Qualcomm Linux Lite Runtime Reference

80-70018-54 AA

April 7, 2025

Contents

1 LiteRT overview

Lite Runtime (LiteRT) is an open-source deep learning framework designed for on-device inference. The TensorFlow framework provides tools and APIs to convert a standard pretrained TensorFlow model from the SavedModel or Keras format into a LiteRT format.

Topics covered describe the available delegates and methods to run LiteRT models using the Qualcomm® software stack, and explain how to:

- Run LiteRT models using the GStreamer-based Qualcomm® Intelligent Multimedia SDK (Qualcomm IM SDK) or the native LiteRT application.
- Convert TensorFlow models to LiteRT models and optimize them for on-device inference.
- Run LiteRT models using a delegate on hardware accelerators, such as CPU, GPU, and the Qualcomm® Hexagon™ Tensor Processor.
- Benchmark LiteRT models.

1.1 Next steps

Run a LiteRT model

→ Use the GStreamer-based Qualcomm IM SDK

→ Use a native LiteRT sample application

LiteRT developer workflow

→ Convert a TensorFlow model to a LiteRT model

→ Create an application and run inference

→ Develop a custom application

Sample applications

→ Download models and sample images

→ Run a LiteRT model using an available delegate

→ Run a QNN delegate using an external delegate

Note: See Hardware SoCs that are supported on Qualcomm® Linux®.

2 Get started with LiteRT

This information explains how to run LiteRT models on the Qualcomm Linux development kit.

Before you get started with running LiteRT models, do the following:

1. Set up the Qualcomm Linux development kit. For instructions, see the following:
   - QCS6490/QCS5430: Qualcomm® RB3 Gen 2 Quick Start Guide
   - QCS9075: Qualcomm® IQ-9 Beta Evaluation Kit Quick Start Guide
   - QCS8275: Qualcomm® IQ-8 Beta Evaluation Kit Quick Start Guide

   Note: The QCS9075 and QCS8275 quick start guides are available for authorized users only. To upgrade your access, go to www.qualcomm.com/support/working-with-qualcomm.

2. Connect the Qualcomm Linux development kit to a monitor using HDMI.
3. Upgrade the Qualcomm Linux development kit to the latest available software release. For instructions, see Download the Platform eSDK.
4. Flash the image to the device. For instructions, see Flash images.

2.1 Run a LiteRT model using the GStreamer-based Qualcomm IM SDK

The Qualcomm Linux development kit contains precompiled LiteRT sample applications to run sample LiteRT models.

The gst-ai-classification sample application uses the Qualcomm IM SDK plug-ins to run a LiteRT classification model on the Qualcomm Linux development kit. The sample application achieves hardware acceleration using LiteRT delegates. The following figure shows the pipeline, which receives a video stream from a camera, does the preprocessing, runs the inference on the AI hardware, and displays the results.

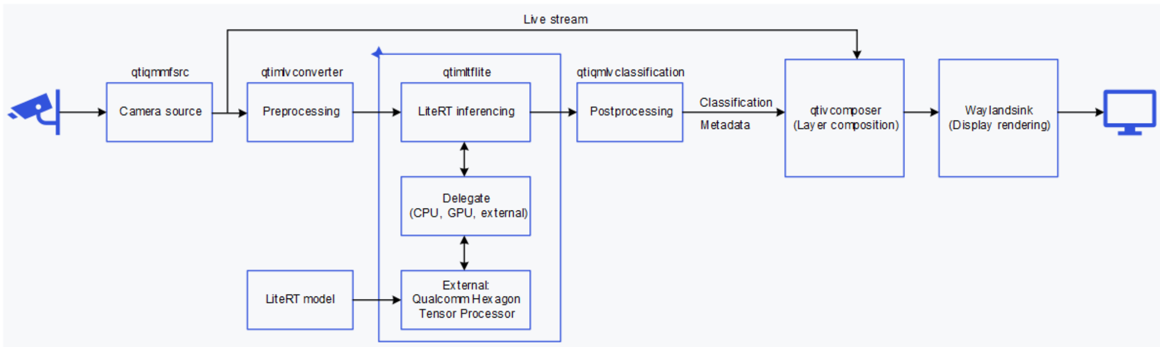


Figure: Workflow to run a LiteRT model using Qualcomm IM SDK

The gst-ai-classification sample application does the following:

1. Opens the IMX577 camera on the Qualcomm Linux development kit with a specific resolution and frame rate; for example, 1080p at 30 fps
2. Preprocesses each camera frame to provide the input data to a classification model

