bw FLOAT_BW] [--float_bias_bw FLOAT_BIAS_BW] [--overwrite
                                [--exclude_named_tensors] [--op_package_lib OP_PACKAGE_LIB]
                                [--restrict_quantization_steps ENCODING_MIN, ENCODING_MAX]
                                [--converter_op_package_lib CONVERTER_OP_PACKAGE_LIB]
                                [-p PACKAGE_NAME | --op_package_config CUSTOM_OP_CONFIG_PATHS [CUS
                                [-h] [--arch_checker]

Script to convert TF model into QNN

required arguments:
  -d INPUT_NAME INPUT_DIM, --input_dim INPUT_NAME INPUT_DIM
                        The names and dimensions of the network input layers specified in the form
                        [input_name comma-separated-dimensions], for example:
                            'data' 1,224,224,3
                        Note that the quotes should always be included in order to
                        handlespecial characters, spaces, etc.
                        For multiple inputs specify multiple --input_dim on the command line like:
                            --input_dim 'data1' 1,224,224,3 --input_dim 'data2' 1,50,100,3
  --out_node OUT_NODE,  --out_name OUT_NAMES
                        Name of the graph's output nodes. Multiple output nodes should be
                        provided separately like:
                            --out_node out_1 --out_node out_2
  --input_network INPUT_NETWORK, -i INPUT_NETWORK
                        Path to the source framework model.

optional arguments:
  --input_type INPUT_NAME INPUT_TYPE, -t INPUT_NAME INPUT_TYPE
```

```
                        Type of data expected by each input op/layer. Type for each input is
                        |default| if not specified. For example: "data" image.Note that the quotes
                        should always be included in order to handle special characters, spaces,et
                        For multiple inputs specify multiple --input_type on the command line.
                        Eg:
                            --input_type "data1" image --input_type "data2" opaque
                        These options get used by DSP runtime and following descriptions state how
                        input will be handled for each option.
                        Image:
                        Input is float between 0-255 and the input's mean is 0.0f and the input's
                        max is 255.0f. We will cast the float to uint8ts and pass the uint8ts to t
                        DSP.
                        Default:
                        Pass the input as floats to the dsp directly and the DSP will quantize it.
                        Opaque:
                        Assumes input is float because the consumer layer(i.e next layer) requires
                        it as float, therefore it won't be quantized.
                        Choices supported:
                            image
                            default
                            opaque
  --input_dtype INPUT_NAME INPUT_DTYPE
                        The names and datatype of the network input layers specified in the format
                        [input_name datatype], for example:
                            'data' 'float32'.
                        Default is float32 if not specified.
                        Note that the quotes should always be included in order to handle special
                        characters, spaces, etc.
                        For multiple inputs specify multiple --input_dtype on the command line lik
                            --input_dtype 'data1' 'float32' --input_dtype 'data2' 'float32'
  --input_encoding INPUT_ENCODING [INPUT_ENCODING ...], -e INPUT_ENCODING [INPUT_ENCODING ...]
                        Usage:     --input_encoding "INPUT_NAME" INPUT_ENCODING_IN
                        [INPUT_ENCODING_OUT]
                        Input encoding of the network inputs. Default is bgr.
                        e.g.
                            --input_encoding "data" rgba
                        Quotes must wrap the input node name to handle special characters,
                        spaces, etc. To specify encodings for multiple inputs, invoke
                        --input_encoding for each one.
                        e.g.
                            --input_encoding "data1" rgba --input_encoding "data2" other
                        Optionally, an output encoding may be specified for an input node by
                        providing a second encoding. The default output encoding is bgr.
                        e.g.
                            --input_encoding "data3" rgba rgb
```

```
                        Input encoding types:
                            image color encodings: bgr,rgb, nv21, nv12, ...
                            time_series: for inputs of rnn models;
                            other: not available above or is unknown.
                        Supported encodings:
                            bgr
                            rgb
                            rgba
                            argb32
                            nv21
                            nv12
                            time_series
                            other
--input_layout INPUT_NAME INPUT_LAYOUT, -l INPUT_NAME INPUT_LAYOUT
                        Layout of each input tensor. If not specified, it will use the default
                        based on the Source Framework, shape of input and input encoding.
                        Accepted values are-
                            NCDHW, NDHWC, NCHW, NHWC, NFC, NCF, NTF, TNF, NF, NC, F, NONTRIVIAL
                        N = Batch, C = Channels, D = Depth, H = Height, W = Width, F = Feature, T
                        NDHWC/NCDHW used for 5d inputs
                        NHWC/NCHW used for 4d image-like inputs
                        NFC/NCF used for inputs to Conv1D or other 1D ops
                        NTF/TNF used for inputs with time steps like the ones used for LSTM op
                        NF used for 2D inputs, like the inputs to Dense/FullyConnected layers
                        NC used for 2D inputs with 1 for batch and other for Channels (rarely used
                        F used for 1D inputs, e.g. Bias tensor
                        NONTRIVIAL for everything elseFor multiple inputs specify multiple
                        --input_layout on the command line.
                        Eg:
                           --input_layout "data1" NCHW --input_layout "data2" NCHW
--custom_io CUSTOM_IO
                        Use this option to specify a yaml file for custom IO
--show_unconsumed_nodes
                        Displays a list of unconsumed nodes, if there any are found. Nodes which a
                        unconsumed do not violate the structural fidelity of thegenerated graph.
--saved_model_tag SAVED_MODEL_TAG
                        Specify the tag to seletet a MetaGraph from savedmodel. ex:
                        --saved_model_tag serve. Default value will be 'serve' when it is not
                        assigned.
--saved_model_signature_key SAVED_MODEL_SIGNATURE_KEY
                        Specify signature key to select input and output of the model. ex:
                        --saved_model_signature_key serving_default. Default value will be
                        'serving_default' when it is not assigned
--disable_batchnorm_folding
--keep_disconnected_nodes
```

```
                        Disable Optimization that removes Ops not connected to the main graph.
                        This optimization uses output names provided over commandline OR
                        inputs/outputs extracted from the Source model to determine the main graph
  --debug [DEBUG]       Run the converter in debug mode.
  -o OUTPUT_PATH, --output_path OUTPUT_PATH
                        Path where the converted Output model should be saved.If not specified, th
                        converter model will be written to a file with same name as the input mode
  --copyright_file COPYRIGHT_FILE
                        Path to copyright file. If provided, the content of the file will be added
                        to the output model.
  --float_bw FLOAT_BW   Option to select the bitwidth to use when using float for parameters (weig
                        bias) and activations for all ops or a specific op (via encodings) selecte
                        through encoding; values are 32 (default) or 16.
  --overwrite_model_prefix
                        If option passed, model generator will use the output path name to use as
                        model prefix to name functions in <qnn_model_name>.cpp. (Useful for runnin
                        multiple models at once) eg: ModelName_composeGraphs. Default is to use
                        generic "QnnModel_".
  --exclude_named_tensors
                        Remove using source framework tensorNames; instead use a counter for namin
                        tensors. Note: This can potentially help to reduce the final model library
                        that will be generated(Recommended for deploying model). Default is False.
  -h, --help            show this help message and exit

Quantizer Options:
  --quantization_overrides QUANTIZATION_OVERRIDES
                        Use this option to specify a json file with parameters to use for
                        quantization. These will override any quantization data carried from
                        conversion (eg TF fake quantization) or calculated during the normal
                        quantization process. Format defined as per AIMET specification.
  --keep_quant_nodes    Use this option to keep activation quantization nodes in the graph rather
                        than stripping them.
  --input_list INPUT_LIST
                        Path to a file specifying the input data. This file should be a plain text
                        file, containing one or more absolute file paths per line. Each path is
                        expected to point to a binary file containing one input in the "raw" forma
                        ready to be consumed by the quantizer without any further preprocessing.
                        Multiple files per line separated by spaces indicate multiple inputs to th
                        network. See documentation for more details. Must be specified for
                        quantization. All subsequent quantization options are ignored when this is
                        not provided.
  --param_quantizer PARAM_QUANTIZER
                        Optional parameter to indicate the weight/bias quantizer to use. Must be
                        followed by one of the following options: "tf": Uses the real min/max of t
                        data and specified bitwidth (default) "enhanced": Uses an algorithm useful
```

```
                        for quantizing models with long tails present in the weight distribution
                        "adjusted": Uses an adjusted min/max for computing the range, particularly
                        good for denoise models "symmetric": Ensures min and max have the same
                        absolute values about zero. Data will be stored as int#_t data such that t
                        offset is always 0.
  --act_quantizer ACT_QUANTIZER
                        Optional parameter to indicate the activation quantizer to use. Must be
                        followed by one of the following options: "tf": Uses the real min/max of t
                        data and specified bitwidth (default) "enhanced": Uses an algorithm useful
                        for quantizing models with long tails present in the weight distribution
                        "adjusted": Uses an adjusted min/max for computing the range, particularly
                        good for denoise models "symmetric": Ensures min and max have the same
                        absolute values about zero. Data will be stored as int#_t data such that t
                        offset is always 0.
  --algorithms ALGORITHMS [ALGORITHMS ...]
                        Use this option to enable new optimization algorithms. Usage is:
                        --algorithms <algo_name1> ... The available optimization algorithms are:
                        "cle" - Cross layer equalization includes a number of methods for equalizi
                        weights and biases across layers in order to rectify imbalances that cause
                        quantization errors.
  --bias_bw BIAS_BW     Use the --bias_bw option to select the bitwidth to use when quantizing the
                        biases, either 8 (default) or 32.
  --act_bw ACT_BW       Use the --act_bw option to select the bitwidth to use when quantizing the
                        activations, either 8 (default) or 16.
  --weight_bw WEIGHT_BW
                        Use the --weight_bw option to select the bitwidth to use when quantizing t
                        weights, currently only 8 bit (default) supported.
  --float_bias_bw FLOAT_BIAS_BW
                        Use the --float_bias_bw option to select the bitwidth to use when biases a
                        in float, either 32 or 16.
  --ignore_encodings    Use only quantizer generated encodings, ignoring any user or model provide
                        encodings.
                        Note: Cannot use --ignore_encodings with --quantization_overrides
  --use_per_row_quantization
                        Use this option to enable rowwise quantization of Matmul and FullyConnecte
                        ops.
  --use_per_channel_quantization [USE_PER_CHANNEL_QUANTIZATION [USE_PER_CHANNEL_QUANTIZATION ...]]
                        Use per-channel quantization for convolution-based op weights.
                        Note: This will replace built-in model QAT encodings when used for a given
                        weight.Usage "--use_per_channel_quantization" to enable or "--
                        use_per_channel_quantization false" (default) to disable
  --use_native_input_files
                        Boolean flag to indicate how to read input files:
                        1. float (default): reads inputs as floats and quantizes if necessary base
                        on quantization parameters in the model.
```

```
                    2. native: reads inputs assuming the data type to be native to the
                    model. For ex., uint8_t.
--use_native_dtype  Note: This option is deprecated, use --use_native_input_files option in
                    future.
                    Boolean flag to indicate how to read input files:
                    1. float (default): reads inputs as floats and quantizes if necessary base
                    on quantization parameters in the model.
                    2. native: reads inputs assuming the data type to be native to the
                    model. For ex., uint8_t.
--use_native_output_files
                    Use this option to indicate the data type of the output files
                    1. float (default): output the file as floats.
                    2. native: outputs the file that is native to the model. For ex.,
                    uint8_t.
--restrict_quantization_steps ENCODING_MIN, ENCODING_MAX
                    Specifies the number of steps to use for computing quantization encodings
                    such that scale = (max - min) / number of quantization steps.
                    The option should be passed as a space separated pair of hexadecimal strin
                    minimum and maximum values. i.e. --restrict_quantization_steps "MIN MAX".
                    Please note that this is a hexadecimal string literal and not a signed
                    integer, to supply a negative value an explicit minus sign is required.
                    E.g.--restrict_quantization_steps "-0x80 0x7F" indicates an example 8 bit
                    --restrict_quantization_steps "-0x8000 0x7F7F" indicates an example 16
                    bit range. This argument is required for 16-bit Matmul operations.


Custom Op Package Options:
 --op_package_lib OP_PACKAGE_LIB, -opl OP_PACKAGE_LIB
                    Use this argument to pass an op package library for quantization. Must be
                    the form
                    <op_package_lib_path:interfaceProviderName> and be separated by a
                    comma for multiple package libs
-p PACKAGE_NAME, --package_name PACKAGE_NAME
                    A global package name to be used for each node in the Model.cpp file.
                    Defaults to Qnn header defined package name
--converter_op_package_lib CONVERTER_OP_PACKAGE_LIB, -cpl CONVERTER_OP_PACKAGE_LIB
                    Absolute path to converter op package library compiled by the OpPackage
                    generator. Must be separated by a comma for multiple package libraries.
                    Note: Libraries must follow the same order as the xml files.
                    E.g.1: --converter_op_package_lib absolute_path_to/libExample.so
                    E.g.2: -cpl absolute_path_to/libExample1.so,absolute_path_to/libExample2.s
--op_package_config OP_PACKAGE_CONFIG [OP_PACKAGE_CONFIG ...], -opc OP_PACKAGE_CONFIG [OP_PACKAG
                    Path to a Qnn Op Package XML configuration file that contains user defined
                    custom operations.
```

```
Architecture Checker Options(Experimental):
  --arch_checker        Note: This option will be soon deprecated. Use the qnn-architecture-checke

Note: Only one of: {'package_name', 'op_package_config'} can be specified
```

Basic command line usage looks like:

```
$ qnn-tensorflow-converter -i <path>/frozen_graph.pb
                    -d <network_input_name> <dims>
                    --out_node <network_output_name>
                    -o <optional_output_path>
                    --allow_unconsumed_nodes  # optional, but most likely will be need for larger
                    -p <optional_package_name> # Defaults to "qti.aisw"
```

qnn-tflite-converter

The **qnn-tflite-converter** tool converts a TFLite model to a CPP file representing the model as a series of QNN API calls. Additionally, a binary file containing static weights of the model is produced.

```
usage: qnn-tflite-converter -d INPUT_NAME INPUT_DIM [--out_node OUT_NAMES]
                            [--input_type INPUT_NAME INPUT_TYPE]
                            [--input_dtype INPUT_NAME INPUT_DTYPE] [--input_encoding INPUT_ENCODIN
                            [--input_layout INPUT_NAME INPUT_LAYOUT] [--custom_io CUSTOM_IO]
                            [--dump_relay DUMP_RELAY]
                            [--quantization_overrides QUANTIZATION_OVERRIDES] [--keep_quant_nodes]
                            [--disable_batchnorm_folding] [--keep_disconnected_nodes]
                            [--input_list INPUT_LIST] [--param_quantizer PARAM_QUANTIZER]
                            [--act_quantizer ACT_QUANTIZER]
                            [--algorithms ALGORITHMS [ALGORITHMS ...]] [--bias_bw BIAS_BW]
                            [--act_bw ACT_BW] [--weight_bw WEIGHT_BW] [--ignore_encodings]
                            [--use_per_row_quantization]
                            [--use_per_channel_quantization [USE_PER_CHANNEL_QUANTIZATION [USE_PER
                            [--use_native_input_files] [--use_native_dtype]
                            [--use_native_output_files] --input_network INPUT_NETWORK
                            [--debug [DEBUG]] [-o OUTPUT_PATH] [--copyright_file COPYRIGHT_FILE]
                            [--float_bw FLOAT_BW] [--float_bias_bw FLOAT_BIAS_BW] [--overwrite_mod
                            [--exclude_named_tensors] [--op_package_lib OP_PACKAGE_LIB]
                            [--restrict_quantization_steps ENCODING_MIN, ENCODING_MAX]
                            [--converter_op_package_lib CONVERTER_OP_PACKAGE_LIB]
                            [-p PACKAGE_NAME | --op_package_config CUSTOM_OP_CONFIG_PATHS [CUSTOM_
```

```
                        [-h] [--arch_checker]

Script to convert TFLite model into QNN

required arguments:
  -d INPUT_NAME INPUT_DIM, --input_dim INPUT_NAME INPUT_DIM
                        The names and dimensions of the network input layers specified in the form
                        [input_name comma-separated-dimensions], for example:
                            'data' 1,224,224,3 Note that the quotes should always be included in o
                        characters, spaces, etc. For multiple inputs specify multiple --input_dim
                        line like:
                            --input_dim 'data1' 1,224,224,3 --input_dim 'data2' 1,50,100,3
  --input_network INPUT_NETWORK, -i INPUT_NETWORK
                        Path to the source framework model.

optional arguments:
  --out_node OUT_NAMES, --out_name OUT_NAMES
                        Name of the graph's output Tensor Names. Multiple output names should be
                        provided separately like:
                            --out_name out_1 --out_name out_2
  --input_type INPUT_NAME INPUT_TYPE, -t INPUT_NAME INPUT_TYPE
                        Type of data expected by each input op/layer. Type for each input is
                        |default| if not specified. For example: "data" image.Note that the quotes
                        should always be included in order to handle special characters, spaces,et
                        For multiple inputs specify multiple --input_type on the command line.
                        Eg:
                            --input_type "data1" image --input_type "data2" opaque
                        These options get used by DSP runtime and following descriptions state how
                        input will be handled for each option.
                        Image:
                        Input is float between 0-255 and the input's mean is 0.0f and the input's
                        max is 255.0f. We will cast the float to uint8ts and pass the uint8ts to t
                        DSP.
                        Default:
                        Pass the input as floats to the dsp directly and the DSP will quantize it.
                        Opaque:
                        Assumes input is float because the consumer layer(i.e next layer) requires
                        it as float, therefore it won't be quantized.
                        Choices supported:
                            image
                            default
                            opaque
  --input_dtype INPUT_NAME INPUT_DTYPE
                        The names and datatype of the network input layers specified in the format
                        [input_name datatype], for example:
```

```
                        'data' 'float32'
                    Default is float32 if not specified
                    Note that the quotes should always be included in order to handlespecial
                    characters, spaces, etc.
                    For multiple inputs specify multiple --input_dtype on the command line lik
                        --input_dtype 'data1' 'float32' --input_dtype 'data2' 'float32'
--input_encoding INPUT_ENCODING [INPUT_ENCODING ...], -e INPUT_ENCODING [INPUT_ENCODING ...]
                    Usage:     --input_encoding "INPUT_NAME" INPUT_ENCODING_IN
                    [INPUT_ENCODING_OUT]
                    Input encoding of the network inputs. Default is bgr.
                    e.g.
                        --input_encoding "data" rgba
                    Quotes must wrap the input node name to handle special characters,
                    spaces, etc. To specify encodings for multiple inputs, invoke
                    --input_encoding for each one.
                    e.g.
                        --input_encoding "data1" rgba --input_encoding "data2" other
                    Optionally, an output encoding may be specified for an input node by
                    providing a second encoding. The default output encoding is bgr.
                    e.g.
                        --input_encoding "data3" rgba rgb
                    Input encoding types:
                        image color encodings: bgr,rgb, nv21, nv12, ...
                        time_series: for inputs of rnn models;
                        other: not available above or is unknown.
                    Supported encodings:
                        bgr
                        rgb
                        rgba
                        argb32
                        nv21
                        nv12
                        time_series
                        other
--input_layout INPUT_NAME INPUT_LAYOUT, -l INPUT_NAME INPUT_LAYOUT
                    Layout of each input tensor. If not specified, it will use the default
                    based on the Source Framework, shape of input and input encoding.
                    Accepted values are-
                        NCDHW, NDHWC, NCHW, NHWC, NFC, NCF, NTF, TNF, NF, NC, F, NONTRIVIAL
                    N = Batch, C = Channels, D = Depth, H = Height, W = Width, F = Feature, T
                    NDHWC/NCDHW used for 5d inputs
                    NHWC/NCHW used for 4d image-like inputs
                    NFC/NCF used for inputs to Conv1D or other 1D ops
                    NTF/TNF used for inputs with time steps like the ones used for LSTM op
                    NF used for 2D inputs, like the inputs to Dense/FullyConnected layers
```

```
                        NC used for 2D inputs with 1 for batch and other for Channels (rarely used
                        F used for 1D inputs, e.g. Bias tensor
                        NONTRIVIAL for everything elseFor multiple inputs specify multiple
                        --input_layout on the command line.
                        Eg:
                           --input_layout "data1" NCHW --input_layout "data2" NCHW
  --custom_io CUSTOM_IO
                        Use this option to specify a yaml file for custom IO.
  --dump_relay DUMP_RELAY
                        Dump Relay ASM and Params at the path provided with the argument
                        Usage: --dump_relay <path_to_dump>
  --show_unconsumed_nodes
                        Displays a list of unconsumed nodes, if there any are
                        found. Nodeswhich are unconsumed do not violate the
                        structural fidelity of thegenerated graph.
  --disable_batchnorm_folding
  --keep_disconnected_nodes
                        Disable Optimization that removes Ops not connected to the main graph.
                        This optimization uses output names provided over commandline OR
                        inputs/outputs extracted from the Source model to determine the main graph
  -o OUTPUT_PATH, --output_path OUTPUT_PATH
                        Path where the converted Output model should be saved.If not specified, th
                        converter model will be written to a file with same name as the input mode
  --copyright_file COPYRIGHT_FILE
                        Path to copyright file. If provided, the content of the file will be added
                        to the output model.
  --float_bw FLOAT_BW   Option to select the bitwidth to use when using float for parameters (weig
                        bias) and activations for all ops or a specific op (via encodings) selecte
                        through encoding; values are 32 (default) or 16.
  --overwrite_model_prefix
                        If option passed, model generator will use the output path name to use as
                        model prefix to name functions in <qnn_model_name>.cpp. (Useful for runnin
                        multiple models at once) eg: ModelName_composeGraphs. Default is to use
                        generic "QnnModel_".
  --exclude_named_tensors
                        Remove using source framework tensorNames; instead use a counter for namin
                        tensors. Note: This can potentially help to reduce the final model library
                        that will be generated(Recommended for deploying model). Default is False.
  -h, --help            show this help message and exit

Quantizer Options:
  --quantization_overrides QUANTIZATION_OVERRIDES
                        Use this option to specify a json file with parameters to use for
                        quantization. These will override any quantization data carried from
                        conversion (eg TF fake quantization) or calculated during the normal
```

```
                      quantization process. Format defined as per AIMET specification.
--keep_quant_nodes    Use this option to keep activation quantization nodes in the graph rather
                      than stripping them.
--input_list INPUT_LIST
                      Path to a file specifying the input data. This file should be a plain text
                      file, containing one or more absolute file paths per line. Each path is
                      expected to point to a binary file containing one input in the "raw" forma
                      ready to be consumed by the quantizer without any further preprocessing.
                      Multiple files per line separated by spaces indicate multiple inputs to th
                      network. See documentation for more details. Must be specified for
                      quantization. All subsequent quantization options are ignored when this is
                      not provided.
--param_quantizer PARAM_QUANTIZER
                      Optional parameter to indicate the weight/bias quantizer to use. Must be
                      followed by one of the following options: "tf": Uses the real min/max of t
                      data and specified bitwidth (default) "enhanced": Uses an algorithm useful
                      for quantizing models with long tails present in the weight distribution
                      "adjusted": Uses an adjusted min/max for computing the range, particularly
                      good for denoise models "symmetric": Ensures min and max have the same
                      absolute values about zero. Data will be stored as int#_t data such that t
                      offset is always 0.
--act_quantizer ACT_QUANTIZER
                      Optional parameter to indicate the activation quantizer to use. Must be
                      followed by one of the following options: "tf": Uses the real min/max of t
                      data and specified bitwidth (default) "enhanced": Uses an algorithm useful
                      for quantizing models with long tails present in the weight distribution
                      "adjusted": Uses an adjusted min/max for computing the range, particularly
                      good for denoise models "symmetric": Ensures min and max have the same
                      absolute values about zero. Data will be stored as int#_t data such that t
                      offset is always 0.
--algorithms ALGORITHMS [ALGORITHMS ...]
                      Use this option to enable new optimization algorithms. Usage is:
                      --algorithms <algo_name1> ... The available optimization algorithms are:
                      "cle" - Cross layer equalization includes a number of methods for equalizi
                      weights and biases across layers in order to rectify imbalances that cause
                      quantization errors.
--bias_bw BIAS_BW     Use the --bias_bw option to select the bitwidth to use when quantizing the
                      biases, either 8 (default) or 32.
--act_bw ACT_BW       Use the --act_bw option to select the bitwidth to use when quantizing the
                      activations, either 8 (default) or 16.
--weight_bw WEIGHT_BW
                      Use the --weight_bw option to select the bitwidth to use when quantizing t
                      weights, currently only 8 bit (default) supported.
--float_bias_bw FLOAT_BIAS_BW
                      Use the --float_bias_bw option to select the bitwidth to use when biases a
```

```
                        in float, either 32 or 16.
  --ignore_encodings    Use only quantizer generated encodings, ignoring any user or model provide
                        encodings.
                        Note: Cannot use --ignore_encodings with --quantization_overrides
  --use_per_row_quantization
                        Use this option to enable rowwise quantization of Matmul and FullyConnecte
                        ops.
  --use_per_channel_quantization [USE_PER_CHANNEL_QUANTIZATION [USE_PER_CHANNEL_QUANTIZATION ...]]
                        Use per-channel quantization for convolution-based op weights.
                        Note: This will replace built-in model QAT encodings when used for a given
                        weight.Usage "--use_per_channel_quantization" to enable or "--
                        use_per_channel_quantization false" (default) to disable
  --use_native_input_files
                        Boolean flag to indicate how to read input files:
                        1. float (default): reads inputs as floats and quantizes if necessary base
                        on quantization parameters in the model.
                        2. native: reads inputs assuming the data type to be native to the
                        model. For ex., uint8_t.
  --use_native_dtype    Note: This option is deprecated, use --use_native_input_files option in
                        future.
                        Boolean flag to indicate how to read input files:
                        1. float (default): reads inputs as floats and quantizes if necessary base
                        on quantization parameters in the model.
                        2. native: reads inputs assuming the data type to be native to the
                        model. For ex., uint8_t.
  --use_native_output_files
                        Use this option to indicate the data type of the output files
                        1. float (default): output the file as floats.
                        2. native: outputs the file that is native to the model. For ex.,
                        uint8_t.
  --restrict_quantization_steps ENCODING_MIN, ENCODING_MAX
                        Specifies the number of steps to use for computing quantization encodings
                        such that scale = (max - min) / number of quantization steps.
                        The option should be passed as a space separated pair of hexadecimal strin
                        minimum and maximum values. i.e. --restrict_quantization_steps "MIN MAX".
                        Please note that this is a hexadecimal string literal and not a signed
                        integer, to supply a negative value an explicit minus sign is required.
                        E.g.--restrict_quantization_steps "-0x80 0x7F" indicates an example 8 bit
                        --restrict_quantization_steps "-0x8000 0x7F7F" indicates an example 16
                        bit range.

Custom Op Package Options:
  --op_package_lib OP_PACKAGE_LIB, -opl OP_PACKAGE_LIB
                        Use this argument to pass an op package library for quantization. Must be
                        the form <op_package_lib_path:interfaceProviderName> and be separated by a
```

```
                        comma for multiple package libs
  --converter_op_package_lib CONVERTER_OP_PACKAGE_LIB, -cpl CONVERTER_OP_PACKAGE_LIB
                        Absolute path to converter op package library compiled by the OpPackage
                        generator. Must be separated by a comma for multiple package libraries.
                        Note: Libraries must follow the same order as the xml files.
                        E.g.1: --converter_op_package_lib absolute_path_to/libExample.so
                        E.g.2: -cpl absolute_path_to/libExample1.so,absolute_path_to/libExample2.s
  -p PACKAGE_NAME, --package_name PACKAGE_NAME
                        A global package name to be used for each node in the Model.cpp file.
                        Defaults to Qnn header defined package name
  --op_package_config OP_PACKAGE_CONFIG [OP_PACKAGE_CONFIG ...], -opc OP_PACKAGE_CONFIG [OP_PACKAG
                        Path to a Qnn Op Package XML configuration file that contains user defined
                        custom operations.

Architecture Checker Options(Experimental):
  --arch_checker        Note: This option will be soon deprecated. Use the qnn-architecture-checke

Note: Only one of: {'package_name', 'op_package_config'} can be specified
```

Basic command line usage looks like:

```
$ qnn-tflite-converter -i <path>/model.tflite
                       -d <network_input_name> <dims>
                       -o <optional_output_path>
                       -p <optional_package_name> # Defaults to "qti.aisw"
```

qnn-pytorch-converter

The **qnn-pytorch-converter** tool converts a PyTorch model to a CPP file representing the model as a series of QNN API calls. Additionally, a binary file containing static weights of the model is produced.

```
usage: qnn-pytorch-converter -d INPUT_NAME INPUT_DIM [--out_node OUT_NAMES]
                             [--input_type INPUT_NAME INPUT_TYPE]
                             [--input_dtype INPUT_NAME INPUT_DTYPE] [--input_encoding INPUT_ENCODING
                             [--input_layout INPUT_NAME INPUT_LAYOUT] [--custom_io CUSTOM_IO]
                             [--preserve_io [PRESERVE_IO [PRESERVE_IO ...]]]
                             [--dump_relay DUMP_RELAY] [--dry_run] [--dump_out_names]
                             [--pytorch_custom_op_lib PYTORCH_CUSTOM_OP_LIB]
                             [--quantization_overrides QUANTIZATION_OVERRIDES] [--keep_quant_nodes]
                             [--disable_batchnorm_folding] [--keep_disconnected_nodes]
```

```
                        [--input_list INPUT_LIST] [--param_quantizer PARAM_QUANTIZER]
                        [--act_quantizer ACT_QUANTIZER]
                        [--algorithms ALGORITHMS [ALGORITHMS ...]] [--bias_bw BIAS_BW]
                        [--act_bw ACT_BW] [--weight_bw WEIGHT_BW] [--ignore_encodings]
                        [--use_per_row_quantization]
                        [--use_per_channel_quantization [USE_PER_CHANNEL_QUANTIZATION [USE_PER_C
                        [--use_native_input_files] [--use_native_dtype]
                        [--use_native_output_files] --input_network INPUT_NETWORK
                        [--debug [DEBUG]] [-o OUTPUT_PATH] [--copyright_file COPYRIGHT_FILE]
                        [--float_bw FLOAT_BW] [--float_bias_bw FLOAT_BIAS_BW] [--overwrite_model
                        [--exclude_named_tensors] [--op_package_lib OP_PACKAGE_LIB]
                        [--restrict_quantization_steps ENCODING_MIN, ENCODING_MAX]
                        [--converter_op_package_lib CONVERTER_OP_PACKAGE_LIB]
                        [-p PACKAGE_NAME | --op_package_config CUSTOM_OP_CONFIG_PATHS [CUSTOM_OP
                        [-h] [--arch_checker]

Script to convert PyTorch model into QNN

required arguments:
  -d INPUT_NAME INPUT_DIM, --input_dim INPUT_NAME INPUT_DIM
                        The names and dimensions of the network input layers specified in the form
                        [input_name comma-separated-
                        dimensions], for example:
                            'data' 1,3,224,224
                        Note that the quotes should always be included in order to handle special
                        characters, spaces, etc.
                        For multiple inputs specify multiple --input_dim on the command line like:
                            --input_dim 'data1' 1,3,224,224 --input_dim 'data2' 1,50,100,3
  --input_network INPUT_NETWORK, -i INPUT_NETWORK
                        Path to the source framework model.

optional arguments:
  --out_node OUT_NAMES, --out_name OUT_NAMES
                        Name of the graph's output Tensor Names. Multiple output names should be
                        provided separately like:
                            --out_name out_1 --out_name out_2
  --input_type INPUT_NAME INPUT_TYPE, -t INPUT_NAME INPUT_TYPE
                        Type of data expected by each input op/layer. Type for each input is
                        |default| if not specified. For example: "data" image.Note that the quotes
                        should always be included in order to handle special characters, spaces, e
                        For multiple inputs specify multiple --input_type on the command line.
                        Eg:
                            --input_type "data1" image --input_type "data2" opaque
                        These options get used by DSP runtime and following descriptions state how
                        input will be handled for each option.
```

```
                        Image:
                        Input is float between 0-255 and the input's mean is 0.0f and the input's
                        max is 255.0f. We will cast the float to uint8ts and pass the uint8ts to t
                        DSP.
                        Default:
                        Pass the input as floats to the dsp directly and the DSP will quantize it.
                        Opaque:
                        Assumes
                        input is float because the consumer layer(i.e next layer) requires
                        it as float, therefore it won't be quantized.
                        Choices supported:
                            image
                            default
                            opaque
--input_dtype INPUT_NAME INPUT_DTYPE
                        The names and datatype of the network input layers specified in the format
                        [input_name datatype], for example:
                            'data' 'float32'
                        Default is float32 if not specified
                        Note that the quotes should always be included in order to handlespecial
                        characters, spaces, etc.
                        For multiple inputs specify multiple --input_dtype on the command line lik
                            --input_dtype 'data1' 'float32' --input_dtype 'data2' 'float32'
--input_encoding INPUT_ENCODING [INPUT_ENCODING ...], -e INPUT_ENCODING [INPUT_ENCODING ...]
                        Usage:     --input_encoding "INPUT_NAME" INPUT_ENCODING_IN
                        [INPUT_ENCODING_OUT]
                        Input encoding of the network inputs. Default is bgr.
                        e.g.
                            --input_encoding "data" rgba
                        Quotes must wrap the input node name to handle special characters,
                        spaces, etc. To specify encodings for multiple inputs, invoke
                        --input_encoding for each one.
                        e.g.
                            --input_encoding "data1" rgba --input_encoding "data2" other
                        Optionally, an output encoding may be specified for an input node by
                        providing a second encoding. The default output encoding is bgr.
                        e.g.
                            --input_encoding "data3" rgba rgb
                        Input encoding types:
                            image color encodings: bgr,rgb, nv21, nv12, ...
                            time_series: for inputs of rnn models;
                            other: not available above or is unknown.
                        Supported encodings:
                            bgr
                            rgb
```

```
                        rgba
                        argb32
                        nv21
                        nv12
                        time_series
                        other
--input_layout INPUT_NAME INPUT_LAYOUT, -l INPUT_NAME INPUT_LAYOUT
                    Layout of each input tensor. If not specified, it will use the default
                    based on the Source Framework, shape of input and input encoding.
                    Accepted values are-
                        NCDHW, NDHWC, NCHW, NHWC, NFC, NCF, NTF, TNF, NF, NC, F, NONTRIVIAL
                    N = Batch, C = Channels, D = Depth, H = Height, W = Width, F = Feature, T
                    NDHWC/NCDHW used for 5d inputs
                    NHWC/NCHW used for 4d image-like inputs
                    NFC/NCF used for inputs to Conv1D or other 1D ops
                    NTF/TNF used for inputs with time steps like the ones used for LSTM op
                    NF used for 2D inputs, like the inputs to Dense/FullyConnected layers
                    NC used for 2D inputs with 1 for batch and other for Channels (rarely used
                    F used for 1D inputs, e.g. Bias tensor
                    NONTRIVIAL for everything elseFor multiple inputs specify multiple
                    --input_layout on the command line.
                    Eg:
                       --input_layout "data1" NCHW --input_layout "data2" NCHW
--custom_io CUSTOM_IO
                    Use this option to specify a yaml file for custom IO.
--preserve_io [PRESERVE_IO [PRESERVE_IO ...]]
                    Use this option to preserve IO layout and datatype. The different ways of
                    using this option are as follows:
                        --preserve_io layout <space separated list of names of inputs and
                    outputs of the graph>
                        --preserve_io datatype <space separated list of names of inputs and
                    outputs of the graph>
                    In this case, user should also specify the string - layout or datatype in
                    the command to indicate that converter needs to
                    preserve the layout or datatype. e.g.
                    --preserve_io layout input1 input2 output1
                    --preserve_io datatype input1 input2 output1
                    Optionally, the user may choose to preserve the layout and/or datatype for
                    all the inputs and outputs of the graph.
                    This can be done in the following two ways:
                        --preserve_io layout
                        --preserve_io datatype
                    Additionally, the user may choose to preserve both layout and datatypes fo
                    all IO tensors by just passing the option as follows:
                        --preserve_io
```

```
                        Note: Only one of the above usages are allowed at a time.
                        Note: --custom_io gets higher precedence than --preserve_io.
  --dump_relay DUMP_RELAY
                        Dump Relay ASM and Params at the path provided with the argument
                        Usage: --dump_relay <path_to_dump>
  --dry_run             Evaluates the model without actually converting any ops, and
                         returns unsupported ops if any.
  --dump_out_names      Dump output names mapped from QNN CPP stored names to converter used
                        names and save to file 'model_output_names.json'.
  --pytorch_custom_op_lib PYTORCH_CUSTOM_OP_LIB, -pcl PYTORCH_CUSTOM_OP_LIB
                        Absolute path to the PyTorch library containing the custom op definition.
                        Multiple custom op libraries must be comma-separated.
                        For PyTorch custom op details, refer to:
                             https://pytorch.org/tutorials/advanced/torch_script_custom_ops.html
                        For custom C++ extension details, refer to:
                             https://pytorch.org/tutorials/advanced/cpp_extension.html
                        Eg. 1: --pytorch_custom_op_lib absolute_path_to/Example.so
                        Eg. 2: -pcl absolute_path_to/Example1.so,absolute_path_to/Example2.so
  --disable_batchnorm_folding
  --keep_disconnected_nodes
                        Disable Optimization that removes Ops not connected to the main graph.
                        This optimization uses output names provided over commandline OR
                        inputs/outputs extracted from the Source model to determine the main graph
  --debug [DEBUG]       Run the converter in debug mode.
  -o OUTPUT_PATH, --output_path OUTPUT_PATH
                        Path where the converted Output model should be saved.If not specified, th
                        converter model will be written to a file with same name as the input mode
  --copyright_file COPYRIGHT_FILE
                        Path to copyright file. If provided, the content of the file will be added
                        to the output model.
  --float_bw FLOAT_BW   Option to select the bitwidth to use when using float for parameters (weig
                        bias) and activations for all ops or a specific op (via encodings) selecte
                        through encoding; values are 32 (default) or 16.
  --overwrite_model_prefix
                        If option passed, model generator will use the output path name to use as
                        model prefix to name functions in <qnn_model_name>.cpp. (Useful for runnin
                        multiple models at once) eg: ModelName_composeGraphs. Default is to use
                        generic "QnnModel_".
  --exclude_named_tensors
                        Remove using source framework tensorNames; instead use a counter for namin
                        tensors. Note: This can potentially help to reduce the final model library
                        that will be generated(Recommended for deploying model). Default is False.
  -h, --help            show this help message and exit

Quantizer Options:
```

```
--quantization_overrides QUANTIZATION_OVERRIDES
                      Use this option to specify a json file with parameters to use for
                      quantization. These will override any quantization data carried from
                      conversion (eg TF fake quantization) or calculated during the normal
                      quantization process. Format defined as per AIMET specification.
--keep_quant_nodes    Use this option to keep activation quantization nodes in the graph rather
                      than stripping them.
--input_list INPUT_LIST
                      Path to a file specifying the input data. This file should be a plain text
                      file, containing one or more absolute file paths per line. Each path is
                      expected to point to a binary file containing one input in the "raw" forma
                      ready to be consumed by the quantizer without any further preprocessing.
                      Multiple files per line separated by spaces indicate multiple inputs to th
                      network. See documentation for more details. Must be specified for
                      quantization. All subsequent quantization options are ignored when this is
                      not provided.
--param_quantizer PARAM_QUANTIZER
                      Optional parameter to indicate the weight/bias quantizer to use. Must be
                      followed by one of the following options: "tf": Uses the real min/max of t
                      data and specified bitwidth (default) "enhanced": Uses an algorithm useful
                      for quantizing models with long tails present in the weight distribution
                      "adjusted": Uses an adjusted min/max for computing the range, particularly
                      good for denoise models "symmetric": Ensures min and max have the same
                      absolute values about zero. Data will be stored as int#_t data such that t
                      offset is always 0.
--act_quantizer ACT_QUANTIZER
                      Optional parameter to indicate the activation quantizer to use. Must be
                      followed by one of the following options: "tf": Uses the real min/max of t
                      data and specified bitwidth (default) "enhanced": Uses an algorithm useful
                      for quantizing models with long tails present in the weight distribution
                      "adjusted": Uses an adjusted min/max for computing the range, particularly
                      good for denoise models "symmetric": Ensures min and max have the same
                      absolute values about zero. Data will be stored as int#_t data such that t
                      offset is always 0.
--algorithms ALGORITHMS [ALGORITHMS ...]
                      Use this option to enable new optimization algorithms. Usage is:
                      --algorithms <algo_name1> ... The available optimization algorithms are:
                      "cle" - Cross layer equalization includes a number of methods for equalizi
                      weights and biases across layers in order to rectify imbalances that cause
                      quantization errors.
--bias_bw BIAS_BW     Use the --bias_bw option to select the bitwidth to use when quantizing the
                      biases, either 8 (default) or 32.
--act_bw ACT_BW       Use the --act_bw option to select the bitwidth to use when quantizing the
                      activations, either 8 (default) or 16.
--weight_bw WEIGHT_BW
```

```
                      Use the --weight_bw option to select the bitwidth to use when quantizing t
                      weights, currently only 8 bit (default) supported.
--float_bias_bw FLOAT_BIAS_BW
                      Use the --float_bias_bw option to select the bitwidth to use when biases a
                      in float, either 32 or 16.
--ignore_encodings    Use only quantizer generated encodings, ignoring any user or model provide
                      encodings.
                      Note: Cannot use --ignore_encodings with --quantization_overrides
--use_per_row_quantization
                      Use this option to enable rowwise quantization of Matmul and FullyConnecte
                      ops.
--use_per_channel_quantization [USE_PER_CHANNEL_QUANTIZATION [USE_PER_CHANNEL_QUANTIZATION ...]]
                      Use per-channel quantization for convolution-based op weights.
                      Note: This will replace built-in model QAT encodings when used for a given
                      weight.Usage "--use_per_channel_quantization" to enable or "--
                      use_per_channel_quantization false" (default) to disable
--use_native_input_files
                      Boolean flag to indicate how to read input files:
                      1. float (default): reads inputs as floats and quantizes if necessary base
                      on quantization parameters in the model.
                      2. native: reads inputs assuming the data type to be native to the
                      model. For ex., uint8_t.
--use_native_dtype    Note: This option is deprecated, use --use_native_input_files option in
                      future.
                      Boolean flag to indicate how to read input files:
                      1. float (default): reads inputs as floats and quantizes if necessary base
                      on quantization parameters in the model.
                      2. native: reads inputs assuming the data type to be native to the
                      model. For ex., uint8_t.
--use_native_output_files
                      Use this option to indicate the data type of the output files
                      1. float (default): output the file as floats.
                      2. native: outputs the file that is native to the model. For ex.,
                      uint8_t.
--restrict_quantization_steps ENCODING_MIN, ENCODING_MAX
                      Specifies the number of steps to use for computing quantization encodings
                      such that scale = (max - min) / number of quantization steps.
                      The option should be passed as a space separated pair of hexadecimal strin
                      minimum and maximum values. i.e. --restrict_quantization_steps "MIN MAX".
                      Please note that this is a hexadecimal string literal and not a signed
                      integer, to supply a negative value an explicit minus sign is required.
                      E.g.--restrict_quantization_steps "-0x80 0x7F" indicates an example 8 bit
                      --restrict_quantization_steps "-0x8000 0x7F7F" indicates an example 16
                      bit range.
```

```
Custom Op Package Options:
  --op_package_lib OP_PACKAGE_LIB, -opl OP_PACKAGE_LIB
                        Use this argument to pass an op package library for quantization. Must be
                        the form <op_package_lib_path:interfaceProviderName> and be separated by a
                        comma for multiple package libs
  --converter_op_package_lib CONVERTER_OP_PACKAGE_LIB, -cpl CONVERTER_OP_PACKAGE_LIB
                        Absolute path to converter op package library compiled by the OpPackage
                        generator. Must be separated by a comma for multiple package libraries.
                        Note: Libraries must follow the same order as the xml files.
                        E.g.1: --converter_op_package_lib absolute_path_to/libExample.so
                        E.g.2: -cpl absolute_path_to/libExample1.so,absolute_path_to/libExample2.s
  -p PACKAGE_NAME, --package_name PACKAGE_NAME
                        A global package name to be used for each node in the Model.cpp file.
                        Defaults to Qnn header defined package name
  --op_package_config CUSTOM_OP_CONFIG_PATHS [CUSTOM_OP_CONFIG_PATHS ...], -opc CUSTOM_OP_CONFIG_P
                        Path to a Qnn Op Package XML configuration file that contains user defined
                        custom operations.

Architecture Checker Options(Experimental):
  --arch_checker        Note: This option will be soon deprecated. Use the qnn-architecture-checke
```

Note: Only one of: {'package_name', 'op_package_config'} can be specified

Basic command line usage looks like:

```
$ qnn-pytorch-converter -i <path>/model.pt
                        -d <network_input_name> <dims>
                        -o <optional_output_path>
                        -p <optional_package_name> # Defaults to "qti.aisw"
```

qnn-onnx-converter

The **qnn-onnx-converter** tool converts a model from the ONNX framework to a CPP file representing the model as a series of QNN API calls. Additionally, a binary file containing static weights of the model is produced.