   For example, the gst-ai-classification sample application:

   a. Downscales a 1080p frame to a 224 x 224 resolution

   b. Normalizes the input frame based on the model requirements
3. The qtimltflite Qualcomm IM SDK plug-in, built on top of the LiteRT C++ API, does the following:

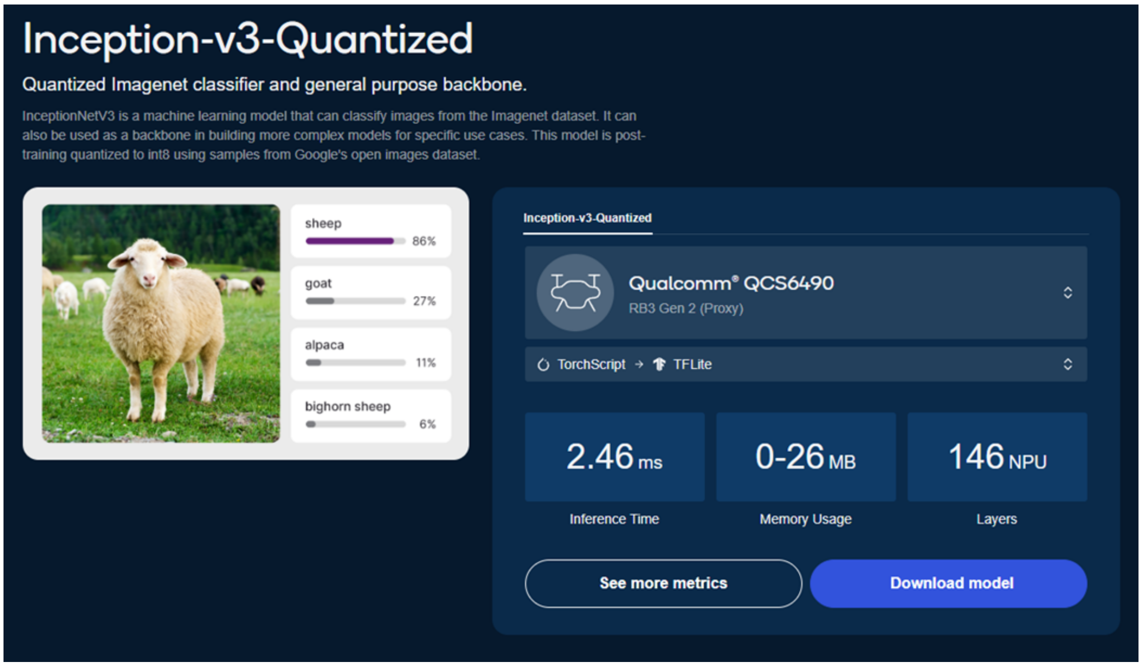
   a. Loads the sample LiteRT classification model

   b. Performs inference on the model using hardware acceleration
4. Postprocesses the output from the inference, that is, extracts the label with the highest predicted probability within the output tensor
5. Overlays the inference result on the original camera input image and displays it on the connected monitor

Download and copy a sample model

To download and copy a model and a label file to the device, do the following:

1. Go to Qualcomm ® AI Hub, and download the Inception-v3-Quantized model.



Note: The gst-ai-classification sample application is demonstrated for QCS6490.

2. To download the corresponding label file, run the following command:

```
wget https://raw.githubusercontent.com/quic/ai-hub-models/refs/
heads/main/qai_hub_models/labels/imagenet_labels.txt
```

Note: The model is available on Qualcomm AI Hub and the corresponding label file is available on QUIC GitHub.

3. To copy the models and label files to the device using the secure copy protocol (SCP), run the following commands:

```
# For SCP, run the following command:
ssh root@[ip-addr]
mount -o remount,rw /usr
cd /etc
mkdir labels
mkdir media
exit
```

```
# Copy files securely
scp imagenet_labels.txt root@[ip-addr]:/etc/labels
scp inception_v3_quantized.tflite root@[ip-addr]:/etc/models
```

Note: To get the IP address of the Qualcomm Linux development kit, run the following command:

```
ifconfig wlan0
```

Note: When prompted for a password, enter *oelinux123*.