```
usage: qnn-onnx-converter [--out_node OUT_NAMES] [--input_type INPUT_NAME INPUT_TYPE]
           [--input_dtype INPUT_NAME INPUT_DTYPE] [--input_encoding INPUT_ENCODING [INPUT_ENCODIN
```

```
                [--input_layout INPUT_NAME INPUT_LAYOUT] [--custom_io CUSTOM_IO]
                [--dry_run [DRY_RUN]] [-d INPUT_NAME INPUT_DIM] [-n] [-b BATCH]
                [-s SYMBOL_NAME VALUE] [--preserve_io PRESERVE_IO]
                [--dump_custom_io_config_template DUMP_CUSTOM_IO_CONFIG_TEMPLATE]
                [--quantization_overrides QUANTIZATION_OVERRIDES] [--keep_quant_nodes]
                [--disable_batchnorm_folding] [--keep_disconnected_nodes]
                [--input_list INPUT_LIST] [--param_quantizer PARAM_QUANTIZER]
                [--act_quantizer ACT_QUANTIZER] [--algorithms ALGORITHMS [ALGORITHMS ...]]
                [--bias_bw BIAS_BW] [--act_bw ACT_BW] [--weight_bw WEIGHT_BW]
                [--ignore_encodings] [--use_per_row_quantization]
                [--use_per_channel_quantization [USE_PER_CHANNEL_QUANTIZATION [USE_PER_CHANNEL_QUANTIZ
                [--use_native_input_files] [--use_native_dtype]
                [--use_native_output_files] --input_network INPUT_NETWORK
                [--debug [DEBUG]] [-o OUTPUT_PATH] [--copyright_file COPYRIGHT_FILE]
                [--float_bw FLOAT_BW] [--float_bias_bw FLOAT_BIAS_BW] [--overwrite_model_prefix] [--ex
                [--restrict_quantization_steps ENCODING_MIN, ENCODING_MAX]
                [--op_package_lib OP_PACKAGE_LIB]
                [--converter_op_package_lib CONVERTER_OP_PACKAGE_LIB]
                [-p PACKAGE_NAME | --op_package_config CUSTOM_OP_CONFIG_PATHS [CUSTOM_OP_CONFIG_PATHS
                [-h] [--arch_checker]

Script to convert ONNX model into QNN

required arguments:
  --input_network INPUT_NETWORK, -i INPUT_NETWORK
                        Path to the source framework model.

optional arguments:
  --out_node OUT_NAMES, --out_name OUT_NAMES
                        Name of the graph's output tensor names. Multiple output
                        nodes should be provided separately like:
                            --out_name out_1 --out_name out_2
  --input_type INPUT_NAME INPUT_TYPE, -t INPUT_NAME INPUT_TYPE
                        Type of data expected by each input op/layer. Type for
                        each input is |default| if not specified. For example:
                        "data" image.Note that the quotes should always be
                        included in order to handle special characters,
                        spaces,etc. For multiple inputs specify multiple
                        --input_type on the command line. Eg:
                            --input_type "data1" image --input_type "data2" opaque
                        These options get used by DSP runtime and following
                        descriptions state how input will be handled for each
                        option.
                        Image:
                        Input is float between 0-255 and the input's mean is 0.0f and the input's
```

```
                        max is 255.0f. We will cast the float to uint8ts and pass the uint8ts to
                        the DSP.
                        Default:
                        Pass the input as floats to the dsp
                        directly and the DSP will quantize it.
                        Opaque:
                        Assumes input is float because the consumer layer(i.e next
                        layer) requires it as float, therefore it won't be
                        quantized.
                        Choices supported:
                            image
                            default
                            opaque
--input_dtype INPUT_NAME INPUT_DTYPE
                        The names and datatype of the network input layers
                        specified in the format [input_name datatype], for
                        example:
                            'data' 'float32'.
                        Default is float32 if not specified.
                        Note that the quotes should always be included in order to handle special
                        characters, spaces, etc.
                        For multiple inputs specify multiple --input_dtype on the command line lik
                            --input_dtype 'data1' 'float32' --input_dtype 'data2' 'float32'
--input_encoding INPUT_ENCODING [INPUT_ENCODING ...], -e INPUT_ENCODING [INPUT_ENCODING ...]
                        Usage:     --input_encoding "INPUT_NAME" INPUT_ENCODING_IN
                        [INPUT_ENCODING_OUT]
                        e.g.
                            --input_encoding "data" rgba
                        Quotes must wrap the input node name to handle special characters,
                        spaces, etc. To specify encodings for multiple inputs, invoke
                        --input_encoding for each one.
                        e.g.
                            --input_encoding "data1" rgba --input_encoding "data2" other
                        Optionally, an output encoding may be specified for an input node by
                        providing a second encoding. The default output encoding is bgr.
                        e.g.
                            --input_encoding "data3" rgba rgb
                        Input encoding types:
                            image color encodings: bgr,rgb, nv21, nv12, ...
                            time_series: for inputs of rnn models;
                            other: not available above or is unknown.
                        Supported encodings:
                            bgr
                            rgb
                            rgba
```

```
                        argb32
                        nv21
                        nv12
                        time_series
                        other
--input_layout INPUT_NAME INPUT_LAYOUT, -l INPUT_NAME INPUT_LAYOUT
                    Layout of each input tensor. If not specified, it will use the default
                    based on the Source Framework, shape of input and input encoding.
                    Accepted values are-
                        NCDHW, NDHWC, NCHW, NHWC, NFC, NCF, NTF, TNF, NF, NC, F, NONTRIVIAL
                    N = Batch, C = Channels, D = Depth, H = Height, W = Width, F = Feature, T
                    NDHWC/NCDHW used for 5d inputs
                    NHWC/NCHW used for 4d image-like inputs
                    NFC/NCF used for inputs to Conv1D or other 1D ops
                    NTF/TNF used for inputs with time steps like the ones used for LSTM op
                    NF used for 2D inputs, like the inputs to Dense/FullyConnected layers
                    NC used for 2D inputs with 1 for batch and other for Channels (rarely used
                    F used for 1D inputs, e.g. Bias tensor
                    NONTRIVIAL for everything elseFor multiple inputs specify multiple
                    --input_layout on the command line.
                    Eg:
                        --input_layout "data1" NCHW --input_layout "data2" NCHW
                    Note: This flag does not set the layout of the input tensor in the convert
                        Please use --custom_io for that.
--custom_io CUSTOM_IO
                    Use this option to specify a yaml file for custom IO.
--preserve_io PRESERVE_IO
                    Use this option to preserve IO layout and datatype. The different ways of
                    this option are as follows:
                        --preserve_io layout <space separated list of names of inputs and outp
                        --preserve_io datatype <space separated list of names of inputs and ou
                    In this case, user should also specify the string - layout or datatype in
                    to indicate that converter needs to preserve the layout or datatype. e.g.
                        --preserve_io layout input1 input2 output1
                        --preserve_io datatype input1 input2 output1
                    Optionally, the user may choose to preserve the layout and/or datatype for
                    the inputs and outputs of the graph. This can be done in the following two
                        --preserve_io layout
                        --preserve_io datatype
                    Additionally, the user may choose to preserve both layout and datatypes fo
                    IO tensors by just passing the option as follows:
                        --preserve_io
                    Note: Only one of the above usages are allowed at a time.
                    Note: --custom_io gets higher precedence than --preserve_io.
--dry_run [DRY_RUN]   Evaluates the model without actually converting any ops, and returns
```

```
                      unsupported ops/attributes as well as unused inputs and/or outputs if any.
                      Leave empty or specify "info" to see dry run as a table, or specify "debug
                      to show more detailed messages only"
-d INPUT_NAME INPUT_DIM, --input_dim INPUT_NAME INPUT_DIM
                      The name and dimension of all the input buffers to the network specified i
                      the format [input_name comma-separated-dimensions],
                      for example: 'data' 1,224,224,3.
                      Note that the quotes should always be included in order to handle special
                      characters, spaces, etc.
                      NOTE: This feature works only with Onnx 1.6.0 and above
-n, --no_simplification
                      Do not attempt to simplify the model automatically. This may prevent some
                      models from properly converting
-b BATCH, --batch BATCH
                      The batch dimension override. This will take the first dimension of all
                      inputs and treat it as a batch dim, overriding it with the value provided
                      here. For example:
                      --batch 6
                      will result in a shape change from [1,3,224,224] to [6,3,224,224].
                      If there are inputs without batch dim this should not be used and each inp
                      should be overridden independently using -d option for input dimension
                      overrides.
-s SYMBOL_NAME VALUE, --define_symbol SYMBOL_NAME VALUE
                      This option allows overriding specific input dimension symbols. For instan
                      you might see input shapes specified with variables such as :
                      data: [1,3,height,width]
                      To override these simply pass the option as:
                      --define_symbol height 224 --define_symbol width 448
                      which results in dimensions that look like:
                      data: [1,3,224,448]
--dump_custom_io_config_template
                      Dumps the yaml template for Custom I/O configuration. This file can be edi
                      as per the custom requirements and passed using the option --custom_ioUse
                      this option to specify a yaml file to which the custom IO config template
                      dumped.
--disable_batchnorm_folding
--keep_disconnected_nodes
                      Disable Optimization that removes Ops not connected to the main graph.
                      This optimization uses output names provided over commandline OR
                      inputs/outputs extracted from the Source model to determine the main graph
--debug [DEBUG]       Run the converter in debug mode.
-o OUTPUT_PATH, --output_path OUTPUT_PATH
                      Path where the converted Output model should be saved.If not specified, th
                      converter model will be written to a file with same name as the input mode
--copyright_file COPYRIGHT_FILE
```

```
                        Path to copyright file. If provided, the content of the file will be added
                        to the output model.
  --float_bw FLOAT_BW   Option to select the bitwidth to use when using float for parameters (weig
                        bias) and activations for all ops or a specific op (via encodings) selecte
                        through encoding; values are 32 (default) or 16.
  --overwrite_model_prefix
                        If option passed, model generator will use the output path name to use as
                        model prefix to name functions in <qnn_model_name>.cpp. (Useful for runnin
                        multiple models at once) eg: ModelName_composeGraphs. Default is to use
                        generic "QnnModel_".
  --exclude_named_tensors
                        Remove using source framework tensorNames; instead use a counter for namin
                        tensors. Note: This can potentially help to reduce  the final model librar
                        that will be generated(Recommended for deploying model). Default is False.
  -h, --help            show this help message and exit

Quantizer Options:
  --quantization_overrides QUANTIZATION_OVERRIDES
                        Use this option to specify a json file with parameters to use for
                        quantization. These will override any quantization data carried from
                        conversion (eg TF fake quantization) or calculated during the normal
                        quantization process. Format defined as per AIMET specification.

  --keep_quant_nodes    Use this option to keep activation quantization nodes in the graph rather
                        than stripping them.
  --input_list INPUT_LIST
                        Path to a file specifying the input data. This file should be a plain text
                        file, containing one or more absolute file paths per line. Each path is
                        expected to point to a binary file containing one input in the "raw" forma
                        ready to be consumed by the quantizer without any further preprocessing.
                        Multiple files per line separated by spaces indicate multiple inputs to th
                        network. See documentation for more details. Must be specified for
                        quantization. All subsequent quantization options are ignored when this is
                        not provided.
  --param_quantizer PARAM_QUANTIZER
                        Optional parameter to indicate the weight/bias quantizer to use. Must be
                        followed by one of the following options: "tf": Uses the real min/max of t
                        data and specified bitwidth (default) "enhanced": Uses an algorithm useful
                        for quantizing models with long tails present in the weight distribution
                        "adjusted": Uses an adjusted min/max for computing the range, particularly
                        good for denoise models "symmetric": Ensures min and max have the same
                        absolute values about zero. Data will be stored as int#_t data such that t
                        offset is always 0.
  --act_quantizer ACT_QUANTIZER
                        Optional parameter to indicate the activation quantizer to use. Must be
```

```
                      followed by one of the following options: "tf": Uses the real min/max of t
                      data and specified bitwidth (default) "enhanced": Uses an algorithm useful
                      for quantizing models with long tails present in the weight distribution
                      "adjusted": Uses an adjusted min/max for computing the range, particularly
                      good for denoise models "symmetric": Ensures min and max have the same
                      absolute values about zero. Data will be stored as int#_t data such that t
                      offset is always 0.
--algorithms ALGORITHMS [ALGORITHMS ...]
                      Use this option to enable new optimization algorithms. Usage is:
                      --algorithms <algo_name1> ... The available optimization algorithms are:
                      "cle" - Cross layer equalization includes a number of methods for equalizi
                      weights and biases across layers in order to rectify imbalances that cause
                      quantization errors.

--bias_bw BIAS_BW     Use the --bias_bw option to select the bitwidth to use when quantizing the
                      biases, either 8 (default) or 32.
--act_bw ACT_BW       Use the --act_bw option to select the bitwidth to use when quantizing the
                      activations, either 8 (default) or 16.
--weight_bw WEIGHT_BW
                      Use the --weight_bw option to select the bitwidth to use when quantizing t
                      weights, currently only 8 bit (default) supported.
--float_bias_bw FLOAT_BIAS_BW
                      Use the --float_bias_bw option to select the bitwidth to use when biases a
                      in float, either 32 or 16.
--ignore_encodings    Use only quantizer generated encodings, ignoring any user or model provide
                      encodings.
                      Note: Cannot use --ignore_encodings with --quantization_overrides
--use_per_row_quantization
                      Use this option to enable rowwise quantization of Matmul and FullyConnecte
                      ops.
--use_per_channel_quantization [USE_PER_CHANNEL_QUANTIZATION [USE_PER_CHANNEL_QUANTIZATION ...]]
                      Use per-channel quantization for convolution-based op weights.
                      Note: This will replace built-in model QAT encodings when used for a given
                      weight.Usage "--use_per_channel_quantization" to enable or "--
                      use_per_channel_quantization false" (default) to disable
--use_native_input_files
                      Boolean flag to indicate how to read input files:
                      1. float (default): reads inputs as floats and quantizes if necessary base
                      on quantization parameters in the model.
                      2. native: reads inputs assuming the data type to be native to the
                      model. For ex., uint8_t.
--use_native_dtype    Note: This option is deprecated, use --use_native_input_files option in
                      future.
                      Boolean flag to indicate how to read input files:
                      1. float (default): reads inputs as floats and quantizes if necessary base
```

```
                        on quantization parameters in the model.
                        2. native: reads inputs assuming the data type to be native to the
                        model. For ex., uint8_t.
  --use_native_output_files
                        Use this option to indicate the data type of the output files
                        1. float (default): output the file as floats.
                        2. native:          outputs the file that is native to the model. For ex.,
                        uint8_t.
  --restrict_quantization_steps ENCODING_MIN, ENCODING_MAX
                        Specifies the number of steps to use for computing quantization encodings
                        such that scale = (max - min) / number of quantization steps.
                        The option should be passed as a space separated pair of hexadecimal strin
                        minimum and maximum values. i.e. --restrict_quantization_steps "MIN MAX".
                        Please note that this is a hexadecimal string literal and not a signed
                        integer, to supply a negative value an explicit minus sign is required.
                        E.g.--restrict_quantization_steps "-0x80 0x7F" indicates an example 8 bit
                        --restrict_quantization_steps "-0x8000 0x7F7F" indicates an example 16
                        bit range. This argument is required for 16-bit Matmul operations.

Custom Op Package Options:
  --op_package_lib OP_PACKAGE_LIB, -opl OP_PACKAGE_LIB
                        Use this argument to pass an op package library for quantization. Must be
                        the form <op_package_lib_path:interfaceProviderName> and be separated by a
                        comma for multiple package libs
  --converter_op_package_lib CONVERTER_OP_PACKAGE_LIB, -cpl CONVERTER_OP_PACKAGE_LIB
                        Absolute path to converter op package library compiled by the OpPackage
                        generator. Must be separated by a comma for multiple package libraries.
                        Note: Libraries must follow the same order as the xml files.
                        E.g.1: --converter_op_package_lib absolute_path_to/libExample.so
                        E.g.2: -cpl absolute_path_to/libExample1.so,absolute_path_to/libExample2.s
  -p PACKAGE_NAME, --package_name PACKAGE_NAME
                        A global package name to be used for each node in the Model.cpp file.
                        Defaults to Qnn header defined package name
  --op_package_config OP_PACKAGE_CONFIG [OP_PACKAGE_CONFIG ...], -opc OP_PACKAGE_CONFIG [OP_PACKAG
                        Path to a Qnn Op Package XML configuration file that contains user defined
                        custom operations.

Architecture Checker Options(Experimental):
  --arch_checker        Note: This option will be soon deprecated. Use the qnn-architecture-checke

Note: Only one of: {'package_name', 'op_package_config'} can be specified
```

qairt-converter

The **qairt-converter** tool converts a model from the one of Onnx/TensorFlow/TFLite/PyTorch framework to a DLC file representing the QNN graph format that can enable inference on Qualcomm AI IP/HW. The converter auto detects the framework based on the source model extension.

```
Basic command line usage looks like:

usage: qairt-converter [--desired_input_shape INPUT_NAME INPUT_DIM] [--out_tensor_node OUT_NAMES]
                       [--source_model_input_datatype INPUT_NAME INPUT_DTYPE]
                       [--source_model_input_layout INPUT_NAME INPUT_LAYOUT]
                       [--desired_input_color_encoding   [  ...]]
                       [--dump_io_config_template DUMP_IO_CONFIG_TEMPLATE] [--io_config IO_CONFIG]
                       [--dry_run [DRY_RUN]] [--quantization_overrides QUANTIZATION_OVERRIDES]
                       [--onnx_no_simplification] [--onnx_batch BATCH]
                       [--onnx_define_symbol SYMBOL_NAME VALUE] [--tf_no_optimization]
                       [--tf_show_unconsumed_nodes] [--tf_saved_model_tag SAVED_MODEL_TAG]
                       [--tf_saved_model_signature_key SAVED_MODEL_SIGNATURE_KEY]
                       [--tf_validate_models] [--tflite_signature_name SIGNATURE_NAME]
                       --input_network INPUT_NETWORK [-h] [--debug [DEBUG]]
                       [--output_path OUTPUT_PATH] [--copyright_file COPYRIGHT_FILE]
                       [--float_bitwidth FLOAT_BITWIDTH] [--float_bias_bitwidth FLOAT_BIAS_BITWIDT
                       [--model_version MODEL_VERSION] [--converter_op_package_lib CONVERTER_OP_PA
                       [--package_name PACKAGE_NAME | --op_package_config CUSTOM_OP_CONFIG_PATHS [

required arguments:
  --input_network INPUT_NETWORK, -i INPUT_NETWORK
                        Path to the source framework model.

optional arguments:
  --desired_input_shape INPUT_NAME INPUT_DIM, -d INPUT_NAME INPUT_DIM
                        The name and dimension of all the input buffers to the network specified i
                        the format [input_name comma-separated-dimensions],
                        for example: 'data' 1,224,224,3.
                        Note that the quotes should always be included in order to handle special
                        characters, spaces, etc.
                        NOTE: Required for TensorFlow and PyTorch. Optional for Onnx and Tflite
                        In case of Onnx, this feature works only with Onnx 1.6.0 and above
  --out_tensor_node OUT_NAMES, --out_tensor_name OUT_NAMES
                        Name of the graph's output Tensor Names. Multiple output names should be
                        provided separately like:
                            --out_name out_1 --out_name out_2
                        NOTE: Required for TensorFlow. Optional for Onnx, Tflite and PyTorch
  --source_model_input_datatype INPUT_NAME INPUT_DTYPE
```

```
                        The names and datatype of the network input layers specified in the format
                        [input_name datatype], for example:
                            'data' 'float32'
                        Default is float32 if not specified
                        Note that the quotes should always be included in order to handlespecial
                        characters, spaces, etc.
                        For multiple inputs specify multiple --input_dtype on the command line lik
                            --input_dtype 'data1' 'float32' --input_dtype 'data2' 'float32'
--source_model_input_layout INPUT_NAME INPUT_LAYOUT, -l INPUT_NAME INPUT_LAYOUT
                        Layout of each input tensor. If not specified, it will use the default
                        based on the Source Framework, shape of input and input encoding.
                        Accepted values are-
                            NCDHW, NDHWC, NCHW, NHWC, NFC, NCF, NTF, TNF, NF, NC, F, NONTRIVIAL
                        N = Batch, C = Channels, D = Depth, H = Height, W = Width, F = Feature, T
                        Time
                        NDHWC/NCDHW used for 5d inputs
                        NHWC/NCHW used for 4d image-like inputs
                        NFC/NCF used for inputs to Conv1D or other 1D ops
                        NTF/TNF used for inputs with time steps like the ones used for LSTM op
                        NF used for 2D inputs, like the inputs to Dense/FullyConnected layers
                        NC used for 2D inputs with 1 for batch and other for Channels (rarely used
                        F used for 1D inputs, e.g. Bias tensor
                        NONTRIVIAL for everything elseFor multiple inputs specify multiple
                        --input_layout on the command line.
                        Eg:
                            --input_layout "data1" NCHW --input_layout "data2" NCHW
--desired_input_color_encoding   [  ...], -e   [  ...]
                        Usage:     --input_color_encoding "INPUT_NAME" INPUT_ENCODING_IN
                        [INPUT_ENCODING_OUT]
                        Input encoding of the network inputs. Default is bgr.
                        e.g.
                            --input_color_encoding "data" rgba
                        Quotes must wrap the input node name to handle special characters,
                        spaces, etc. To specify encodings for multiple inputs, invoke
                        --input_color_encoding for each one.
                        e.g.
                            --input_color_encoding "data1" rgba --input_color_encoding "data2" oth
                        Optionally, an output encoding may be specified for an input node by
                        providing a second encoding. The default output encoding is bgr.
                        e.g.
                            --input_color_encoding "data3" rgba rgb
                        Input encoding types:
                             image color encodings: bgr,rgb, nv21, nv12, ...
                            time_series: for inputs of rnn models;
                            other: not available above or is unknown.
```

```
                        Supported encodings:
                           bgr
                           rgb
                           rgba
                           argb32
                           nv21
                           nv12
  --dump_io_config_template DUMP_IO_CONFIG_TEMPLATE
                        Dumps the yaml template for I/O configuration. This file can be edited as
                        per the custom requirements and passed using the option --io_configUse thi
                        option to specify a yaml file to which the IO config template is dumped.
  --io_config IO_CONFIG
                        Use this option to specify a yaml file for input and output options.
  --dry_run [DRY_RUN]   Evaluates the model without actually converting any ops, and returns
                        unsupported ops/attributes as well as unused inputs and/or outputs if any.
  -h, --help            show this help message and exit
  --debug [DEBUG]       Run the converter in debug mode.
  --output_path OUTPUT_PATH, -o OUTPUT_PATH
                        Path where the converted Output model should be saved.If not specified, th
                        converter model will be written to a file with same name as the input mode
  --copyright_file COPYRIGHT_FILE
                        Path to copyright file. If provided, the content of the file will be added
                        to the output model.
  --float_bitwidth FLOAT_BITWIDTH
                        Use the --float_bitwidth option to convert the graph to the specified floa
                        bitwidth, either 32 (default) or 16.
  --float_bias_bitwidth FLOAT_BIAS_BITWIDTH
                        Use the --float_bias_bitwidth option to select the bitwidth to use for flo
                        bias tensor
  --model_version MODEL_VERSION
                        User-defined ASCII string to identify the model, only first 64 bytes will
                        stored

Custom Op Package Options:
  --converter_op_package_lib CONVERTER_OP_PACKAGE_LIB, -cpl CONVERTER_OP_PACKAGE_LIB
                        Absolute path to converter op package library compiled by the OpPackage
                        generator. Must be separated by a comma for multiple package libraries.
                        Note: Order of converter op package libraries must follow the order of xml
                        Ex1: --converter_op_package_lib absolute_path_to/libExample.so
                        Ex2: -cpl absolute_path_to/libExample1.so,absolute_path_to/libExample2.so
  --package_name PACKAGE_NAME, -p PACKAGE_NAME
                        A global package name to be used for each node in the Model.cpp file.
                        Defaults to Qnn header defined package name
  --op_package_config CUSTOM_OP_CONFIG_PATHS [CUSTOM_OP_CONFIG_PATHS ...], -opc CUSTOM_OP_CONFIG_P
                        Path to a Qnn Op Package XML configuration file that contains user defined
```

```
                        custom operations.

Quantizer Options:
  --quantization_overrides QUANTIZATION_OVERRIDES
                        Use this option to specify a json file with parameters to use for
                        quantization. These will override any quantization data carried from
                        conversion (eg TF fake quantization) or calculated during the normal
                        quantization process. Format defined as per AIMET specification.

Onnx Converter Options:
  --onnx_no_simplification
                        Do not attempt to simplify the model automatically. This may prevent some
                        models from properly converting
                        when sequences of unsupported static operations are present.
  --onnx_batch BATCH    The batch dimension override. This will take the first dimension of all
                        inputs and treat it as a batch dim, overriding it with the value provided
                        here. For example:
                        --batch 6
                        will result in a shape change from [1,3,224,224] to [6,3,224,224].
                        If there are inputs without batch dim this should not be used and each inp
                        should be overridden independently using -d option for input dimension
                        overrides.
  --onnx_define_symbol SYMBOL_NAME VALUE
                        This option allows overriding specific input dimension symbols. For instan
                        you might see input shapes specified with variables such as :
                        data: [1,3,height,width]
                        To override these simply pass the option as:
                        --define_symbol height 224 --define_symbol width 448
                        which results in dimensions that look like:
                        data: [1,3,224,448]

TensorFlow Converter Options:
  --tf_no_optimization  Do not attempt to optimize the model automatically.
  --tf_show_unconsumed_nodes
                        Displays a list of unconsumed nodes, if there any are found. Nodeswhich ar
                        unconsumed do not violate the structural fidelity of thegenerated graph.
  --tf_saved_model_tag SAVED_MODEL_TAG
                        Specify the tag to seletet a MetaGraph from savedmodel. ex:
                        --saved_model_tag serve. Default value will be 'serve' when it is not
                        assigned.
  --tf_saved_model_signature_key SAVED_MODEL_SIGNATURE_KEY
                        Specify signature key to select input and output of the model. ex:
                        --saved_model_signature_key serving_default. Default value will be
                        'serving_default' when it is not assigned
  --tf_validate_models  Validate the original TF model against optimized TF model.
```

```
                        Constant inputs with all value 1s will be generated and will be used
                        by both models and their outputs are checked against each other.
                        The % average error and 90th percentile of output differences will be
                        calculated for this.
                        Note: Usage of this flag will incur extra time due to inference of the
                        models.

Tflite Converter Options:
  --tflite_signature_name SIGNATURE_NAME
                        Use this option to specify a specific Subgraph signature to convert
```

Model Preparation

Quantization Support

Quantization is supported through the converter interface and is performed at conversion time. The only required option to enable quantization along with conversion is the –input_list option, which provides the quantizer with the required input data for the given model. The following options are available in each converter listed above to enable and configure quantization:

```
Quantizer Options:
--quantization_overrides QUANTIZATION_OVERRIDES
                        Use this option to specify a json file with parameters
                        to use for quantization. These will override any
                        quantization data carried from conversion (eg TF fake
                        quantization) or calculated during the normal
                        quantization process. Format defined as per AIMET
                        specification.
--input_list INPUT_LIST
                      Path to a file specifying the input data. This file
                      should be a plain text file, containing one or more
                      absolute file paths per line. Each path is expected to
                      point to a binary file containing one input in the
                      "raw" format, ready to be consumed by the quantizer
                      without any further preprocessing. Multiple files per
                      line separated by spaces indicate multiple inputs to
                      the network. See documentation for more details. Must
                      be specified for quantization. All subsequent
                      quantization options are ignored when this is not
                      provided.
--param_quantizer PARAM_QUANTIZER
```

```
                      Optional parameter to indicate the weight/bias
                      quantizer to use. Must be followed by one of the
                      following options: "tf": Uses the real min/max of the
                      data and specified bitwidth (default) "enhanced": Uses
                      an algorithm useful for quantizing models with long
                      tails present in the weight distribution "adjusted":
                      Uses an adjusted min/max for computing the range,
                      particularly good for denoise models "symmetric":
                      Ensures min and max have the same absolute values
                      about zero. Data will be stored as int#_t data such
                      that the offset is always 0.
--act_quantizer ACT_QUANTIZER
                      Optional parameter to indicate the activation
                      quantizer to use. Must be followed by one of the
                      following options: "tf": Uses the real min/max of the
                      data and specified bitwidth (default) "enhanced": Uses
                      an algorithm useful for quantizing models with long
                      tails present in the weight distribution "adjusted":
                      Uses an adjusted min/max for computing the range,
                      particularly good for denoise models "symmetric":
                      Ensures min and max have the same absolute values
                      about zero. Data will be stored as int#_t data such
                      that the offset is always 0.
--algorithms ALGORITHMS [ALGORITHMS ...]
                      Use this option to enable new optimization algorithms.
                      Usage is: --algorithms <algo_name1> ... The
                      available optimization algorithms are: "cle" - Cross
                      layer equalization includes a number of methods for
                      equalizing weights and biases across layers in order
                      to rectify imbalances that cause quantization errors.
--bias_bw BIAS_BW     Use the --bias_bw option to select the bitwidth to use
                      when quantizing the biases, either 8 (default) or 32.
--act_bw ACT_BW       Use the --act_bw option to select the bitwidth to use
                      when quantizing the activations, either 8 (default) or
                      16.
--weight_bw WEIGHT_BW
                      Use the --weight_bw option to select the bitwidth to
                      use when quantizing the weights, currently only 8 bit
                      (default) supported.
--float_bias_bw FLOAT_BIAS_BW
                      Use the --float_bias_bw option to select the bitwidth to
                      use when biases are in float, either 32 or 16.
--ignore_encodings    Use only quantizer generated encodings, ignoring any
                      user or model provided encodings. Note: Cannot use
                      --ignore_encodings with --quantization_overrides
```

```
--use_per_channel_quantization [USE_PER_CHANNEL_QUANTIZATION [USE_PER_CHANNEL_QUANTIZATION ...]]
                      Use per-channel quantization for
                      convolution-based op weights. Note: This will replace
                      built-in model QAT encodings when used for a given
                      weight.Usage "--use_per_channel_quantization" to
                      enable or "--use_per_channel_quantization false"
                      (default) to disable
--use_per_row_quantization [USE_PER_ROW_QUANTIZATION [USE_PER_ROW_QUANTIZATION ...]]
                      Use this option to enable rowwise quantization of Matmul and
                      FullyConnected op. Usage "--use_per_row_quantization" to enable
                      or "--use_per_row_quantization false" (default) to
                      disable. This option may not be supported by all backends.
```

Basic command line usage to convert and quantize a model using the TF converter would look like:

```
$ qnn-tensorflow-converter -i <path>/frozen_graph.pb
                      -d <network_input_name> <dims>
                      --out_node <network_output_name>
                      -o <optional_output_path>
                      --allow_unconsumed_nodes  # optional, but most likely will be need for larger
                      -p <optional_package_name> # Defaults to "qti.aisw"
                      --input_list input_list.txt
```

This will quantize the network using the default quantizer and bitwidths (8 bits for activations, weights, and biases).

For more detailed information on quantization, options, and algorithms please refer to Quantization.

qairt-quantizer

The **qairt-quantizer** tool converts non-quantized DLC models into quantized DLC models.

```
Basic command line usage looks like:

usage: qairt-quantizer --input_dlc INPUT_DLC [-h] [--output_dlc OUTPUT_DLC]
                       [--input_list INPUT_LIST] [--float_fallback]
                       [--algorithms ALGORITHMS [ALGORITHMS ...]] [--bias_bitwidth BIAS_BITWIDTH]
                       [--act_bitwidth ACT_BITWIDTH] [--weights_bitwidth WEIGHTS_BITWIDTH]
```

```
                    [--float_bitwidth FLOAT_BITWIDTH] [--float_bias_bitwidth FLOAT_BIAS_BITWIDTH]
                    [--ignore_encodings] [--use_per_channel_quantization]
                    [--use_per_row_quantization] [--use_native_input_files]
                    [--use_native_output_files]
                    [--restrict_quantization_steps ENCODING_MIN, ENCODING_MAX]
                    [--act_quantizer_calibration ACT_QUANTIZER_CALIBRATION]
                    [--param_quantizer_calibration PARAM_QUANTIZER_CALIBRATION]
                    [--act_quantizer_schema ACT_QUANTIZER_SCHEMA]
                    [--param_quantizer_schema PARAM_QUANTIZER_SCHEMA]
                    [--percentile_calibration_value PERCENTILE_CALIBRATION_VALUE]
                    [--use_aimet_quantizer]
                    [--op_package_lib OP_PACKAGE_LIB]
                    [--dump_encoding_json] [--debug [DEBUG]]

required arguments:
  --input_dlc INPUT_DLC
                        Path to the dlc container containing the model for which fixed-point
                        encoding metadata should be generated. This argument is required

optional arguments:
  -h, --help            show this help message and exit
  --output_dlc OUTPUT_DLC
                        Path at which the metadata-included quantized model container should be
                        written.If this argument is omitted, the quantized model will be written a
                        <unquantized_model_name>_quantized.dlc
  --input_list INPUT_LIST
                        Path to a file specifying the input data. This file should be a plain text
                        file, containing one or more absolute file paths per line. Each path is
                        expected to point to a binary file containing one input in the "raw" forma
                        ready to be consumed by the quantizer without any further preprocessing.
                        Multiple files per line separated by spaces indicate multiple inputs to th
                        network. See documentation for more details. Must be specified for
                        quantization. All subsequent quantization options are ignored when this is
                        not provided.
  --float_fallback      Use this option to enable fallback to floating point (FP) instead of fixed
                        point.
                        This option can be paired with --float_bitwidth to indicate the bitwidth f
                        FP (by default 32).
                        If this option is enabled, then input list must not be provided and
                        --ignore_encodings must not be provided.
                        The external quantization encodings (encoding file/FakeQuant encodings)
                        might be missing quantization parameters for some interim tensors.
                        First it will try to fill the gaps by propagating across math-invariant
                        functions. If the quantization params are still missing,
                        then it will apply fallback to nodes to floating point.
```

```
--algorithms ALGORITHMS [ALGORITHMS ...]
                      Use this option to enable new optimization algorithms. Usage is:
                      --algorithms <algo_name1> ... The available optimization algorithms are:
                      "cle" - Cross layer equalization includes a number of methods for equalizi
                      weights and biases across layers in order to rectify imbalances that cause
                      quantization errors.
--bias_bitwidth BIAS_BITWIDTH
                      Use the --bias_bitwidth option to select the bitwidth to use when quantizi
                      the biases, either 8 (default) or 32.
--act_bitwidth ACT_BITWIDTH
                      Use the --act_bitwidth option to select the bitwidth to use when quantizin
                      the activations, either 8 (default) or 16.
--weights_bitwidth WEIGHTS_BITWIDTH
                      Use the --weights_bitwidth option to select the bitwidth to use when
                      quantizing the weights, either 4 or 8 (default).
--float_bitwidth FLOAT_BITWIDTH
                      Use the --float_bitwidth option to select the bitwidth to use for float
                      tensors,either 32 (default) or 16.
--float_bias_bitwidth FLOAT_BIAS_BITWIDTH
                      Use the --float_bias_bitwidth option to select the bitwidth to use when
                      biases are in float, either 32 or 16.
--ignore_encodings    Use only quantizer generated encodings, ignoring any user or model provide
                      encodings.
                      Note: Cannot use --ignore_encodings with --quantization_overrides
--use_per_channel_quantization
                      Use this option to enable per-channel quantization for convolution-based o
                      weights.
                      Note: This will replace built-in model QAT encodings when used for a given
                      weight.
--use_per_row_quantization
                      Use this option to enable rowwise quantization of Matmul and FullyConnecte
                      ops.
--use_native_input_files
                      Boolean flag to indicate how to read input files:
                      1. float (default): reads inputs as floats and quantizes if necessary base
                      on quantization parameters in the model.
                      2. native:          reads inputs assuming the data type to be native to th
                      model. For ex., uint8_t.
--use_native_output_files
                      Use this option to indicate the data type of the output files
                      1. float (default): output the file as floats.
                      2. native:          outputs the file that is native to the model. For ex.,
                      uint8_t.
--restrict_quantization_steps ENCODING_MIN, ENCODING_MAX
                      Specifies the number of steps to use for computing quantization encodings
```

```
                      such that scale = (max - min) / number of quantization steps.
                      The option should be passed as a space separated pair of hexadecimal strin
                      minimum and maximum valuesi.e. --restrict_quantization_steps "MIN MAX".
                       Please note that this is a hexadecimal string literal and not a signed
                      integer, to supply a negative value an explicit minus sign is required.
                      E.g.--restrict_quantization_steps "-0x80 0x7F" indicates an example 8 bit
                      range,
                          --restrict_quantization_steps "-0x8000 0x7F7F" indicates an example 16
                      bit range.
                      This argument is required for 16-bit Matmul operations.
--act_quantizer_calibration ACT_QUANTIZER_CALIBRATION
                      Specify which quantization calibration method to use for activations
                      supported values: min-max (default), sqnr, entropy, mse, percentile
                      This option can be paired with --act_quantizer_schema to override the
                      quantization
                      schema to use for activations otherwise default schema(asymmetric) will be
                      used
--param_quantizer_calibration PARAM_QUANTIZER_CALIBRATION
                      Specify which quantization calibration method to use for parameters
                      supported values: min-max (default), sqnr, entropy, mse, percentile
                      This option can be paired with --param_quantizer_schema to override the
                      quantization
                      schema to use for parameters otherwise default schema(asymmetric) will be
                      used
--act_quantizer_schema ACT_QUANTIZER_SCHEMA
                      Specify which quantization schema to use for activations
                      supported values: asymmetric (default), symmetric
--param_quantizer_schema PARAM_QUANTIZER_SCHEMA
                      Specify which quantization schema to use for parameters
                      supported values: asymmetric (default), symmetric
--percentile_calibration_value PERCENTILE_CALIBRATION_VALUE
                      Specify the percentile value to be used with Percentile calibration method
                      The specified float value must lie within 90 and 100, default: 99.99
--use_aimet_quantizer
                      Use AIMET Quantizer in place of IR Quantizer. The following arguments are
                      not allowed together with this option, --float_bw, --float_bitwidth, --flo
                      --float_bias_bitwidth, --disable_relu_squashing, --restrict_quantization_s
                      --float_fallback, --use_dynamic_16_bit_weights, --pack_4_bit_weights, --op
--op_package_lib OP_PACKAGE_LIB, -opl OP_PACKAGE_LIB
                      Use this argument to pass an op package library for quantization. Must be
                      the form <op_package_lib_path:interfaceProviderName> and be separated by a
                      comma for multiple package libs
--dump_encoding_json  Use this argument to dump encoding of all the tensors in a json file
--debug [DEBUG]       Run the quantizer in debug mode.
```

For more information on usage, please refer to SNPE documentation on the snpe-dlc-quant tool.

qnn-model-lib-generator

> **Note**
>
> For developers who want to execute the model preparation tools under Windows-PC, or on a Qualcomm device with a Windows operating system.
> The qnn-model-lib-generator are located under /bin/x86_64-windows-msvc within the SDK for native Windows-PC usage.
> For developers who want to run qnn-model-lib-generator on a device with a Windows OS, it is located under /bin/aarch64-windows-msvc.
> qnn-model-lib-generator will try to use the CMake command from your platform to generate libraries.
> Please make sure the CMake in Windows-OS is feasible by making sure the compile tools are installed(windows-platform compiling tools).

The **qnn-model-lib-generator** tool compiles QNN model source code into artifacts for a specific target.

```
usage: qnn-model-lib-generator [-h] [-c <QNN_MODEL>.cpp] [-b <QNN_MODEL>.bin]
       [-t LIB_TARGETS ] [-l LIB_NAME] [-o OUTPUT_DIR]
Script compiles provided Qnn Model artifacts for specified targets.

Required argument(s):
 -c <QNN_MODEL>.cpp                  Filepath for the qnn model .cpp file

optional argument(s):
 -b <QNN_MODEL>.bin                  Filepath for the qnn model .bin file
                                     (Note: if not passed, runtime will fail if .cpp needs any i

 -t LIB_TARGETS                      Specifies the targets to build the models for. Default: aar
 -l LIB_NAME                         Specifies the name to use for libraries. Default: uses name
                                     else generic qnn_model.so
  -o OUTPUT_DIR                       Location for saving output libraries.
```

> **Note**
>
> For Windows users, please execute this tool with python3.

qnn-op-package-generator

The **qnn-op-package-generator** tool is used to generate skeleton code for a QNN op package using an XML config file that describes the attributes of the package. The tool creates the package as a directory containing skeleton source code and makefiles that can be compiled to create a shared library object.

```
usage: qnn-op-package-generator [-h] --config_path CONFIG_PATH [--debug]
                                [--output_path OUTPUT_PATH] [-f]

optional arguments:
  -h, --help            show this help message and exit

required arguments:
  --config_path CONFIG_PATH, -p CONFIG_PATH
                        The path to a config file that defines a QNN Op
                        package(s).

optional arguments:
  --debug               Returns debugging information from generating the
                        package
  --output_path OUTPUT_PATH, -o OUTPUT_PATH
                        Path where the package should be saved
  -f, --force-generation
                        This option will delete the entire existing package
                        Note appropriate file permissions must be set to use
                        this option.
  --converter_op_package, -cop
                        Generates Converter Op Package skeleton code needed
                        by the output shape inference for converters
```

qnn-context-binary-generator

The **qnn-context-binary-generator** tool is used to create a context binary by using a particular backend and consuming a model library created by the qnn-model-lib-generator.

```
usage: qnn-context-binary-generator --model QNN_MODEL.so --backend QNN_BACKEND.so
                                    --binary_file BINARY_FILE_NAME
                                    [--model_prefix MODEL_PREFIX]
                                    [--output_dir OUTPUT_DIRECTORY]
                                    [--op_packages ONE_OR_MORE_OP_PACKAGES]
                                    [--config_file CONFIG_FILE.json]
                                    [--profiling_level PROFILING_LEVEL]
                                    [--verbose] [--version] [--help]

REQUIRED ARGUMENTS:
-------------------
  --model                       <FILE>      Path to the <qnn_model_name.so> file containing a QN
                                            To create a context binary with multiple graphs, use
                                            comma-separated list of model.so files. The syntax i
                                            <qnn_model_name_1.so>,<qnn_model_name_2.so>.

  --backend                     <FILE>      Path to a QNN backend .so library to create the cont

  --binary_file                 <VAL>       Name of the binary file to save the context binary t
                                            Saved in the same path as --output_dir option with .
                                            as the binary file extension. If not provided, no ba
                                            is created.


OPTIONAL ARGUMENTS:
-------------------
  --model_prefix                            Function prefix to use when loading <qnn_model_name.
                                            containing a QNN network. Default: QnnModel.

  --output_dir                  <DIR>       The directory to save output to. Defaults to ./outpu

  --op_packages                 <VAL>       Provide a comma separated list of op packages
                                            and interface providers to register. The syntax is:
                                            op_package_path:interface_provider[,op_package_path:

  --profiling_level             <VAL>       Enable profiling. Valid Values:
                                            1. basic:    captures execution and init time.
                                            2. detailed: in addition to basic, captures per Op t
                                                for execution.
                                            3. backend:  backend-specific profiling level specif
```

```
                                                in the backend extension related JSON config fil

--profiling_option                  <VAL>       Set profiling options:
                                                1. optrace:    Generates an optrace of the run.

--config_file                       <FILE>      Path to a JSON config file. The config file currentl
                                                supports options related to backend extensions and
                                                context priority. Please refer to SDK documentation
                                                for more details.

--enable_intermediate_outputs                   Enable all intermediate nodes to be output along wit
                                                default outputs in the saved context.
                                                Note that options --enable_intermediate_outputs and
                                                are mutually exclusive. Only one of the options can

--set_output_tensors                <VAL>       Provide a comma-separated list of intermediate outpu
                                                will be written in addition to final graph output te
                                                Note that options --enable_intermediate_outputs and
                                                are mutually exclusive. Only one of the options can
                                                The syntax is: graphName0:tensorName0,tensorName1;gr

--backend_binary                    <VAL>       Name of the file to save a backend-specific context
                                                binary to.
                                                Saved in the same path as --output_dir option with .
                                                as the binary file extension.

--log_level                                     Specifies max logging level to be set. Valid setting
                                                "error", "warn", "info" and "verbose"

--dlc_path                          <VAL>       Paths to a comma separated list of Deep Learning Con
                                                Necessitates libQnnModelDlc.so as the --model argume
                                                To compose multiple graphs in the context, use comma
                                                The syntax is <qnn_model_name_1.dlc>,<qnn_model_name
                                                Default: None

--input_output_tensor_mem_type      <VAL>       Specifies mem type to be used for input and output t
                                                Valid settings:"raw" and "memhandle"

--platform_options                  <VAL>       Specifies values to pass as platform options. Multip
                                                using the syntax: key0:value0;key1:value1;key2:value

--version                                       Print the QNN SDK version.

--help                                          Show this help message.
```

See qnn-net-run section for more details about `--op_packages` and `--config_file` options.

Execution

qnn-net-run

The **qnn-net-run** tool is used to consume a model library compiled from the output of the QNN converter, and run it on a particular backend.

```
DESCRIPTION:
------------
Example application demonstrating how to load and execute a neural network
using QNN APIs.


REQUIRED ARGUMENTS:
-------------------
  --model              <FILE>       Path to the model containing a QNN network.
                                    To compose multiple graphs, use comma-separated list of
                                    model.so files. The syntax is
                                    <qnn_model_name_1.so>,<qnn_model_name_2.so>.

  --backend            <FILE>       Path to a QNN backend to execute the model.

  --input_list         <FILE>       Path to a file listing the inputs for the network.
                                    If there are multiple graphs in model.so, this has
                                    to be comma-separated list of input list files.
                                    When multiple graphs are present, to skip execution of a graph
                                    "__"(double underscore without quotes) as the file name in the
                                    comma-seperated list of input list files.

  --retrieve_context   <VAL>       Path to cached binary from which to load a saved
                                   context from and execute graphs. --retrieve_context and
                                   --model are mutually exclusive. Only one of the options
                                   can be specified at a time.


OPTIONAL ARGUMENTS:
-------------------
  --model_prefix                             Function prefix to use when loading <qnn_model_name.s
                                             Default: QnnModel

  --debug                                    Specifies that output from all layers of the network
```

```
                                                  will be saved. This option can not be used when loadi
                                                  a saved context through --retrieve_context option.

--output_dir                    <DIR>             The directory to save output to. Defaults to ./output

--use_native_output_files                         Specifies that the output files will be generated in
                                                  type native to the graph. If not specified, output fi
                                                  be generated in floating point.

--use_native_input_files                          Specifies that the input files will be parsed in the
                                                  type native to the graph. If not specified, input fil
                                                  be parsed in floating point. Note that options --use_
                                                  and --native_input_tensor_names are mutually exclusiv
                                                  Only one of the options can be specified at a time.

--native_input_tensor_names     <VAL>             Provide a comma-separated list of input tensor names,
                                                  for which the input files would be read/parsed in nat
                                                  Note that options --use_native_input_files and
                                                  --native_input_tensor_names are mutually exclusive.
                                                  Only one of the options can be specified at a time.
                                                  The syntax is: graphName0:tensorName0,tensorName1;gra

--op_packages                   <VAL>             Provide a comma-separated list of op packages, interf
                                                  providers, and, optionally, targets to register. Vali
                                                  for target are CPU and HTP. The syntax is:
                                                  op_package_path:interface_provider:target[,op_package

--profiling_level               <VAL>             Enable profiling. Valid Values:
                                                    1. basic:    captures execution and init time.
                                                    2. detailed: in addition to basic, captures per Op
                                                                 for execution, if a backend supports i
                                                    3. client:   captures only the performance metrics
                                                                 measured by qnn-net-run.

--perf_profile                  <VAL>             Specifies performance profile to be used. Valid setti
                                                  low_balanced, balanced, default, high_performance,
                                                  sustained_high_performance, burst, low_power_saver,
                                                  power_saver, high_power_saver, extreme_power_saver
                                                  and system_settings.
                                                  Note: perf_profile argument is now deprecated for
                                                  HTP backend, user can specify performance profile
                                                  through backend config now. Please refer to config_fi
                                                  backend extensions usage section below for more detai
```

--config_file	<FILE>	Path to a JSON config file. The config file currently supports options related to backend extensions, context priority and graph configs. Please refer to S documentation for more details.
--log_level	<VAL>	Specifies max logging level to be set. Valid settings error, warn, info, debug, and verbose.
--shared_buffer		Specifies creation of shared buffers for graph I/O be and the device/coprocessor associated with a backend This option is currently supported on Android only.
--synchronous		Specifies that graphs should be executed synchronousl If a backend does not support asynchronous execution,
--num_inferences	<VAL>	Specifies the number of inferences. Loops over the in the number of inferences has transpired.
--duration	<VAL>	Specifies the duration of the graph execution in seco Loops over the input_list until this amount of time h
--keep_num_outputs	<VAL>	Specifies the number of outputs to be saved. Once the number of outputs reach the limit, subsequen
--batch_multiplier	<VAL>	Specifies the value with which the batch value in inp will be multiplied. The modified input and output ten the execute graphs. Composed graphs will still use th
--timeout	<VAL>	Specifies the value of the timeout for execution of g using this option with a backend that does not suppor
--retrieve_context_timeout	<VAL>	Specifies the value of the timeout for initialization using this option with a backend that does not suppor Also note that this option can only be used when load --retrieve_context option.
--max_input_cache_tensor_sets	<VAL>	Specifies the maximum number of input tensor sets tha Use value "-1" to cache all the input tensors created Note that options --max_input_cache_tensor_sets and - Only one of the options can be specified at a time.
--max_input_cache_size_mb	<VAL>	Specifies the maximum cache size in mega bytes(MB). Note that options --max_input_cache_tensor_sets and - Only one of the options can be specified at a time.

```
 --set_output_tensors          <VAL>       Provide a comma-separated list of intermediate output
                                           will be written in addition to final graph output ten
                                           --set_output_tensors are mutually exclusive. Only one
                                           Also note that this option can not be used when graph
                                           since the graph is already finalized when retrieved f
                                           The syntax is: graphName0:tensorName0,tensorName1;gra

 --use_mmap                                Specifies that the context binary that is being read
                                           using the Memory-mapped (MMAP) file I/O. Please note
                                           may not support this due to OS limitations in which c
                                           is thrown when this option is used.

 --validate_binary                         Specifies that the context binary will be validated b
                                           This option can only be used with backends that suppo

 --platform_options            <VAL>       Specifies values to pass as platform options. Multipl
                                           using the syntax: key0:value0;key1:value1;key2:value2

 --graph_profiling_start_delay <VAL>       Specifies graph profiling start delay in seconds. Ple
                                           in conjunction with graph-level profiling handles.

 --dlc_path                    <VAL>       Paths to a comma separated list of Deep Learning Cont
                                           Necessitates libQnnModelDlc.so as the --model argumen
                                           To compose multiple graphs in the context, use comma-
                                           The syntax is <qnn_model_name_1.dlc>,<qnn_model_name_
                                           Default: None

 --graph_profiling_num_executions  <VAL>   Specifies the maximum number of QnnGraph_execute/QnnG
                                           Please Note that this option can only be used in conj

 --version                                 Print the QNN SDK version.

 --help                                    Show this help message.
```

See `<QNN_SDK_ROOT>/examples/QNN/NetRun` folder for reference example on how to use `qnn-net-run` tool.

Typical arguments:

`--backend` - The appropriate argument depends on what target and backend you want to run on

Android (aarch64): `<QNN_SDK_ROOT>/lib/aarch64-android/`

- CPU - `libQnnCpu.so`
- GPU - `libQnnGpu.so`
- HTA - `libQnnHta.so`
- DSP (Hexagon v65) - `libQnnDspV65Stub.so`
- DSP (Hexagon v66) - `libQnnDspV66Stub.so`
- DSP - `libQnnDsp.so`
- HTP (Hexagon v68) - `libQnnHtp.so`
- [Deprecated] HTP Alternate Prepare (Hexagon v68) - `libQnnHtpAltPrepStub.so`
- Saver - `libQnnSaver.so`

Linux x86: `<QNN_SDK_ROOT>/lib/x86_64-linux-clang/`

- CPU - `libQnnCpu.so`
- HTP (Hexagon v68) - `libQnnHtp.so`
- Saver - `libQnnSaver.so`

Windows x86: `<QNN_SDK_ROOT>/lib/x86_64-windows-msvc/`

- CPU - `QnnCpu.dll`
- Saver - `QnnSaver.dll`

WoS: `<QNN_SDK_ROOT>/lib/aarch64-windows-msvc/`

- CPU - `QnnCpu.dll`
- DSP (Hexagon v66) - `QnnDspV66Stub.dll`
- DSP - `QnnDsp.dll`
- HTP (Hexagon v68) - `QnnHtp.dll`
- Saver - `QnnSaver.dll`

> **Note**
>
> Hexagon based backend libraries are emulations on x86_64 platforms

`--input_list` - This argument provides a file containing paths to input files to be used for graph execution. Input files can be specified with the below format:

```
<input_layer_name>:=<input_layer_path>[<space><input_layer_name>:=<input_layer_path>]
[<input_layer_name>:=<input_layer_path>[<space><input_layer_name>:=<input_layer_path>]]
...
```

Below is an example containing 3 sets of inputs with layer names "Input_1" and "Input_2", and files located in the relative path "Placeholder_1/real_input_inputs_1/":

```
Input_1:=Placeholder_1/real_input_inputs_1/0-0#e6fb51.rawtensor Input_2:=Placeholder_1/real_inp
Input_1:=Placeholder_1/real_input_inputs_1/1-0#67c965.rawtensor Input_2:=Placeholder_1/real_inp
Input_1:=Placeholder_1/real_input_inputs_1/2-0#b42dc6.rawtensor Input_2:=Placeholder_1/real_inp
```

Note: If the batch dimension of the model is greater than 1, the number of batch elements in the input file has to either match the batch dimension specified in the model or it has to be one. In the latter case, qnn-net-run will combine multiple lines into a single input tensor.

`--op_packages` - This argument is only needed if you are using custom op packages. The native QNN ops are already included as part of the backend libraries.

When using custom op packages, each provided op package requires a colon separated command line argument containing the path to the op package shared library (.so) file, as well as the name of the interface provider, formatted as `<op_package_path>:<interface_provider>`.

The interface_provider argument must be the name of the function in the op package library that satisfies the QnnOpPackage_InterfaceProvider_t interface. In the skeleton code created by `qnn-op-package-generator`, this function will be named `<package_name><backend>InterfaceProvider`.

See Generating Op Packages for more information.

`--config_file` - This argument is only needed if you need to specify context priority or provide backend extensions related parameters. These parameters are specified through a JSON file. The template of the JSON file is shown below:

```
{
  "backend_extensions" :
    {
      "shared_library_path" :  "path_to_shared_library",
      "config_file_path" :  "path_to_config_file"
    },
  "context_configs" :
    {
      "context_priority" :  "low | normal | normal_high | high",
      "async_execute_queue_depth" : uint32_value,
      "enable_graphs" :  ["<graph_name_1>", "<graph_name_2>", ...],
      "memory_limit_hint"  : uint64_value,
      "is_persistent_binary" : boolean_value,
      "cache_compatibility_mode" : "permissive | strict"
    },
  "graph_configs" : [
    {
      "graph_name" :  "graph_name_1",
      "graph_priority" :  "low | normal | normal_high | high"
      "graph_profiling_start_delay" : double_value
      "graph_profiling_num_executions" : uint64_value
    }
  ],
  "profile_configs" :
    {
      "num_max_events" : uint64_value
    },
  "async_graph_execution_config" :
```

```
    {
      "input_tensors_creation_tasks_limit" : uint32_value,
      "execute_enqueue_tasks_limit" : uint32_value
    }
}
```

All the options in the JSON file are optional. *context_priority* is used to specify priority of the context as a context config. *async_execute_queue_depth* is used to specify the number of executions that can be in the queue at a given time. While using a context binary, *enable_graphs* is used to implement the *graph selection* functionality. *memory_limit_hint* is used to set the peak memory limit hint of a deserialized context in MBs. *is_persistent_binary* indicates that the context binary pointer is available during QnnContext_createFromBinary and until QnnContext_free is called.

Set Cache Compatibility Mode : *cache_compatibility_mode* specifies the mode used to check whether cache record is optimal for the device. The available modes indicate binary cache compatibility:

- "permissive": Binary cache is compatible if it could run on the device; default.
- "strict": Binary cache is compatible if it could run on the device and fully utilize hardware capability. If it cannot fully utilize hardware, selecting this option results in a recommendation to prepare the cache again. This option returns an error if it is not supported by the selected backend.

Graph Selection : Allows to specify a subset of graphs in a context to be loaded and executed. If *enable_graphs* is specified, only those graphs are loaded. If a graph name is selected and it doesn't exist, that would be an error. If *enable_graphs* is not specified or passed as an empty list, default behaviour continues where all graphs in a context are loaded.

graph_configs can be used to specify asynchronous execution order and depth, if a backend supports asynchronous execution. Every set of graph configs has to be specified along with a graph name. *graph_profiling_start_delay* is used to set the profiling start delay time in seconds.

graph_profiling_num_executions is used to set the maximum number of QnnGraph_execute/QnnGraph_executeAsync calls that will be profiled.

profile_configs can be used to specify the max profile events per profiling handle.

async_graph_execution_config can be used to specify the limits on number of tasks that run in parallel when graphs are executed asynchronously using graphExecuteAsync. *input_tensors_creation_tasks_limit* specifies the maximum number of tasks in which input tensor sets are populated, which can be used for graph execution. *execute_enqueue_tasks_limit* specifies the maximum number of tasks in which the backend graphExecuteAsync will be called using the pre-populated input tensors. If unspecified, these values will be set to the specified "async_execute_queue_depth" or 10 which is the default for "async_execute_queue_depth".

backend_extensions is used to exercise custom options in a particular backend. This can be done by providing an extensions shared library (.so) and a config file, if necessary. This is also required to enable various performance modes, which can be exercised using backend config. Currently, HTP supports it through `libQnnHtpNetRunExtensions.so` shared library and DSP supports it through `libQnnDspNetRunExtensions.so` . For different custom options which can be enabled with HTP see HTP Backend Extensions

`--shared_buffer` - This argument is only needed to indicate qnn-net-run to use shared buffers for zero-copy use case with a device/coprocessor associated with a particular backend (for ex., DSP with HTP backend) for graph input and output tensor data. This option is supported on Android only. qnn-net-run implements this feature using rpcmem APIs, which further create shared buffers using ION/DMA-BUF memory allocator on Android, available through the shared library libcdsprpc.so. In addition to specifying this option, for qnn-net-run to be able to discover libcdsprpc.so, the path in which the shared library is present needs to be appended to LD_LIBRARY_PATH variable.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/vendor/lib64
```

Running Quantized Model on HTP backend with qnn-net-run

The HTP backend currently allows to finalize / create an optimized version of a quantized QNN model offline, on Linux development host (using `x86_64-linux-clang` backend library) and then execute the finalized model on device (using `hexagon-v68` backend libraries).

First, configure the environment by following instructions in Setup section. Next, build QNN Model library from your network, using artifacts produced by one of QNN converters. See Building Example Model for reference. Lastly, use the `qnn-context-binary-generator` utility to generate a serialized representation of the finalized graph to execute the serialized binary on device.

```
# Generate the optimized serialized representation of QNN Model on Linux development host.
$ qnn-context-binary-generator --binary_file qnngraph.serialized.bin \
                               --model <path_to_model_library>/libQnnModel.so \ # a x86_64-linux
                               --backend ${QNN_SDK_ROOT}/lib/x86_64-linux-clang/libQnnHtp.so \
                               --output_dir <output_dir_for_result_and_qnngraph_serialized_binar
```

To use produced serialized representation of the finalized graph (`qnngraph.serialized.bin`) ensure the below binaries are available on the android device:

- `libQnnHtpV68Stub.so` (ARM)
- `libQnnHtpPrepare.so` (ARM)
- `libQnnModel.so` (ARM)
- `libQnnHtpV68Skel.so` (cDSP v68)
- `qnngraph.serialized.bin` (serialized binary from run on Linux development host)

See `<QNN_SDK_ROOT>/examples/QNN/NetRun/android/android-qnn-net-run.sh` script for reference on how to use `qnn-net-run` tool on android device.

```
# Run the optimized graph on HTP target
$ qnn-net-run --retrieve_context qnngraph.serialized.bin \
              --backend <path_to_model_library>/libQnnHtp.so \
```

```
4               --output_dir <output_dir_for_result> \
5               --input_list <path_to_input_list.txt>
```

Running Float Model on HTP backend with qnn-net-run

The QNN HTP backend can support running float32 models on select Qualcomm SoCs.

First, configure the environment by following instructions in Setup section. Next, build QNN Model library from your network, using artifacts produced by one of QNN converters. See Building Example Model for reference.

Lastly, configure *backend_extensions* parameters through a JSON file and set custom options for the HTP backend. Pass this file to qnn-net-run using `--config_file` argument. *backend_extensions* take two parameters, an extensions shared library (.so) (for HTP use `libQnnHtpNetRunExtensions.so`) and a config file for the backend.

Below is the template for the JSON file:

```
{
  "backend_extensions" :
    {
      "shared_library_path" : "path_to_shared_library",
      "config_file_path" : "path_to_config_file"
    }
}
```

For HTP backend extensions configurations, you can set “vtcm_mb”, “fp16_relaxed_precision” and “graph_names” through a config file.

Here is an example of the config file:

```
1 {
2    "graphs": [
3       {
```

```
        "vtcm_mb": 8,   // Provides performance infrastructure configuration options that are me
                        // Optional; if not set, QNN HTP defaults to 4.
        "fp16_relaxed_precision": 1,  // Ensures that operations will run with relaxed precisio

        "graph_names": [ "qnn_model" ]  // Provide the list of names of the graph for the infe
                                        // "qnn_model" must be the name of the .cpp file genera
        .....
      },
      {
         .....  // Other graph object
      }
   ]
}
```

Note

"fp16_relaxed_precision" is the key configuration to enable running QNN float models on HTP float runtime. HTP Graph Configurations such as fp16_relaxed_precision, vtcm_mb etc are only applied if at least one "graph_name" is provided in backend extensions config.

See `<QNN_SDK_ROOT>/examples/QNN/NetRun/android/android-qnn-net-run.sh` script for reference on how to use `qnn-net-run` tool on android device.

```
# Run the optimized graph on HTP target
$ qnn-net-run --model <path_to_model_library>/libQnnModel.so \ # a x86_64-linux-clang built floa
              --backend ${QNN_SDK_ROOT}/lib/x86_64-linux-clang/libQnnHtp.so \
              --config_file <path_to_JSON_file.json> \
              --output_dir <output_dir_for_result> \
              --input_list <path_to_input_list.txt>
```

qnn-throughput-net-run

The **qnn-throughput-net-run** tool is used to exercise the execution of multiple models on a QNN backend or on different backends in a multi-threaded fashion. It allows repeated execution of models on a specified backend for a specified duration or number of iterations.

```
Usage:
------
qnn-throughput-net-run [--config <config_file>.json]
                       [--output <results>.json]

REQUIRED argument(s):
 --config        <FILE>.json       Path to the json config file .

OPTIONAL argument(s):
 --output        <FILE>.json       Specify the json file used to save the performance test results
```

Configuration JSON File:

qnn-throughput-net-run uses configuration file as input to run the models on the backends. The configuration json file comprises of four objects (required) - **backends**, **models**, **contexts** and **testCase**.

Below is an example of a json configuration file. Please refer the following section for detailed information on the four configuration objects backends, models, contexts and testCase.

```
{
  "backends": [
    {
      "backendName": "cpu_backend",
      "backendPath": "libQnnCpu.so",
      "profilingLevel": "BASIC",
      "backendExtensions": "libQnnHtpNetRunExtensions.so",
      "perfProfile": "high_performance"
    },
    {
      "backendName": "gpu_backend",
      "backendPath": "libQnnGpu.so",
      "profilingLevel": "OFF"
    }
  ],
  "models": [
    {
      "modelName": "model_1",
```

```
      "modelPath": "libqnn_model_1.so",
      "loadFromCachedBinary": false,
      "inputPath": "model_1-input_list.txt",
      "inputDataType": "FLOAT",
      "postProcessor": "MSE",
      "outputPath": "model_1-output",
      "outputDataType": "FLOAT_ONLY",
      "saveOutput": "NATIVE_ALL",
      "groundTruthPath": "model_1-golden_list.txt"
    },
    {
      "modelName": "model_2",
      "modelPath": "libqnn_model_2.so",
      "loadFromCachedBinary": false,
      "inputPath": "model_2-input_list.txt",
      "inputDataType": "FLOAT",
      "postProcessor": "MSE",
      "outputPath": "model_2-output",
      "outputDataType": "FLOAT_ONLY",
      "saveOutput": "NATIVE_LAST"
    }
  ],
  "contexts": [
    {
      "contextName": "cpu_context_1"
    },
    {
      "contextName": "gpu_context_1"
    }
  ],
  "testCase": {
    "iteration": 5,
    "logLevel": "error",
    "threads": [
      {
        "threadName": "cpu_thread_1",
        "backend": "cpu_backend",
        "context": "cpu_context_1",
        "model": "model_1",
        "interval": 10,
        "loopUnit": "count",
        "loop": 1
      },
      {
        "threadName": "gpu_thread_1",
```

```
        "backend": "gpu_backend",
        "context": "gpu_context_1",
        "model": "model_2",
        "interval": 0,
        "loopUnit": "count",
        "loop": 10
      }
    ]
  }
}
```

backends : Property value is an array of json objects, where each object contains the needed backend information on which the models are executed. Each object of the array has the following properties as key/value pairs.

Key	Value Type	Default Value	Optional / Required	
backendName	string	-	Required	Is a u
backendPath	string	-	Required	
profilingLevel	string	OFF	Optional	S • BASIC - (• DETAILE
backendExtensions	string	-	Optional	Enables This is required
perfProfile	string	default	Optional	F s
opPackagePath	string	Native QNN Ops. part of the backend libraries	Optional	C Syntax: op_pa

platformOption	string	-	Optional	

models : Property value is an array of json objects, where each object contatins details about a model and corresponding input data and post-processing information. Each object of the array has the following properties as key/value pairs.

Key	Value Type	Default Value	Optional / Required	
modelName	string	-	Required	Is a unique identifier f
modelPath	string	-	Required	Specifies the <model>.s
loadFromCachedBinary	bool	false	Optional	Set to `true` if <se
inputPath	string	-	Optional	Path to a file lis If there are multi <serialized_context>.b list of input patl Graph1_input_ If not set, Ra
inputDataType	string	NATIVE	Optional	Possible v

postProcessor	string	-	Optional	Possible values: NONI
				MSE_INT16. If there are
				<serialized_context>.b
				list of postProcessc
				MSE will output a m
				execution with the go
				`groundTruthPath`
				specified, the first e
				compute the MSE. If
				`groundTruthPath` i
				type, users need to s
				`postPr`

outputPath	string	-	Optional	If `postProcessor` is logs will be

outputDataType	string	NATIVE_ONLY	Optional	Possible values: FLO
saveOutput	string	NONE	Optional	Possible values: NO • NATIVE_LAST - network execu • NATIVE_ALL - S executions to t
groundTruthPath	string	NONE	Optional	Specifies the golden file are multiple g <serialized_context>.b list of ground truth Graph1_ground_truth_p

contexts : Property value is an array of json objects, where each object contains all the context information. Each object of the array has the following properties as key/value pairs.

Key	Value Type	Default Value	Optional / Required	Description
contextName	string	-	Required	Is a unique identifier for the testcase to designate the context in which a model should be created.
priority	string	DEFAULT	Optional	Specifies the priority of the context. Possible values: DEFAULT, LOW, NORMAL, HIGH.
executeAsyncQueueDepth	int	-	Optional	Specfies the queue depth for async execution.
cacheCompatibilityMode	string	-	Optional	Specifies the cache compatibility check mode; valid values

				are: “permissive” (default), and “strict”.

testCase : Property value is a json object that specifies the testing configuration that controls multi-threaded execution.

Key	Value Type	Default Value	Optional / Required	Description
`iteration`	`int`	-	`Required`	Number of times the entire use case is repeated. If the value is `negative` , test runs forever until keyboard interrupt.
`logLevel`	`string`	-	`Optional`	Specifies max logging level to be set. Valid settings: `error` , `warn` , `info` , `debug` , and `verbose`
`threads`	`string`	-	`Required`	Property value is an array of json objects, where each object contains all the thread details, that are to be executed by the qnn-throughput-net-run. Each object of the array has the below properties listed under threads as key/value pairs.

`threads` : Property value is an array containing all the threads and corresponding backend, context and models information. Each element of the array can have the following required/optional property.

Key	Value Type	Default Value	Optional / Required	Description
`threadName`	`string`	-	`Required`	Is a unique identifier for the testc the thread and save the outp
`backend`	`string`	-	`Required`	Specifies the backend to be us thread executes the graph. The v should match with one of the `b` entry in the `backends` prop configuration json

context	string	-	Required	Specifies the context to be use thread executes the graph. The v should match with one of the c entry in the contexts prop configuration json
model	string	-	Required	Specifies the model to be used b execution. The value specified s with one of the modelName models property of the config
initModelInLoop	bool	false	Optional	Set it to true if the model r initialized repeatedly for every i value cannot be set to t loadFromCachedBinary fro property is true
loadInputDataInLoop	bool	false	Optional	Set it to true if the input needs for every loop of execu
useRandomData	bool	false	Optional	Set it to true if random data is used as input.
interval	int	0	Optional	Repesents the interval (in mic between each graph execution
loopUnit	string	count	Optional	Possible values: count, s
loop	int	1	Optional	Value is taken either as seconds on the value for the loopUnit . is second , the value specifies seconds the threads repeats e loopUnit is count , the va number of times thread repeat
executeAsynchronous	bool	false	Optional	Set it to true if the graphs shou asynchronously rather than synch backend does not support as execution, this option results
backendConfig	string	-	Optional	Specifies the backend config f backend specific options backendExtensions sha Syntax: path_to_backend_c

An example json file `sample_config.json` file can be found at `<QNN_SDK_ROOT>/examples/QNN/ThroughputNetRun`.

Analysis

qnn-quantization-checker (Beta)

The **qnn-quantization-checker** tool is used to analyze the activations, weights and biases of all the possible quantization options available in the qnn-converter for each subsequent layer of a single model file or a directory of models. The analysis involves comparing the weight, bias and activation tensors of the quantized options to their unquantized counterparts. The tool runs the model to generate floating-point activations and analyzes floating-point weights, biases and activations to determine the quality of the encodings. It finally outputs the problematic weight, bias and activation tensors for all quantization options.

```
X86-Linux/ WSL Usage: qnn-quantization-checker [--model MODEL_PATH]
                                [--input_list INPUT_LIST_PATH]
                                [--activation_width ACT_BW]
                                [--bias_width BIAS_BW]
                                [--output_dir OUTPUT_DIR_PATH]
                                [--skip_building_model]
                                [--skip_generator] [--skip_runner]
                                [--generate_histogram] [--per_channel_histogram]
                                [--output_csv]
                                --config_file CONFIG_FILE_PATH

X86-Windows/ Windows on Snapdragon Usage: python qnn-quantization-checker [--model MODEL_PATH]
                                [--input_list INPUT_LIST_PATH]
                                [--activation_width ACT_BW]
                                [--bias_width BIAS_BW]
                                [--output_dir OUTPUT_DIR_PATH]
                                [--skip_building_model]
                                [--skip_generator] [--skip_runner]
                                [--generate_histogram] [--per_channel_histogram]
                                [--output_csv]
                                --config_file CONFIG_FILE_PATH
```

required arguments:

–config_file - Path to the config file specifying all possible options required for execution.

E.g., [RUN_IN_REPAIR_MODE, MODEL_PATH, INPUT_LIST_PATH, ACTIVATION_WIDTH, BIAS_WIDTH, WEIGHT_WIDTH, INPUT_LAYOUT, OUTPUT_DIR_PATH, CLANG_PATH, BASH_PATH, BIN_PATH, PY3_PATH, TENSORFLOW_HOME, TFLITE_HOME, ONNX_HOME, QUANTIZATION_OVERRIDES, WEIGHT_COMPARISON_ALGORITHMS, BIAS_COMPARISON_ALGORITHMS, ACT_COMPARISON_ALGORITHMS, INPUT_DATA_ANALYSIS_ALGORITHMS, OUTPUT_CSV, GENERATE_HISTOGRAM, QUANTIZATION_VARIATIONS, QUANTIZATION_ALGORITHMS]

optional arguments:

–model - Path to model graph file, it is required if MODEL_PATH is not set in the config_file. –input_list - Path to a text file containing a list of input files. It is required if the MODEL_PATH/model refers to a single model file and INPUT_LIST_PATH is not present in the config_file. –activation_width - [Optional] Bit-width to use for activations. E.g., 8, 16. Default is 8. It can also be set using ACTIVATION_WIDTH field in the config_file. –bias_width - [Optional] Bit-width to use for biases. E.g., 8, 32. Default is 8. It can also be set using BIAS_WIDTH in the config_file. –output_dir - [Optional] Path to store the unquantized output files. It can also be set using OUTPUT_DIR_PATH in the config_file. –skip_building_model - [Optional] Stop the tool from building the model. It is assumed the model.so is pre-built. –skip_generator - [Optional] Stop the tool from running the QNN converter. It is assumed that the necessary files are already available. –skip_runner - [Optional] Stop the tool from running the model. It is assumed that the necessary files are already available. –generate_histogram - [Optional] Generate histogram analysis for weights/biases. Default is to skip histgoram generation. –per_channel_histogram - [Optional] Generate per channel histogram analysis for weights/biases. Default is to skip histgoram generation. –output_csv - [Optional] Store analysis results in csv files in the output directory.

Please note: the arguments accepted on the command line will override those in the config file if there is overlap.

Here is an example of different algorithms and thresholds mentioned in the config file:

```
{
   // All other required user options can be added here

   "WEIGHT_COMPARISON_ALGORITHMS": [{"algo_name":"minmax", "threshold":"10"}, {"algo_name":"maxdif
   "BIAS_COMPARISON_ALGORITHMS": [{"algo_name":"minmax", "threshold":"10"}, {"algo_name":"maxdiff"
   "ACT_COMPARISON_ALGORITHMS": [{"algo_name":"minmax", "threshold":"10"}, {"algo_name":"data_rang
}
```

The tool generates a **html** directory to include all .html files containing clear indicators of failures and successes for different combinations of quantization options and input files. The tool also produces a **csv** directory to include all .csv files storing detailed computation results for each metric. Additionally, the tool generates an output log file under the **qnn-quantization-checker-log** directory which contains all log outputs.

Sample Command

```
qnn-quantization-checker --config_file ${SDK_ROOT}/lib/python/qti/aisw/quantization_checker/config
```

Sample Config File

```
{
"CLANG_PATH": "{Path_To_Clang_Lib}/clang-14.0.0/usr/bin",
"PY3_PATH": "/{PY3_PATH}/py3env/bin",
"BASH_PATH": "/bin",
"BIN_PATH": "/usr/bin",
"TENSORFLOW_HOME": "/{Path_To_Tensorflow}/bionic/py3",
"TFLITE_HOME": "/{Path_To_TFLITE}/bionic/py3",
"ONNX_HOME": "/{Path_To_Onnx}/bionic/py3",
"MODEL_PATH": "/{Path_To_Model}/model_frozen.pb",
"INPUT_LIST_PATH": "/{Path_To_Input}/input_list.txt",
"OUTPUT_DIR_PATH": "{Path_To_Output_Dir}"
"WEIGHT_COMPARISON_ALGORITHMS": [{"algo_name":"minmax", "threshold":"10"}, {"algo_name":"maxdiff",
"BIAS_COMPARISON_ALGORITHMS": [{"algo_name":"minmax", "threshold":"10"}, {"algo_name":"maxdiff", "
```

```
"ACT_COMPARISON_ALGORITHMS": [{"algo_name":"minmax", "threshold":"10"}, {"algo_name":"data_range_a
"INPUT_DATA_ANALYSIS_ALGORITHMS": [{"algo_name":"stats", "threshold":"2"}],
"OUTPUT_CSV": "True",
}
```

Viewing the results (html, csv or log files)

All results are stored in output directories under the user specified directory.

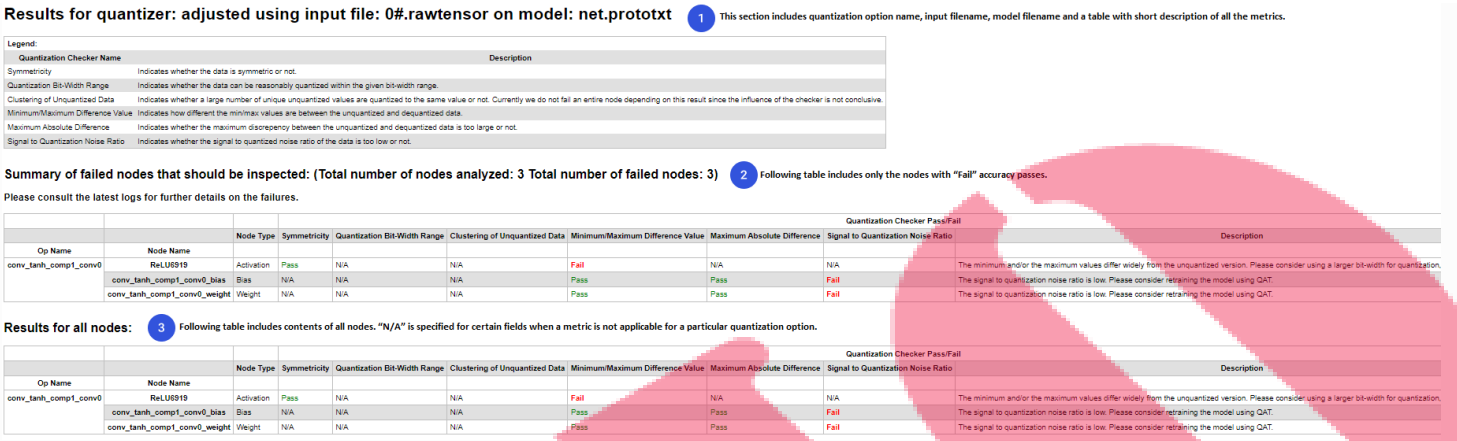
```
# In user_specified_dir, all .html files are stored in "user_specified_dir/html", .csv files are s
cd ${Path_to_the_model}/user_specified_dir/html
# This directory contains all the html files produced by the snpe-quantization-checker which can b
cd ${Path_to_the_model}/user_specified_dir/csv
# This directory contains all the csv files produced by the snpe-quantization-checker which can be
cd ${Path_to_the_model}/user_specified_dir/snpe-quantization-checker-log
# This directory contains all the log files produced by the snpe-quantization-checker. The files a

If histogram generation is specified in the config file, histogram analysis for weights and biases
cd ${Path_to_the_model}/user_specified_dir/hist_analysis_weights
# This directory contains all the png files representing pixelwise data distribution for unquantiz
cd ${Path_to_the_model}/user_specified_dir/hist_analysis_biases
# This directory contains all the png files representing pixelwise data distribution for unquantiz
```

HTML Results Files

Each HTML file contains a summary of the results for each quantization option and for each input file provided.

Note

Snapshot of HTML content has been added for clarity.



Results for quantizer: adjusted using input file: 0#.rawtensor on model: net.prototxt 1 This section includes quantization option name, input filename, model filename and a table with short description of all the metrics.

Legend:

Quantization Checker Name	Description
Symmetricity	Indicates whether the data is symmetric or not.
Quantization Bit-Width Range	Indicates whether the data can be reasonably quantized within the given bit-width range.
Clustering of Unquantized Data	Indicates whether a large number of unique unquantized values are quantized to the same value or not. Currently we do not fail an entire node depending on this result since the influence of the checker is not conclusive.
Minimum/Maximum Difference Value	Indicates how different the min/max values are between the unquantized and dequantized data.
Maximum Absolute Difference	Indicates whether the maximum discrepency between the unquantized and dequantized data is too large or not.
Signal to Quantization Noise Ratio	Indicates whether the signal to quantized noise ratio of the data is too low or not.

Summary of failed nodes that should be inspected: (Total number of nodes analyzed: 3 Total number of failed nodes: 3) 2 Following table includes only the nodes with "Fail" accuracy passes.

Please consult the latest logs for further details on the failures.

			Quantization Checker Pass/Fail						
		Node Type	Symmetricity	Quantization Bit-Width Range	Clustering of Unquantized Data	Minimum/Maximum Difference Value	Maximum Absolute Difference	Signal to Quantization Noise Ratio	Description
Op Name	Node Name								
conv_tanh_comp1_conv0	ReLU6919	Activation	Pass	N/A	N/A	Fail	N/A	N/A	The minimum and/or the maximum values differ widely from the unquantized version. Please consider using a larger bit-width for quantization.
	conv_tanh_comp1_conv0_bias	Bias	N/A	N/A	N/A	Pass	Pass	Fail	The signal to quantization noise ratio is low. Please consider retraining the model using QAT.
	conv_tanh_comp1_conv0_weight	Weight	N/A	N/A	N/A	Pass	Pass	Fail	The signal to quantization noise ratio is low. Please consider retraining the model using QAT.

Results for all nodes: 3 Following table includes contents of all nodes. "N/A" is specified for certain fields when a metric is not applicable for a particular quantization option.

			Quantization Checker Pass/Fail						
		Node Type	Symmetricity	Quantization Bit-Width Range	Clustering of Unquantized Data	Minimum/Maximum Difference Value	Maximum Absolute Difference	Signal to Quantization Noise Ratio	Description
Op Name	Node Name								
conv_tanh_comp1_conv0	ReLU6919	Activation	Pass	N/A	N/A	Fail	N/A	N/A	The minimum and/or the maximum values differ widely from the unquantized version. Please consider using a larger bit-width for quantization.
	conv_tanh_comp1_conv0_bias	Bias	N/A	N/A	N/A	Pass	Pass	Fail	The signal to quantization noise ratio is low. Please consider retraining the model using QAT.
	conv_tanh_comp1_conv0_weight	Weight	N/A	N/A	N/A	Pass	Pass	Fail	The signal to quantization noise ratio is low. Please consider retraining the model using QAT.

CSV Results Files

Each CSV file contains detailed computation results for a specific node type (activation/weight/bias) and quantization option. Each row in the csv file displays the op name, node name, passes accuracy (True/False), computation result (accuracy differences), threshold being used for each algorithm and the algorithm name. Computation results(accuracy differences) format can be somewhat different according to different algorithms/metrics.

The following are a few short notes about the different algorithms and the information each csv row contains:

- **minmax: Indicates the difference between the unquantized minimum and the dequantized minimum value. Correspondingly, indicates the same difference for the maximum unquantized and dequantized value.**

  Example computation result: "min: #VALUE max: #VALUE"

- **maxdiff: Calculates the absolute difference between the unquantized and dequantized data for all data points and displays the maximum value of the result.**

  Example computation result: "#VALUE"

- **sqnr: Calculates the signal to quantization noise ratio between the two tensors of unquantized and dequantized data.**

Example computation result: “#VALUE”

- **data_range_analyzer: Calculates the difference between the maximum and minimum values in a tensor and compares that to the maximum value supported by the bit-width used to determine if the range of values can be reasonably represented by the selected quantization bit width.**

  Example computation result: “unique dec places: #INT_VALUE data range : #VALUE”. Information in the computation results field includes how many unique decimal places we need to express the unquantized data in quantized format and what is the actual data range.

- **data_distribution_analyzer: Calculates the clustering of the data to find whether a large number of unique unquantized values are quantized to the same value or not.**

  Example computation result: “Distribution of pixels above threshold: #VALUE”

- **stats: Calculates some basic statistics on the received data such as the min, max, median, variance, standard deviation, the mode and the skew. The skew is used to indicate how symmetric the data is.**

  Example computation result: skew: #VALUE min: #VALUE max: #VALUE median: #VALUE variance: #VALUE stdDev: #VALUE mode: #VALUE

> **Note**
>
> Snapshot of a CSV file content for weight data for one of the quantization options has been added below.

	A	B	C	D	E	F
conv_tanh_comp1_conv0	conv_tanh_comp1_conv0_weight	TRUE	min: 0.43918341398239136 max: 0.37056002020835876	10	minmax
conv_tanh_comp1_conv0	conv_tanh_comp1_conv0_weight	TRUE	0.439183414	10	maxdiff
conv_tanh_comp1_conv0	conv_tanh_comp1_conv0_weight	FALSE	-2.53E-07	26	sqnr

Separate .csv files are available for activations, weights and biases for each quantization option. The activation related results also include analysis for each input file provided.

Log Result File

The log files contain the following information:

1. All the commands executed as part of the script's run. This will include different runs of the qnn-converter tool with different quantization options
2. Activations Analysis Failures
3. Weights Analysis Failures
4. Biases Analysis Failures

A sample log output looks like below:

```
<====ACTIVATIONS ANALYSIS FAILURES====>

Results for the enhanced quantization:
|          Op Name          | Activation Node | Passes Accuracy |          Accuracy Difference
|  conv_tanh_comp1_conv0    |    ReLU_6919    |      False      | minabs_diff: 0.59 maxabs_diff: 17.
```

where,

1. Op Name : Op Name as expressed in qnn_model.cpp
2. Activation Node : Activation Node Name in the Op
3. Passes Accuracy : Pass if the quantized activation(or weight or bias) meets threshold when compared with values from float32 graph
4. Accuracy Difference : Details about the accuracy per the algorithm used
5. Threshold Used : The threshold used to influence the result of "Passes Accuracy" column
6. Algorithm Used : Metric used to compare actual quantized activations/weights/biases against unquantized float data or analyze the quality of unquantized float data. Metrics can be minmax, maxdiff, sqnr, stats, data_range_analyzer, data_distribution_analyzer.

Repair Mode

This feature adds the ability to 'repair' nodes in a graph that demonstrate poor accuracy due to quantization. The tool is run and identifies nodes that demonstrate poor accuracy due to quantization.

Based on the rule that was violated a corrective action is determined to improve the performance (accuracy). The tool is then rerun using the corrective action decided upon and the nodes evaluated again to see if there is an improvement in accuracy. Determining the performance of nodes is accomplished using the current set of comparison algorithms already used within the quantization checker tool. The current implementation is limited to an increase in the bit-width for the worst performing tensor (either weights, biases or activations) as the corrective action to be taken, i.e., the 'repair'.

Note

The repair mode does not guarantee an improvement in accuracy and is merely an attempt at the easiest change that can be made, thus eliminating one additional debug step for the end user.

The repair mode also supports manual mode by supplying the following argument to the config file:

RUN_IN_REPAIR_MODE type: bool (True/False)

Description: run the [snpe, qnn]-quantization-checker in repair mode. This mode will identify nodes that demonstrate poor accuracy due to quantization and further adjust the quantization encodings to improve the accuracy of identified nodes. The tool is then rerun with the updated encoding values and a report of the accuracy improvement is given upon completion.

Example output from the repair mode:

```
MINMAX: Percentage of Failed Nodes

-------------------------------------------------------------------------------------------------
| Tensor Type | unquantized |         tf          |       tf_cle        |       tf_pcq        |
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
|   Weights   |     N/A     |         0.0         |         0.0         |         0.0         |
-------------------------------------------------------------------------------------------------
|   Biases    |     N/A     |         0.0         |         0.0         |         0.0         |
-------------------------------------------------------------------------------------------------
```

```
| Activations |     N/A     | 0.11634671320535195 | 0.11634671320535195 | 0.11634671320535195 | 0.
-------------------------------------------------------------------------------------------------
```

Troubleshooting Steps

The following steps are intended for end users who are having difficulty with using the tool. These are common areas to investigate and are not comprehensive.

The tool has some requirements/dependencies which originate from the snpe binaries requirements:

- clang 9 must be installed
- a python 3 venv must be used
- the bash shell interpreter must be installed
- **at least one of the following frameworks must be installed depending on which type of model is being analyzed:**
  - Tensorflow
  - Onnx
  - TFLite

Additionally a few checks can be made to ensure the model runs on the first try:

- ensure the path specified to the model to be analyzed is correct
- ensure the path to the output directory exists and is writeable
- use the provided pre-defined config files and fill out the details according to your local setup, i.e., local paths
- failures highlighted by the tool may be acceptable failures depending on the model, it is up to the user to determine whether a failure indicates a problem or not based on the users knowledge of the particular model being analyzed
- python errors are generally due to a version mismatch and the tool will only work with Python 3.6 and above and currently is focused on Python 3.10

qnn-accuracy-evaluator (Beta)

The **qnn-accuracy-evaluator** tool provides a framework to evaluate end-to-end accuracy metrics for a model on a given dataset. In addition, the tool can be used to identify the best quantization options for a model on a given set of inputs.

Dependencies

The QNN Accuracy Evaluator assumes that the platform dependencies and environment setup instructions have been followed as outlined in the Setup page. Certain additional python packages are required by this tool, refer to Optional Python packages.

Usage

User needs to set QNN_SDK_ROOT environment variable to root directory of QNN SDK. The following environment variables might need to be set with appropriate values: QNN_MODEL_ZOO : Path to model zoo. If not set model_zoo base directory path is assumed to present at "/home/model_zoo". Note: This environment variable is required only if the model path supplied is not absolute and relative to the set model zoo path. ADB_PATH : Set the path to the ADB binary. If not set ADB is assumed to be present at "/opt/bin/adb".

There are two modes of usage for the tool - minimal mode and config mode.

Minimal mode option is to perform accuracy analysis on all possible quantization options and rank them in order based on the given comparator when compared against the cpu fp32 outputs.

Config mode supports accuracy analysis of a model on a given dataset by customizing the backends, quantization options and reference inference schemas.

Supported models

The qnn-accuracy-evaluator currently supports ONNX and PyTorch (torchscript) models.

Note: For PyTorch models, it is mandatory to provide the model's input and output node info, wherever applicable, as per QNN's naming convention which can be observed when such models are executed using qnn-pytorch-converter.

Minimal Mode

Minimal mode option is to perform accuracy analysis on all possible quantization options and rank them in order based on the given comparator when compared against the cpu fp32 outputs.

```
qnn-accuracy-evaluator Options

minimal mode options:
    -model MODEL          path to model or model directory
    -target_arch {aarch64-android,x86_64-linux-clang}
                          Target architecture to compile.
    -backend BACKEND      Backend to run the inference.
                          For target_arch x86_64-linux-clang, allowed backends are {htp}
                          For target_arch aarch64-android, allowed backends are {cpu,gpu,dspv69,ds
    -preproc_file PREPROC_FILE
                          Path to a file specifying paths to raw inputs.
    -comparator COMPARATOR
                          comparator to be used.
    -tol_thresh TOL_THRESH
                          Tolerance threshold to be used for the comparator
    -act_bw ACT_BW        [Optional] bitwidth to use for activations. E.g., 8,
                          1.  Default is 8.
    -bias_bw BIAS_BW      [Optional] bitwidth to use for biases. either 8
                          (default) or 32.
    -box_input BOX_INPUT  Path to the json file. Used only with the box
                          comparator

other options:
    -input_info INPUT_INFO INPUT_INFO
                           The name and dimension of all the input buffers to the
                           network specified in the format [input_name comma-
                           separated-dimensions], for example: 'data'
                           1,224,224,3. This option is mandatory for pytorch
                           models in minimal mode.
    -onnx_symbol ONNX_SYMBOL [ONNX_SYMBOL ...]
                           Replace onnx symbols in input/output shapes.Can be
                           passed multiple timesDefault replaced by 1. e.g
                           __unk_200:1
```

```
-device_id DEVICE_ID  Target device-id/device-serial to be provided
-work_dir WORK_DIR    working directory path. default is ./qacc_temp
-silent               Run in silent mode
-debug                Set logging level to DEBUG within the tool
```

Note

The HTP backend emulation on the host is not guaranteed to generate the same output tensor values as when running on an HTP on a Qualcomm SoC.
For HTP backend execution on Android device, user must specify the backend along with the version such as "dspv69/v73/v75" and target-arch as "aarch64-android" in the CLI command.

Sample Command

```
qnn-accuracy-evaluator -model ${QNN_MODEL_ZOO}/onnx-cnns_mobilenet/Source_model/model.onnx
                       -preproc_file ${QNN_MODEL_ZOO}/onnx-cnns_mobilenet/inputs/input_list.txt
                       -backend htp
                       -target_arch x86_64-linux-clang
                       -comparator abs
                       -tol_thresh 0.1
```

Results

The tool displays a table with quantization options ordered by output match based on the selected comparator and also generates a csv file with the same data. Comparator column shows output match percentage/value based on the selected comparator. Quant params column displays the quantization params used for that run. Other columns also show backend, runtime/compile params used. The information is also stored in a csv file at *{work_dir}/metrics-info.csv*.

The quantization option combinations that are run in minimal mode:

```
param_quantizer:  tf | symmetric | enhanced | adjusted
algorithms: default | cle
use_per_channel_quantization: True | False
use_per_row_quantization: True | False
```

Each quantization option work directory is stored at *{work_dir}/infer/schema{i}_qnn_{quant_option}*. QNN IR files are stored at *{work_dir}/infer/schema{i}_qnn_{quant_option}/qnn_ir* and outputs are stored at *{work_dir}/infer/schema{i}_qnn_{quant_option}/Result_{i}*.

Note

Snapshot of console log has been added for clarity.

Inference schema	Status	Precision	Backend	Backend extensions	CLI Params	Metrics	Comparator
schema1_qnn_symmetric_pcq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:default param_quantizer:symmetric use_per_channel_quantization:True use_per_row_quantization:False	-	Avg Match (all outputs): 88.011 (avg) ArgMax_0: 97.037 (avg) softmax_tensor_0: 78.985
schema1_qnn_symmetric_cle_pcq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:cle param_quantizer:symmetric use_per_channel_quantization:True use_per_row_quantization:False	-	Avg Match (all outputs): 88.011 (avg) ArgMax_0: 97.037 (avg) softmax_tensor_0: 78.985
schema1_qnn_enhanced_pcq_prq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:default param_quantizer:enhanced use_per_channel_quantization:True use_per_row_quantization:True	-	Avg Match (all outputs): 87.857 (avg) ArgMax_0: 94.341 (avg) softmax_tensor_0: 81.373
schema1_qnn_enhanced_cle_pcq_prq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:cle param_quantizer:enhanced use_per_channel_quantization:True use_per_row_quantization:True	-	Avg Match (all outputs): 87.857 (avg) ArgMax_0: 94.341 (avg) softmax_tensor_0: 81.373
schema1_qnn_enhanced_pcq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:default param_quantizer:enhanced use_per_channel_quantization:True use_per_row_quantization:False	-	Avg Match (all outputs): 87.764 (avg) ArgMax_0: 93.842 (avg) softmax_tensor_0: 81.686
schema1_qnn_enhanced_cle_pcq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:cle param_quantizer:enhanced use_per_channel_quantization:True use_per_row_quantization:False	-	Avg Match (all outputs): 87.764 (avg) ArgMax_0: 93.842 (avg) softmax_tensor_0: 81.686
schema1_qnn_tf_pcq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:default param_quantizer:tf use_per_channel_quantization:True use_per_row_quantization:False	-	Avg Match (all outputs): 87.2 (avg) ArgMax_0: 95.092 (avg) softmax_tensor_0: 79.307
schema1_qnn_adjusted_pcq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:default param_quantizer:adjusted use_per_channel_quantization:True use_per_row_quantization:False	-	Avg Match (all outputs): 87.2 (avg) ArgMax_0: 95.092 (avg) softmax_tensor_0: 79.307
schema1_qnn_tf_cle_pcq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:cle param_quantizer:tf use_per_channel_quantization:True use_per_row_quantization:False	-	Avg Match (all outputs): 87.2 (avg) ArgMax_0: 95.092 (avg) softmax_tensor_0: 79.307

> **Note**
>
> Snapshot of csv file has been added for clarity.

Inference schema	Status	Precision	Backend	Backend extensions	CLI Params	Metrics	Comparator
schema1_qnn_symmetric_pcq	Success	quant	htp x86_64-linux-clang	rpc_control_latency:100 vtcm_mb:4	Converter params: algorithms:default param_quantizer:symmetric use_per_channel_quantization:True use_per_row_quantization:False	-	Avg Match (all outputs): 88.011 (avg) ArgMax_0: 97.037 (avg) softmax_tensor_0: 78.985

Config Mode

Config mode supports accuracy analysis of a model on a given dataset by customizing the backends, quantization options and reference inference schemas. Sample config files can be found at *${QNN_SDK_ROOT}/lib/python/qti/aisw/accuracy_evaluator/configs/samples/model_configs*.

The high-level structure of a model config is shown below:

```
model
    info
    globals
    dataset
    processing
        preprocessing
        postprocessing
    inference-engine
    evaluator
```

User needs to provide all dataset information under the dataset section in the model config file, failing which, an error is thrown. An example of this is shown below:

```
dataset:
    name: COCO2014
    path: '/home/ml-datasets/COCO/2014/'
    inputlist_file: inputlist.txt
    calibration:
```

```
        type: index
        file: calibration-index.txt
```

Details of the dataset fields is as follows:

Field	Description
name	Name of the dataset
path	Base directory of the dataset files
inputlist_file	Text file containing all the pre-processed input files relative to the path field, one input per line. For models having multiple inputs, the inputs in each line have to be comma separated
calibration	**Specifies the calibration file type to be used with quantization. Optional. It has following params** • **type: Type can be 'index', 'raw' or 'dataset'** ◦ index - File provided contains the indexes to be picked from inputlist for calibration ◦ raw - File provided contains entries of pre-processed raw files for calibration ◦ dataset - File provided contains images processed separately and passed to inference • file: pre-processed calibration file name

The inference engine is used to run the model on multiple inference schemas. A sample inference engine section is shown below, followed by the description of the different configurable entries in the inference section.

```
inference-engine:
    model_path: MLPerfModels/ResNetV1.5/modelFiles/ONNX/resnet50_v1.onnx
    simplify_model : True
```

```
inference_schemas:
    - inference_schema:
        name: qnn
        backend: htp
        target_arch: x86_64-linux-clang
        precision: quant
        tag: htp_int8
        converter_params:
            param_quantizer: tf | symmetric | enhanced | adjusted
        contextbin_params:
            profiling_level: basic
        netrun_params:
            use_native_input_files: True
        backend_extensions:
            num_cores: 8
            soc_id: 0
inputs_info:
    - input_tensor_0:
        type: float32
        shape: ["*", 3, 224, 224]
outputs_info:
    - ArgMax_0:
        type: int64
        shape: ["*"]
    - softmax_tensor_0:
        type: float32
        shape: ["*", 1001]
```

Details of each configurable entry is given below:

Field	Description
model_path	Absolute or relative path of the model. If the path is relative, it would be taken relative to MODEL_ZOO_PATH, if set, else default /home/model_zoo
simplify_model	Flag to enable or disable model simplification for ONNX models. By default, this flag is set to True and the model would be simplified. Note: Model simplification would be skipped for models having custom operators or for inference schemas having quantization_overrides parameter configured.
inference_schemas	**List of inference schemas to perform inference on. Each inference_schema has further entries as the following:**

	• name - Name of the inference schema. Options: qnn, onnxrt, tensorflow, torchscript, tensorflow-session • precision - Precision to run inference on. Options: fp32, fp16, int8/quant • target_arch - Target architecture to run inference on. Options: x86_64-linux-clang, aarch64-android • backend - Backend to run inference on. Allowed backends for x86_64-linux-clang: {cpu,htp} and aarch64: {cpu,gpu,dspv69,dspv73,dspv75}. • tag - Tag unique for a inference schema • converter_params - Params to be passed as arguments to converter • contextbin_params - Params to be passed as arguments to context-binary-generator • netrun_params - Params to be passed as arguments to net-run • backend_extensions - Params to be passed as backend extensions config file to context-binary-generator and net-run
input_info	**Information about each model input. Requires following params in the given order** • type - numpy type (float16, float32, float64, int8, int16, int32/int, int64) • shape - list of dimensions
output_info	**Information about each model output. Requires following params in the given order** • type - numpy type (float16, float32, float64, int8, int16, int32/int, int64) • shape - list of dimensions

Note

For HTP backend emulation on host, set backend to “htp” and target_arch as “x86_64-linux-clang” in the config file.

For HTP backend execution on Android device, user must specify the backend along with the version such as dspv69 / dspv73 / dspv75 and target_arch as “aarch64-android” in the config file.

The evaluator section provides information about the comparator being used to compare the inference outputs, in case of multiple inference schemas. A sample evaluator section is shown below, followed by the description of the different configurable entries in the section.

```
evaluator:
    comparator:
        enabled: True
        fetch-top: 1
        type: avg
        tol: 0.001
```

Details of each configurable entry is given below:

Field	Description
comparator	**If multiple inference schemas are provided, compare the inference outputs with the reference inference schema. If reference inference schema is not defined, the first inference schema is considered the reference. If only one inference schema is defined, comparator is not executed. Following params need to be provided** • enabled - By default enabled (True) • fetch-top - Fetch top ‘n’ highest mismatching outputs, Default 1 • type - One of in-built comparators (abs, cos, topk, avg, l1norm, l2norm). Default avg • tol - Tolerance value. Default 0.001

Command line options available for config mode are as follows:

```
qnn-acc-evaluator options

required options:
    -config CONFIG        path to model config yaml

    pipeline options:
    -preproc_file PREPROC_FILE
                          preprocessed input file, overrides inputfile provided in model config
    -calib_file CALIB_FILE
                          calibration input file, overrides calibration file provided in model c

other options:
    -device_id DEVICE_ID    Target device id to be provided
    -work_dir WORK_DIR      working directory path. default is ./qacc_temp
    -silent                 Run in silent mode
    -debug                  Set logging level to DEBUG within the tool
    -inference_schema INFERENCE_SCHEMA
                          run only on this inference schema
    -inference_schema_tag INFERENCE_SCHEMA_TAG
                          run only this inference schema tag
    -batchsize BATCHSIZE    overrides batchsize provided in model config
    -onnx_symbol ONNX_SYMBOL [ONNX_SYMBOL ...]
                          Replace onnx symbols in input/output shapes.Can be
                          passed multiple timesDefault replaced by 1. e.g
                          __unk_200:1
    -set_global SET_GLOBAL [SET_GLOBAL ...]
                          Replace global symbols with given value.Can be passed
                          multiple times. e.g <symbol>:2
    -use_memory_plugins     Flag to enable memory plugins.
```

Note

Users can accelerate their evaluations using memory plugins to minimize unnecessary reading and writing of data during evaluation by passing the `-use_memory_plugins` flag to the evaluator command.

Sample Command

```
qnn-accuracy-evaluator -config {path to configs}/qnn_resnet50_config.yaml
```

Results

Refer to the minimal mode section

Config file options

```
- inference_schema:
    name: qnn
    target_arch: x86_64-linux-clang
    backend: cpu
    precision: fp32
    tag: qnn_cpu_x86

- inference_schema:
    name: qnn
    target_arch: aarch64-android
    backend: cpu
    precision: fp32
    tag: qnn_cpu_android

- inference_schema:
    name: qnn
    target_arch: aarch64-android
    backend: gpu
    precision: fp32
    tag: qnn_gpu_android

- inference_schema:
    name: qnn
    target_arch: x86_64-linux-clang
    backend: htp
    precision: quant
    tag: htp_int8
    backend_extensions:
        vtcm_mb: 4
        rpc_control_latency: 100
    converter_params:
```

```
        param_quantizer: tf | symmetric | enhanced | adjusted
        act_quantizer: tf | symmetric
        algorithms: default | cle
        use_per_channel_quantization: True | False
        use_per_row_quantization: True | False
        quantization_overrides: "path to the ext quant json"
        act_bw: 8 | 16
        bias_bw: 8 | 32
        weight_bw: 8

- inference_schema:
    name: qnn
    target_arch: aarch64-android
    backend: dspv69/dspv73/dspv75
    precision: quant
    tag: htp_int8
    backend_extensions:
        vtcm_mb: 4
        rpc_control_latency: 100
    converter_params:
        param_quantizer: tf | symmetric | enhanced | adjusted
        act_quantizer: tf | symmetric
        algorithms: default | cle
        use_per_channel_quantization: True | False
        use_per_row_quantization: True | False
        quantization_overrides: "path to the ext quant json"
        act_bw: 8 | 16
        bias_bw: 8 | 32
        weight_bw: 8
```

Plugins

Plugins are Python classes used to implement different stages of the inference pipeline, such as dataset handling, preprocessing, postprocessing, and metrics logic.

Dataset and **pre-processing** plugins perform transformations to the input before they are passed to inference.

Post-processing plugins transform inference outputs.

Metric plugins analyze inference outputs to assess their accuracy

Sample plugins are provided in the SDK at *${QNN_SDK_ROOT}/lib/python/qti/aisw/accuracy_evaluator/plugins*.

Users can implement their own plugins (custom plugins) to meet their specific requirements. To include custom plugins, export the *CUSTOM_PLUGIN_PATH* environment variable pointing to the location of the custom plugin(s), so that they are also included while registering the plugin(s).

```
export CUSTOM_PLUGIN_PATH=/path/to/custom/plugins/directory
```

In the model configuration file, plugins are defined as a transformation chain, as shown below:

```
transformations:
    - plugin:
        name: resize
        params:
            dims: 416,416
            channel_order: RGB
            type: letterbox

    - plugin:
        name: normalize
    - plugin:
        name: convert_nchw
```

Plugins required for dataset transformation are configured in the dataset section as shown below.

```
dataset:
    name: ILSVRC2012
    path: '/home/ml-datasets/imageNet/'
    inputlist_file: inputlist.txt
    annotation_file: ground_truth.txt
    calibration:
```

```
        type: dataset
        file: calibration.txt
    transformations:
        - plugin:
              name: filter_dataset
              params:
                  random: False
                  max_inputs: -1
                  max_calib: -1
```

The preprocessing and postprocessing plugins that the user wishes to use are configured in the processing section as shown below:

```
processing:
    preprocessing:
        transformations:
            - plugin:
                  name: resize
                  params:
                      dims: 416,416
                      channel_order: RGB
                      type: letterbox

            - plugin:
                  name: normalize

    postprocessing:
        squash_results: True
        transformations:
            - plugin:
                  name: object_detection
                  params:
                      dims: 416,416
                      type: letterbox
                      dtypes: [float32, float32, float32, float32]
```

Metric calculation plugins are configured in the evaluator section as shown below.

```
evaluator:
    metrics:
        - plugin:
            name: topk
            params:
                kval: 1,5
                softmax_index: 1
                round: 7
                label_offset: 1
```

Plugins that need to be executed for a pipeline stage are listed under 'transformations' (except metric plugins) and preceded by the 'plugin' keyword. The following table lists details of each configurable entry for a plugin.

Field	Description
name	Name of the plugin
params	Parameters expected and required by the plugin

A complete list of all plugins and their parameters can be found at Accuracy Evaluator Plugins

Comparators

Following are the comparators that can be used to compare the outputs. Some of the comparators output percentage match between the two tensors and some output the absolute value, corresponding to the selected comparator.

1. **abs** - Percentage match between the two tensors based on the relative tolerance threshold value
2. **cos** - Percentage match between the two tensors based on the Cosine Similarity score
3. **topk** - Percentage match between the two tensors based on the topk match between the two tensors
4. **avg** - Percentage match between the two tensors based on the average difference between the two tensors

5. **l1norm** - Percentage match between the two tensors based on the L1 Norm of the diff
6. **l2norm** - Percentage match between the two tensors based on the L2 Norm of the diff
7. **std** - Percentage match between the two tensors based on the standard deviation difference
8. **rme** - Percentage match between the two tensors based on the RMSE between the tensors
9. **snr** - Signal to Noise Ratio between the two tensors
10. **maxerror** - max error value between the two tensors
11. **kld** - KL Divergence value between the two tensors
12. **pixelbypixel** - pixel by pixel plot difference between the two tensors. For each input i, plot is saved at *{work_dir}/{schema}/Result_{i}*
13. **box** - The box verifier requires the *–box_input* parameter which accepts the filename of a json file with the following format:

```
{"box":"Result_0/detection_boxes_0.raw",
"class":"Result_0/detection_classes_0.raw",
"score":"Result_0/detection_scores_0.raw"}
```

Directory of models

In Minimal mode, User can also pass a directory of models. The directory structure is expected to be in the following format:

```
model_dir/
  |__model1/
  |      |__Source_model/
  |      |       |__model.onnx
  |      |__inputs/
  |      |       |__data/images.raw
  |      |       |__input_list.txt
  |__model2/
  |      |__Source_model/
  |      |       |__model.onnx
  |      |__inputs/
  |      |       |__data/images.raw
  |      |       |__input_list.txt
```

The output directory for each model will be available at *{work_dir}/{model_dir}*

qnn-architecture-checker (Beta)

Architecture Checker is a tool made for models running with HTP backend, including quantized 8-bit, quantized 16-bit and FP16 models. It outputs a list of issues in the model that keep the model from getting better performance while running on the HTP backend. Architecture checker tool can be invoked with the modifier feature which will apply the recommended modifications for these issues. This will help in visualizing the changes that can be applied to the model to make it a better fit on the HTP backend.

```
X86-Linux/ WSL Usage:
$ qnn-architecture-checker -i <path>/model.json
                           -b <optional_path>/model.bin
                           -o <optional_output_path>
                           -m <optional_modifier_argument>


X86-Windows/ Windows on Snapdragon Usage:
$ python qnn-architecture-checker -i <path>/model.json
                           -b <optional_path>/model.bin
                           -o <optional_output_path>
                           -m <optional_modifier_argument>


 required arguments:
     -i INPUT_JSON, --input_json INPUT_JSON
                             Path to json file

 optional arguments:
     -b BIN, --bin BIN
                     Path to a bin file
     -o OUTPUT_PATH, --output_path OUTPUT_PATH
                     Path where the output csv should be saved. If not specified, the output csv w
     -m MODIFY, --modify MODIFY
                     The query to select the modifications to apply.
                         --modify or --modify show - To see all the possible modifications. Displa
                         --modify all - To apply all the possible modifications found for the mode
                         --modify apply=rule_name1,rule_name2 - To apply modifications for specifi
```

Note:

If running on a quantized model, the quantized model generated with one input image is good enough to satisfy the quantization requirement to have the tool run properly.
QNN_SDK_ROOT environment variable must be configured before running the tool.

Deprecation Note:
The option of enabling architecture checker by passing '–arch_checker' in each converter listed above will be deprecated. E.g: Running qnn-tflite-converter -i <path>/model.tflite -d <network_input_name> <dims> -o <optional_output_path> -p <optional_package_name> –arch_checker will be deprecated.
To enable the Architecture checker, run the converter tool without passing the '–arch_checker' argument, then run the qnn-architecture-checker command to see the architecture checker output.
The usage of "–modify" is only supported with the qnn-architecture-checker command.

The output is a csv file and will be saved as <optional_output_path>/<model_name>_architecture_checker.csv. An example output is shown below:

	Graph/Node_name	Issue	Recommendation	Type	Input_tensor_name: [dims]	O
1	Graph	This model uses 16-bit activation data. 16-bit activation data takes twice the amount of memory than 8-bit activation data does.	Try to use a smaller datatype to get better performance. E.g., 8-bit	N/A	N/A	
2	Node_name_1	The number of channels in the input/output tensor of this	Try increasing the number of channels in the input/output tensor to 32 or greater to get	Conv2d	input_1:[1, 250, 250, 3], __param_1: [5, 5, 3, 32], convolution_0_bias: [32]	

		convolution node is low (smaller than 32).	better performance.			

How to read the example output csv?

Row 1: This is an issue on the graph, the graph is using 16-bit activation data, as said in the recommendation, changing the activation from 16 bit to 8 bit gives better performance.

Row 2: The issue is on the node with QNN node name as "Node_name_1". This node has three inputs: input_1, __param_1 and convolution_0_bias where the dimensions are [1, 250, 250, 3], [5, 5, 3, 32] and [32] respectively. This node has one output with QNN tensor name output_1 and the dimension of this tensor is [1, 123, 123, 32]. The type of this node is Conv2d. The previous/next node names and the full set of additional node parameters available in the Parameters column that can be used to locate the node inside the original model. The issue for this node is the channel of the input tensor is low, as the channel is smaller than 32, would recommend to increase the channel to at least 32 to get better performance on HTP backend. Currently the input dimension is [1, 250, 250, 3] and ideally have that to be [1, x, x, 32]. The Modification and Modification_info columns provide details about the modifications applied to the node. If the Architecture Checker is not invoked with modifier or if there aren't any modifications applicable, then these value will be N/A.

Is the QNN node/tensor name the same in the original model?

It is not the same but should be similar. There is naming sanitization in converter in order to meet the QNN naming standard. The input tensor, output tensor, previous node, next node and all the additional parameters are avaliable in the output csv file to help locate the correct node inside the original model.

Sample Command

```
qnn-architecture-checker --input_json ./model_net.json
                         --bin ./model.bin
                         --output_path ./archCheckerOutput
```

Architecture Checker - Model Modifier

For appying modifications to the model, the Architecture Checker can be invoked with "–modify" or "–modify show" which will display a list of possible modifications. In this case, the Architecture Checker tool will only show the rule names and modification detail. It will run without making any changes to the model and generate the csv output. Using the rule names from the above run, the Architecture Checker can be invoked with "–modify all" or "–modify apply=rule_name1,rule_name2". In this case, the rule specific changes will be applied to the model and the changes can be viewed in the updated model json. Additionally, the output csv will also contain information related to the modifications.

Consider the below csv output generated after applying "–modify apply=elwisediv" modification on an example model.

	Graph/Node_name	Issue	Recommendation	Type	Input_t
1	Node_name_1	ElementWiseDivide usually has poor performance compared to ElementWiseMultiply.	Try replacing ElementWiseDivide with ElementWiseMultiply using the reciprocal value to get better performance.	Eltwise_Binary	input_ 6], ir
2	Node_name_2	The number of channels in the input/output tensor of this convolution node is low (smaller than 32).	Try increasing the number of channels in the input/output tensor to 32 or greater to get better performance.	Conv2d	input 250, 3] [5, convolu

How to read the example output csv?

Row 1: The issue on the node with QNN node name as "Node_name_1" is that it has element wise divide which gives a poor performance as compared to elementwise multipy. After invoking architecture checker with "–modify apply=elwisediv", the modifications have been successfully applied i.e. the element wise divide is replaced by element wise multiply with a reciprocal value. This information is available in the Modification and Modification_info columns.

Row 2: The issue on the node with QNN node name as “Node_name_2” is that the node has input tensor with number of channels less than 32. Its recommended to increase the number of channels to 32 or greater for better performance. For this issue, the modification through the tool is not applicable hence the Modification and Modification_info columns are N/A.
After modifying the model, the above run will generate updated model.cpp, model_net.json and/or model.bin along with the csv output. Running the Architecture Checker on the updated model json will no longer show the element wise divide issue on Node_Name_1.

Following are the commands to invoke Architecture Checker with Modifier to display list of modifications:

Sample Command

```
qnn-architecture-checker --input_json ./model_net.json
                         --bin ./model.bin
                         --output_path ./archCheckerOutput
                         --modify
```

Sample Command

```
qnn-architecture-checker --input_json ./model_net.json
                         --bin ./model.bin
                         --output_path ./archCheckerOutput
                         --modify show
```

Following are the commands to apply the modifications either on all possible modifications or specific rules:

Sample Command

```
qnn-architecture-checker --input_json ./model_net.json
                         --bin ./model.bin
                         --output_path ./archCheckerOutput
                         --modify all
```

Sample Command

```
qnn-architecture-checker --input_json ./model_net.json
                         --bin ./model.bin
                         --output_path ./archCheckerOutput
                         --modify apply=prelu,elwisediv
```

Note:
The Architecture Checker with modifier is an enchancement to help visualize the changes that can be applied on the model to better fit it on the HTP. To see the actual performance improvements, the model may require retraining/redesigning.

qnn-accuracy-debugger (Beta)

Dependencies

The Accuracy Debugger depends on the setup outlined in Setup. In particular, the following are required:

1. Platform dependencies are need to be met as per Platform Dependencies
2. The desired ML frameworks need to be installed. Accuracy debugger is verified to work with the ML framework versions mentioned at Environment Setup

The following environment variables are used inside this guide (User may change the following path depending on their needs):

1. RESOURCESPATH = {Path to the directory where all models and input files reside}
2. PROJECTREPOPATH = {Path to your accuracy debugger project directory}

Supported models

The qnn-accuracy-debugger currently supports ONNX, TFLite, and Tensorflow 1.x models. Pytorch models are supported only in oneshot-layerwise debugging algorithm of tool.

Overview

The **accuracy-debugger** tool finds inaccuracies in a neural-network at the layer level. The tool compares the golden outputs produced by running a model through a specific ML framework (ie. Tensorflow, Onnx, TFlite) with the results produced by running the same model through Qualcomm's QNN Inference Engine. The inference engine can be run on a variety of computing mediums including GPU, CPU and DSP.

The following features are available in Accuracy Debugger. Each feature can be run with its corresponding option; for example, `qnn-accuracy-debugger --{option}` .

1. **qnn-accuracy-debugger —framework_diagnosis** This feature uses a ML framework e.g. tensorflow, tflite or onnx, to run the model to get intermediate outputs.
2. **qnn-accuracy-debugger —inference_engine** This feature uses the QNN engine to run a model to retrieve intermediate outputs.
3. **qnn-accuracy-debugger —verification** This feature compares the output generated by the framework diagnosis and inference engine features using verifiers such as CosineSimilarity, RtolAtol, etc.
4. **qnn-accuracy-debugger –compare_encodings** This feature extracts encodings from a given QNN net JSON file, compares them with the given AIMET encodings, and outputs an Excel sheet highlighting mismatches.
5. **qnn-accuracy-debugger –tensor_inspection** This feature compares given target outputs with reference outputs.

Tip:

- You can use –help after the bin commands to see what other options (required or optional) you can add.
- If no option is provided, Accuracy Debugger runs framework_diagnosis, inference_engine, and verification sequentially.

Below are the instructons for running the Accuracy Debugger:

Framework Diagnosis

The Framework Diagnosis feature is designed to run models with different machine learning frameworks (e.g. Tensorflow, etc). A selected model is run with a specific ML framework. Golden outputs are produced for future comparison with inference results from the Inference Engine step.

Usage

```
usage: qnn-accuracy-debugger --framework_diagnosis [-h]
                             -f FRAMEWORK [FRAMEWORK ...]
                             -m MODEL_PATH
                             -i INPUT_TENSOR [INPUT_TENSOR ...]
                             -o OUTPUT_TENSOR
                             [-w WORKING_DIR]
                             [--output_dirname OUTPUT_DIRNAME]
                             [-v]
                             [--disable_graph_optimization]
                             [--onnx_custom_op_lib ONNX_CUSTOM_OP_LIB]

Script to generate intermediate tensors from an ML Framework.

optional arguments:
     -h, --help            show this help message and exit

required arguments:
     -f FRAMEWORK [FRAMEWORK ...], --framework FRAMEWORK [FRAMEWORK ...]
                           Framework type and version, version is optional. Currently
                           supported frameworks are ["tensorflow","onnx","tflite"] case
                           insensitive but spelling sensitive
     -m MODEL_PATH, --model_path MODEL_PATH
                           Path to the model file(s).
```

```
    -i INPUT_TENSOR [INPUT_TENSOR ...], --input_tensor INPUT_TENSOR [INPUT_TENSOR ...]
                          The name, dimensions, raw data, and optionally data
                          type of the network input tensor(s) specifiedin the
                          format "input_name" comma-separated-dimensions path-
                          to-raw-file, for example: "data" 1,224,224,3 data.raw
                          float32. Note that the quotes should always be
                          included in order to handle special characters,
                          spaces, etc. For multiple inputs specify multiple
                          --input_tensor on the command line like:
                          --input_tensor "data1" 1,224,224,3 data1.raw
                          --input_tensor "data2" 1,50,100,3 data2.raw float32.
    -o OUTPUT_TENSOR, --output_tensor OUTPUT_TENSOR
                          Name of the graph's specified output tensor(s).

    optional arguments:
    -w WORKING_DIR, --working_dir WORKING_DIR
                          Working directory for the framework_diagnosis to store
                          temporary files. Creates a new directory if the
                          specified working directory does not exist
    --output_dirname OUTPUT_DIRNAME
                          output directory name for the framework_diagnosis to
                          store temporary files under
                          <working_dir>/framework_diagnosis. Creates a new
                          directory if the specified working directory does not
                          exist
    -v, --verbose         Verbose printing
    --disable_graph_optimization
                          Disables basic model optimization
    --onnx_custom_op_lib ONNX_CUSTOM_OP_LIB
                          path to onnx custom operator library

Please note: All the command line arguments should either be provided through command line or thro
```

Sample Commands

```
qnn-accuracy-debugger \
    --framework_diagnosis \
    --framework tensorflow \
    --model_path $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen.pb \
    --input_tensor "input:0" 1,299,299,3 $RESOURCESPATH/samples/InceptionV3Model/data/chairs.raw \
    --output_tensor InceptionV3/Predictions/Reshape_1:0
```

```
qnn-accuracy-debugger \
    --framework_diagnosis \
    --framework onnx \
    --model_path $RESOURCESPATH/samples/dlv3onnx/dlv3plus_mbnet_513-513_op9_mod_basic.onnx \
    --input_tensor Input 1,3,513,513 $RESOURCESPATH/samples/dlv3onnx/data/00000_1_3_513_513.raw \
    --output_tensor Output

To run model with custom operator:
qnn-accuracy-debugger \
    --framework_diagnosis \
    --framework onnx \
    -input_tensor "image" 1,3,640,640 $RESOURCESPATH/models/yolov3/batched-inp-107-0.raw \
    --model_path $RESOURCESPATH/models/yolov3/yolov3_640_640_with_abp_qnms.onnx \
    --output_tensor detection_boxes \
    --onnx_custom_op_lib $RESOURCESPATH/models/libCustomQnmsYoloOrt.so
```

TIP:

- a working_directory, if not otherwise specified, is generated from wherever you are calling the script from; it is recommended to call all scripts from the same directory so all your outputs and results are stored under the same directory without having outputs everywhere
- for tensorflow it is sometimes necessary to add the :0 after the input and output node name to signify the index of the node. Notice the :0 is dropped for onnx models.

Output

The program also creates a directory named *latest* in *working_directory/framework_diagnosis* which is symbolically linked to the most recently generated directory. In the example below, *latest* will have data that is symlinked to the data in the most recent directory *YYYY-MM-DD_HH:mm:ss*. Users may choose to override the directory name by passing it to –output_dirname (i.e. –output_dirname myTest1Ouput).

The *float data* produced by the **Framework Diagnosis** step offers precise reference material for the **Verification** component to diagnose the accuracy of the network generated by the **Inference Engine**. Unless a path is otherwise specified, the Accuracy Debugger will create directories within the *working_directory/framework_diagnosis* directory found in the current working directory. The

directories will be named with the date and time of the program's execution, and contain tensor data. Depending on the tensor naming convention of the model, there may be numerous sub-directories within the new directory. This occurs when tensor names include a slash "/". For example, for the tensor names 'inception_3a/1x1/bn/sc', 'inception_3a/1x1/bn/sc_internal' and 'inception_3a/1x1/bn', subdirectories will be generated.

The figure above shows a sample output from a framework_diagnosis run. InceptionV3 and Logits contain the outputs of each layer before the last layer. Each output directory contains the .raw files corresponding to each node. Every raw file that can be seen is the output of an operation. The outputs of the final layer are saved inside the Predictions directory. The file framework_diagnosis_options.json contains all the options used to run this feature.

Inference Engine

The Inference Engine feature is designed to find the outputs for a QNN model. The output produced by this step can be compared with the golden outputs produced by the framework diagnosis step.

Usage

```
usage: qnn-accuracy-debugger --inference_engine [-h]
                                    -p ENGINE_PATH
                                    -l INPUT_LIST
                                    -r {cpu,gpu,dsp,dspv65,dspv66,dspv68,dspv69,dspv73,htp}
```

```
-a {aarch64-android,x86_64-linux-clang}
[--stage {source,converted,compiled}]
[-i INPUT_TENSOR [INPUT_TENSOR ...]]
[-o OUTPUT_TENSOR] [-m MODEL_PATH]
[-f FRAMEWORK [FRAMEWORK ...]]
[-qmcpp QNN_MODEL_CPP_PATH]
[-qmbin QNN_MODEL_BIN_PATH]
[-qmb QNN_MODEL_BINARY_PATH]
[--deviceId DEVICEID] [-v]
[--host_device {x86}] [-w WORKING_DIR]
[--output_dirname OUTPUT_DIRNAME]
[--engine_version ENGINE_VERSION]
[--debug_mode_off]
[--print_version PRINT_VERSION]
[--offline_prepare] [-bbw {8,32}]
[-abw {8,16}]
[--golden_dir_for_mapping GOLDEN_DIR_FOR_MAPPING]
[-wbw {8}] [--lib_name LIB_NAME]
[-bd BINARIES_DIR] [-qmn MODEL_NAME]
[-pq {tf,enhanced,adjusted,symmetric}]
[-qo QUANTIZATION_OVERRIDES]
[--act_quantizer {tf,enhanced,adjusted,symmetric}]
[--algorithms ALGORITHMS]
[--ignore_encodings]
[--per_channel_quantization]
[-idt {float,native}]
[-odt {float_only,native_only,float_and_native}]
[--profiling_level {basic,detailed}]
[--perf_profile {low_balanced,balanced,high_performance,sustain
[--log_level {error,warn,info,debug,verbose}]
[--qnn_model_net_json QNN_MODEL_NET_JSON]
[--qnn_netrun_config_file QNN_NETRUN_CONFIG_FILE]
[--extra_converter_args EXTRA_CONVERTER_ARGS]
[--extra_runtime_args EXTRA_RUNTIME_ARGS]
[--compiler_config COMPILER_CONFIG]
[--precision {int8,fp16}]
[--add_layer_outputs ADD_LAYER_OUTPUTS]
[--add_layer_types ADD_LAYER_TYPES]
[--skip_layer_types SKIP_LAYER_TYPES]
[--skip_layer_outputs SKIP_LAYER_OUTPUTS]
```

Script to run QNN inference engine.

```
optional arguments:
     -h, --help            show this help message and exit

Core Arguments:
     --stage {source,converted,compiled}
                           Specifies the starting stage in the Accuracy Debugger
                           pipeline.
                           Source: starting with source framework model [default].
                           Converted: starting with model.cpp and .bin files.
                           Compiled: starting with a model's .so binary.
     -r {cpu,gpu,dsp,dspv65,dspv66,dspv68,dspv69,dspv73,htp}, --runtime {cpu,gpu,dsp,dspv65,dspv66
                           Runtime to be used.
                           Use HTP runtime for emulation on x86 host.
     -a {aarch64-android,x86_64-linux-clang}, --architecture {aarch64-android,x86_64-linux-clang}
                           Name of the architecture to use for inference engine.
     -l INPUT_LIST, --input_list INPUT_LIST
                           Path to the input list text.

Arguments required for SOURCE stage:
     -i INPUT_TENSOR [INPUT_TENSOR ...], --input_tensor INPUT_TENSOR [INPUT_TENSOR ...]
                           The name, dimension, and raw data of the network input
                           tensor(s) specified in the format "input_name" comma-
                           separated-dimensions path-to-raw-file, for example:
                           "data" 1,224,224,3 data.raw. Note that the quotes
                           should always be included in order to handle special
                           characters, spaces, etc. For multiple inputs specify
                           multiple --input_tensor on the command line like:
                           --input_tensor "data1" 1,224,224,3 data1.raw
                           --input_tensor "data2" 1,50,100,3 data2.raw.
     -o OUTPUT_TENSOR, --output_tensor OUTPUT_TENSOR
                           Name of the graph's output tensor(s).
     -m MODEL_PATH, --model_path MODEL_PATH
                           Path to the model file(s).
     -f FRAMEWORK [FRAMEWORK ...], --framework FRAMEWORK [FRAMEWORK ...]
                           Framework type to be used, followed optionally by
                           framework version.

Arguments required for CONVERTED stage:
     -qmcpp QNN_MODEL_CPP_PATH, --qnn_model_cpp_path QNN_MODEL_CPP_PATH
                           Path to the qnn model .cpp file
     -qmbin QNN_MODEL_BIN_PATH, --qnn_model_bin_path QNN_MODEL_BIN_PATH
                           Path to the qnn model .bin file

Arguments required for COMPILED stage:
     -qmb QNN_MODEL_BINARY_PATH, --qnn_model_binary_path QNN_MODEL_BINARY_PATH
```

```
                        Path to the qnn model .so binary.

Optional Arguments:
     --deviceId DEVICEID   The serial number of the device to use. If not
                           available, the first in a list of queried devices will
                           be used for validation.
     -v, --verbose         Verbose printing
     --host_device {x86}   The device that will be running conversion. Set to x86
                           by default.
     -w WORKING_DIR, --working_dir WORKING_DIR
                           Working directory for the inference_engine to store
                           temporary files. Creates a new directory if the
                           specified working directory does not exist
     --output_dirname OUTPUT_DIRNAME
                           output directory name for the inference_engine to
                           store temporary files under
                           <working_dir>/inference_engine .Creates a new
                           directory if the specified working directory does not
                           exist
     -p ENGINE_PATH, --engine_path ENGINE_PATH
                           Path to the inference engine.
     --debug_mode_off      Specifies if wish to turn off debug_mode mode.
     --print_version PRINT_VERSION
                           Print the QNN SDK version alongside the output.
     --offline_prepare     Use offline prepare to run qnn model.
     -bbw {8,32}, --bias_bitwidth {8,32}
                           option to select the bitwidth to use when quantizing
                           the bias. default 8
     -abw {8,16}, --act_bitwidth {8,16}
                           option to select the bitwidth to use when quantizing
                           the activations. default 8
     --golden_output_reference_directory GOLDEN_OUTPUT_REFERENCE_DIRECTORY, --golden_dir_for_mappi
                           Optional parameter to indicate the directory of the
                           goldens, it's used for tensor mapping without
                           framework.
     -wbw {8}, --weights_bitwidth {8}
                           option to select the bitwidth to use when quantizing
                           the weights. Only support 8 atm
     --float_bitwidth {32,16}
                           option to select the bitwidth to use when using float
                           for parameters(weights/bias) and activations for all
                           ops or specific Op (via encodings) selected through
                           encoding; either 32 (default) or 16
     -nif, --use_native_input_files
                           Specifies that the input files will be parsed in the
```

```
                        data type native to the graph. If not specified, input
                        files will be parsed in floating point.
-nof, --use_native_output_files
                        Specifies that the output files will be generated in
                        the data type native to the graph. If not specified,
                        output files will be generated in floating point.
--lib_name LIB_NAME   Name to use for model library (.so file)
-bd BINARIES_DIR, --binaries_dir BINARIES_DIR
                        Directory to which to save model binaries, if they
                        don't yet exist.
-mn MODEL_NAME, --model_name MODEL_NAME
                        Name of the desired output qnn model
-pq {tf,enhanced,adjusted,symmetric}, --param_quantizer {tf,enhanced,adjusted,symmetric}
                        Param quantizer algorithm used.
-qo QUANTIZATION_OVERRIDES, --quantization_overrides QUANTIZATION_OVERRIDES
                        Path to quantization overrides json file.
--act_quantizer {tf,enhanced,adjusted,symmetric}
                        Optional parameter to indicate the activation
                        quantizer to use
-fbw {16,32}, --float_bias_bitwidth {16,32}
                        option to select the bitwidth to use when biases are in float; defaul
-rqs RESTRICT_QUANTIZATION_STEPS, --restrict_quantization_steps RESTRICT_QUANTIZATION_STEPS
                        ENCODING_MIN, ENCODING_MAX
                        Specifies the number of steps to use to compute quantization encoding
                        scale = (max - min) / number of quantization steps.
                        The option should be passed as a space separated pair of hexadecimal
                        i.e. --restrict_quantization_steps 'MIN MAX'. Note that this is a hex
                        literal and not a signed integer. To supply a negative value an expli
                        e.g.: 8-bit range: --restrict_quantization_steps '-0x80 0x7F'
                        16-bit range: --restrict_quantization_steps '-0x8000 0x7F7F'
--algorithms ALGORITHMS
                        Use this option to enable new optimization algorithms.
                        Usage is: --algorithms <algo_name1> ... The available
                        optimization algorithms are: 'cle ' - Cross layer
                        equalization includes a number of methods for
                        equalizing weights and biases across layers in order
                        to rectify imbalances that cause quantization errors.
--ignore_encodings    Use only quantizer generated encodings, ignoring any
                        user or model provided encodings.
--per_channel_quantization
                        Use per-channel quantization for convolution-based op
                        weights.
-idt {float,native}, --input_data_type {float,native}
                        the input data type, must match with the supplied
                        inputs
```

```
    -odt {float_only,native_only,float_and_native}, --output_data_type {float_only,native_only,fl
                          the desired output data type
    --profiling_level {basic,detailed,backend}
                          Enables profiling and sets its level.
    --perf_profile {low_balanced,balanced,high_performance,sustained_high_performance,burst,low_p
    --log_level {error,warn,info,debug,verbose}
                          Enable verbose logging.
    --qnn_model_net_json QNN_MODEL_NET_JSON
                          Path to the qnn model net json. Only necessary if its being run from
                          This file is required to generate model_graph_struct.json file which
    --qnn_netrun_config_file QNN_NETRUN_CONFIG_FILE
                          allow backend_extention features to be applied during
                          qnn-net-run
    --extra_converter_args EXTRA_CONVERTER_ARGS
                          additional convereter arguments in a string. example:
                          --extra_converter_args input_dtype=data
                          float;input_layout=data1 NCHW
    --extra_contextbin_args EXTRA_CONTEXTBIN_ARGS
                          additional context binary generator arguments in a quoted string.
                          example: --extra_contextbin_args 'arg1=value1;arg2=value2'
    --extra_runtime_args EXTRA_RUNTIME_ARGS
                          additional convereter arguments in a quoted string.
                          example: --extra_runtime_args
                          profiling_level=basic;log_level=debug
    --compiler_config COMPILER_CONFIG
                          Path to the compiler config file.
    --precision {int8,fp16}
                          select precision
    --add_layer_outputs ADD_LAYER_OUTPUTS
                          Output layers to be dumped, e.g., 1579,232
    --add_layer_types ADD_LAYER_TYPES
                          Outputs of layer types to be dumped, e.g., Resize, Transpose; all ena
    --skip_layer_types SKIP_LAYER_TYPES
                          Comma delimited layer types to skip snooping, e.g., Resize, Transpose
    --skip_layer_outputs SKIP_LAYER_OUTPUTS
                          Comma delimited layer output names to skip debugging, e.g., 1171, 117


Please note: All the command line arguments should either be provided through command line or thro
```

The inference engine config file can be found in *{accuracy_debugger tool root directory}/python/qti/aisw/accuracy_debugger/lib/inference_engine/configs/config_files* and is a **JSON**

file. This config file stores information that helps the inference engine determine which tool and parameters to read in.

Sample Command

```
qnn-accuracy-debugger \
    --inference_engine \
    --framework tensorflow \
    --runtime dspv73 \
    --model_path $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen.pb \
    --input_tensor "input:0" 1,299,299,3 $RESOURCESPATH/samples/InceptionV3Model/data/chairs.raw \
    --output_tensor InceptionV3/Predictions/Reshape_1 \
    --architecture x86_64-linux-clang \
    --input_list $RESOURCESPATH/samples/InceptionV3Model/data/image_list.txt \
    --verbose
```

Sample Command

```
qnn-accuracy-debugger \
    --inference_engine \
    --deviceId 357415c4 \
    --framework tensorflow \
    --runtime dspv73 \
    --architecture aarch64-android \
    --model_path $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen.pb \
    --input_tensor "input:0" 1,299,299,3 $RESOURCESPATH/samples/InceptionV3Model/data/chairs.raw \
    --output_tensor InceptionV3/Predictions/Reshape_1 \
    --input_list $RESOURCESPATH/samples/InceptionV3Model/data/image_list.txt \
    --verbose
```

Tip:

- for runtime (choose from ‘cpu’, ‘gpu’, ‘dsp’, ‘dspv65’, ‘dspv66’, ‘dspv68’, ‘dspv69’, ‘dspv73’, ‘htp’). Make sure the runtime is 73 for kailua, 69 for waipio, etc. Choose HTP runtime for emulation on x86 host.

- the input_tensor (–i) and output_tensor (-o) does not need the :0 indexing like when runing tensorflow framework diagnosis
- two files, namely tensor_mapping.json and qnn_model_graph_struct.json are generated to be used in verification, be sure to locate these 2 files in the working_directory/inference_engine/latest

More example commands running from different stages:

Sample Command

```
source file stage: same as example from above section (stage default is "source")

running from converted stage (x86):
qnn-accuracy-debugger \
    --inference_engine \
    --stage converted \
    -qmcpp $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen_qnn_model.cpp \
    -qmbin $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen_qnn_model.bin \
    --runtime dspv73 \
    --architecture x86_64-linux-clang \
    --input_list $RESOURCESPATH/samples/InceptionV3Model/data/image_list.txt \
    --qnn_model_net_json $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen_qn
    --verbose \
    --framework tensorflow \
    --golden_output_reference_directory $RESOURCESPATH/samples/InceptionV3Model/golden_from_framew

Android Devices (ie. MTP):
qnn-accuracy-debugger \
    --inference_engine \
    --stage converted \
    -qmcpp $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen_qnn_model.cpp \
    -qmbin $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen_qnn_model.bin \
    --deviceId f366ce60 \
    --runtime dspv73 \
    --architecture aarch64-android \
    --input_list $RESOURCESPATH/samples/InceptionV3Model/data/image_list.txt \
    --qnn_model_net_json $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen_qn
    --verbose \
    --framework tensorflow \
    --golden_output_reference_directory $RESOURCESPATH/samples/InceptionV3Model/golden_from_framew
```

```
running in compiled stage (x86):

qnn-accuracy-debugger \
    --inference_engine \
    --stage compiled \
    --qnn_model_binary $RESOURCESPATH/samples/InceptionV3Model/qnn_model_binaries/x86_64-linux-cla
    --runtime dspv73 \
    --architecture x86_64-linux-clang \
    --input_list $RESOURCESPATH/samples/InceptionV3Model/data/image_list.txt \
    --verbose \
    --qnn_model_net_json $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen_qn
    --golden_output_reference_directory $RESOURCESPATH/samples/InceptionV3Model/golden_from_framew

Android devices (ie MTP):
qnn-accuracy-debugger \
    --inference_engine \
    --stage compiled \
    --qnn_model_binary $RESOURCESPATH/samples/InceptionV3Model/qnn_model_binaries/aarch64-android/
    --runtime dspv73 \
    --architecture aarch64-android \
    --input_list $RESOURCESPATH/samples/InceptionV3Model/data/image_list.txt \
    --verbose \
    --qnn_model_net_json $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen_qn
    --framework tensorflow \
    --golden_output_reference_directory $RESOURCESPATH/samples/InceptionV3Model/golden_from_framew

To run onnx model with custom operator:
qnn-accuracy-debugger \
    --inference_engine \
    --framework onnx \
    --runtime dspv75
    --architecture aarch64_android \
    --model_path $RESOURCESPATH/AISW-77095/model.onnx \
    --input_tensor "image" 1,3,640,1794 $RESOURCESPATH/inputs/image.raw \
    --output_tensor uncertainty_jacobian_bb \
    --input_list $RESOURCESPATH/input_list.txt \
    --default_verifier mse \
    --engine QNN \
    --engine_path $QNN_SDK_ROOT \
    --extra_converter_args 'op_package_config=$RESOURCESPATH/CustomPreTopKOpPackageCPU_v2.xml;op_p
    --extra_contextbin_args 'op_packages=$RESOURCESPATH/libQnnCustomPreTopKOpPackageHtp.so:CustomP
    --extra_runtime_args 'op_packages=$RESOURCESPATH/AISW-77095/libQnnCustomPreTopKOpPackageHtp_v7
    --debug_mode_off \
```

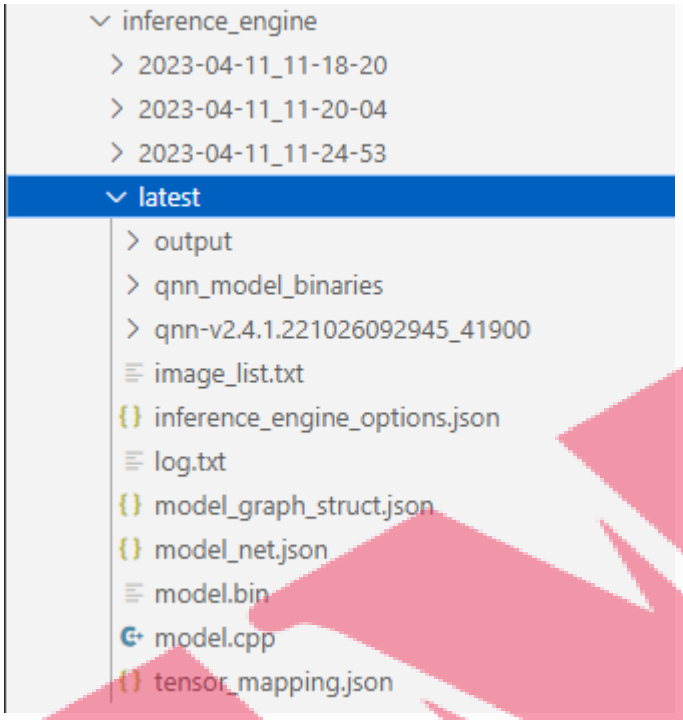
```
--offline_prepare \
--verbose
```

Tip:

- The qnn_model_net_json file is not required to run this step. However, it is needed to build the qnn_model_graph_struct.json, which can be used in the Verification step. The model_net.json file is generated when the original model is converted into a converted model. Hence if you are debugging this model from the converted model stage, it is recommended to ask for this model_net.json file.
- framework and golden_dir_for_mapping, or just golden_dir_for_mapping itself is an alternative to the original model to be provided to generate the tensor_mapping.json. However, providing only the golden_dir_for_mapping, the get_tensor_mapping module will try its best to map but it is not guaranteed this mapping will be 100% accurate.

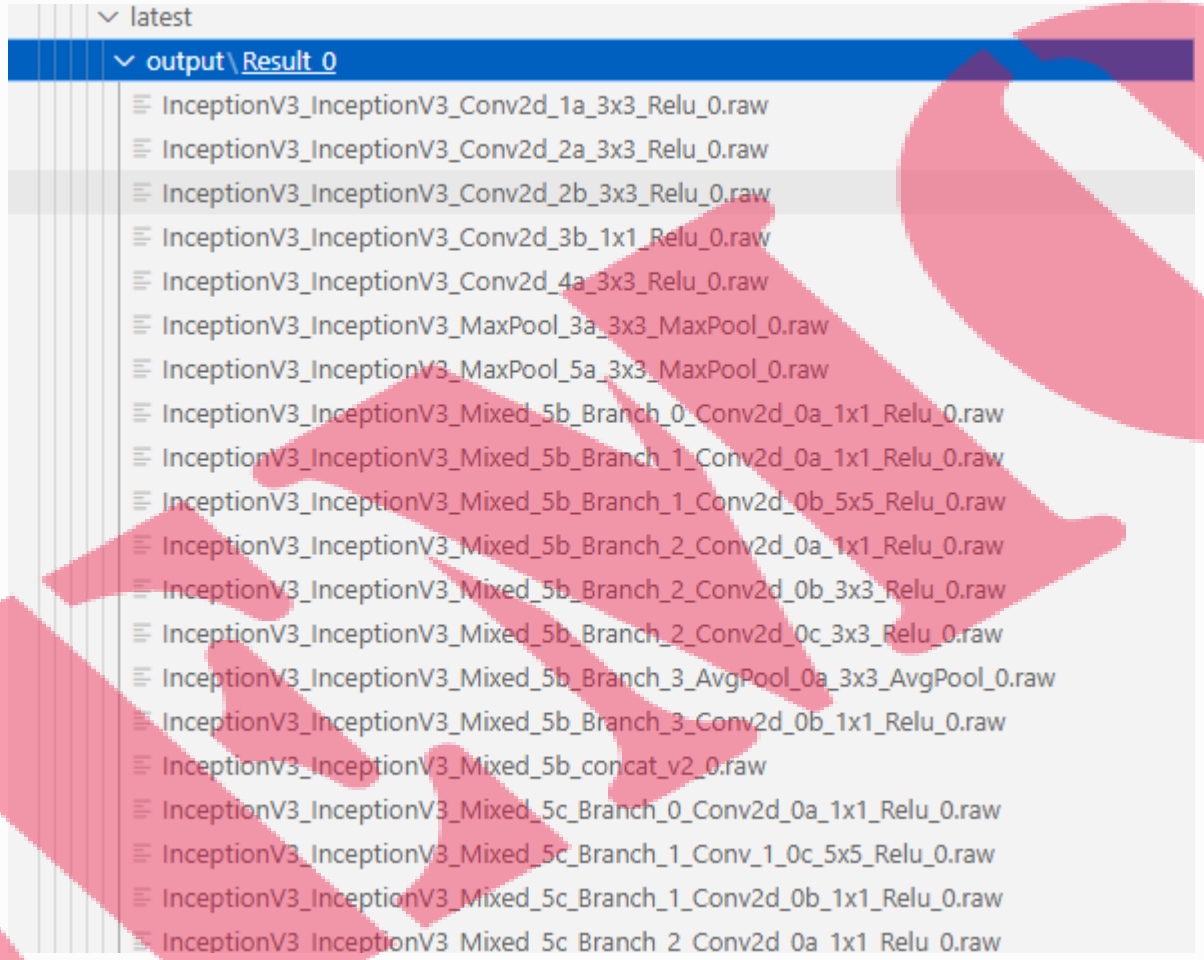
Output

Once the inference engine has finished running, it will store the output in the specified directory (or the current working directory by default) and store the files in that directory. By default, it will store the output in *working_directory/inference_engine* in the current working directory.



The figure above shows the sample output from one of the runs of inference engine step. The output directory contains raw files. Each raw file is an output of an operation in the network. The model.bin and model.cpp files are created by the model converter. The qnn_model_binaries directory contains the .so file that is generated by the modellibgenerator utility. The file image_list.txt contains the path for sample test images. The inference_engine_options.json file contains all the options with which this run was launched. In addition to generating the .raw files, the inference_engine also generates the model's graph structure in a .json file. The name of the file is the same as the name of the protobuf model file. The model_graph_struct.json aids in providing structure related information of the converted model graph during the verification step. Specifically, it helps with organizing the nodes in order (for i.e. the beginning nodes should come earlier than ending nodes). The model_net.json has information about what structure the data is in within the framework_diagnosis and inference_engine steps (data can be in different formats for e.g. channels first vs channels last). The verification step uses this information so that data can be properly transposed and compared. It is an optional parameter which can be provided during inference engine step for generating the model_graph_struct.json file (mandated only when running inference engine from the converted

stage). Finally, the tensor_mapping file contains a mapping of the various intermediate output file names generated from the framework diagnosis step and the inference engine step.



The created .raw files are organized in the same manner as framework_diagnosis (see above).

Verification

The Verification step compares the output (from the intermediate tensors of a given model) produced by the framework diagnosis step with the output produced by the inference engine step. Once the comparison is complete, the verification results are compiled and displayed visually in a format that can be easily interpreted by the user.

There are different types of verifiers for e.g.: CosineSimilarity, RtolAtol, etc. To see available verifiers please use the –help option (qnn-accuracy-debugger –verification –help). Each verifier compares the Framework Diagnosis and Inference Engine output using an error metric. It also prepares reports and/or visualizations to help the user analyze the network's error data.

Usage

```
usage: qnn-accuracy-debugger --verification [-h]

Script to run verification.

required arguments:
     --default_verifier DEFAULT_VERIFIER [DEFAULT_VERIFIER ...]
                            Default verifier used for verification. The options
                            "RtolAtol", "AdjustedRtolAtol", "TopK", "L1Error",
                            "CosineSimilarity", "MSE", "MAE", "SQNR", "MeanIOU",
                            "ScaledDiff" are supported. An optional list of
                            hyperparameters can be appended. For example:
                            --default_verifier
                            rtolatol,rtolmargin,0.01,atolmargin,0,01. An optional
                            list of placeholders can be appended. For example:
                            --default_verifier CosineSimilarity param1 1 param2 2.
                            to use multiple verifiers, add additional
                            --default_verifier CosineSimilarity
     --golden_output_reference_directory GOLDEN_OUTPUT_REFERENCE_DIRECTORY, --framework_results FR
                            Path to root directory generated from framework
                            diagnosis. Paths may be absolute, or relative to the
                            working directory.
     --inference_results INFERENCE_RESULTS
                            Path to root directory generated from inference engine
                            diagnosis. Paths may be absolute, or relative to the
                            working directory.

optional arguments:
     --tensor_mapping TENSOR_MAPPING
                            Path to the file describing the tensor name mapping
                            between inference and golden tensors.can be generated
                            with in the inference engine step.
     --verifier_config VERIFIER_CONFIG
                            Path to the verifiers' config file
     --graph_struct GRAPH_STRUCT
                            Path to the inference graph structure .json file. This file
                            aids in providing the structure related information of the converted
```

```
      -v, --verbose         Verbose printing
      -w WORKING_DIR, --working_dir WORKING_DIR
                            Working directory for the verification to store
                            temporary files. Creates a new directory if the
                            specified working directory does not exist
      --output_dirname OUTPUT_DIRNAME
                            output directory name for the verification to store
                            temporary files under <working_dir>/verification.
                            Creates a new directory if the specified working
                            directory does not exist
      --qnn_model_json_path QNN_MODEL_JSON_PATH
                            Path to the model json for transforming intermediate
                            tensors to spatial-first axis order.

arguments for generating a new tensor_mapping.json:
      -m MODEL_PATH, --model_path MODEL_PATH
                            path to original model for tensor_mapping uses here.
      -e ENGINE_NAME [ENGINE_VERSION ...], --engine ENGINE_NAME [ENGINE_VERSION ...]
                            Name of engine that will be running inference,
                            optionally followed by the engine version. Used here
                            for tensor_mapping.
      -f FRAMEWORK [FRAMEWORK ...], --framework FRAMEWORK [FRAMEWORK ...]
                            Framework type to be used, followed optionally by
                            framework version. Used here for tensor_mapping.

Please note: All the command line arguments should either be provided through command line or thro
```

The main verification process run using qnn-accuracy-debugger –verification optionally uses –tensor_mapping and –graph_struct to find files to compare. These files are generated by the inference engine step, and should be supplied to verification for best results. By default they are named tensor_mapping.json and {model name}_graph_struct.json, and can be found in the output directory of the inference engine results.

Sample Command

```
Compare output of framework diagnosis with inference engine:

qnn-accuracy-debugger \
      --verification \
```

```
    --default_verifier CosineSimilarity param1 1 param2 2 \
    --default_verifier SQNR param1 5 param2 1 \
    --golden_output_reference_directory $PROJECTREPOPATH/working_directory/framework_diagnosis/20
    --inference_results $PROJECTREPOPATH/working_directory/inference_engine/latest/output/Result_
    --tensor_mapping $PROJECTREPOPATH/working_directory/inference_engine/latest/tensor_mapping.js
    --graph_struct $PROJECTREPOPATH/working_directory/inference_engine/latest/qnn_model_graph_str
    --qnn_model_json_path $PROJECTREPOPATH/working_directory/inference_engine/latest/qnn_model_ne
```

Tip:

- If you passed multiple images in the image_list.txt from run inference engine diagnosis, you'll receive multiple output/Result_x, choose result that matches the input you used for framework diagnosis for comparison (ie. in framework you used chair.raw and inference chair.raw was the first item in the image_list.txt then choose output/Result_0, if chair.raw was the second item in image_list.txt, then choose output/Result_1).
- It is recommended to always supply 'graph_struct' and 'tensor_mapping' to the command as it is used to line up the report and find the corresponding files for comparison. if tensor_mapping did not get generated from previous steps, you can supplement with 'model_path', 'engine', 'framework' to have module generate 'tensor_mapping' during runtime.
- You can also compare inference_engine outputs to inference_engine outputs by passing the /output of the inference_engine output to the 'framework_results'. If you want the outputs to be exact-name-matching, then you do not need to provide a tensor_mapping file.
- Note that if you need to generate a tensor mapping instead of providing a path to prexisting tensor mapping file. You can provide the 'model_path' option.

Verifier uses two optional config files. The first file is used to set parameters for specific verifiers, as well as which tensors to use these verifiers on. The second file is used to map tensor names from framework_diagnosis to the inference_engine, since certain tensors generated by framework_diagnosis may have different names than tensors generated by inference_engine.

Verifier Config:

The verifier config file is a JSON file that tells verification which verifiers (asides from the default verifier) to use and with which parameters and on what specific tensors. If no config file is provided, the tool will only use the default verifier specified from the command line, with its default parameters, on all the tensors. The JSON file is keyed by verifier names, with each verifier as its own dictionary keyed by "parameters" and "tensors".

Config File

```json
{
    "MeanIOU": {
        "parameters": {
            "background_classification": 1.0
        },
        "tensors": [["Postprocessor/BatchMultiClassNonMaxSuppression_boxes", "detection_classes:0"
    },
    "TopK": {
        "parameters": {
            "k": 5,
            "ordered": false
        },
        "tensors": [["Reshape_1:0"], ["detection_classes:0"]]
    }
}
```

Note that the "tensors" field is a list of lists. This is done because specific verifiers (e.g. MeanIOU) runs on two tensor at a time. Hence the two tensors are placed in a list. Otherwise if a verifier only runs on one tensor, it will have a list of lists with only one tensor name in each list.

Tensor Mapping:

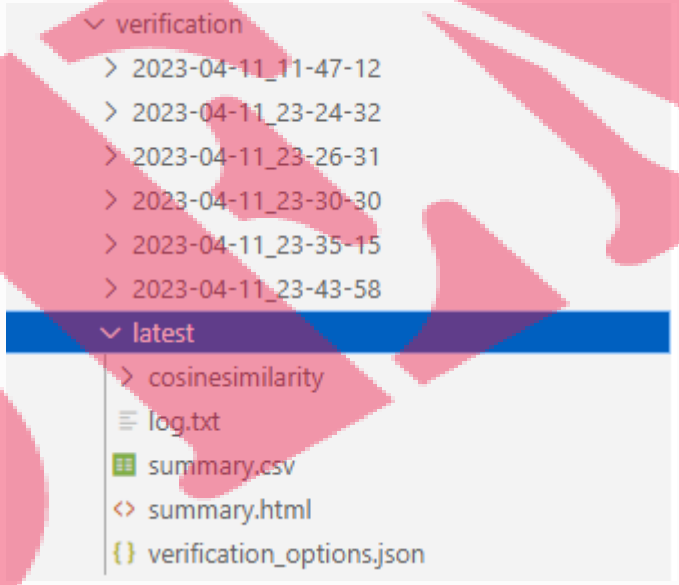
Tensor mapping is a JSON file keyed by inference tensor names, of framework tensor names. If the tensor mapping is not provided, the tool will assume inference and golden tensor names are identical.

Tensor Mapping File

```json
{
    "Postprocessor/BatchMultiClassNonMaxSuppression_boxes": "detection_boxes:0",
    "Postprocessor/BatchMultiClassNonMaxSuppression_scores": "detection_scores:0"
}
```

Output

Verification's output is divided into different verifiers. For example, if both RtolAtol and TopK verifiers are used, there will be two sub-directories named "RtolAtol" and "TopK". Availble verifiers can be found by issuing –help option.



Under each sub-directory, the verification analysis for each tensor is organized similar to how framework_diagnosis (see above) and inference_engine are organized. For each tensor, a CSV and HTML file is generated. In addition to the tensor-specific analysis, the tool also generates a summary CSV and HTML file which summarizes the data from all verifiers and their subsequent tensors. The

following figure shows how a sample summary generated in the verification step looks. Each row in this summary corresponds to one tensor name that is identified by the framework diagnosis and inference engine steps. The final column shows cosinesimilarity score which can vary between 0 to 1 (this range might be different for other verifiers). Higher scores denote similarity while lower scores indicate variance. The developer can then further investigate those specific tensor details. Developer should inspect tensors from top-to-bottom order, meaning if a tensor is broken at an earlier node, anything that was generated post that node is unreliable until that node is properly fixed.

Name	LayerType	Size	Tensor_dims	cosinesimilarity
InceptionV3_InceptionV3_Conv2d_1a_3x3_Relu_0	Conv2d	710432	[1, 149, 149, 32]	0.999751
InceptionV3_InceptionV3_Conv2d_2a_3x3_Relu_0	Conv2d	691488	[1, 147, 147, 32]	0.996600
InceptionV3_InceptionV3_Conv2d_2b_3x3_Relu_0	Conv2d	1382976	[1, 147, 147, 64]	0.996976
InceptionV3_InceptionV3_MaxPool_3a_3x3_MaxPool_0	PoolMax2d	341056	[1, 73, 73, 64]	0.997646
InceptionV3_InceptionV3_Conv2d_3b_1x1_Relu_0	Conv2d	426320	[1, 73, 73, 80]	0.994449
InceptionV3_InceptionV3_Conv2d_4a_3x3_Relu_0	Conv2d	967872	[1, 71, 71, 192]	0.994425
InceptionV3_InceptionV3_MaxPool_5a_3x3_MaxPool_0	PoolMax2d	235200	[1, 35, 35, 192]	0.995875

Compare Encodings

The Compare Encodings feature is designed to compare QNN and AIMET encodings. This feature takes QNN model net and AIMET encoding JSON files as inputs. This feature executes in the following order.

1. Extracts encodings from the given QNN model net JSON.
2. Compares extracted QNN encodings with given AIMET encodings.
3. Writes results to an Excel file that highlights mismatches.
4. Throws warnings if some encodings are present in QNN but not in AIMET and vice-versa.
5. Writes the extracted QNN encodings JSON file (for reference).

Usage

```
usage: qnn-accuracy-debugger --compare_encodings [-h]
                             --input INPUT
```

```
                                --aimet_encodings_json AIMET_ENCODINGS_JSON
                                [--precision PRECISION]
                                [--params_only]
                                [--activations_only]
                                [--specific_node SPECIFIC_NODE]
                                [--working_dir WORKING_DIR]
                                [--output_dirname OUTPUT_DIRNAME]
                                [-v]

Script to compare QNN encodings with AIMET encodings

optional arguments:
  -h, --help            Show this help message and exit

required arguments:
  --input INPUT
                        Path to QNN model net JSON file
  --aimet_encodings_json AIMET_ENCODINGS_JSON
                        Path to AIMET encodings JSON file

optional arguments:
  --precision PRECISION
                        Number of decimal places up to which comparison will be done (default: 17)
  --params_only         Compare only parameters in the encodings
  --activations_only    Compare only activations in the encodings
  --specific_node SPECIFIC_NODE
                        Display encoding differences for the given node
  --working_dir WORKING_DIR
                        Working directory for the compare_encodings to store temporary files.
                        Creates a new directory if the specified working directory does not exist.
  --output_dirname OUTPUT_DIRNAME
                        Output directory name for the compare_encodings to store temporary files
                        under <working_dir>/compare_encodings. Creates a new directory if the
                        specified working directory does not exist.
  -v, --verbose         Verbose printing
```

Sample Commands

```
# Compare both params and activations
qnn-accuracy-debugger \
    --compare_encodings \
    --input QNN_model_net.json \
```

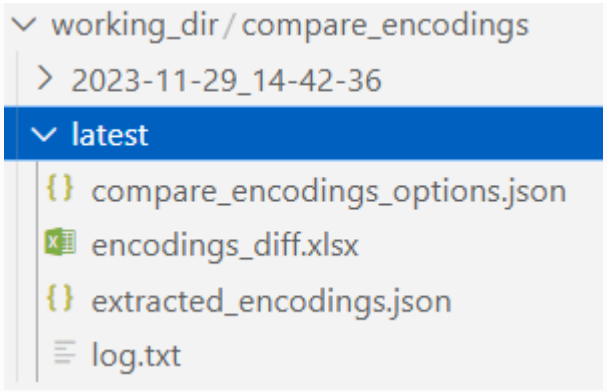
```
    --aimet_encodings_json aimet_encodings.json

# Compare only params
qnn-accuracy-debugger \
    --compare_encodings \
    --input QNN_model_net.json \
    --aimet_encodings_json aimet_encodings.json \
    --params_only

# Compare only activations
qnn-accuracy-debugger \
    --compare_encodings \
    --input QNN_model_net.json \
    --aimet_encodings_json aimet_encodings.json \
    --activations_only

# Compare only a specific encoding
qnn-accuracy-debugger \
    --compare_encodings \
    --input QNN_model_net.json \
    --aimet_encodings_json aimet_encodings.json \
    --specific_node _2_22_Conv_output_0
```

Tip

A working_directory is generated from wherever this script is called from unless otherwise specified.

Output

The program creates a directory named *latest* in *working_directory/compare_encodings* which is symbolically linked to the most recently generated directory. In the example below, *latest* will have data that is symlinked to the data in the most recent directory *YYYY-MM-DD_HH:mm:ss*. Users may choose to override the directory name by passing it to –output_dirname, e.g., *–output_dirname myTest*.



The figure above shows a sample output from a compare_encodings run. The following details what each file contains.

- compare_encodings_options.json contains all the options used to run this feature
- encodings_diff.xlsx contains comparison results with mismatches highlighted
- log.txt contains log statements for the run
- extracted_encodings.json contains extracted QNN encodings

Tensor inspection

Tensor inspection compares given reference output and target output tensors and dumps various statistics to represent differences between them.

The Tensor inspection feature can:

1. Plot histograms for golden and target tensors
2. Plot a graph indicating deviation between golden and target tensors
3. Plot a cumulative distribution graph (CDF) for golden vs target tensors
4. Plot a density (KDE) graph for target tensor highlighting target min/max and calibrated min/max values
5. Create a CSV file containing information about: target min/max; calibrated min/max; golden output min/max; target/calibrated min/max differences; and computed metrics (verifiers).

Note

Only data with matching target/golden filenames is inspected; other data is ignored

Calibrated min/max values are extracted from a user provided encodings file. If an encodings file is not provided, density plot will be skipped and also the CSV summary output will not include calibrated min/max information.

Usage

```
usage: qnn-accuracy-debugger --tensor_inspection [-h]
                        --golden_data GOLDEN_DATA
                        --target_data TARGET_DATA
                        --verifier VERIFIER [VERIFIER ...]
                        [-w WORKING_DIR]
                        [--data_type {int8,uint8,int16,uint16,float32}]
                        [--target_encodings TARGET_ENCODINGS]
                        [-v]

Script to inspection tensor.

required arguments:
  --golden_data GOLDEN_DATA
                        Path to golden/framework outputs folder. Paths may be absolute or
                        relative to the working directory.
  --target_data TARGET_DATA
                        Path to target outputs folder. Paths may be absolute or relative to the
                        working directory.
  --verifier VERIFIER [VERIFIER ...]
                        Verifier used for verification. The options "RtolAtol",
                        "AdjustedRtolAtol", "TopK", "L1Error", "CosineSimilarity", "MSE", "MAE",
                        "SQNR", "MeanIOU", "ScaledDiff" are supported.
                        An optional list of hyperparameters can be appended, for example:
                        --verifier rtolatol,rtolmargin,0.01,atolmargin,0,01.
                        To use multiple verifiers, add additional --verifier CosineSimilarity

optional arguments:
  -w WORKING_DIR, --working_dir WORKING_DIR
                        Working directory to save results. Creates a new directory if the
                        specified working directory does not exist
  --data_type {int8,uint8,int16,uint16,float32}
                        DataType of the output tensor.
  --target_encodings TARGET_ENCODINGS
```

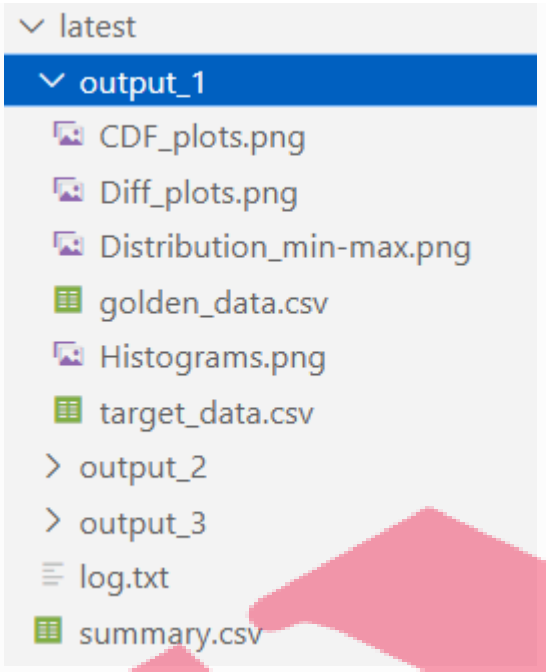
```
                        Path to target encodings json file.
  -v, --verbose         Verbose printing
```

Sample Commands

```
# Basic run
qnn-accuracy-debugger --tensor_inspection \
    --golden_data golden_tensors_dir \
    --target_data target_tensors_dir \
    --verifier sqnr

# Pass target encodings file and enable multiple verifiers
qnn-accuracy-debugger --tensor_inspection \
    --golden_data golden_tensors_dir \
    --target_data target_tensors_dir \
    --verifier mse \
    --verifier sqnr \
    --verifier rtolatol,rtolmargin,0.01,atolmargin,0.01 \
    --target_encodings qnn_encoding.json
```

Tip

A working_directory is generated from wherever this script is called from unless otherwise specified.



The figure above shows a sample output from a Tensor inspection run. The following details what each file contains.

- Each tensor will have its own directory; the directory name matches the tensor name.
  - CDF_plots.png – Golden vs target CDF graph
  - Diff_plots.png – Golden and target deviation graph
  - Distribution_min-max.png – Density plot for target tensor highlighting target vs calibrated min/max values
  - Histograms.png – Golden and target histograms
  - golden_data.csv – Golden tensor data
  - target_data.csv – Target tensor data
- log.txt – Log statements from the entire run
- summary.csv – Target min/max, calibrated min/max, golden output min/max, target vs calibrated min/max differences, and verifier outputs

Run QNN Accuracy Debugger E2E

This feature is designed to run the framework diagnosis, inference engine, and verification features sequentially with a single command to debug the model. The following debugging algorithms are available.

1. **Oneshot-layerwise(default):**
    - **This algorithm is designed to debug all layers of model at a time by performing below steps**
        - Execute framework diagnosis to collect reference outputs in fp32
        - Execute inference engine to collect backend outputs in provided target precision.
        - Execute verification for comparison of intermediate outputs from the above 2 steps
        - Execute tensor inspection (when –enable_tensor_inspection is passed) to dump various plots, e.g., scatter, line, CDF, etc., for intermediate outputs
    - It provides quick analysis to identify layers of model causing accuracy deviation.
    - User can chose cumulative-layerwise(below) for deeper analysis of accuracy deviation.
2. **Cumulative-layerwise:**
    - **This algorithm is designed to debug one layer at a time by performing below steps**
        - Execute framework diagnosis to collect reference outputs from all layers of model in fp32.
        - **Execute inference engine and verification in iterative manner to perform below operations**
            - to collect backend outputs in target precision for each layer while removing the effect of its preceeding layers on final output.
            - to compare intermediate outputs from framework diagnosis and inference engine
    - It provides deeper analysis to identify all layers of model causing accuracy deviation.
    - Currently this option supports only onnx models.
3. **Layerwise:**
    - **This algorithm is designed to debug a single layer model at a time by performing the following steps**

- Get golden reference per layer outputs from an external tool or, if a golden reference is not given, run framework diagnosis to collect intermediate layer outputs.
- **Iteratively execute inference engine and verification to:**
  - Collect backend outputs in target precision for each single layer model by removing the preceding and following layers
  - Compare intermediate output from golden reference with inference engine single layer model output
- Layerwise snooping provides deeper analysis to identify all model layers causing accuracy deviation on hardware with respect to framework/simulation outputs.
- Layerwise snooping only supports ONNX models.

Usage

```
usage: qnn-accuracy-debugger [--framework_diagnosis] [--inference_engine] [--verification] [-h]

Script that runs Framework Diagnosis, Inference Engine or Verification.

Arguments to select which component of the tool to run.  Arguments are mutually exclusive (at
most 1 can be selected).  If none are selected, then all components are run:
--framework_diagnosis Run framework
--inference_engine    Run inference engine
--verification        Run verification

optional arguments:
-h, --help            Show this help message. To show help for any of the components, run
                      script with --help and --<component>. For example, to show the help
                      for Framework Diagnosis, run script with the following: --help
                      --framework_diagnosis

usage: qnn-accuracy-debugger [-h] -f FRAMEWORK [FRAMEWORK ...] -m MODEL_PATH -i INPUT_TENSOR
                             [INPUT_TENSOR ...] -o OUTPUT_TENSOR -r RUNTIME -a
                             {aarch64-android,x86_64-linux-clang,aarch64-android-clang6.0}
                             -l INPUT_LIST --default_verifier DEFAULT_VERIFIER
                             [DEFAULT_VERIFIER ...] [-v] [-w WORKING_DIR]
                             [--output_dirname OUTPUT_DIRNAME]
                             [--deep_analyzer {modelDissectionAnalyzer}]
                             [--debugging_algorithm {layerwise,cumulative-layerwise,oneshot-layerwi

Options for running the Accuracy Debugger components
```

```
optional arguments:
-h, --help            show this help message and exit

Arguments required by both Framework Diagnosis and Inference Engine:
-f FRAMEWORK [FRAMEWORK ...], --framework FRAMEWORK [FRAMEWORK ...]
                      Framework type and version, version is optional. Currently supported
                      frameworks are [tensorflow, tflite, onnx]. For example, tensorflow
                      2.3.0
-m MODEL_PATH, --model_path MODEL_PATH
                      Path to the model file(s).
-i INPUT_TENSOR [INPUT_TENSOR ...], --input_tensor INPUT_TENSOR [INPUT_TENSOR ...]
                      The name, dimensions, raw data, and optionally data type of the
                      network input tensor(s) specifiedin the format "input_name" comma-
                      separated-dimensions path-to-raw-file, for example: "data"
                      1,224,224,3 data.raw float32. Note that the quotes should always be
                      included in order to handle special characters, spaces, etc. For
                      multiple inputs specify multiple --input_tensor on the command line
                      like: --input_tensor "data1" 1,224,224,3 data1.raw --input_tensor
                      "data2" 1,50,100,3 data2.raw float32.
-o OUTPUT_TENSOR, --output_tensor OUTPUT_TENSOR
                      Name of the graph's specified output tensor(s).

Arguments required by Inference Engine:
-r RUNTIME, --runtime RUNTIME
                      Runtime to be used for inference.
-a {aarch64-android,x86_64-linux-clang,aarch64-android-clang6.0}, --architecture {aarch64-an
droid,x86_64-linux-clang,aarch64-android-clang6.0}
                      Name of the architecture to use for inference engine.
-l INPUT_LIST, --input_list INPUT_LIST
                      Path to the input list text.
Arguments required by Verification:                                          [3/467]
--default_verifier DEFAULT_VERIFIER [DEFAULT_VERIFIER ...]
                      Default verifier used for verification. The options "RtolAtol",
                      "AdjustedRtolAtol", "TopK", "L1Error", "CosineSimilarity", "MSE",
                      "MAE", "SQNR", "MeanIOU", "ScaledDiff" are supported. An optional
                      list of hyperparameters can be appended. For example:
                      --default_verifier rtolatol,rtolmargin,0.01,atolmargin,0,01. An
                      optional list of placeholders can be appended. For example:
                      --default_verifier CosineSimilarity param1 1 param2 2. to use
                      multiple verifiers, add additional --default_verifier
                      CosineSimilarity

optional arguments:
-v, --verbose         Verbose printing
```

```
-w WORKING_DIR, --working_dir WORKING_DIR
                        Working directory for the wrapper to store temporary files. Creates
                        a new directory if the specified working directory does not exitst.
--output_dirname OUTPUT_DIRNAME
                        output directory name for the wrapper to store temporary files under
                        <working_dir>/wrapper. Creates a new directory if the specified
                        working directory does not exist
--deep_analyzer {modelDissectionAnalyzer}
                        Deep Analyzer to perform deep analysis
--golden_output_reference_directory
                        Optional parameter to indicate the directory of the golden reference outpu
                        When this option is provided, the framework diagnosis is stage skipped.
                        In inference stage, it's used for tensor mapping without a framework.
                        In verification stage, it's used as a reference to compare
                        outputs produced in the inference engine stage.
--enable_tensor_inspection
                        Plots graphs (line, scatter, CDF etc.) for each
                        layer's output. Additionally, summary sheet will have
                        more details like golden min/max, target min/max etc.,
--debugging_algorithm {layerwise,cumulative-layerwise,oneshot-layerwise}
                        Performs model debugging layerwise, cumulative-layerwise or in oneshot-
                        layerwise based on choice.

(below options are allowed only for Layerwise and Cumulative layerwise run)
--start_layer START_LAYER
                        Extracts the given model from mentioned start layer
                        output name
--end_layer END_LAYER
                        Extracts the given model from mentioned end layer
                        output name
```

Sample Command for oneshot-layerwise

```
Command for Oneshot-layerwise using DSP backend:

qnn-accuracy-debugger \
    --framework tensorflow \
    --runtime dspv73 \
    --model_path $RESOURCESPATH/samples/InceptionV3Model/inception_v3_2016_08_28_frozen.pb \
    --input_tensor "input:0" 1,299,299,3 $PATHTOGOLDENI/samples/InceptionV3Model/data/chairs.raw \
    --output_tensor InceptionV3/Predictions/Reshape_1:0 \
    --architecture x86_64-linux-clang \
```

```
    --debugging_algorithm oneshot-layerwise
    --input_list $RESOURCESPATH/samples/InceptionV3Model/data/image_list.txt \
    --default_verifier CosineSimilarity \
    --framework_results $PROJECTREPOPATH/working_directory/framework_diagnosis/latest/ \
    --inference_results $PROJECTREPOPATH/working_directory/inference_engine/latest/output/Result_0
    --tensor_mapping $PROJECTREPOPATH/working_directory/inference_engine/latest/tensor_mapping.jso
    --graph_struct $PROJECTREPOPATH/working_directory/inference_engine/latest/qnn_model_graph_stru
    --qnn_model_json_path $PROJECTREPOPATH/working_directory/inference_engine/latest/qnn_model_net
    --enable_tensor_inspection \
    --verbose
```

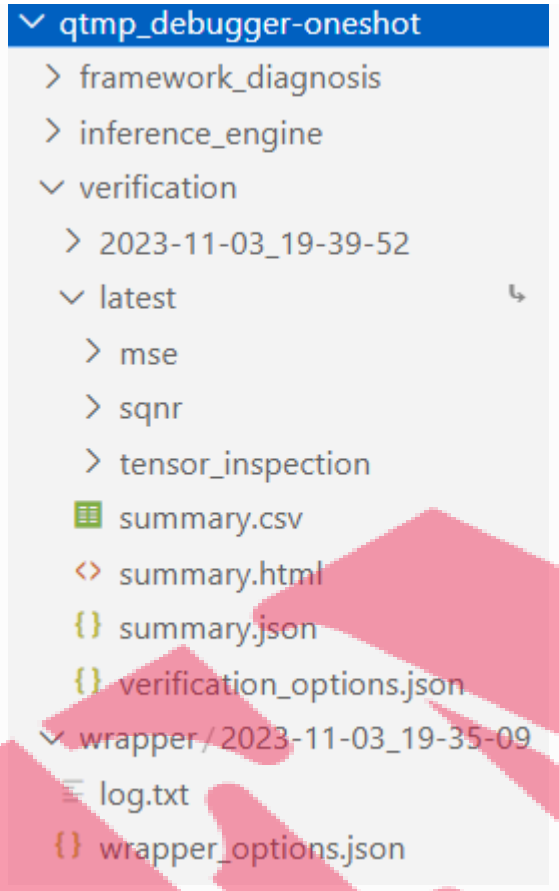
```
Command for Oneshot-layerwise using HTP emulation on x86 host:

qnn-accuracy-debugger \
    --framework onnx \
    --runtime htp \
    --model_path /local/mnt/workspace/models/vit/vit_base_16_224.onnx \
    --input_tensor "input.1" 1,3,224,224 /local/mnt/workspace/models/vit/000000039769_1_3_224_224.
    --output_tensor 1597 \
    --architecture x86_64-linux-clang \
    --input_list /local/mnt/workspace/models/vit/list.txt \
    --default_verifier CosineSimilarity \
    --offline_prepare \
    --debugging_algorithm oneshot-layerwise
    --engine QNN \
    --enable_tensor_inspection \
    --verbose
```

Note

The –enable_tensor_inspection argument significantly increases overall execution time when used with large models. To speed up execution, omit this argument.

Output

The program creates framework_diagnosis, inference_engine, verification, and wrapper output directories as below:



- framework_diagnosis – Contains a timestamped directory that contains the intermediate layer outputs (framework) stored in .raw format as described in the framework diagnosis step.
- inference_engine – Contains a timestamped directory that contains the intermediate layer outputs (inference engine) stored in .raw format as described in the inference engine step.
- verification directory – Contains a timestamped directory that contains the following:
  - A directory for each verifier specified while running oneshot; it contains CSV and HTML files with metric details for each layer output
  - tensor_inspection – Individual directories for each layer's output with the following contents:
    - CDF_plots.png – Golden vs target CDF graph
    - Diff_plots.png – Golden and target deviation graph
    - Histograms.png – Golden and target histograms
    - golden_data.csv – Golden tensor data

    - target_data.csv – Target tensor data
  - summary.csv – Report for verification results of each layers output
- Wrapper directory containing log.txt with the entire log for the run.

Note: Except wrapper directory all other directories will have a folder called latest which is a symlink to the latest run's corresponding timestamped directory.

Snapshot of summary.csv file:

Name	LayerType	Size	Tensor_dims	mse	sqnr	golden_min	golden_max	target_min	target_max
_317	Conv2d	401408	[1, 112, 112, 32]	0.186607793	-0.278973952	0	2.407759905	0	2.420042276
_320	DepthWiseConv2d	401408	[1, 112, 112, 32]	0.766735554	-0.769651905	0	6	0	6
_321	Conv2d	200704	[1, 112, 112, 16]	0.641227365	-2.960772812	-4.36497736	4.387659073	-4.362733841	4.427849293
_325	Conv2d	1204224	[1, 112, 112, 96]	0.091154471	-0.304804537	0	4.334304333	0	4.329411983
_328	DepthWiseConv2d	301056	[1, 56, 56, 96]	0.164875686	-1.010350212	0	3.02080822	0	3.388235331
_329	Conv2d	75264	[1, 56, 56, 24]	0.438003272	-2.947278917	-2.925263882	3.320363045	-3.000357628	3.417073965
_333	Conv2d	451584	[1, 56, 56, 144]	0.049246728	-0.189710017	0	1.309156179	0	1.349412322
_336	DepthWiseConv2d	451584	[1, 56, 56, 144]	0.097275533	-1.24253273	0	2.830595016	0	2.86246419
_337	Conv2d	75264	[1, 56, 56, 24]	0.598135471	-2.964283824	-3.501322269	3.29065752	-3.49203968	3.22749114
_339	Eltwise_Binary	75264	[1, 56, 56, 24]	1.214788675	-2.962860167	-4.299339771	3.909160137	-4.263907909	4.060864925
_342	Conv2d	451584	[1, 56, 56, 144]	0.050069354	-0.656715855	0	1.932192683	0	1.921604753
_345	DepthWiseConv2d	112896	[1, 28, 28, 144]	0.124174997	-0.239327699	0	1.965788126	0	2.063059807
_346	Conv2d	25088	[1, 28, 28, 32]	0.537808657	-2.682052255	-2.24165535	1.956404686	-2.382666588	2.184111118
_350	Conv2d	150528	[1, 28, 28, 192]	0.029806027	0.77684857	0	1.00565505	0	1.099466801
_353	DepthWiseConv2d	150528	[1, 28, 28, 192]	0.041249752	-1.5549317	0	1.507732391	0	1.375646114
_354	Conv2d	25088	[1, 28, 28, 32]	0.227637917	-2.866997719	-1.996557236	2.015232801	-1.958809853	1.958809853
_356	Eltwise_Binary	25088	[1, 28, 28, 32]	0.770535588	-2.747408748	-2.74501133	2.556851864	-3.004191399	2.496440649
_359	Conv2d	150528	[1, 28, 28, 192]	0.023810335	0.280860178	0	0.935381889	0	1.02562952
_362	DepthWiseConv2d	150528	[1, 28, 28, 192]	0.038466841	-1.788970232	0	1.720778108	0	1.65101409
_363	Conv2d	25088	[1, 28, 28, 32]	0.179694474	-2.893956602	-2.161521673	1.705373526	-2.147612333	1.649615169
_365	Eltwise_Binary	25088	[1, 28, 28, 32]	1.023316503	-2.779848874	-3.470533371	3.124724865	-3.505088806	3.144270658
_368	Conv2d	150528	[1, 28, 28, 192]	0.036799919	-0.28568564	0	1.403604627	0	1.400200129
_371	DepthWiseConv2d	37632	[1, 14, 14, 192]	0.113418125	0.399082154	0	1.849825263	0	1.859656453
_372	Conv2d	12544	[1, 14, 14, 64]	0.4055686	-2.846610248	-1.95641458	1.767401457	-1.888438225	1.91580689
_376	Conv2d	75264	[1, 14, 14, 384]	0.019188538	0.860826448	0	0.865441978	0	0.880366147
_379	DepthWiseConv2d	75264	[1, 14, 14, 384]	0.022653537	-1.486739665	0	1.080931664	0	1.031657577
_380	Conv2d	12544	[1, 14, 14, 64]	0.163591459	-2.89067179	-1.573169947	1.262223005	-1.527808189	1.273173571
_382	Eltwise_Binary	12544	[1, 14, 14, 64]	0.577057838	-2.884348333	-2.036406994	1.97029829	-1.944982886	2.066544294
_385	Conv2d	75264	[1, 14, 14, 384]	0.015156617	-0.271802191	0	0.645082116	0	0.631709516
_388	DepthWiseConv2d	75264	[1, 14, 14, 384]	0.023140276	-1.390685141	0	1.179039478	0	1.264452577
_389	Conv2d	12544	[1, 14, 14, 64]	0.11270529	-2.866001725	-1.22379303	1.437875628	-1.186528206	1.336933255

Understanding the oneshot-layerwise report:

Column	Description
Name	Output name of the current layer

Layer Type	Type of the current layer
Size	Size of this layer's output
Tensor_dims	Shape of this layer's output
<Verifier name>	Verifier value of the current layer output compared to reference output
golden_min	minimum value in the reference output for current layer
golden_max	maximum value in the reference output for current layer
target_min	minimum value in the target output for current layer
target_max	maximum value in the target output for current layer

Sample Command for cumulative-layerwise

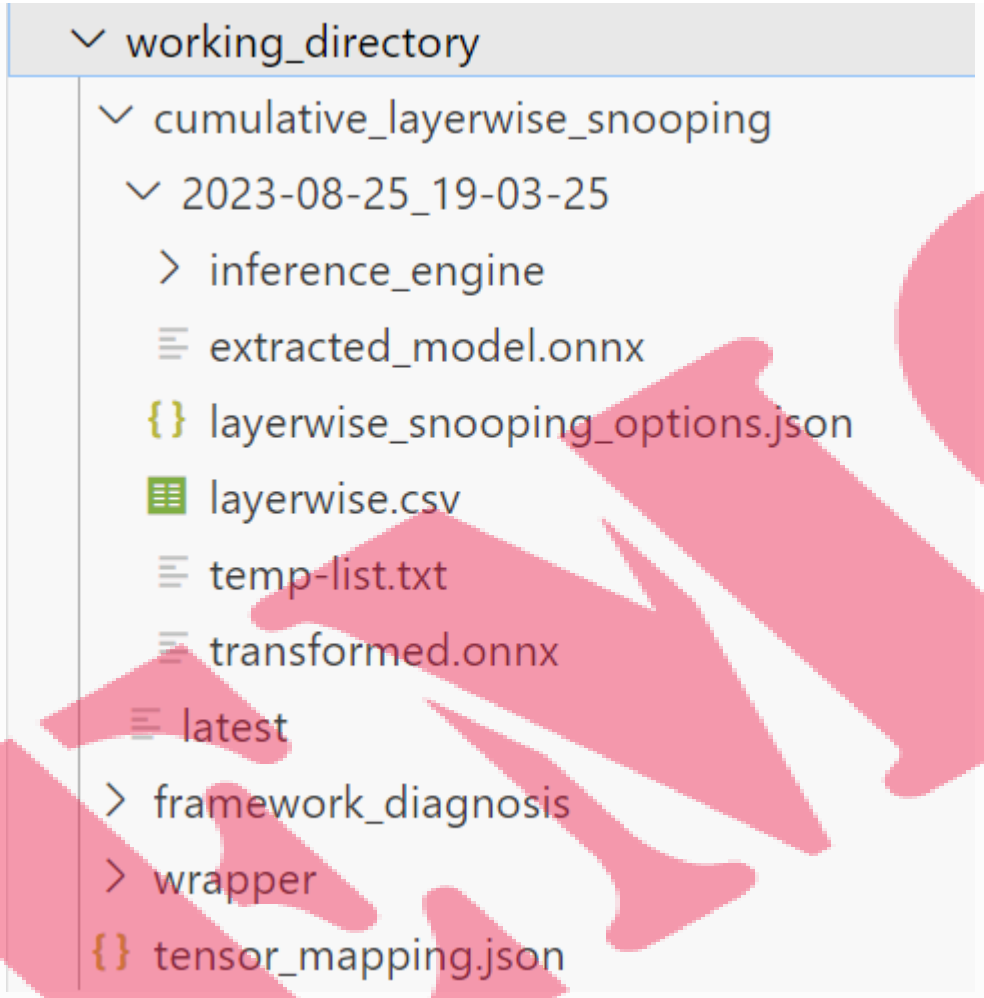
```
Command for Cumulative-layerwise using DSP backend:

qnn-accuracy-debugger \
    --framework onnx \
    --runtime dspv73 \
    --model_path /local/mnt/workspace/models/vit/vit_base_16_224.onnx \
    --input_tensor "input.1" 1,3,224,224 /local/mnt/workspace/models/vit/000000039769_1_3_224_224.
    --output_tensor 1597 \
    --architecture x86_64-linux-clang \
    --input_list /local/mnt/workspace/models/vit/list.txt \
    --default_verifier CosineSimilarity \
    --offline_prepare \
    --debugging_algorithm cumulative-layerwise
    --engine QNN \
    --verbose
```

```
Command for Cumulative-layerwise using HTP emulation on x86 host:

qnn-accuracy-debugger \
    --framework onnx \
    --runtime htp \
```

```
--model_path /local/mnt/workspace/models/vit/vit_base_16_224.onnx \
--input_tensor "input.1" 1,3,224,224 /local/mnt/workspace/models/vit/000000039769_1_3_224_224.
--output_tensor 1597 \
--architecture x86_64-linux-clang \
--input_list /local/mnt/workspace/models/vit/list.txt \
--default_verifier CosineSimilarity \
--offline_prepare \
--debugging_algorithm cumulative-layerwise
--engine QNN \
--verbose
```

Output

The program creates output directories framework_diagnosis, cumulative_layerwise_snooping and wrapper directories as below



- framework_diagnosis directory contains timestamped directory that contains the intermediate layers outputs stored in .raw format just as mentioned in Framework Diagnosis step.
- cumulative_layerwise_snooping directory contains intemediate outputs obtained from inference engine step stored in separate directories with respective layer names. Also it contains final report named cumulative_layerwise.csv which contains verifier scores for each layer. User can identify layers with most deviating scores as problematic nodes.
- Wrapper directory consists a log.txt where user can refer entire logs for the whole run.

O/P Name	Status	Layer Type	Shape	Activations (Min,Ma	mse	Orig Outputs	Info
317	part	Clip	[1, 32, 112, 112]	['0.0', '2.962', '0.244']	0.0463	{'473': {'mse': 15.393}	mse: -
320	part	Clip	[1, 32, 112, 112]	['0.0', '6.0', '0.321']	0.0011	{'473': {'mse': 0.3995}	mse: -
322	part	BatchNormalization	[1, 16, 112, 112]	['-6.109', '5.047', '-0.0	0.0004	{'473': {'mse': 0.0969}	mse: -
325	part	Clip	[1, 96, 112, 112]	['0.0', '4.287', '0.154']	0	{'473': {'mse': 0.1097}	mse: -
328	part	Clip	[1, 96, 56, 56]	['0.0', '3.188', '0.098']	0.0001	{'473': {'mse': 0.0982}	mse: -
330	part	BatchNormalization	[1, 24, 56, 56]	['-2.589', '3.013', '-0.0	0.0001	{'473': {'mse': 0.0712}	mse: -

Understanding the cumulative-layerwise report

At the end of cumulative-layerwise run, the tool generates .csv with below information for each layer

Column	Description
O/P Name	Output name of the current layer.
Status	**If empty, indicates normal execution.Other possible values:** • skip - This layer was not debugged as requested by the user. • part - Due to the mismatch at this layer, the model was partitioned after this layer • err_part - error occured while partitioning model at that layer. • err_con - coverter error occurred at this layer. • err_lib - lib-generator error occurred at this layer. • err_cntx - context-bin-generator error occurred at this layer. • err-exec - Failed to execute the compiled model at this layer. • err-compare - Failed to compare the backend output of this layer with reference.
Layer Type	Type of the current layer.
Shape	Shape of this layer's output.
Activations	The Min, Max and Median of the outputs at this layer taken from reference execution.
<Verifier name>	Absolute verifier value of the current layer compared to reference platform.
Orig outputs	Displays the original outputs verifier score observed when the model was run with the current

	layer output enabled starting from the last partitioned layer.
Info	Displays information for the output verifiers, if the values are abnormal.

```
Command for Layerwise:

qnn-accuracy-debugger \
    --framework onnx \
    --runtime dspv73 \
    --model_path /local/mnt/workspace/models/vit/vit_base_16_224.onnx \
    --input_tensor "input.1" 1,3,224,224 /local/mnt/workspace/models/vit/000000039769_1_3_224_224.
    --output_tensor 1597 \
    --architecture x86_64-linux-clang \
    --input_list /local/mnt/workspace/models/vit/list.txt \
    --default_verifier CosineSimilarity \
    --offline_prepare \
    --debugging_algorithm layerwise
    --golden_directory /local/mnt/workspace/aimet_layer_output_dump/outputs/layer_outputs_0/
    --quantization_overrides /local/mnt/workspace/layer_output_dump/vit_base_16_224.encodings
    --engine QNN \
    --verbose
```

Output

The program creates layerwise_snooping and wrapper output directories as well as framework_diagnosis if a golden reference is not provided (like described for cumulative-layerwise).

- layerwise_snooping directory – Contains each single layer model outputs obtained from the inference engine stage stored in separate directories and the final report named layerwise.csv which contains verifier scores for each layer model. Users can identify layers with the most deviating scores as problematic nodes.
- wrapper directory – Contains log.txt which stores the full logs for the run.
- The output .csv is similar to the cumulative-layerwise output, but the original outputs column will not be present in layerwise snooping, since we are not dealing with final outputs of the model.

Debugging Accuracy issue with Quantized model using Cumulative Layerwise Snooping

- With quantized models, it is expected to have some mismatch at most data intensive layers - arising due to quantization error.
- The debugger can be used to identify operators which are most sensitive with high verifier score and run those at higher precision to improve overall accuracy.
- The sensitivity is determined by the verifier score seen at that layer regarding the reference platform (like ONNXRT).
- Note that Cumulative-layerwise debugging takes considerable time as the partitioned model shall be quantized and compiled at every layer that does not have a 100% match with reference.
- Below is one strategy to debug larger models:
    - Run Oneshot-layerwise on the model which helps to identify the starting point of sensitivity in the model.
    - Run Cumulative-layerwise at different parts of the model using start-layer and end-layer options (if the model has 100 nodes, use start layer at starting node from Oneshot-layerwise run and end layer at the 25th node for run 1, start layer at 26th and end layer at 50th node for run 2, start layer at 51st node and end layer at 75th node for run 3 .. and so on).The final reports of all runs help to identify the most sensitive layers in the model. Let's say node A,B,C have high verifier scores which indicates high sensitivity
        - Run the original model with those specific layers (A/B/C - one at a time or combinations) in FP16 and observe the improvement in accuracy.

Debugging Accuracy issue for models exhibiting Accuracy discrepancy between golden reference (for ex. - AIMET/framework runtime output) vs target output using Layerwise Snooping

- **One of the popular usecase for layerwise snooping is debugging accuracy difference between AIMET vs target**
    - Though we are creating an exact simulation of hardware using tools like AIMET, still it is expected to have a very minute mismatch due to environment differences. This can be because simulation executes on GPU FP32 kernels and is simulating noise rather than actual execution on integer kernels in the case of hardware execution.

- If we have a higher deviation between simulation and hardware, then layerwise snooping could be used to point out to the nodes having higher deviations. The nodes showing higher deviation as per layerwise.csv can be identified as the erroneous nodes.

- Other usecases include debugging Framework runtime's FP32 output vs target INT16 output deviations.

qnn-platform-validator

qnn-platform-validator checks the QNN compatibility/capability of a device. The output is saved in a CSV file in the "output" directory, in a csv format. Basic logs are also displayed on the console.

```
DESCRIPTION:
------------
Helper script to set up the environment for and launch the qnn-platform-
validator executable.

REQUIRED ARGUMENTS:
-------------------
--backend           <BACKEND>           Specify the backend to validate: <gpu>, <dsp>
                                        <all>.

--directory         <DIR>               Path to the root of the unpacked SDK directory containing
                                        the executable and library files

--dsp_type          <DSP_VERSION>       Specify DSP variant: v66 or v68

OPTIONALS ARGUMENTS:
--------------------
--buildVariant      <TOOLCHAIN>         Specify the build variant
                                        aarch64-android or aarch64-windows-msvc to be validated.
                                        Default: aarch64-android

--testBackend                           Runs a small program on the runtime and Checks if QNN is s
                                        backend.

--deviceId          <DEVICE_ID>         Uses the device for running the adb command.
                                        Defaults to first device in the adb devices list..

--coreVersion                           Outputs the version of the runtime that is present on the

--libVersion                            Outputs the library version of the runtime that is present
```

```
--targetPath          <DIR>           The path to be used on the device.
                                      Defaults to /data/local/tmp/platformValidator

--remoteHost          <REMOTEHOST>    Run on remote host through remote adb server.
                                      Defaults to localhost.

--debug                               Set to turn on Debug log
```

Additional details:

- The following files need to be pushed to the device for the DSP to pass validator test.
  Note that the stub and skel libraries are specific to the DSP architecture version(e.g., v73):

```
// Android
bin/aarch64-android/qnn-platform-validator
lib/aarch64-android/libQnnHtpV73CalculatorStub.so
lib/hexagon-${DSP_ARCH}/unsigned/libCalculator_skel.so

// Windows
bin/aarch64-windows-msvc/qnn-platform-validator.exe
lib/aarch64-windows-msvc/QnnHtpV73CalculatorStub.dll
lib/hexagon-${DSP_ARCH}/unsigned/libCalculator_skel.so
```

- The following example pushes the aarch64-android variant to /data/local/tmp/platformValidator

```
adb push $SNPE_ROOT/bin/aarch64-android/snpe-platform-validator /data/local/tmp/platformVal
adb push $SNPE_ROOT/lib/aarch64-android/ /data/local/tmp/platformValidator/lib
adb push $SNPE_ROOT/lib/dsp /data/local/tmp/platformValidator/dsp
```

qnn-profile-viewer

The **qnn-profile-viewer** tool is used to parse profiling data that is generated using **qnn-net-run**. Additionally, the same data can be saved to a csv file.

```
usage: qnn-profile-viewer --input_log PROFILING_LOG [--help] [--output=CSV_FILE] [--extract_opaque

Reads a profiling log and outputs the contents to stdout

Note: The IPS calculation takes the following into account: graph execute time, tensor file IO tim

required arguments:
  --input_log                   PROFILING_LOG
                                Profiling log file.


optional arguments:
  --output                      PATH
                                Output file with processed profiling data. File formats vary dep
                                (see --reader). If not provided, not output is created.

  --help                        Displays this help message.

  --reader                      CUSTOM_READER_SHARED_LIB
                                Path to a reader library. If not specified, the default reader o

  --schematic                   SCHEMATIC_BINARY
                                Path to the schematic binary file.
                                Please note that this option is specific to the QnnHtpOptracePro

  --version                     Displays version information.

  --extract_opaque_objects      Specifies that the opaque objects will be dumped to output files
```

qnn-netron (Beta)

Overview

QNN Netron tool is making model debugging and visualization less daunting. qnn-netron is an extension of the netron graph tool. It provides for easier graph debugging and convenient runtime information. There are currently two key functionalities of the tool:

1. The Visualize section allows customers to view their desired models after using the QNN Converter by importing the JSON representation of the model
2. The Diff section allows customers to run networks of their choosing on different runtimes in order to compare network accuracy and performance

Launching Tool

Dependencies

The QNN netron tool leverages electron JS framework for building GUI frontend and depends on npm/node_js to be available in system. Additionally, python libraries for accuracy analysis are required by backend of tool. A convenient script is available in the QNN SDK to download necessary dependencies for building and running the tool.

```
# Note: following command should be run as administrator/root to be able to install system librari
$ sudo bash ${QNN_SDK_ROOT}/bin/check-linux-dependency.sh
$ ${QNN_SDK_ROOT}/bin/check-python-dependency
```

Launching Application

qnn-netron script is used to be able to launch the QNN Netron application. This script:

1. Clones vanilla netron git project
2. Applies custom patches for enabling Netron for QNN
3. Build the npm project
4. Launches application

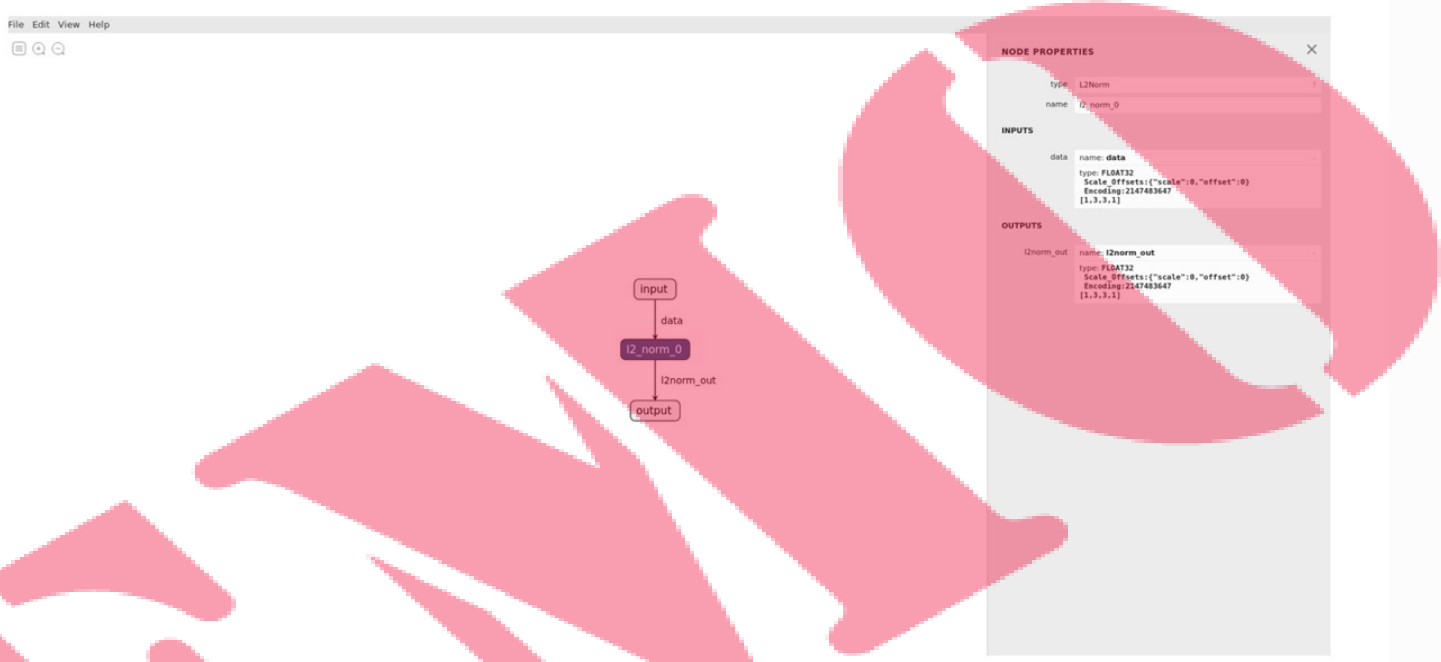
```
$ qnn-netron -h
usage: qnn-netron [-h] [-w <working_dir>]
Script to build and launch QNN Netron tool for visualizing and running analysis on Qnn Models.

Optional argument(s):
 -w <working_dir>                       Location for building QNN Netron tool. Default: current_dir


# To build and run application use
$ qnn-netron -w <my_working_dir>
```

QNN Netron Visualize Deep Dive

First, the user is prompted to open a JSON file that represents their converted model. This JSON comes from the converter tool. Please refer to this Overview for more details.

Once the file is loaded into the tool, the graph should be displayed in the UI as shown below:

After loading in the model, the user can click on any of the nodes and a side pop-up section will display node information such as the type and name as well as vital parameter information such as inputs and outputs (datatypes, encodings, and shapes)

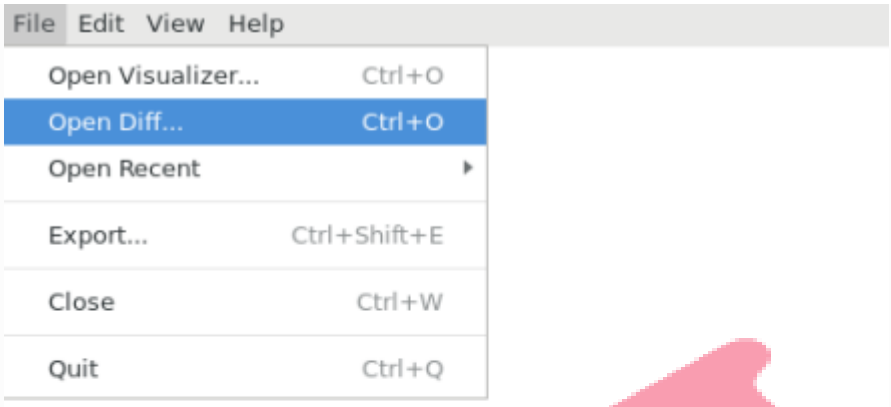


Netron Diff Customization Deep Dive

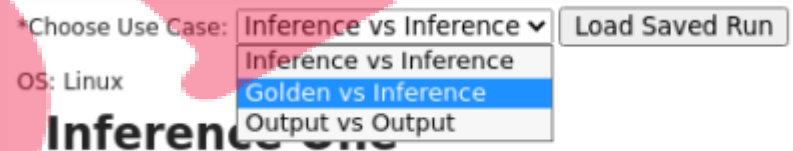
Limitations

1. Diff Tool comparison between source framework goldens only works for framework goldens that are spatial first axis order. (NHWC)
2. For usecases where source framework golden is used for comparison, Diff Tool is only tested to work for tensorflow and tensorflow variant frameworks.

In order for the user to open the Diff Customization tool, they can either click file and then “Open Diff...” or on tool startup by clicking “Diff...” as shown below:



Upon launch of the Diff Customization tool, at the top, the user is prompted to select a use case for the tool. There are 3 options to choose from:

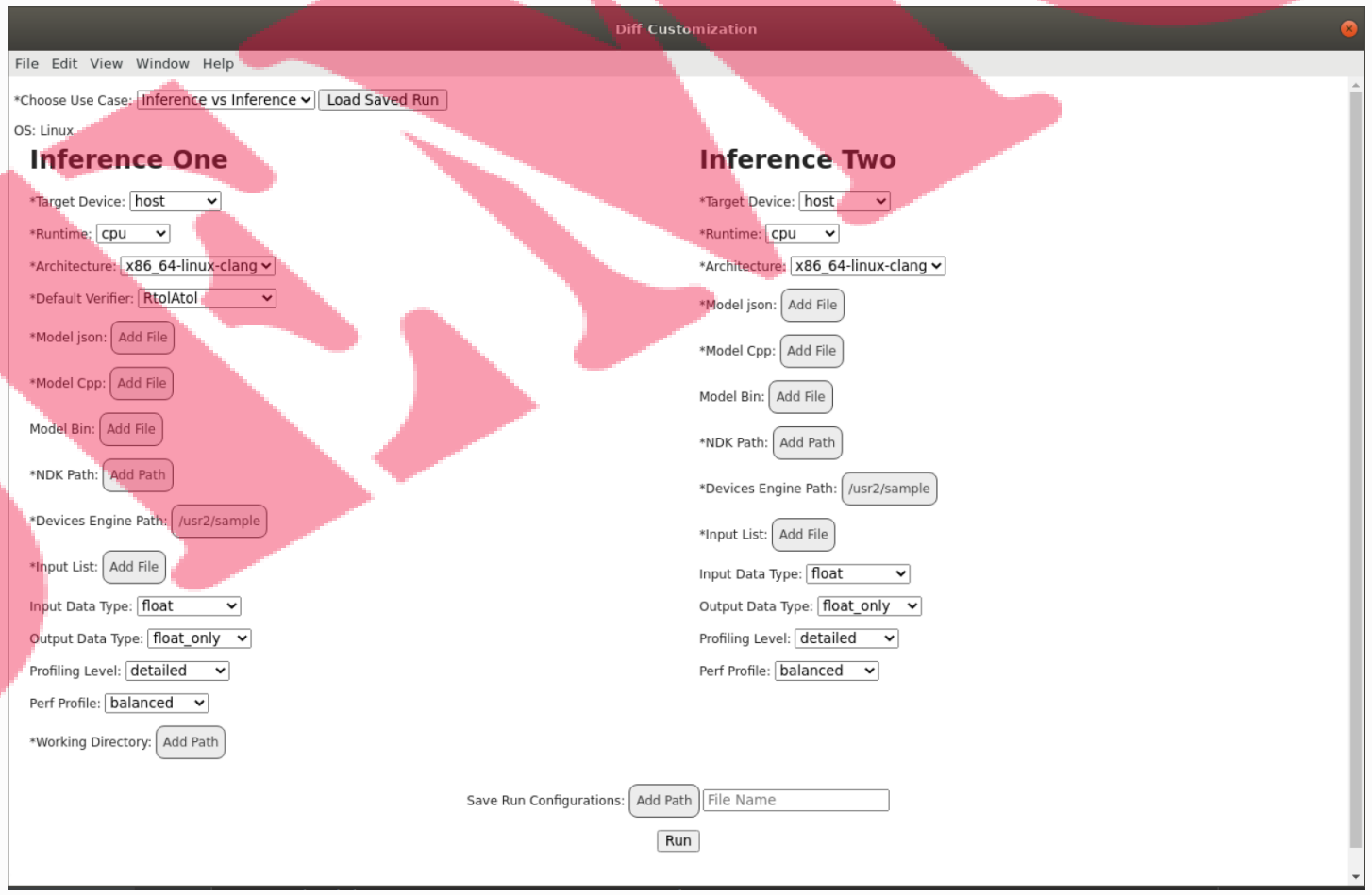


For the purposes of this documentation, only inference vs inference will be detailed. The setup procedure for the other use cases is similar. The other two use cases are explained below:

1. Golden vs Inference: Used to test inference run using goldens from a particular ML framework and comparing against the output of a QNN backend
2. Output vs Output: Used to test existing inference results against ML framework goldens OR used to test differences between two existing inference results
3. Inference Vs Inference: Used to test inference between two converted QNN models or the same QNN model on different QNN backends

Inference vs Inference

If this use case is selected, the user is presented with various form fields for the purposes of running two jobs asynchronously with the option of choosing different runtimes for each QNN network being run.



A more detailed view of what the user is prompted is displayed below:

*Choose Use Case: Inference vs Inference | Load Saved Run

OS: Linux

Inference One

*Target Device: host

*Runtime: cpu

*Architecture: x86_64-linux-clang

*Default Verifier: RtolAtol

*Model json: /usr2/sample/model_net.json

*Model Cpp: /usr2/sample/model.cpp

Model Bin: N/A

*NDK Path: Add Path

*Devices Engine Path: /usr2/sample/qnn_sdk

*Input List: /usr2/sample/input_list.txt

Input Data Type: float

Output Data Type: float_only

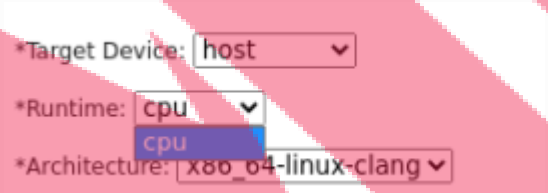
Profiling Level: detailed

Perf Profile: balanced

In order to execute the networks, the user has two options:

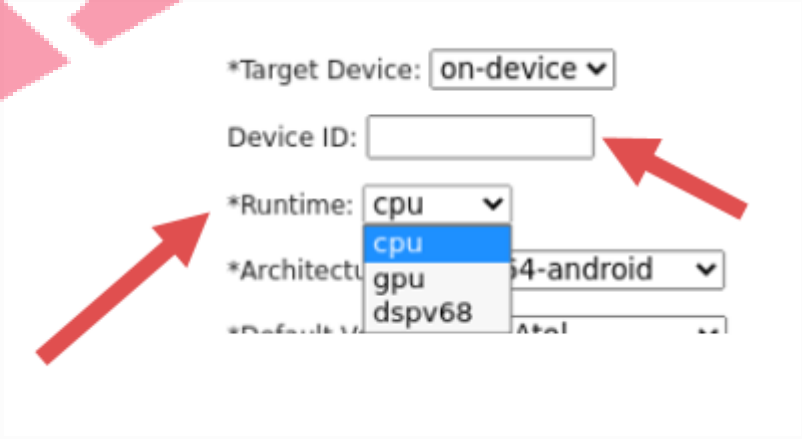
Running on Host machine

When the Target Device is selected as “host”, the user can only use the CPU as a runtime. In addition, the user can only select “x86_64-linux-clang” as the architecture in this use case.



Running On-Device

When the Target Device is selected as “on-device”, a Device ID is required to connect to the device via adb. Thereafter, the user can select any of the three QNN backend runtimes available (CPU, GPU, or DSPv[68, 69, 73]) and the user can select architecture “aarch64-android”



After choosing the desired target device and runtime configurations, the rest of the fields are explained in detail below:

> **Note**
>
> Users are able to click again and change the location to any of the path fields

Setup Parameters	Configurations to Select
The options for what verifier to run on the outputs of the model are (See Note below table for custom verifier (accuracy + performance) thresholds and see table below for providing custom accuracy verifier hyperparameters):	RtolAtol, AdjustedRtolAtol, TopK, MeanIOU, L1Error, CosineSimilarity, MSE, SQNR
Model JSON	upload <model>_net.json file that was outputted from the QNN converters.
Model Cpp	upload <model>.cpp that was outputted from the QNN converters.
Model Bin	upload <model>.bin that was outputted from the QNN converters.
NDK Path	upload the path to your Android NDK
Devices Engine Path	upload the path to the top-level of the unzipped qnn-sdk
Input List	provide a path to the input file for the model
Save Run Configurations	provide a location where the inference and runtime results from the Diff customization tool will be stored

Note

Users have the option of providing a custom accuracy and performance verifier threshold when running diff. A custom accuracy verifier threshold can be provided for any of the accuracy verifiers. By default the verifier thresholds are 0.01. The custom thresholds can be provided in the text boxes labelled “Accuracy Threshold” and “Perf Threshold”.

Users now have the option to enter accuracy verifier specific hyperparameters inside textboxes. The Default Values are displayed inside the text-boxes and can be customized as per user needs. The table below highlights the hyperparameters for each verifier that can be customized.

Verifier	Hyperparameters
AdjustedRtolAtol	Number of Levels
RtolAtol	Rtol Margin, Atol Margin
Topk	K, Ordered
MeanIOU	Background Classification
L1Error	Multiplier, Scale
CosineSimilarity	Multiplier, Scale
MSE (Mean Square Error)	N/A
SQNR (Signal-To-Noise Ratio)	N/A

Below is an example of what the fields should look like once filled to completion:

*Choose Use Case: Inference vs Inference | Load Saved Run

OS: Linux

Inference One

*Target Device: host
*Runtime: cpu
*Architecture: x86_64-linux-clang
*Default Verifier: RtolAtol
*Model json: /usr2/sample/model_net.json
*Model Cpp: /usr2/sample/model.cpp
Model Bin: N/A
*NDK Path: Add Path
*Devices Engine Path: /usr2/sample/qnn_sdk
*Input List: /usr2/sample/input_list.txt
Input Data Type: float
Output Data Type: float_only
Profiling Level: detailed
Perf Profile: balanced
*Working Directory: /usr2/sample

Inference Two

*Target Device: host
*Runtime: cpu
*Architecture: x86_64-linux-clang
*Model json: /usr2/sample/model_net.json
*Model Cpp: /usr2/sample/model.cpp
Model Bin: N/A
*NDK Path: Add Path
*Devices Engine Path: /usr2/sample/qnn_sdk
*Input List: /usr2/sample/input_list.txt
Input Data Type: float
Output Data Type: float_only
Profiling Level: detailed
Perf Profile: balanced

Save Run Configurations: /usr2/sample run_configs

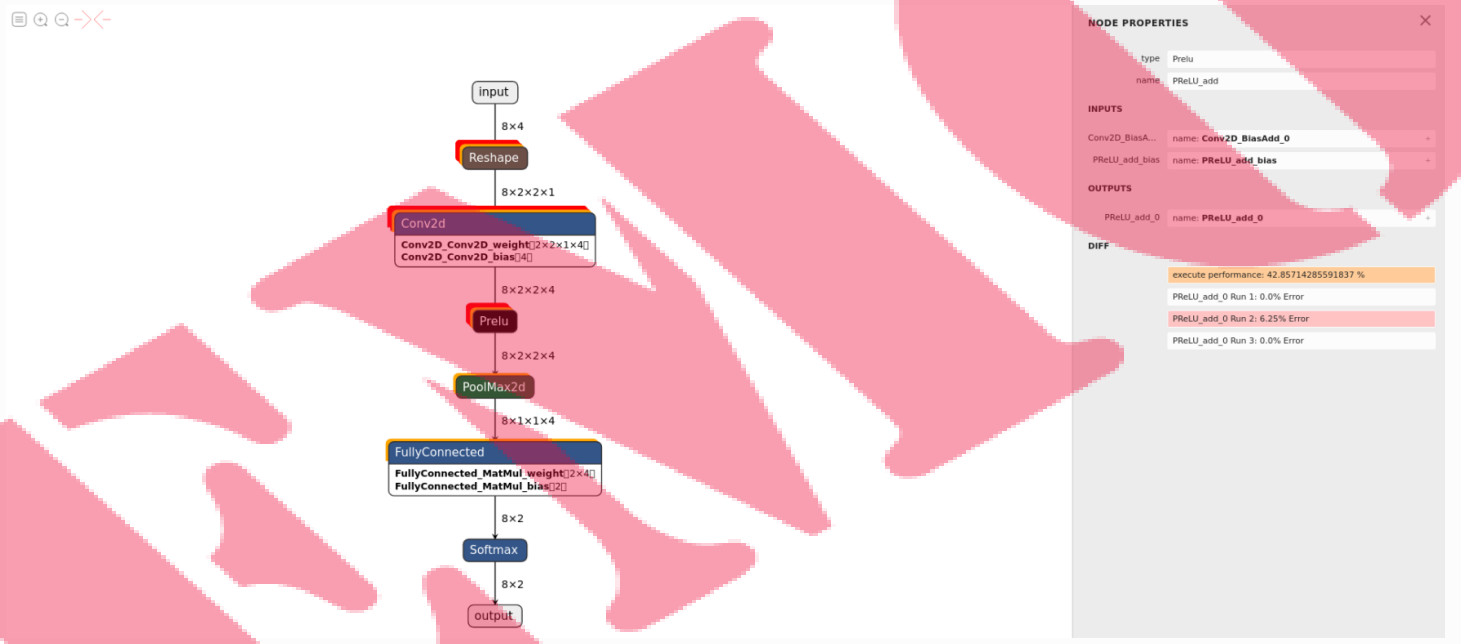
Run

After running the Diff Customization tool, the output directories/files should be present in the working directory file path provided in the last field

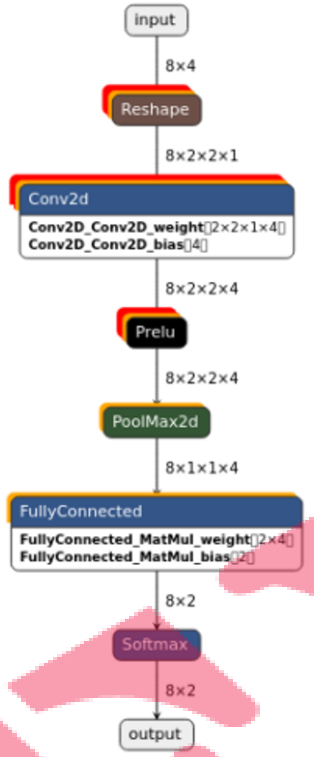
csv_outputs	2/17/2022 3:22 PM	File folder
inference_engine	2/17/2022 3:22 PM	File folder
intermediate_files	2/17/2022 3:20 PM	File folder
verification	2/17/2022 3:22 PM	File folder

Results and Outputs:

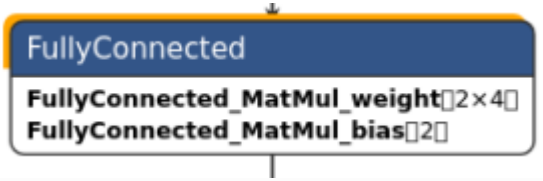
After pressing the Run button as mentioned above, the visualization of the network should pop-up. Nodes will be highlighted if there are any accuracy and/or performance variations. Clicking on each node will show more information about the accuracy and performance diff information as shown below.



Performance and Accuracy Diff Visualizations:

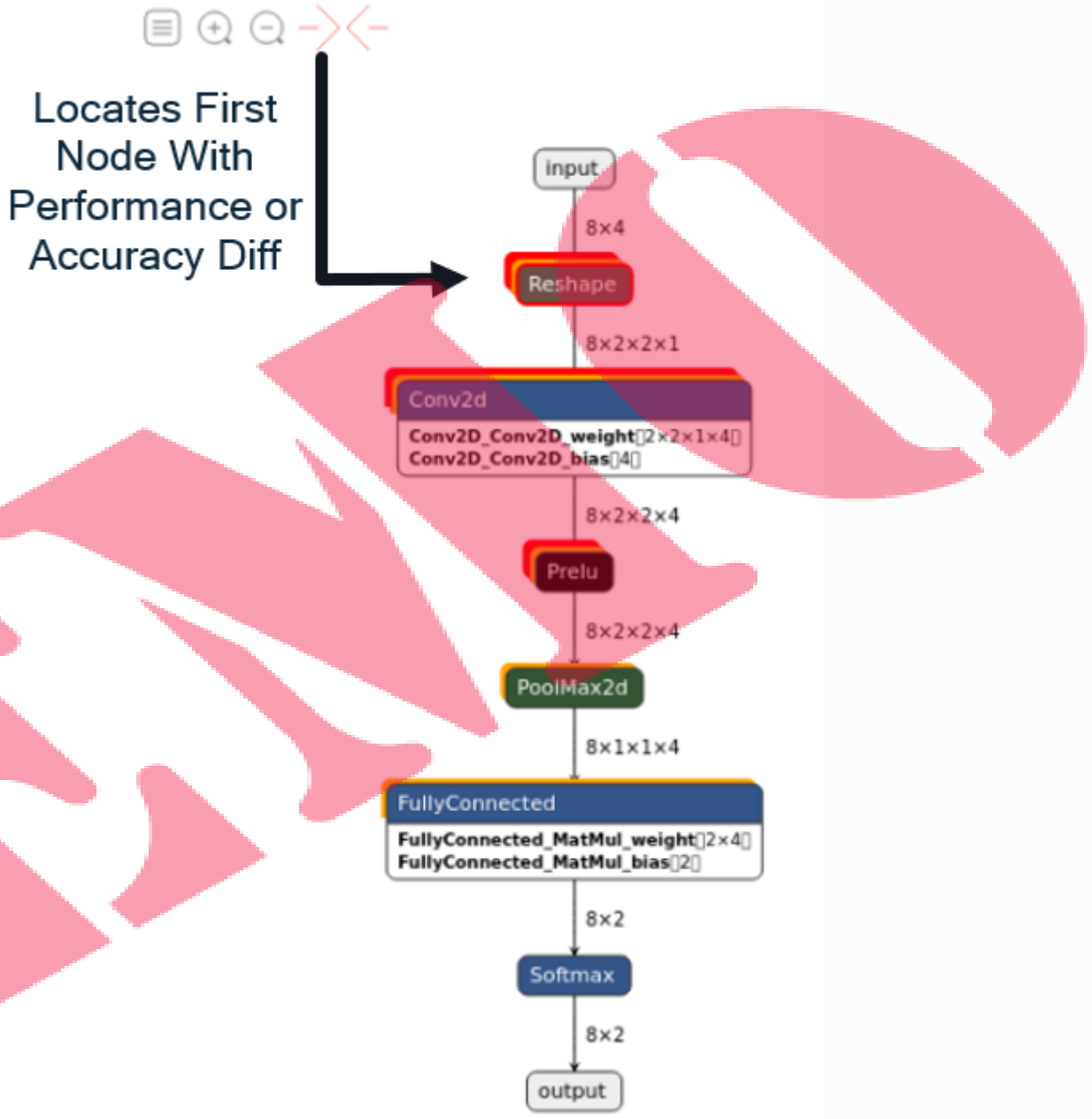


As seen above, the performance and accuracy diff information is shown under the Diff section of any given node. The color of the node boundary in the viewer represents whether a performance or accuracy error (above the default verifier threshold of 0.01) was reported. For example, in the Conv2d node shown below, there are two boundaries of orange and red indicating that this node has both an accuracy and performance difference across the runs. The FullyConnected node shown only has a yellow boundary indicating that only a performance difference was found.



QNN Netron Diff Navigation

QNN Netron has the ability to locate the first node in the graph with any performance or accuracy diffs. When the user clicks on the next and previous arrows, the visualization of the graph will zoom into the desired node with the first performance or accuracy difference. This makes model debugging much easier for larger models as the user doesn't have to look for the nodes themselves to find where the network performance and accuracy errors starts to diverge.



qnn-context-binary-utility

The **qnn-context-binary-utility** tool validates and serializes the metadata of context binary into a json file. This json file can then be used for inspecting the context binary aiding in debugging. A QNN

context can be serialized to binary using QNN APIs or qnn-context-binary-generator tool.

```
usage: qnn-context-binary-utility --context_binary CONTEXT_BINARY_FILE --json_file JSON_FILE_NAME

Reads a serialized context binary and validates its metadata.
If --json_file is provided, it outputs the metadata to a json file

required arguments:
  --context_binary  CONTEXT_BINARY_FILE
                    Path to cached context binary from which the binary info will be extracted
                    and written to json.

  --json_file       JSON_FILE_NAME
                    Provide path along with the file name <DIR>/<FILE_NAME> to serialize
                    context binary info into json.
                    The directory path must exist. File with the FILE_NAME will be created at DIR.

optional arguments:
  --help          Displays this help message.

  --version       Displays version information.
```

Accuracy Evaluator plugins

File-based plugins

This section lists the built-in file-based plugins.

Dataset plugins

create_squad_examples - Extracts examples from given squad dataset file and save them to a file.

Parameters	Description	Type	Default
squad_version	Squad version 1 or 2	Integer	1

filter_dataset - Filters the dataset including the input list, calibration and annotation files.

Parameters	Description	Type	Default
max_inputs	Maximum number of inputs in inputlist to be considered for execution	Integer	Mandatory
max_calib	Maximum number of inputs in calibration to be considered for execution	Integer	Mandatory
random	Shuffles the inputlist and calibration files	Boolean	False

gpt2_tokenizer - Tokenizes data from files using GPT2TokenizerFast.

Parameters	Description	Type	Default
vocab_file	Path to the vocabulary file	String	Mandatory
merges_file	Path to the merges file	String	Mandatory
seq_length	Sequence length for the generated model inputs	Integer	Mandatory
past_seq_length	Sequence length for the "past" inputs	Integer	Mandatory
past_shape	Shape of the 'past' inputs	List	
num_past	Number of 'past' inputs	Integer	0

split_txt_data - Saves individual text files for each line present in the given input text file.

Preprocessing plugins

centernet_preproc - Performs preprocessing on CenterNet dataset examples.

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 416,416	String	Mandatory
scale	Scale factor for image	Float	1.0
fix_res	Resolution of the image	Boolean	True

pad	Image padding	Integer	0

convert_nchw - Transposes WHC to CHW or CHW to WHC and adds an extra N dimension.

Parameters	Description	Type	Default
expand-dims	Add the Nth dimension	Boolean	True

create_batch - Concatenates raw input files into a single file using numpy.

Parameters	Description	Type	Default
delete_prior	To delete prior unbatched data to save space	Boolean	True
truncate	If num inputs are not a multiple of batch size, then truncate left over inputs in the last batch or not	Boolean	False

crop - Center crops an image to the given dimensions using numpy or torchvision based on the library parameter.

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
library	Python library used to crop the given input; valid values are: numpy | torchvision	String	numpy
typecasting_required	To convert final output to numpy or not. Note: This option is specific to torchvision library	Boolean	True

expand_dims - Adds the N dimension for images, e.g., HWC to NHWC.

image_transformers_input - Creates input files with image and/or text for image transformer models like ViT and CLIP.

Parameters	Description	Type	Default
dims	Expected processed output dimension in CHW format	String	Mandatory
num_base_class	Number of base classes in classification; used in the scenario where text input is also provided	Integer	Total classes available
num_prompt	Number of prompts for text classes; used in the scenario where text input is also provided	Integer	Total classes available
image_only	Data type of raw data	Boolean	False

normalize - Normalizes input per the given scheme; data must be of NHWC format.

Parameters	Description	Type	Default
library	Python library used to crop the given input; valid values are: numpy | torchvision	String	numpy
norm	Normalization factor, all values divided by norm	float32	255
means	Dictionary of means to be subtracted, e.g., {“R”:0.485, “G”:0.456, “B”:0.406}	RGB dictionary	{“R”:0, “G”:0, “B”:0}
std	Dictionary of std-dev for rescaling the values, e.g., {“R”:0.229, “G”:0.224, “B”:0.225}	RGB dictionary	{“R”:1, “G”:1, “B”:1}
channel_order	Channel order to specify means and std values per channel - RGB | BGR	String	RGB
normalize_first	To perform normalization before or after mean subtraction and standard deviation. normalize_first=True means perform normalization before. Note: torchvision library does not use this option	Boolean	True

typecasting_required	To convert final output to numpy or not. Note: This option is specific to the Torchvision library	Boolean	True
pil_to_tensor_input	To convert input to tensor before normalization. Note: This option is specific to the Torchvision library	Boolean	True

onmt_preprocess - Performs preprocessing on WMT dataset for FasterTransformer OpenNMT model

Parameters	Description	Type	Default
vocab_path	Path to OpenNMT model vocabulary file (pickle file)	String	Mandatory
src_seq_len	The maximum total input sequence length	Integer	128
skip_sentencepiece	Skip sentencepiece encoding	Boolean	True
sentencepiece_model_path	Path to sentencepiece model for WMT dataset (mandatory when "skip_sentencepiece" is False)	String	None

pad - Image padding with constant pad size or based on target dimensions

Parameters	Description	Type	Default
type	**Type of padding. Valid options:** • constant: Add padding of constant sides on 4 sides (pad_size must be provided) • target_dims: Add padding based on difference in image size and target size (dims param must be provided)	String	Mandatory
dims	Height and width comma delimited, e.g., 416,416 for 'target-dims' type of padding	String	Mandatory
pad_size	Size of padding for 'constant' type of padding	Integer	None

img_position	Parameter to specify position of image, either 'center' or 'corner' (top-left). Padding is added accordingly. Currently used for 'target_dims' type padding	String	center
color	Padding value for all planes	Integer	114

resize - Resizes an image using the specified library parameter: cv2(Default), pillow or torchvision

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
library	Python library to be used for resizing a given input; valid values are: opencv | pillow | torchvision	String	opencv
channel_order	Convert image to specified channel order. At present this parameter only takes the 'RGB' value	String	RGB
interp	**Interpolation Type. Options:** • bilinear (supported by opencv, Torchvision, pillow) • area (supported by opencv only) • nearest (supported by opencv, Torchvision, pillow) • bicubic (supported by Torchvision, pillow) • box (supported by pillow only) • hamming (supported by pillow only) • lanczos (supported by pillow only)	String	For opencv and torchvision: bilinear For pillow: bicubic
type	**Type of resize to be done. Note: Torchvision does not use this option. Options:**	String	auto-resize

	• letterbox : Used for YOLO models. • imagenet : Scale followed by resize. • aspect_ratio : Resize while keeping aspect ratio. • None : The default behavior is to auto-resize the image to the target dims.		
resize_before_typecast	To resize before or after conversion to target datatype e.g., fp32	Boolean	True
typecasting_required	To convert final output to numpy or not. Note: This option is specific to the Torchvision library	Boolean	True
mean	Dictionary of means to be subtracted, e.g., {“R”:0.485, “G”:0.456, “B”:0.406}. Note: This option is specific to the Tensorflow library	RGB dictionary	{“R”:0, “G”:0, “B”:0}
std	Dictionary of std-dev for rescaling the values, e.g., {“R”:0.229, “G”:0.224, “B”:0.225}. Note: This option is specific to the Tensorflow library	RGB dictionary	{“R”:0, “G”:0, “B”:0}
normalize_before_resize	To perform normalization before or after mean subtraction and standard deviation. Note: This option is specific to the Tensorflow library	Boolean	False
crop_before_resize	To perform cropping before resize. Note: This option is specific to the Tensorflow library	Boolean	False

squad_read - Reads the SQuAD dataset JSON file. Preprocesses the question-context pairs into features for language models like BERT-Large

Parameters	Description	Type	Default
vocab_path	Path for local directory containing vocabulary files	String	Mandatory
max_seq_length	The maximum total input sequence length after WordPiece tokenization. Sequences longer than this will be truncated, and sequences shorter than this will be padded	Integer	384
max_query_length	The maximum number of tokens for the question. Questions longer than this will be truncated to this length	Integer	64
doc_stride	When splitting up a long document into chunks, how much stride to take between chunks	Integer	128
packing_strategy	Set this flag when using packing strategy for bert based models	Boolean	False
max_sequence_per_pack	The maximum number of sequences which can be packed together	Integer	3
mask_type	This can take either of three values - 'None', 'Boolean' or 'Compressed' depending on the masking to be done on input_mask	String	None
compressed_mask_length	Set this value if mask_type is set to compressed	Integer	None

Postprocessing plugins

bert_predict - Predicts answers for a SQuAD dataset given start and end logits.

Parameters	Description	Type	Default
vocab_path	Path for a local directory containing vocabulary files	String	Mandatory
max_seq_length	The maximum total input sequence length after WordPiece tokenization. Sequences longer than this will be truncated, and sequences shorter than this will be padded (optional if preprocessing is run)	Integer	384

doc_stride	When splitting up a long document into chunks, how much stride to take between chunks (optional if preprocessing is run)	Integer	128
max_query_length	The maximum number of tokens for the question. Questions longer than this will be truncated to this length (optional if preprocessing is run)	Integer	64
n_best_size	The total number of n-best predictions to generate in the post.json output file	Integer	20
max_answer_length	The maximum length of an answer that can be generated. This is needed because the start and end predictions are not conditioned on one another	Integer	30
packing_strategy	This flag is set to True if using packing strategy	Boolean	False

centerface_postproc - Processes the inference outputs to parse detections and generates a detections file for the metric evaluator. Used for processing CenterFace face detector.

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
dtypes	List of datatypes to be used for bounding boxes, scores, and labels (in order), e.g., [float32, float32, int64]. Defaults to the datatypes fetched from the 'outputs_info' for the model's config.yaml	List	Datatypes from the outputs_info section of the model config.yaml
heatmap_threshold	User input for heatmap threshold	Float	0.05
nms_threshold	User input for nms threshold	Float	0.3

centernet_postprocess - Processes the inference outputs to parse detections and generate a detections file for the metric evaluator. Used for processing CenterNet detector.

Parameters	Description	Type	Default

dtypes	List of datatypes (at least 3) to be used to infer outputs	String	Mandatory
output_dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
top_k	Top K proposals are given from the postprocess plugin	Integer	100
num_classes	Number of classes	Integer	1
score	Threshold to purify the detections	Integer	1

lprnet_predict - Used for LPRNET license plate prediction.

object_detection - Processes the inference outputs to parse detections and generate a detections file for metric evaluator

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
type	Type of post-processing (e.g., letterbox, stretch)	String	None
label_offset	Offset for the labels information	Integer	0
score_threshold	Threshold limit for the detection scores	Float	0.001
xywh_to_xyxy	Convert bounding box format from box center (xywh) to box corner (xyxy) format	Boolean	False
xy_swap	Swap the X and Y coordinates of bbox	Boolean	False
dtypes	List of datatypes used for bounding boxes, scores, and labels in order, e.g., [float32, float32, int64]. Defaults to the datatypes fetched from the 'outputs_info' for the model's config.yaml.	List	Datatypes from the outputs_info section of the model config.yaml
mask	Do postprocessing on mask	Boolean	False

mask_dims	Output dims of model. Provide this only if mask = True. E.g., 100,80,28,28	String	None
padded_outputs	Pad the outputs	Boolean	False
scale	Comma separated scale values	String	'1'
skip_padding	Skip padding while rescaling to original image shape	Boolean	False

onmt_postprocess - Performs preprocessing for OpenNMT model outputs

Parameters	Description	Type	Default
sentencepiece_model_path	Path to sentencepiece model for WMT dataset	String	Mandatory
unrolled_count	Upper limit on the unrolls required for the output (no. of output tokens to be considered for metric)	Integer	26
vocab_path	Path to OpenNMT model vocabulary file (pickle file), optional if preprocessing is run	String	None
skip_sentencepiece	Skip sentencepiece encoding, optional if preprocessing is run	Boolean	None

Metric plugins

bleu - Evaluates bleu score using sacrebleu library

Parameters	Description	Type	Default
round	Number of decimal places to round the result to	Integer	1

map_coco - Evaluates the mAP score 50 and 50:05:95 for COCO dataset

Parameters	Description	Type	Default

map_80_to_90	Mapping of classes in range 0-80 to 0-90	Boolean	False
segm	Flag to calculate mAP for mask	Boolean	False
keypoint_map	Flag to calculate mAP for keypoint	Boolean	False

perplexity - Calculates the perplexity metric. Model outputs are expected to be the logits of proper shape. Ground truth data is expected to be in tokenized format and in the form of token IDs. The ground truth will be automatically generated, if using the "gpt2_tokenizer" dataset plugin.

Parameters	Description	Type	Default
logits_index	Index of the logits output if the model has multiple outputs	Integer	0

precision - Calculates the precision metric, i.e., (correct predictions / total predictions). Ground truth data is expected in the format "filename <space> correct_text". The postprocessed model outputs are expected to be text files with just the "predicted_text".

Parameters	Description	Type	Default
round	Number of decimal places to round the result to	Integer	7
input_image_index	For multi input models, the index of image file in input file list csv	Integer	0

squad_em - Calculates the exact match for SQuAD v1.1 dataset predictions and ground truth.

squad_f1 - Calculates F1 score for SQuAD v1.1 dataset predictions and ground truth.

topk - Evaluates topk value by comparing results and annotations.

Parameters	Description	Type	Default
kval	Top k values, e.g., 1,5 evaluates top1 and top5	String	5

softmax_index	Index of the softmax output in the results file list	Integer	0
label_offset	Offset required in the labels' scores, e,g., if shape is 1x1001, then labels_offset=1	Integer	0
round	Number of decimal places to round the result to	Integer	3
input_image_index	For multi input models, the index of image file in input file list csv	Integer	0

widerface_AP - Computes average precision for easy, medium, and hard cases.

Parameters	Description	Type	Default
IoU_threshold	User input for IoU threshold	Float	0.4

Memory-based plugins

This section lists the built-in memory-based plugins.

Dataset plugins

create_squad_examples - Extracts examples from a given squad dataset file and saves them to a file.

Parameters	Description	Type	Default
squad_version	Squad version 1 or 2	Integer	1
max_inputs	Maximum number of inputs in inputlist to be considered for execution	Integer	-1 (Complete Dataset)
max_calib	Maximum number of inputs in calibration to be considered for execution	Integer	-1 (Complete Dataset)

filter_dataset - Filters the dataset including the input list, calibration, and annotation files.

Parameters	Description	Type	Default

max_inputs	Maximum number of inputs in inputlist to be considered for execution	Integer	Mandatory
max_calib	Maximum number of inputs in calibration to be considered for execution	Integer	Mandatory
random	Shuffles the inputlist and calibration files	Boolean	False

tokenize_wikitext_2 - Tokenizes wikitext-2 dataset into model inputs.

Parameters	Description	Type	Default
seq_length	Sequence length for the generated model inputs	Integer	Mandatory
tokenizer_name	Name of the tokenizer to be used for generating model inputs	String	Mandatory
past_shape	Shape of the 'past' inputs	List	0
num_past	Number of 'past' inputs	Integer	0
pos_id	Flag to configure whether position ids are required	Bool	True
mask_dtype	Data type of the mask used.	String	'float32'
cached_path	Path to cached tokenizer file (if available)	String	

split_txt_data - Saves individual text files for each line present in the given input text file.

Preprocessing memory plugins

centernet_preproc - Performs preprocessing on CenterNet dataset examples.

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 416,416	String	Mandatory
scale	Scale factor for image	Float	1.0

fix_res	Resolution of the image	Boolean	True
pad	Image padding	Integer	0

convert_nchw - Transposes WHC to CHW or CHW to WHC and adds an extra N dimension.

Parameters	Description	Type	Default
expand-dims	Add the Nth dimension	Boolean	True

create_batch - Concatenates raw input files into a single file using numpy.

crop - Center crops an image to the given dimensions using numpy or torchvision based on the library parameter.

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
library	Python library used to crop the given input; valid values are: numpy | torchvision	String	numpy
typecasting_required	To convert final output to numpy or not. Note: This option is specific to torchvision library	Boolean	True

expand_dims - Adds the N dimension for images, e.g., HWC to NHWC.

image_transformers_input - Creates input files with image and/or text for image transformer models like ViT and CLIP. (Note: This plugin requires Pillow package version:10.0.0)

Parameters	Description	Type	Default
dims	Expected processed output dimension in CHW format	String	Mandatory

image_only	Data type of raw data	Boolean	True

normalize - Normalizes input per the given scheme; data must be of NHWC format.

Parameters	Description	Type	Default
library	Python library used to crop the given input; valid values are: numpy | torchvision	String	numpy
norm	Normalization factor, all values divided by norm	float32	255.0
means	Dictionary of means to be subtracted, e.g., {"R":0.485, "G":0.456, "B":0.406}	RGB dictionary	{"R":0, "G":0, "B":0}
std	Dictionary of std-dev for rescaling the values, e.g., {"R":0.229, "G":0.224, "B":0.225}	RGB dictionary	{"R":1, "G":1, "B":1}
channel_order	Channel order to specify means and std values per channel - RGB | BGR	String	'RGB'
normalize_first	To perform normalization before or after mean subtraction and standard deviation. normalize_first=True means perform normalization before. Note: torchvision library does not use this option	Boolean	True
typecasting_required	To convert final output to numpy or not. Note: This option is specific to the Torchvision library	Boolean	True
pil_to_tensor_input	To convert input to tensor before normalization. Note: This option is specific to the Torchvision library	Boolean	True

pad - Image padding with constant pad size or based on target dimensions

Parameters	Description	Type	Default
dims	Height and width comma delimited, e.g., 416,416 for 'target-dims' type of padding	String	Mandatory
type	**Type of padding. Valid options:** • constant: Add padding of constant sides on 4 sides (pad_size must be provided) • target_dims: Add padding based on difference in image size and target size (dims param must be provided)	String	Mandatory
pad_size	Size of padding for 'constant' type of padding	Integer	None
img_position	Parameter to specify position of image, either 'center' or 'corner' (top-left). Padding is added accordingly. Currently used for 'target_dims' type padding	String	'center'
color	Padding value for all planes	Integer	114

resize - Resizes an image using the specified library parameter: cv2(Default), pillow or torchvision

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
library	Python library to be used for resizing a given input; valid values are: opencv | pillow | torchvision	String	opencv
channel_order	Convert image to specified channel order. At present this parameter only takes the 'RGB' value	String	RGB
interp	**Interpolation Type. Options:** • bilinear (supported by opencv, Torchvision, pillow)	String	For opencv and torchvision: bilinear

	• area (supported by opencv only) • nearest (supported by opencv, Torchvision, pillow) • bicubic (supported by Torchvision, pillow) • box (supported by pillow only) • hamming (supported by pillow only) • lanczos (supported by pillow only)		For pillow: bicubic
type	**Type of resize to be done. Note: Torchvision does not use this option. Options:** • letterbox : Used for YOLO models. • imagenet : Scale followed by resize. • aspect_ratio : Resize while keeping aspect ratio. • None : The default behavior is to auto-resize the image to the target dims.	String	auto-resize
resize_before_typecast	To resize before or after conversion to target datatype e.g., fp32	Boolean	True
typecasting_required	To convert final output to numpy or not. Note: This option is specific to the Torchvision library	Boolean	True
mean	Dictionary of means to be subtracted, e.g., {“R”:0.485, “G”:0.456, “B”:0.406}. Note: This option is specific to the Tensorflow library	RGB dictionary	{“R”:0, “G”:0, “B”:0}
std	Dictionary of std-dev for rescaling the values, e.g., {“R”:0.229, “G”:0.224,	RGB dictionary	{“R”:0, “G”:0, “B”:0}

	"B":0.225}. Note: This option is specific to the Tensorflow library		
normalize_before_resize	To perform normalization before or after mean subtraction and standard deviation. Note: This option is specific to the Tensorflow library	Boolean	False
crop_before_resize	To perform cropping before resize. Note: This option is specific to the Tensorflow library	Boolean	False
norm	Normalization factor, all values divided by norm	float32	255.0
normalize_first	To perform normalization before or after mean subtraction and standard deviation. normalize_first=True means perform normalization before. Note: torchvision library does not use this option	Boolean	True

squad_preprocess - Reads the processed files created by the *create_squad_examples* plugin.

Parameters	Description	Type	Default
mask_type	The type of masking to apply. If 'bool', boolean masking is applied. If None, no masking is applied.	String	None

Postprocessing memory plugins

squad_postprocess - Predicts answers for a SQuAD dataset for the given start and end scores.

Parameters	Description	Type	Default
packing_strategy	This flag is set to True if using packing strategy	Boolean	False

centerface_postproc - Processes the inference outputs to parse detections and generates a detections file for the metric evaluator. Used for processing CenterFace face detector.

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
heatmap_threshold	User input for heatmap threshold	Float	0.05
nms_threshold	User input for nms threshold	Float	0.3

centernet_postprocess - Processes the inference outputs to parse detections and generate a detections file for the metric evaluator. Used for processing CenterNet detector.

Parameters	Description	Type	Default
output_dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
top_k	Top K proposals are given from the postprocess plugin	Integer	100
num_classes	Number of classes	Integer	1
score	Threshold to purify the detections	Integer	1

lprnet_predict - Used for LPRNET license plate prediction.

object_detection - Processes the inference outputs to parse detections and generate a detections file for metric evaluator

Parameters	Description	Type	Default
dims	Height and width; comma delimited, e.g., 640,640	String	Mandatory
type	Type of post-processing (e.g., letterbox, stretch)	String	None

label_offset	Offset for the labels information	Integer	0
score_threshold	Threshold limit for the detection scores	Float	0.001
xywh_to_xyxy	Convert bounding box format from box center (xywh) to box corner (xyxy) format	Boolean	False
xy_swap	Swap the X and Y coordinates of bbox	Boolean	False
dtypes	List of datatypes used for bounding boxes, scores, and labels in order, e.g., [float32, float32, int64]. Defaults to the datatypes fetched from the 'outputs_info' for the model's config.yaml.	List	Datatypes from the outputs_info section of the model config.yaml
mask	Do postprocessing on mask	Boolean	False
mask_dims	Output dims of model. Provide this only if mask = True. E.g., 100,80,28,28	String	None
padded_outputs	Pad the outputs	Boolean	False
scale	Comma separated scale values	String	'1'
skip_padding	Skip padding while rescaling to original image shape	Boolean	False

Metric memory plugins

map_coco - Evaluates the mAP score 50 and 50:05:95 for COCO dataset

Parameters	Description	Type	Default
map_80_to_90	Mapping of classes in range 0-80 to 0-90	Boolean	False
segm	Flag to calculate mAP for mask	Boolean	False
keypoint_map	Flag to calculate mAP for keypoint	Boolean	False
data	Dataset used for evaluation. data must be one of 'openimages' or 'coco'	String	'coco'

perplexity - Calculates the perplexity metric. Model outputs are expected to be the logits of proper shape. Ground truth data is expected to be in tokenized format and in the form of token IDs. The ground truth will be automatically generated, if using the "gpt2_tokenizer" dataset plugin.

Parameters	Description	Type	Default
logits_index	Index of the logits output if the model has multiple outputs	Integer	0

precision - Calculates the precision metric, i.e., (correct predictions / total predictions). Ground truth data is expected in the format "filename <space> correct_text". The postprocessed model outputs are expected to be text files with just the "predicted_text".

Parameters	Description	Type	Default
round	Number of decimal places to round the result to	Integer	7
input_image_index	For multi input models, the index of image file in input file list csv	Integer	0

squad_eval - Calculates F1 score and exact match scores for SQuAD dataset based on predictions and ground truth.

Parameters	Description	Type	Default

vocabulary	**vocabulary used for creating the tokenizer which would be used for evaluation.** • vocabulary from huggingface.co and cache (e.g. "bert-base-uncased") • vocabulary from huggingface.co (user-uploaded) and cache (e.g. "deepset/roberta-base-squad2") • path for local directory containing vocabulary files(tokenizer was saved using _save_pretrained('./test/saved_model/')	String	Mandatory
max_answer_length	The maximum length of an answer, after tokenization. In SQuAD v2 this was set to 30 tokens; in SQuAD v1 it was not specified so a default value of 30 was used.	Integer	30
n_best_size	Specifies how many of the possible answers to return for a given question along with corresponding confidence scores.	Integer	20
do_lower_case	Whether or not to lowercase all text before processing.	Bool	False
squad_version	Indicates which version of SQuAD style questions and answers we're dealing with ("v1" or "v2").	Integer	1
round	Number of decimal places to round the result to	Integer	6
cached_vocab_path	Path to cached vocab_file to be used for creating tokenizer	String	None

topk - Evaluates topk value by comparing results and annotations.

Parameters	Description	Type	Default
kval	Top k values, e.g., 1,5 evaluates top1 and top5	String	'1,5'
softmax_index	Index of the softmax output in the results file list	Integer	0

label_offset	Offset required in the labels' scores, e,g., if shape is 1x1001, then labels_offset=1	Integer	0
round	Number of decimal places to round the result to	Integer	3
input_image_index	For multi input models, the index of image file in input file list csv	Integer	0

widerface_AP - Computes average precision for easy, medium, and hard cases.

Parameters	Description	Type	Default
IoU_threshold	User input for IoU threshold	Float	0.4

Previous Next

Converters

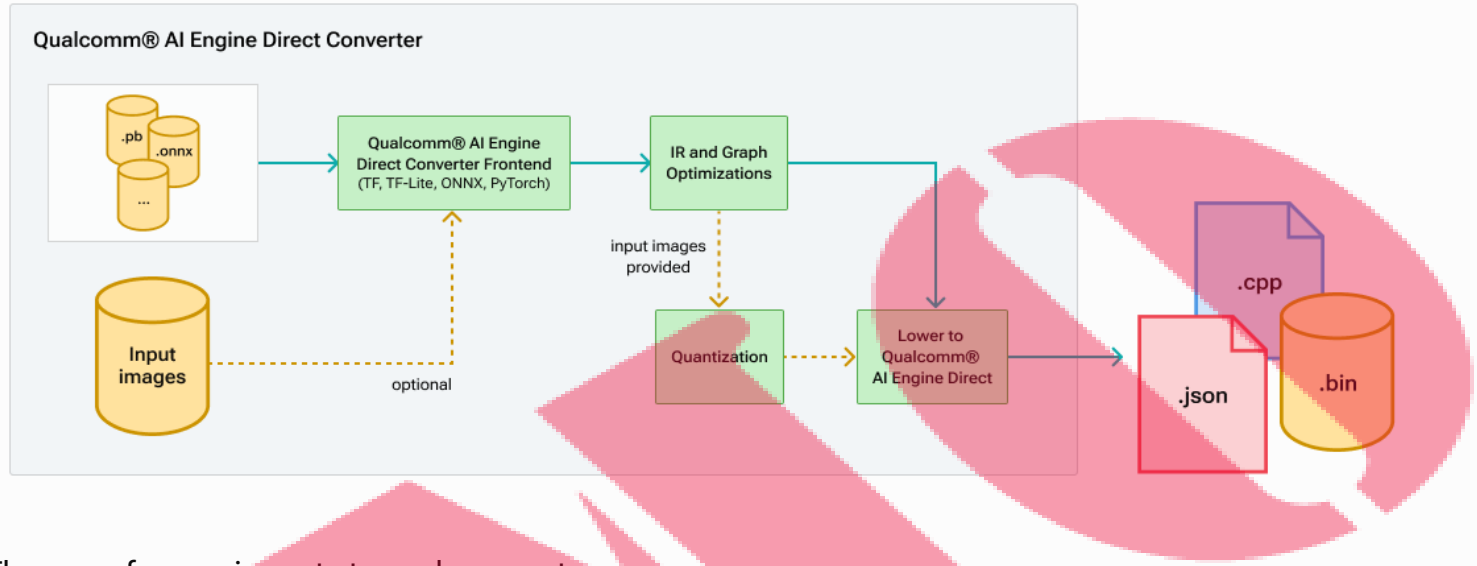
This page describes the general conversion process, the expected inputs and generated outputs, and provides examples of usage.

Overview

Qualcomm® AI Engine Direct currently supports converters for four frameworks: Tensorflow, TFLite, PyTorch, and Onnx. Each converter, at a minimum, requires the original framework model as input to generate a Qualcomm® AI Engine Direct Model. For additional required inputs please refer to the framework specific sections below.

The flow for each converter is the same:

Converter Workflow



There are four main parts to each converter:

1. The front end translation which handles converting the original framework model into the common intermediate represention (IR)
2. The common IR code which contains graph and IR operation definitions as well as various graph optimizations that can be applied to translated graphs.
3. Quantizer, which is optionally invoked to quantize the model prior to the final lowering to QNN. See Quantization for more information.
4. The Qnn converter backend which is responsible for lowering the IR into the final QnnModel API calls.

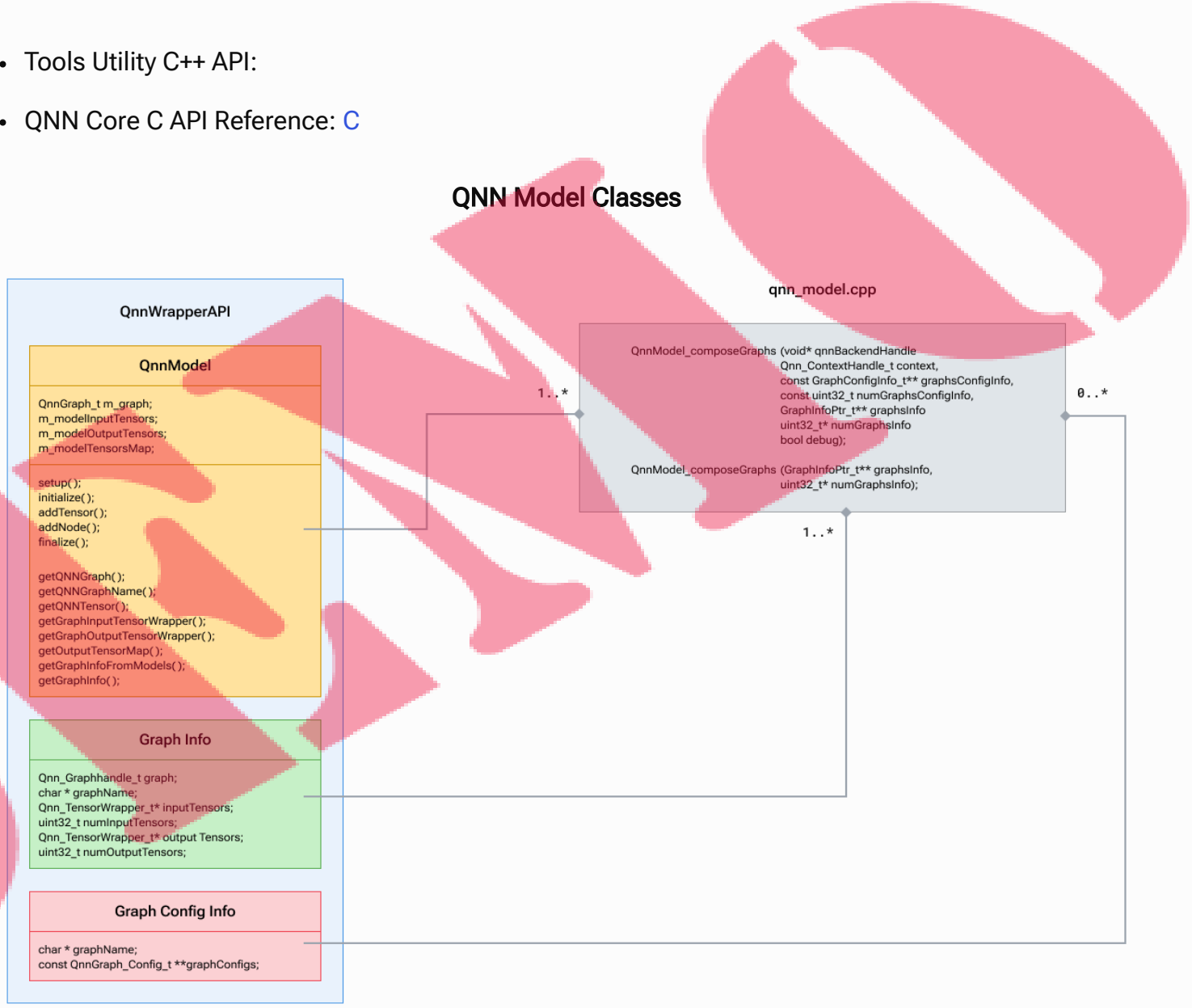
All the converters share the same IR code and QNN converter backend. The output for each converter is the same, a *model.cpp* or *model.cpp/model.bin* which contains the final converted QNN graph. The converted *model.cpp* contains two functions: `QnnModel_composeGraphs` and `QnnModel_freeGraphsInfo`. These two functions leverage the Tools Utility API described below. Additionally, *model_net.json* is saved which is a json format variant to *model.cpp*.

▶ QNN Model JSON Format

Tools Utility API

The tools Utility API contains helper modules to generate QNN API calls. The APIs are light-weight wrappers on-top of the core QNN API and are intended to mitigate repetitive steps for creating QNN graphs.

- Tools Utility C++ API:
- QNN Core C API Reference: C



- **QnnModel**: This class is analogous to a QnnGraph and its tensors inside a given context. The context shall be provided at initialization and a new QnnGraph will be created within it. For more details on these class APIs please see QnnModel.hpp, QnnWrapperUtils.hpp
- **GraphConfigInfo**: This structure is used to pass a list of QNN graph configurations(if applicable) from the client. Refer to QnnGraph API for details on available graph config options.
- **GraphInfo**: This structure is used to communicate constructed graph along with its input and output tensors to the client.
- `QnnModel_composeGraphs` : is responsible for constructing QNN graph on the provided QNN backend using the QnnModel class. It will return the constructed graph via graphsInfo.
- `QnnModel_freeGraphsInfo` : should only be called once the graph is no longer being used.

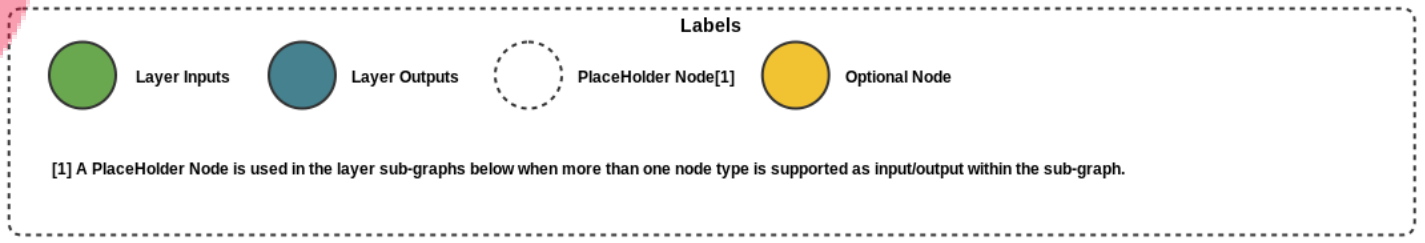
For more information on integrating the model into an application see Integration Workflow

Tensorflow Conversion

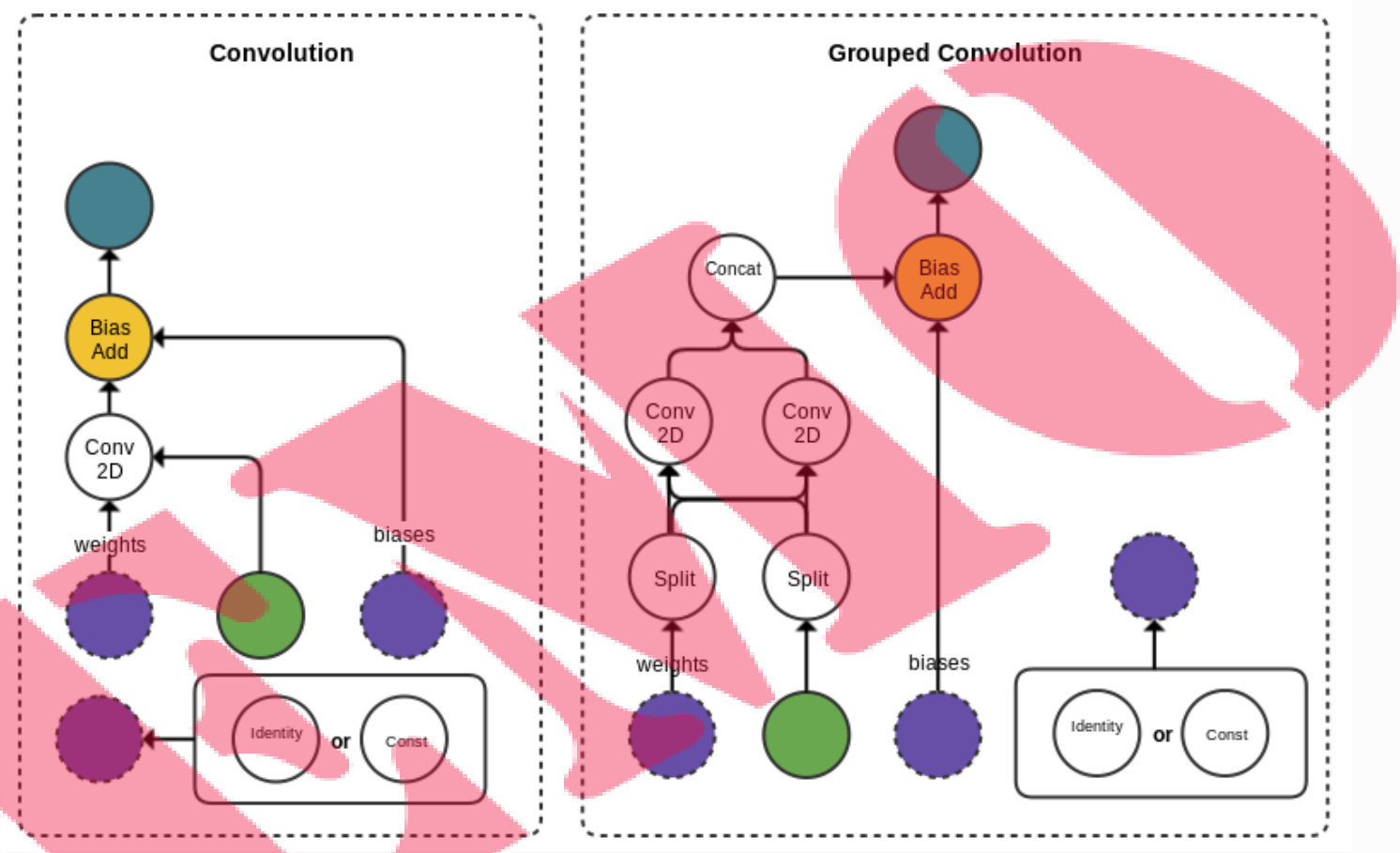
QNN, like many other neural network runtime engines, supports both low level operations (like an elementwise multiply) as well as high level operations (like Prelu). TensorFlow on the other hand, generally supports high level operations by representing them as subgraphs of low level operations. To reconcile these differences the converter must sometimes pattern match subgraphs of small operations into larger “layer-like” operations that can be leveraged in QNN.

Pattern Matching

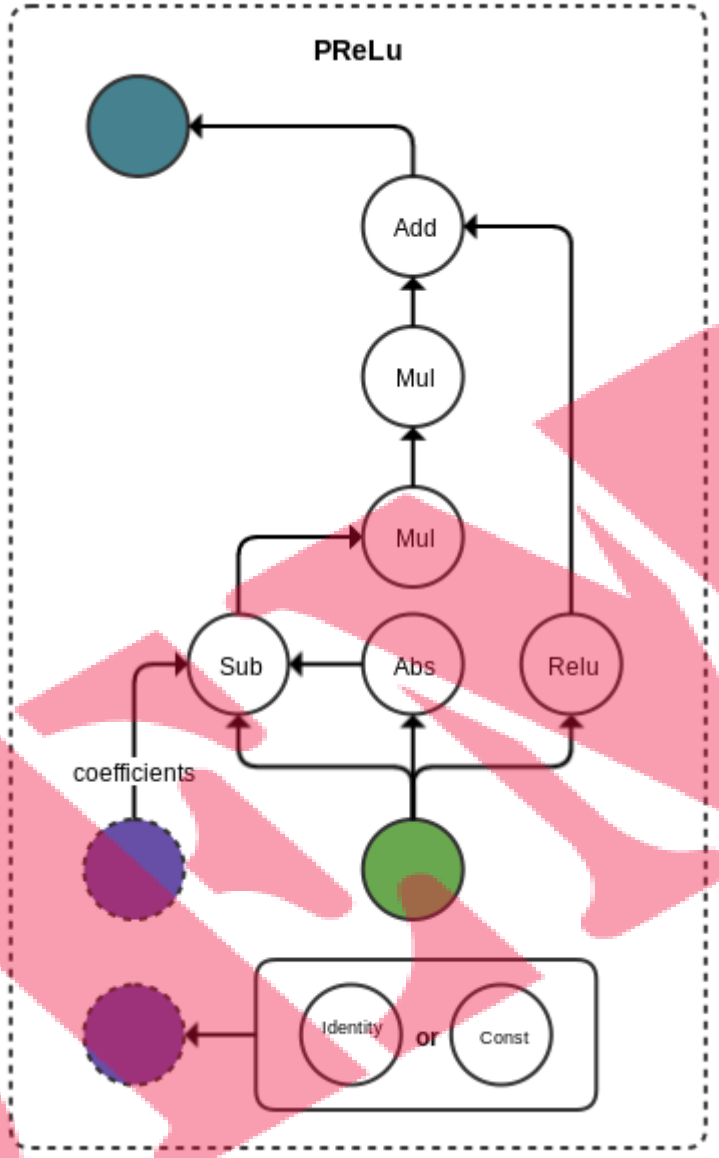
The following are a few examples of pattern matching that occurs in the QNN Tensorflow converter. In each case the pattern generally consists of any operations that fall in between the layer input and output, with additional parameters like weights and biases being absorbed into the final IR op.



Convolution example:



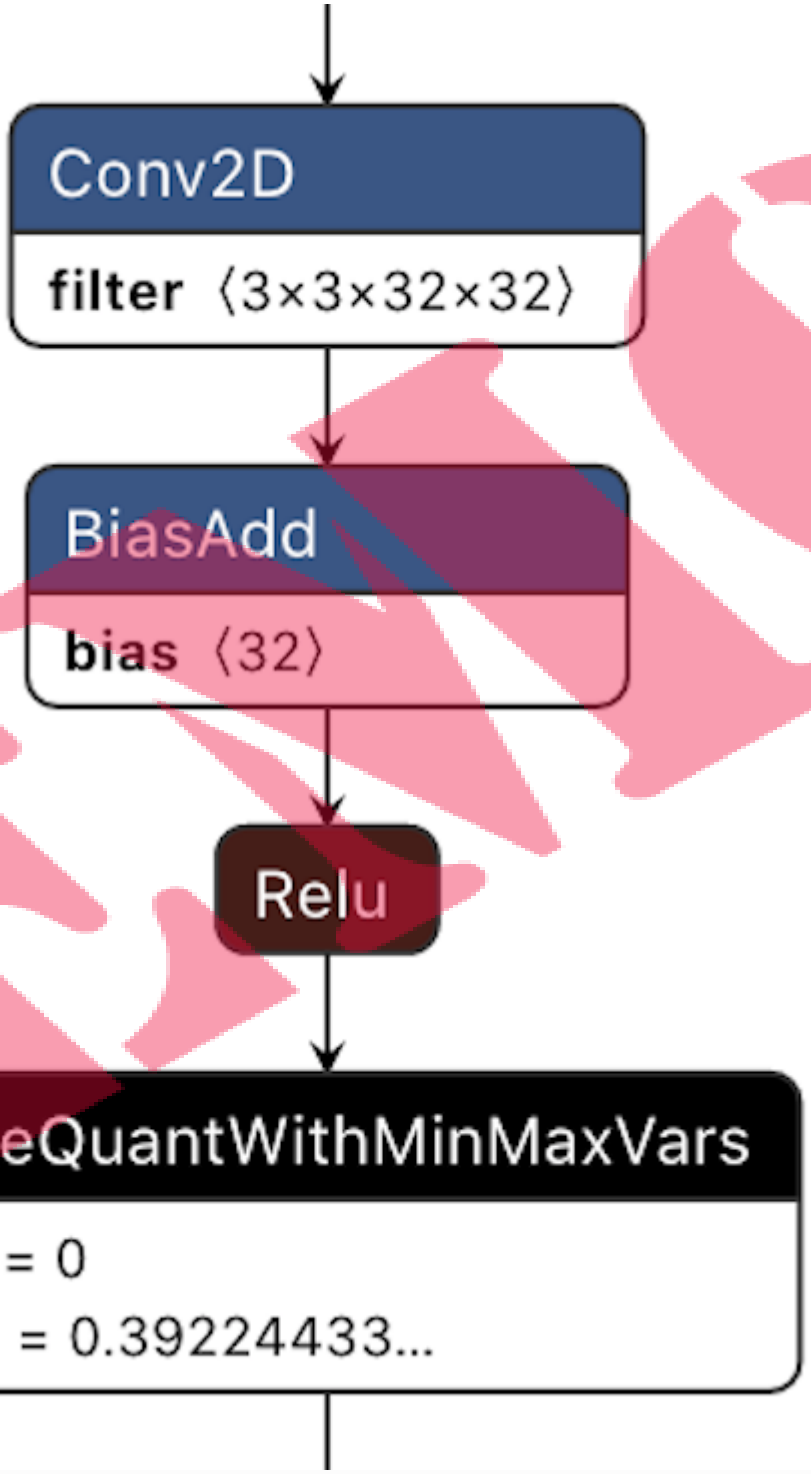
Prelu example:



The important thing to remember is that these patterns are hard coded in the converter. Changes to the model that affect the connectivity and order of the operations in these patterns is also likely to break the conversion as the converter will not be able to identify and map the subgraph to the appropriate layer.

The TF converter also supports propagating quantization aware trained (QAT) model parameters to the final QNN model. This happens automatically during conversion when quantization is invoked. Note that the placement of quantization nodes also determines whether or not they will be propagated. Inserting quantization nodes inside a pattern will cause the pattern matching to break and conversion to fail. The safe place to insert nodes is after "layer-like" layers to capture activation information for a layer. In addition, quantization nodes inserted after weights and biases can capture the quantization information for static parameters.

An example of inserting a quantization node after a Convolution:



See Quantization for more information on initiating quantization as part of the conversion process.

Additional Required Parameters

As Tensorflow graphs often include extraneous nodes that are not required for general inference it is required that the input nodes and dimensions be provided along with the final output nodes required for inference. The converter will then prune unnecessary nodes from the graph ensuring a more compact and efficient graph.

To specify graph's inputs to the converter pass the following on the command line:

```
--input_dim <input_name> <comma separated dims>
```

To specify the graph's output nodes simply pass:

```
--out_node <output_name>
```

Tensorflow also has multiple input formats, but only frozen graphs (.pb files) or .meta files are supported. Saved training sessions are not supported by the converter.

Notes on Tensorflow 2.x Support

The qnn-tensorflow-converter has been updated to support conversion of Tensorflow 2.3 models. Note that while some TF 1.x models may convert using Tensorflow 2.3 as the conversion framework it is generally recommended to use the same TF version for conversion as was used for training the model. Some older 1.x models may not convert at all using TF 2.3 and a TF 1.x instance may be required for successful conversion.

Note that some options have been updated or added to support Tensorflow 2.x models. The first is a change to support the SavedModel format. Users can provide the directory to the SavedModel files by passing it to the same input_network option:

```
--input_network <SavedModel path>
```

Users can optionally pass saved_model_tag to indicate the tag and associated MetaGraph from the SavedModel. Default is “serve”

```
--saved_model_tag <tag>
```

Lastly a user can select the input and output of the model by using the signature key. Default value is ‘serving default’

```
--saved_model_signature_key <signature_key>
```

Example

The following is an example of an SSD model which requires one image input, but has 4 output nodes.

```
qnn-tensorflow-converter --input_network frozen_graph.pb --input_dim Preprocessor/sub 1,300,300,3
```

TFLite Conversion

The `qnn-tflite-converter` converts a TFLite model to an equivalent QNN representation. It takes as input a .tflite model.

Additional Required Parameters

TFlite converter needs the names and dimensions of the input nodes to be provided at commandline for the conversion. Each input must be passed individually using the same argument.

To specify graph’s inputs to the converter pass the following on the command line:

```
--input_dim <input_name_1> <comma separated dims> --input_dim <input_name_2> <comma separated dims
```

Example

The following is an example of converting an Inception_v3 model which requires one image input

```
qnn-tflite-converter --input_network model.tflite --input_dim "input" 1,299,299,3 --output_path mo
```

PyTorch Conversion

The `qnn-pytorch-converter` converts a PyTorch model to an equivalent QNN representation. It takes as input a TorchScript model (.pt).

Additional Required Parameters

PyTorch converter needs the names and dimensions of the input nodes to be provided at commandline for the conversion. Each input must be passed individually using the same argument.

To specify graph's inputs to the converter pass the following on the command line:

```
--input_dim <input_name_1> <comma separated dims> --input_dim <input_name_2> <comma separated dims
```

Example

The following is an example of converting an Inception_v3 model which requires one image input

```
qnn-pytorch-converter --input_network model.pt --input_dim "input" 1,3,299,299 --output_path model
```

Onnx Conversion

The `qnn-onnx-converter` converts a serialized ONNX model to an equivalent QNN representation. By default, it also runs onnx-simplifier if available in user environment(see Setup).

Additionally, onnx-simplifier is only run by default if user has not provided quantization overrides/custom ops as the simplification process could possibly squash layers preventing the custom ops or quantization overrides from being used. If the model contains ONNX functions, converter always does inlining of function nodes. Note: If conversion fails, the onnx converter supports an additional option "–dry_run" which will dump detailed information about unsupported ops and associated parameters.

Example

```
qnn-onnx-converter --input_network model.onnx --output_path model.cpp
```

Custom Operation Output Shape and Datatype Inference

QNN converter requires output shapes and datatypes for all operations to be present in the model for successful conversion. Output shapes and datatypes for custom operations can be inferred from the model if present in the model or inferred using the framework's shape inference script. When the output shapes and datatypes of a custom operation are not present in the model or cannot be inferred from the framework's shape inference script, the logic to infer custom operation output shapes and datatypes can be provided to the converter through a shared library compiled with Convter Op Package Generation. The compiled library can be provided with the `--converter_op_package_lib` or `-cpl` option followed by the absolute path to the compiled library. The converter takes the library, infers the output shapes and datatypes of the custom operations needed for successful model conversion. Multiple libraries must be comma separated.

> **Note**
>
> `--converter_op_package_lib` or `-cpl` is an optional argument and should be used when the output shapes and/or output datatypes for custom operations are not present in the model or cannot be inferred from the framework's shape inference script.

> **Note**

When the output datatypes are present in the model and the `--converter_op_package_lib` with the logic to populate the output datatypes is passed, output datatypes inferred from the library will be given priority and override the output datatypes inferred from the model.

Example

```
qnn-onnx-converter --input_network model.onnx --converter_op_package_lib libExampleLibrary.so
```

Note

- See Convter Op Package Generation for library generation and compilation instructions.
- Custom operation output shape inference is only supported for ONNX and PyTorch converters.
- Tensorflow and TFLite converters do not support custom operation output shape inference.

Custom I/O

Introduction

Custom I/O feature allows users to provide the desired layout and datatype for the inputs and outputs while loading a network. Instead of compiling the network for the inputs and outputs specified in the model, the network is compiled for the inputs and outputs described in custom configuration. This feature is used when the user intends to pre-process (on GPU/CDSP or any other method) or offline process (like allowed by ML commons) the input data and avoid some steps in the input processing. Users can avoid redundant transposes and data-type conversions if they have knowledge of the input pre-processing steps. Similarly, on the post-processing side, if the model output is to be fed to a next stage in a pipeline, the desired format and type can be configured as the output of current stage.

In this section, the term “Model I/O” refers to the input and output datatypes and formats of the original model. The term “Custom I/O” refers to the input and output datatypes and formats desired by the user.

Custom I/O Configuration File

Custom I/O can be applied using a configuration yaml file that contains the following fields for each input and output that needs to be modified.

- IOName: Name of the input or output present in the model that needs to be loaded as per the custom requirement.
- Layout: Layout field (optional) has two sub fields: Model and Custom. Model and Custom fields support valid QNN Layout. Accepted values are: NCDHW, NDHWC, NCHW, NHWC, NFC, NCF, NTF, TNF, NF, NC, F, NONTRIVIAL, where, N = Batch, C = Channels, D = Depth, H = Height, W = Width, F = Feature, T = Time
  - Model: Specify the layout of the buffer in the original model. This is equivalent to the –input_layout option and both cannot be used together.
  - Custom: Specify the custom layout desired for the buffer. This field needs to be filled by the user.
- Datatype: Datatype field (optional) supports float32, float16 and uint8 datatypes.
- QuantParam: QuantParam field (optional) has three sub fields: Type, Scale and Offset.
  - Type: Set to QNN_DEFINITION_DEFINED (default) if the scale and offset are provided by the user else set to QNN_DEFINITION_UNDEFINED.
  - Scale: Float value for the scale of the buffer as desired by the user.
  - Offset: Integer value for the offset as desired by the user.

Example

Consider a ONNX model with the original model I/O and custom I/O configuration as shown in the table below:

Input/Output Name	Model I/O	Custom I/O
'input_0'	float NCHW	int8 NHWC
'output_0'	float NHWC	float NCHW

Then, the content of custom I/O configuration yaml file that should be provided is

```
- IOName: input_0
  Layout:
    Model: NCHW
    Custom: NCHW
  Datatype: uint8
  QuantParam:
    Type:
       QNN_DEFINITION_DEFINED
    Scale:
       0.12
    Offset:
       2

- IOName: output_0
  Layout:
    Model: NHWC
    Custom: NCHW
```

Note:

- If no change is required for an input or output, it can be skipped in the configuration file.
- Datatype can be modified using custom I/O feature only if the model input or output datatype is float, float16, int8 or uint8. For other datatypes, 'Datatype' field should be skipped in the configuration file.

Usage

The custom IO config YAMl file can be provided using the `--custom_io` option of `qnn-onnx-converter`. Sample usage is as follows:

```
$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --custom_io <path/to/YAML/file> ....
```

Custom IO Config Template File

The Custom IO Configuration file filled with default values can be obtained using the `--dump_custom_io_config_template` option of `qnn-onnx-converter`.

```
$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --input_network ${QNN_SDK_ROOT}/examples/Models/InceptionV3/tensorflow/model.onnx \
  --dump_custom_io_config_template <output_folder>/config.yaml
```

The dumped template file has an entry for each input and output of the model provided. Each field in the template file is filled with the default value obtained from the model for that particular input or output. The template file also has comments describing each field for the user.

Supported Use Cases

1. Layout conversions of the input and output buffers of the model. Valid layout conversions are inter-conversions between:
    - NCDHW and NDHWC
    - NHWC and NCHW
    - NFC and NCF
    - NTF and TNF
2. Passing quantized inputs of datatype uint8 or int8 to a non-quantized model. In this case, users must provide the scale and offset for the quantized inputs.
3. Users can provide custom scale and offset for the inputs and outputs of a quantized model. The scale and offset generated by the quantizer are overrriden by those provided by the user in the YAML file.

The user may use the `--input_data_type` and `--output_data_type` options of `qnn-net-run` to provide float or uint8_t type data to model inputs/outputs. Users may pass and get int8/uint8 data to the model using the `native` option. By default, `qnn-net-run` assumes the data to be of type float32 and performs the quantization at input and dequatization at output in case of quantized models.

Limitations

- Custom IO only supports providing the following datatypes: float32, float16, uint8, int8.
- If the user needs to pass quantized inputs (i.e. of type int8 or uint8) to a non-quantized model, the scale and offset must be provided by the user in the YAML file. Not providing the scale and offset in this case would throw an error.

Preserve I/O

Introduction

Preserve I/O feature allows users to retain the layout and datatype of the inputs and outputs as present in the original ONNX model. This feature allows the user to avoid any pre- or post-processing steps to transform the data to the layout and datatype due to the default behavior of QNN converters at the input and output of the model.

Usage

The different ways of using this option are as follows:

1. The user may choose to preserve layouts and datatypes for all IO tensors by just passing the `--preserve_io` option as follows:

```
$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --preserve_io ....
```

2. The user may choose to preserve the only layout or datatype for all the inputs and outputs of the graph as follows:

```
$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --preserve_io layout ....

or,
```

```
$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --preserve_io datatype....
```

3. The user may choose to preserve the layout or datatype for only a few inputs and outputs of the graph as follows:

```
$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --preserve_io layout <space separated list of names of inputs and outputs of the graph>....

or,

$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --preserve_io datatype <space separated list of names of inputs and outputs of the graph>....
```

4. The user can pass a combination of `--preserve_io layout` and `--preserve_io datatype` as follows:

```
$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --preserve_io layout <space separated list of names of inputs and outputs of the graph> \
  --preserve_io datatype <space separated list of names of inputs and outputs of the graph> ....
```

Passing just `--preserve_io layout` and `--preserve_io datatype` together is valid and equivalent to passing `--preserve_io` only. Usage in point 3 cannot be combined with usage in point 1 or point 2 and will result in an error if used together.

Usage in qnn-pytorch-converter

In PyTorch models there may be no tensor names. Input tensor names are named by passing `-d`, but output names in converter are named by internal logic. To preserve layout or datatype for only the specified output tensor user can do as follows:

1. Run a 1st pass of the Converter and use the generated CPP/JSON file to fetch the APP_READ type tensor names.

2. Run a 2nd Converter for preserve layout or datatype for only the specified IO tensor with their names:

```
$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --preserve_io layout <space separated list of names of inputs and outputs of the graph>....

or,

$ ${QNN_SDK_ROOT}/bin/x86_64-linux-clang/qnn-onnx-converter \
  --preserve_io datatype <space separated list of names of inputs and outputs of the graph>....
```

Usage with other converter options

1. `--keep_int64_inputs` need not be passed if preserve IO is used to preserve the datatype of such inputs.
2. `--use_native_input_files` is set to True in case of quantization if preserve IO is used to preserve the datatypes.
3. The layout specified using `--input_layout` is honored.
4. Using `--input_dtype` with preserve IO may result in an error in case of datatype mismatch for any IO tensor.
5. The layouts and datatypes specified using `--custom_io` get higher precedence over `--preserve_io`.

Since preserve IO retains the datatypes of IO tensors in the original model, the user must use `--use_native_input_files` or `--native_input_tensor_names` with `qnn-net-run`.

Common Parameters

There are a number of common parameters that can be passed to all the converters. These are described here: Tools

In addition, quantization parameters are also specified at conversion time. For more information refer to the tools document above and to: Quantization

Qairt Converter

The 2.21 release introduces a new conversion tool, using a new prefix `qairt` for `Qualcomm AI Runtime`. This new prefix communicates that this converter can be used with both the `Qualcomm Neural Processing SDK` API as well as the `Qualcomm AI Engine Direct` API.

> **Note**
>
> This tool is still in a Beta release status.

The `qairt-converter` tool converts a model from one of ONNX/TensorFlow/TFLite/PyTorch framework to a DLC. The DLC contains the model in a Qualcomm graph format to support inference on Qualcomm HW. The converter automatically detects the proper framework based on the source model extension.

Supported frameworks and file types are:

Framework	File Type
Onnx	*.onnx
TensorFlow	*.pb
TFLite	*.tflite
PyTorch	*.pt

Basic Conversion

Basic conversion has only one required argument `--input_network`. Some frameworks may require additional arguments that are otherwise listed as optional. Please check the help text for more details.

a. Onnx Conversion

```
$ qairt-converter --input_network model.onnx
```

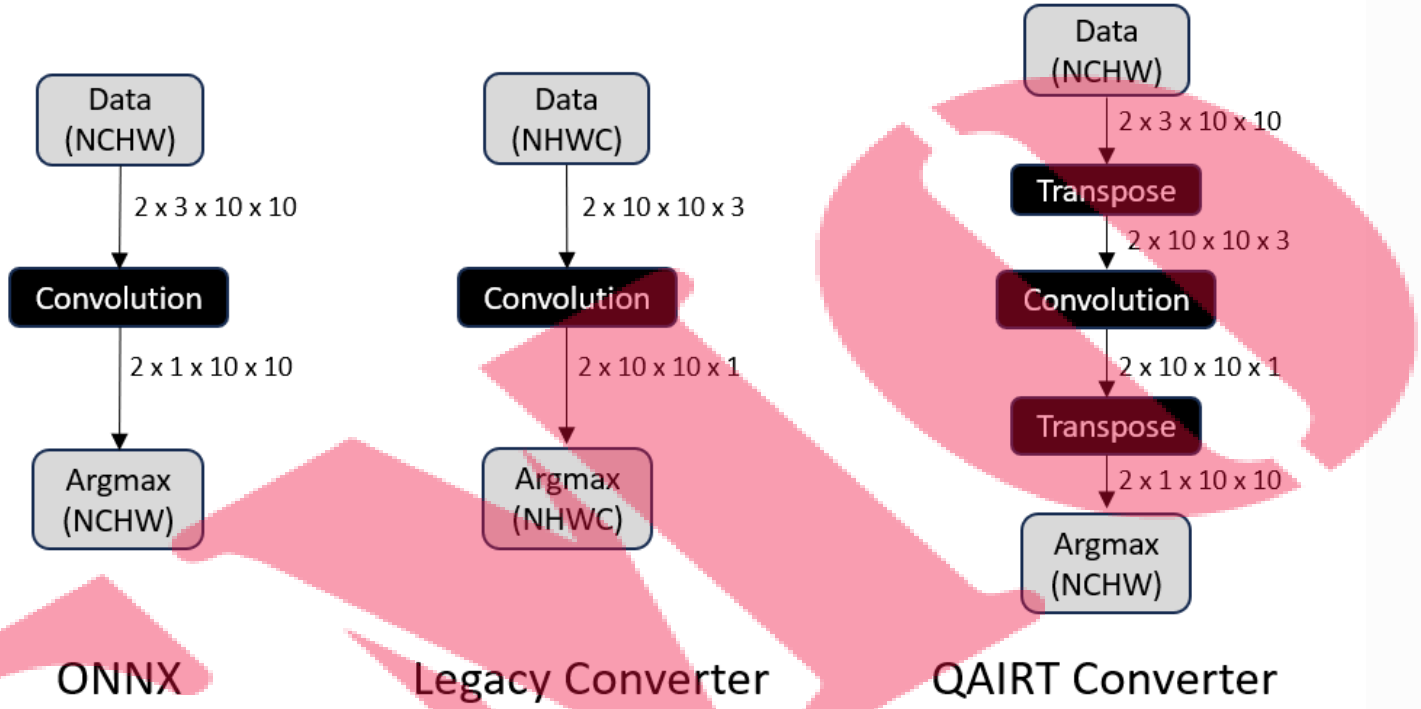
b. Tensorflow Conversion

Tensorflow requires `--desired_input_shape` and `--out_tensor_node`.

```
$ qairt-converter \
     --input_network inception_v3_2016_08_28_frozen.pb \
     --desired_input_shape input 1,299,299,3 \
     --out_tensor_node InceptionV3/Predictions/Reshape_1
```

Input/Output Layouts

The default input and output layouts in the converted graph are the same as per the source model. This behavior differs from the legacy converter which would modify the input and (optionally) the output layout to the spatial first format. An example single layer Onnx model (spatial last) is shown below.



Input/Output Customization using YAML

Note

This feature will allow specification of the desired input/output tensor layout in the converted model.

Users can provide a yaml configuration file to simplify using different input and output configurations over the command line. All configurations in the yaml are optional. If an option is provided in the yaml configuration and an equivalent option is provided on the command line, the command line option takes priority. The yaml configuration schema is shown below.

```
Input Tensor Configuration:
  # Input 1
  - Name:
```

```
    Src Model Parameters:
        DataType:
        Layout:
    Desired Model Parameters:
        DataType:
        Layout:
        Shape:
        Color Conversion:
        QuantParams:
          Scale:
          Offset:

Output Tensor Configuration:
  # Output 1
  - Name:
    Src Model Parameters:
        DataType:
        Layout:
    Desired Model Parameters:
        DType:
        Layout:
        QuantParams:
          Scale:
          Offset:
```

- `Name:` Name of the input or output tensor present in the model that needs to be customized
- `Src Model Parameters`

  These are mandatory if a certain equivalent desired configuration is specified.

  - `DataType:` DataType of the tensor in source model.
  - `Layout:` Layout of the tensor in source model. Accepted values are:
    - NCDHW
    - NDHWC
    - NCHW
    - NHWC
    - NFC
    - NCF

- NTF
- TNF
- NF
- NC
- F

where

- N = Batch
- C = Channels
- D = Depth
- H = Height
- W = Width
- F = Feature
- T = Time

- `Desired Model Parameters`
  - `DataType:` Desired datatype of the tensor in converted model. Supports float32, float16, uint8, int8 datatypes.
  - `Layout:` Desired layout of the tensor in converted model. Supports the same values as source layout.
  - `Shape:` Shape/Dimension of the tensor in converted model. Supports comma separated dimension value (a,b,c,d)
  - `Color Conversion:` Color encoding of the tensor in converted model. Supports BGR, RGB, RGBA, ARGB32, NV21, NV12
  - `QuantParams:` Used when the desired model datatype is a quantized datatype; has two sub fields (Scale and Offset).
    - `Scale:` Float value for the scale of the buffer as desired by the user.
    - `Offset:` Integer value for the offset as desired by the user.

The IO configuration file can be obtained using the `--dump_io_config_template` option of `qairt-converter`.

```
$ qairt-converter \
      --input_network model.onnx \
      --dump_io_config_template <output_folder>/io_config.yaml
```

QAT encodings

QAT encodings are quantization-aware training encodings which are present in the source graph. They can be present in the following form in the source graph.

- FakeQuant Nodes: There can be FakeQuant nodes in the source network.
- Tensor output encodings: Quantization overrides can be associated with the output tensors in the source network.
- Quant-Dequant Nodes: There can be Quant-Dequant nodes present in the source network.

For all the above cases, the FakeQuant and Quant-Dequant nodes are removed and the quantization overrides are cached in the float DLC generated from the `qairt-converter` tool. These can then be used with the `qairt-quantizer` tool.

Quantization Overrides

Provide quantization overrides to `qairt-converter` with a JSON file containing the parameters to use for quantization by using the `--quantization_overrides` option, e.g., `--quantization_overrides <overrides.json>` These will be cached with the float DLC generated by `qairt-converter` and can be used with the `qairt-quantizer` tool.

These will override any quantization data carried from conversion ,e.g., TF fake quantization, or calculated during the normal quantization process. For more details refer to Quantization Overrides.

FP16 Conversion

Users also have the ability to generate a float16 graph where all float32 tensors are converted to float16 by passing the `--float_bitwidth 16` flag to the `qairt-converter` tool. To generate

a float16 graph with the bias still in float32, an additional `--float_bias_bitwidth 32` flag can be passed.

DryRun

Use the `--dry_run` option to evaluate the model without actually converting any ops. This returns unsupported ops/attributes and unused inputs/outputs.

FAQs

- How is QAIRT Converter different from Legacy Converters?
  - Single converter vs independent framework converters

    The qairt-converter is a single converter tool supporting conversion for all supported frameworks based on the model extension while legacy converters had different framework specific tools.
  - Changed some optional arguments as default behavior

    The default input and output layouts in the Converted graph will be same as in the Source graph. The legacy ONNX and Pytorch converters may not always retain the input and output layouts from Source graph.
  - Removed deprecated arguments

    Deprecated arguments on the legacy converters are not enabled on the new converter.
  - Renamed some arguments for clarity

    The –input_encoding argument is renamed to –input_color_encoding. Framework-specific arguments have the framework name present. eg- –define_symbol is renamed to –onnx_define_symbol, –show_unconsumed_nodes is renamed to –tf_show_unconsumed_nodes, –signature_name is renamed to –tflite_signature_name.
  - DLC as the Converter output file format

    The QAIRT Converter uses DLC as output format. The .cpp/.bin & .json format used by `qnn-<framework>-converter` Converter are not supported by QAIRT Converter. In order

to generate the .cpp/.bin and .json output, continue to use the legacy converter.

- Quantizer functionality is separated from Conversion functionality
  - `qnn-<framework>-converter` invokes the quantizer as part of the converter tool when `--input_list` or `--float_fallback` is passed.
  - `qairt-quantizer` however is a standalone tool for quantization like `snpe-dlc-quant`.
  - Please refer to qairt-quantizer for more information and usage details.

- Will the Converted model be any different with QAIRT converter compared to Legacy Converter?
  - The result of the QAIRT Converter will be different from the result of Legacy Converters in terms of the input/output layout.
  - Legacy converters will by default modify the input tensors to Spatial First (e.g. NHWC) layout. This means for Frameworks like ONNX, where the predominant layout is Spatial Last (e.g. NCHW), the input/output layout is different between the source model and the converted model.
  - Since QAIRT Converter preserves the source layouts be default, the QAIRT-converted graphs in case of many ONNX/Pytorch models will be different from the Legacy-converted graphs.
  - The QAIRT Converter will be enhanced in a future release to support the same layouts as the legacy converters.

Previous Next

Quantization

This page describes the general quantization process and supported algorithms and features.

Overview

Non-quantized models files use 32 bit floating point representations of network parameters. Quantized model files use fixed point representations of network parameters, generally 8 bit weights and 8 or 32bit biases. The fixed point representation is the same used in Tensorflow quantized models.

Choosing Between a Quantized or Non-Quantized Model

- CPU - Choose a non-quantized model. Quantized models are currently incompatible with the CPU backend.
- DSP - Choose a quantized model. Quantized models are required when running on the DSP backend.
- GPU - Choose a non-quantized model. Quantized models are currently incompatible with the GPU backend.
- HTP - Choose a quantized model. Quantized models are required when running on the HTP backend.
- HTA - Choose a quantized model. Quantized models are required when running on the HTA backend.

Quantization

This section describes the concepts behind the quantization algorithm used in QNN. These concepts are used by the converters when the developer decides to quantize a graph.

Overview

QNN supports multiple quantization modes. The basics of the quantization, regardless of mode, are described here.

- Quantization converts floating point data to the Tensorflow-style fixed point format using a provided bit width.
- The following requirements are satisfied:
  - Full range of input values is covered.
  - Minimum range of 0.0001 is enforced.
  - Floating point zero is exactly representable.
- Quantization algorithm inputs:
  - Set of floating point values to be quantized.
- Quantization algorithm outputs:
  - Set of 8-bit fixed point values.
  - Encoding parameters:
    - encoding-min - minimum floating point value representable (by fixed point value 0)
    - encoding-max - maximum floating point value representable (by fixed point value 255)
    - scale - The step size for the given range (max - min) / (2^bw-1)
    - offset - The integer value which exactly represents 0. round(-min/scale)
- Algorithm
  1. Compute the true range (min, max) of input data.
  2. Compute the encoding-min and encoding-max.
  3. Quantize the input floating point values.
  4. Output:
     - fixed point values
     - encoding-min and encoding-max parameters
     - scale and offset parameters

Details

1. Compute the true range of the input floating point data.
   - finds the smallest and largest values in the input data

- represents the true range of the input data

2. Compute the encoding-min and encoding-max.

- These parameters are used in the quantization step.
- These parameters define the range and floating point values that will be representable by the fixed point format.
  - encoding-min: specifies the smallest floating point value that will be represented by the fixed point value of 0
  - encoding-max: specifies the largest floating point value that will be represented by the fixed point value of 255
  - floating point values at every step size, where step size = (encoding-max - encoding-min) / (2^bw-1), will be representable
  - offset where zero is exactly represented
- encoding-min and encoding-max are first set to the true min and true max computed in the previous step
- Requirements
  1. Encoding range must be at least a minimum of 0.0001
     - encoding-max is adjusted to max(true max, true min + 0.0001)
  2. Floating point value of 0 must be exactly representable
     - encoding-min or encoding-max may be further adjusted

3. Cases - Handling 0

1. Inputs are strictly positive
   - the encoding-min is set to 0.0
   - zero floating point value is exactly representable by smallest fixed point value 0
   - e.g. input range = [5.0, 10.0]

- encoding-min = 0.0, encoding-max = 10.0

2. Inputs are strictly negative

- encoding-max is set to 0.0
- zero floating point value is exactly representable by the largest fixed point value 255
- e.g. input range = [-20.0, -6.0]
  - encoding-min = -20.0, encoding-max = 0.0

3. Inputs are both negative and positive

- encoding-min and encoding-max are slightly shifted to make the floating point zero exactly representable
- e.g. input range = [-5.1, 5.1]
  - encoding-min and encoding-max are first set to -5.1 and 5.1, respectively
  - encoding range is 10.2 and the step size is 10.2/255 = 0.04
  - zero value is currently not representable. The closest values representable are -0.02 and +0.02 by fixed point values 127 and 128, respectively
  - encoding-min and encoding-max are shifted by -0.02. The new encoding-min is -5.12 and the new encoding-max is 5.08
  - floating point zero is now exactly representable by the fixed point value of 128

4. Quantize the input floating point values.

- encoding-min and encoding-max parameter determined in the previous step are used to quantize all the input floating values to their fixed point representation
- Quantization formula is:
  - quantized value = round(255 * (floating point value - encoding.min) / (encoding.max - encoding.min))
- quantized value is also clamped to be within 0 and 2^bw-1

5. Outputs

- the fixed point values
- encoding-min, encoding-max, scale, and offset parameters

Quantization Example

1. Inputs:

   - input values = [-1.8, -1.0, 0, 0.5]
   - encoding-min is set to -1.8 and encoding-max to 0.5
   - encoding range is 2.3, which is larger than the required 0.0001
   - encoding-min is adjusted to −1.803922 and encoding-max to 0.496078 to make zero exactly representable
   - step size is 0.009020
   - offset is 200

2. Outputs:

   - quantized values are [0, 89, 200, 255]

Dequantization Example

1. Inputs:

   - quantized values = [0, 89, 200, 255]
   - encoding-min = −1.803922, encoding-max = 0.496078
   - step size is 0.009020
   - offset is 200

2. Outputs:

   - dequantized values = [−1.8039, −1.0011, 0.0000, 0.4961]

Bitwidth Selection

QNN currently supports a default quantization bit width of 8 for both weights and biases. The weight, bias, and activation bit widths, however, can be overriden by passing one of –weight_bw, –bias_bw, and/or –act_bw followed by the bitwidth. Please see the converter documentation here for more details on the command line options.

Quantization Modes

QNN supports four quantization modes: tf, symmetric, enhanced, and adjusted. The primary difference is between how they select the quantization range to be used.

TF

The default mode has been described above, and uses the true min/max of the data being quantized, followed by an adjustment of the range to ensure a minimum range and to ensure 0.0 is exactly quantizable.

Symmetric

Symmetric quantization follows the same basic principles as TF quantization but adjusted the range to be symmetric. It does this by selecting a new min and max from the original range such that new_max=max(abs(min), abs(max)) and adjusts the range to be (-new_max, max) such that the range is symmetric around 0. This is typically only used for weights as it helps to reduce computation overhead at runtime. This mode is enabled by passing –param_quantizer symmetric to one of the converters.

Enhanced

Enhanced quantization mode (invoked by passing "enhanced" to either the `--param_quantizer` or `--act_quantizer` options in one of the converters) uses an algorithm to try to determine a better set of quantization parameters to improve accuracy. The algorithm may pick a different min/max value than the default quantizer, and in some cases it may set the range such that some of the original weights and/or activations cannot fall into that range. However, this range does produce better accuracy than simply using the true min/max. The enhanced quantizer can be enabled

independently for weights and activations by appending either “weights” or “activations” after the option.

This is useful for some models where the weights and/or activations may have “long tails”. (Imagine a range with most values between -100 and 1000, but a few values much greater than 1000 or much less than -100.) In some cases these long tails can be ignored and the range -100, 1000 can be used more effectively than the full range.

Enhanced quantizer still enforces a minimum range and ensures 0.0 is exactly quantizable.

TF Adjusted

This mode is used only for quantizing weights to 8 bit fixed point (invoked by passing “adjusted” to either the `--param_quantizer` or `--act_quantizer` options in one of the converters) to, which uses adjusted min or max of the data being quantized other than true min/max or the min/max that exclude the long tail. This has been verified to be able to provide accuracy benefit for denoise model specifically. Using this quantizer, the max will be expanded or the min will be decreased if necessary.

Adjusted weights quantizer still enforces a minimum range and ensures 0.0 is exactly quantizable.

Enhanced Quantization Techniques

Quantization can be a difficult problem to solve due to the myriad of training techniques, model architectures, and layer types. In an attempt to mitigate quantization problems model preprocessing techniques have been added to the quantizer that may improve quantization performance on models which exhibit sharp drops in accuracy upon quantization.

The primary technique introduced is CLE (Cross Layer Equalization).

CLE works by scaling the convolution weight ranges in the network by making use of a scale-equivariance property of activation functions. In addition, the process absorbs high biases which may be result from weight scaling from one convolution layer to a subsequent convolution layer.

Enhanced Quantization Techniques: Limitations

In many cases CLE may enable quantized models to return to close to their original floating-point accuracy. There are some caveats/limitations to the current algorithms:

CLE operates on specific patterns of operations that all exist in a single branch (outputs cannot be consumed by more than one op). The matched operation patterns (r=required, o=optional) are:

Conv(r)->Batchnorm(r)->activation(o)->Conv(r)->Batchnorm(r)->activation(o) Conv(r)->Batchnorm(r)->activation(o)->DepthwiseConv(r)->Batchnorm(r)->activation(o)->Conv(r)->Batchnorm(r)->activation(o)

The CLE algorithm currently only supports Relu activations. Any Relu6 activations will be automatically changed to Relu and any activations other than these will cause the algorithm to ignore the preceding convolution. Typically the switch from Relu6->Relu is harmless and does not cause any degradation in accuracy, however some models may exhibit a slight degradation of accuracy. In this case, CLE can only recover accuracy to that degraded level, and not to the original float accuracy. CLE requires batchnorms (specifically detectable batchnorm beta/gamma data) be present in the original model before conversion to DLC for the complete algorithm to be run and to regain maximum accuracy. For Tensorflow, the beta and gamma can sometimes still be found even with folded batchnorms, so long as the folding didn't fold the parameters into the convolution's static weights and bias. If it does not detect the required information you may see a message that looks like: "Invalid model for HBA quantization algorithm." This indicates the algorithm will only partially run and accuracy issues may likely be present.

To run CLE simply add the option –algorithms cle to the converter command line.

More information about the algorithms can be found here: [https://arxiv.org/abs/1906.04721]

Quantization Impacts

Quantizing a model and/or running it in a quantized runtime (like the HTP) can affect accuracy. Some models may not work well when quantized, and may yield incorrect results. The metrics for measuring

impact of quantization on a model that does classification are typically “Mean Average Precision”, “Top-1 Error” and “Top-5 Error”.

Quantization Overrides

If the option –quantization_overrides is provided the user may provide a json file with parameters to use for quantization. These will override any quantization data carried from conversion (eg TF fake quantization) or calculated during the normal quantization process. Format defined as per AIMET specification.

There are two sections in the json, a section for overriding operator output encodings called “activation_encodings” and a section for overriding parameter (weight and bias) encodings called “param_encodings”. Both must be present in the file, but can be empty if no overrides are desired. An example with all of the currently supported options:


```
{
    "activation_encodings": {
        "Conv1:0": [
            {
                "bitwidth": 8,
                "max": 12.82344407824954,
                "min": 0.0,
                "offset": 0,
                "scale": 0.050288015993135454
            }
        ],
        "input:0": [
            {
                "bitwidth": 8,
                "max": 0.9960872825108046,
                "min": -1.0039304197656937,
                "offset": 127,
                "scale": 0.007843206675594112
            }
        ]
    },
    "param_encodings": {
        "Conv2d/weights": [
            {
```

```
                "bitwidth": 8,
                "max": 1.700559472933134,
                "min": -2.1006477158567995,
                "offset": 140,
                "scale": 0.01490669485799974
            }
        ]
    }
}
```


Note that it is not required to provide scale and offset. If they are provided they will be used, otherwise they will be calulated from the provided bw, min, and max parameters.

Per-channel Quantization Overrides

Per-channel quantization should be used for tensors that are weight inputs to Conv consumers (Conv2d, Conv3d, TransposeConv2d, DepthwiseConv2d). This section provides examples to manually override per-channel encodings for these Conv-based op weight tensors. Per-channel quantization will be used when we provide multiple encodings (equal to the number of channels) for the given tensor. We see an example for convolution weight for the following cases.

- Case 1: Asymmetric encodings without per-channel quantization


```
{
    "features.9.conv.3.weight": [
        {
            "bitwidth": 8,
            "is_symmetric": "False",
            "max": 3.0387749017453665,
            "min": -2.059169834735364,
            "offset": -103,
            "scale": 0.019991940143061618
        }
    ]
}
```


- Case 2: Per-channel quantization encodings with 3 output channels

```
{
    "features.8.conv.3.weight": [
        {
            "bitwidth": 8,
            "is_symmetric": "True",
            "max": 0.7011175155639648,
            "min": -0.7066381259227362,
            "offset": -128.0,
            "scale": 0.005520610358771377
        },
        {
            "bitwidth": 8,
            "is_symmetric": "True",
            "max": 0.5228064656257629,
            "min": -0.5269230519692729,
            "offset": -128.0,
            "scale": 0.004116586343509945
        },
        {
            "bitwidth": 8,
            "is_symmetric": "True",
            "max": 0.7368279099464417,
            "min": -0.7426297045129491,
            "offset": -128.0,
            "scale": 0.005801794566507415
        }
    ]
}
```


Note: Per-channel quantization must use symmetric representation with offset == -2^(bitwidth-1). Per-channel always has is_symmetric = True.

Quantizing a Model

To enable quantization simply pass the option –input_list along with a text file containing raw data inputs to the network. Note that the inputs specified in this file should match exactly with the inputs in the .cpp file generated by conversion. In most cases, these inputs can be obtained directly from the source framework model. However, in rare cases, such as when the inputs are pruned by the converter, these inputs can differ. The format of the file uses a single line for each set of inputs to the network:

```
<inputFile0>
<inputFile1>
<inputFile2>
```

If a network contains multiple inputs they are all listed on a single line separated by a space and prefaced with the input name and a ":="

```
<inputNameA>:=<inputFile0a> <inputNameB>:=<inputFile0b>
<inputNameA>:=<inputFile1a> <inputNameB>:=<inputFile1b>
<inputNameA>:=<inputFile2a> <inputNameB>:=<inputFile2b>
```

Examples

For graph containing a single input the input text file would contain something like:

```
/path/to/file/chair.raw
/path/to/file/mongoose.raw
/path/to/file/honeybadger.raw
```

For a network containing multiple graph inputs:

```
input_left_eye:=left0.rawtensor input_right_eye:=right0.rawtensor
input_left_eye:=left1.rawtensor input_right_eye:=right1.rawtensor
input_left_eye:=left2.rawtensor input_right_eye:=right2.rawtensor
```

Mixed Precision and FP16 Support

Mixed Precision enables specifying different bit widths (e.g. INT8 or INT16) or datatypes (integer or floating point) for different ops within the same graph. Data type conversion ops are automatically inserted when activation precision or data type is different between successive ops. Graphs can have a mix of floating-point and fixed-point data types. Each op can have different precision for weights and activations. However, for a particular op, either all inputs, outputs and parameters

(weights/biases) will be floating-point or all will be integer type. Please refer to the backend supplements for the supported weight/activation bit widths for a particular op.

- CPU
- GPU
- HTP
- HTP FP16

FP16 (half-precision) additionally enables converting the entire models to FP16 or selecting between FP16 and FP32 data-types for the float ops in case of mixed precision graphs with a mix of floating point and integer ops. The different modes of using mixed precision are described below.

Non-quantized Mode

In this mode no calibration images are given (–input_list flag is not given) to the converter. The converted QNN model has only float tensors for both activations and weights.

- Non-quantized FP16: If "–float_bw 16" is added in command line, all activation and weight/bias tensors are converted to FP16.
- Non-quantized FP32: If "–float_bw" is absent from command line or "–float_bw 32" is given, all activation and weight/bias tensors use FP32 format.

Quantized Mode

In this mode calibration images are given (–input_list is given) to converter. The converted QNN model has fixed point tensors for activations and weights.

- No override: If no –quantization_overrides flag is given with an encoding file, all activations are quantized as per –act_bw (default 8) and parameters are quantized as per –weight_bw/–bias_bw (default 8/8) respectively.
- Full override: If –quantization_overrides flag is given along with encoding file specifying encodings for all ops in the model. In this case, the bitwidth with be set as per JSON for all ops defined as

integer/float as per encoding file (dtype='int' or dtype='float' in encoding json).

- Partial override: If –quantization_overrides flag is given along with encoding file specifying partial encodings (i.e. encodings are missing for some ops), the following will happen.
    - Layers for which encoding are NOT available in json file are encoded in the same manner as the no override case i.e. defined as integer with bitwidth defined as per –act_bw/–weight_bw/–bias_bw (or their default values 8/8/8). For some ops (Conv2d, Conv3d, TransposeConv2d, DepthwiseConv2d, FullyConnected, MatMul) even if any of the output/weights/bias are specified as float in the encoding file, all three of them will be overridden to float. The float bitwidth used will be same as the float bitwidth of the overriding tensor in the encodings file. We can also manually control the bitwidth of bias tensors in such case (if encodings for it are absent in encodings json and present for output/weights) with the use of the –float_bias_bw (16/32) flag.
    - Layers for which encoding are available in json are encoded in same manner as full override case.

We show a sample json for network with 3 Conv2d ops. The first and third Conv2d ops are INT8 while the second Conv2d op is marked as FP32. The FP32 op (namely conv2_1) is sandwiched between two INT8 ops in "activation_encodings", hence convert ops will be inserted before and after the FP32 op. The corresponding weights and biases for conv2_1 are also marked as floating-point in the JSON in "param_encodings".

```
{
   "activation_encodings": {
       "data_0": [
           {
               "bitwidth": 8,
               "dtype": "int"
           }
       ],
       "conv1_1": [
           {
               "bitwidth": 8,
               "dtype": "int"
           }
```

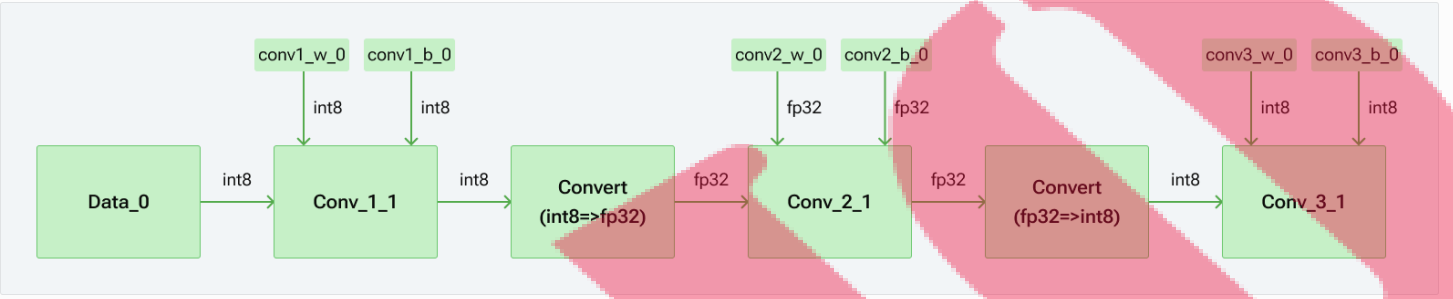
```
        ],
        "conv2_1": [
            {
                "bitwidth": 32,
                "dtype": "float"
            }
        ],
        "conv3_1": [
            {
                "bitwidth": 8,
                "dtype": "int"
            }
        ]
    },
    "param_encodings": {
        "conv1_w_0": [
            {
                "bitwidth": 8,
                "dtype": "int"
            }
        ],
        "conv1_b_0": [
            {
                "bitwidth": 8,
                "dtype": "int"
            }
        ],
        "conv2_w_0": [
            {
                "bitwidth": 32,
                "dtype": "float"
            }
        ],
        "conv2_b_0": [
            {
                "bitwidth": 32,
                "dtype": "float"
            }
        ],
        "conv3_w_0": [
            {
                "bitwidth": 8,
                "dtype": "int"
            }
        ],
```

```
        "conv3_b_0": [
            {
                "bitwidth": 8,
                "dtype": "int"
            }
        ]
    }
}
```


The ops that are not present in json will be assumed to be fixed-point and the bit widths will be selected according to –act_bw/–weight_bw/–bias_bw respectively.


```
{
    "activation_encodings": {
        "conv2_1": [
            {
                "bitwidth": 32,
                "dtype": "float"
            }
        ]
    },
    "param_encodings": {
        "conv2_w_0": [
            {
                "bitwidth": 32,
                "dtype": "float"
            }
        ],
        "conv2_b_0": [
            {
                "bitwidth": 32,
                "dtype": "float"
            }
        ]
    },
    "version": "0.5.0"
}
```

The following quantized mixed-precision graph will be generated based on the JSON shown above. Please note that the convert operations are added appropriately to convert between float and int types and vice-versa.



qairt-quantizer

Note

This tool is still in a Beta release status.

The qairt-converter tool converts non-quantized models into a non-quantized DLC file. Quantizing requires another step. The qairt-quantizer tool is used to quantize the model to one of supported fixed point formats.

For example, the following command will convert an Inception v3 DLC file into a quantized Inception v3 DLC file.

```
$ qairt-quantizer --input_dlc inception_v3.dlc --input_list image_file_list.txt \
                  --output_dlc inception_v3_quantized.dlc
```

To properly calculate the ranges for the quantization parameters, a representative set of input data needs to be used as input into qairt-quantizer using the `--input_list` parameter. The input list specifies paths to raw image files used for quantization. For specifying `--input_list` , refer to input_list argument in qnn-net-run for supported input formats (in order to calculate output activation encoding information for all layers, **do not** include the line which specifies desired outputs).

The tool requires the batch dimension of the DLC input file to be set to 1 during model conversion. The batch dimension can be changed to a different value for inference, by resizing the network during initialization.

Additional details

- qairt-quantizer is majorly similar to snpe-dlc-quant with the following differences:
  - External Overrides and Source Model Encodings (QAT) cached in Float DLC during Conversion stage are applied by default. Use the commandline argument "–ignore_encodings" to ignore Overrides and Source Model Encodings and use Quantizer Runtime to generate encodings using calibration dataset provided through "–input_list".
  - Float_Fallback feature: A commandline option "–float_fallback" is added to enable this feature. When the commandline option is specified, Qairt quantizer produces a fully quantized or mixed precision graph by applying encoding overrides or Source model encodings, propagate encodings accross Data invariant Ops and fallback the missing tensors in float datatype.

    Note: float_fallback and input_list are mutually exclusive options. One of them is mandatory for quantizer
- Outputs can be specified for qairt-quantizer by modifying the input_list in the following ways:

```
#<output_layer_name>[<space><output_layer_name>]
%<output_tensor_name>[<space><output_tensor_name>]
<input_layer_name>:=<input_layer_path>[<space><input_layer_name>:=<input_layer_path>]
```

Note: Output tensors and layers can be specified individually, but when specifying both, the order shown must be used to specify each.

- qairt-quantizer also supports quantization using AIMET, inplace of default Quantizer, when "–use_aimet_quantizer" command line option is provided. To use AIMET Quantizer, run the setup script to create AIMET specific environment, by executing the following command

```
$ source {QNN_SDK_ROOT}/bin/aimet_env_setup.sh --env_path <path where AIMET venv needs to be cr
```

```
--aimet_sdk_tar <AIMET Torch SDK tarball>
```

Note:

1. AIMET Torch Tarball naming convention should be as follows - aimetpro-release-<VERSION (optionally with build ID)>.torch-<cpu/gpu>-.*.tar.gz. For example, aimetpro-release-x.xx.x.torch-xxx-release.tar.gz.
2. Once the setup script is run, ensure that AIMET_ENV_PYTHON environment variable is set to <AIMET virtual environment path>/bin/python
3. Minimum AIMET version supported is, **AIMET-1.32.0**

Previous Next

Tutorials

This section contains helpful tutorials that assist users to become familiar with the Qualcomm® AI Engine Direct workflow, and to be able to interact with Qualcomm® AI Engine Direct API and tools to enable their deep learning based use-cases.

Getting Started

- Converting and executing a CNN model with QNN
- Saver Tutorial: Save execution sequence with Saver and replay on a backend
- Sample App Tutorial: Create and build a sample C++ application

Advanced

- QNN HTP Shared Buffer Tutorial
- Executing with DLCs

Custom Operators

- Executing a shallow model using custom op package
- Converting and executing a CNN model with custom operations

Windows

- Windows Hibernation Tutorial
- ARM64X Tutorial