Next steps

- Run AI/ML sample applications

Run a LiteRT model with a sample application

1. To run inference using LiteRT, run the following command:

```
ssh root@[ip-addr]
```

   a. To set up the Wayland Display environment, run the following command:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && export WAYLAND_
DISPLAY=wayland-1
```

Note: If Weston is not enabled automatically, start two instances of secure shell: one to enable Weston and the other to run the application.

   i. To enable Weston, run the following command in the first shell:

```
export GBM_BACKEND=msm && export XDG_RUNTIME_DIR=/dev/
socket/weston && mkdir -p $XDG_RUNTIME_DIR && weston --
continue-without-input --idle-time=0
```

   ii. To set up the Wayland Display environment, run the following command in the second shell:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && export
WAYLAND_DISPLAY=wayland-1
```

   b. Modify the config_classification.json file in the `/etc/configs` folder, as follows:

```
{
  "file-path":"/etc/media/video.mp4",
  "ml-framework": "tflite",
  "model":"/etc/models/inception_v3_quantized.tflite",
  "labels": "/etc/labels/imagenet_labels.txt",
  "constants": "Inceptionv3,q-offsets=<38.0>,q-scales=<0.
17039915919303894>;"
}
```

Note: You must push the video.mp4 file to the `/etc/media` folder. The default path for the video file is `/etc/media/video.mp4`, labels path is `/etc/labels/classification.labels`, and model is `/etc/model/inception_v3_quantized.tflite`.

   c. Run the classification sample application:

```
gst-ai-classification --config-file=/etc/configs/config_
classification.json
```

2. To run the sample application using a custom classification model and labels file, use the following arguments:

   - `--model`
   - `--labels`

   a. Modify the config_classification.json file in the `/etc/configs` folder, as follows:

   ```
   {
     "file-path": "/etc/media/video.mp4",
     "model":"/etc/models/custom_model.tflite",
     "ml-framework": "tflite",
     "labels": "/etc/labels/custom_labels.txt"
   }
   ```

   b. Run the classification sample application:

   ```
   gst-ai-classification --config-file=/etc/configs/config_
   classification.json
   ```

3. To stop the sample application, select CTRL+C.

When the sample application is running, it displays the video stream on the connected monitor with inference results overlaid on the frame.

2.2 Run a LiteRT model using a native LiteRT sample application

You can run LiteRT models using a sample LiteRT application called label_image, which is a part of the TensorFlow repository.

The label_image sample application and the LiteRT library are cross-compiled with Qualcomm Linux and installed on the target device.

The label_image sample application does the following:

1. Loads a classification LiteRT model
2. Performs inference on an image using a delegate to speed up the model on Qualcomm hardware

To run a model using the label_image sample application, do the following:

1. Download the sample model, corresponding labels, and an example image:

- BMP file from here
- MobileNet LiteRT model from here

2. Run the following commands on the host computer:

```
wget http://download.tensorflow.org/models/mobilenet_v1_2018_08_
02/mobilenet_v1_1.0_224_quant.tgz
```

```
tar -xvf mobilenet_v1_1.0_224_quant.tgz
```

```
wget https://storage.googleapis.com/download.tensorflow.org/
models/mobilenet_v1_1.0_224_frozen.tgz
```

```
tar -xvf mobilenet_v1_1.0_224_frozen.tgz
```

```
# For SCP, run the following command:
ssh root@[ip-addr]
mount -o remount,rw /usr
cd /etc
mkdir artifacts
exit
```

```
scp mobilenet_v1_1.0_224_quant.tflite root@[ip-addr]:/etc/
artifacts
scp grace_hopper.bmp root@[ip-addr]:/etc/artifacts
scp mobilenet_v1_1.0_224/labels.txt root@[ip-addr]:/etc/
artifacts
scp mobilenet_v1_1.0_224.tflite root@[ip-addr]:/etc/artifacts
```

3. To run an inference using either of the following delegates, do the following:

   - To run the model on the Arm® CPU using the XNNPACK delegate:

   ```
   label_image -l /etc/artifacts/labels.txt -i /etc/artifacts/
   grace_hopper.bmp -m /etc/artifacts/mobilenet_v1_1.0_224_
   quant.tflite -c 10 -p 1 --xnnpack_delegate 1
   ```

   - To run the model on the Qualcomm® Adreno™ GPU using the GPU delegate:

   ```
   label_image -l /etc/artifacts/labels.txt -i /etc/artifacts/
   grace_hopper.bmp -m /etc/artifacts/mobilenet_v1_1.0_224.
   tflite -c 10 -p 1 --gl_backend 1
   ```

Known issue

The LiteRT native sample application (label_image) might crash during inferencing on the GPU or external delegate.

2.3 Next steps

- Run LiteRT sample applications

3 LiteRT architecture

The LiteRT framework optimizes models for latency, model size, and power consumption. It helps to run them on devices with low-power requirements, such as mobile, embedded, and edge platforms.

The framework runs models with the help of delegates. Delegates are software layers that use libraries to run a neural network model efficiently on specific hardware. The following figure shows the delegates the LiteRT framework uses to run models.

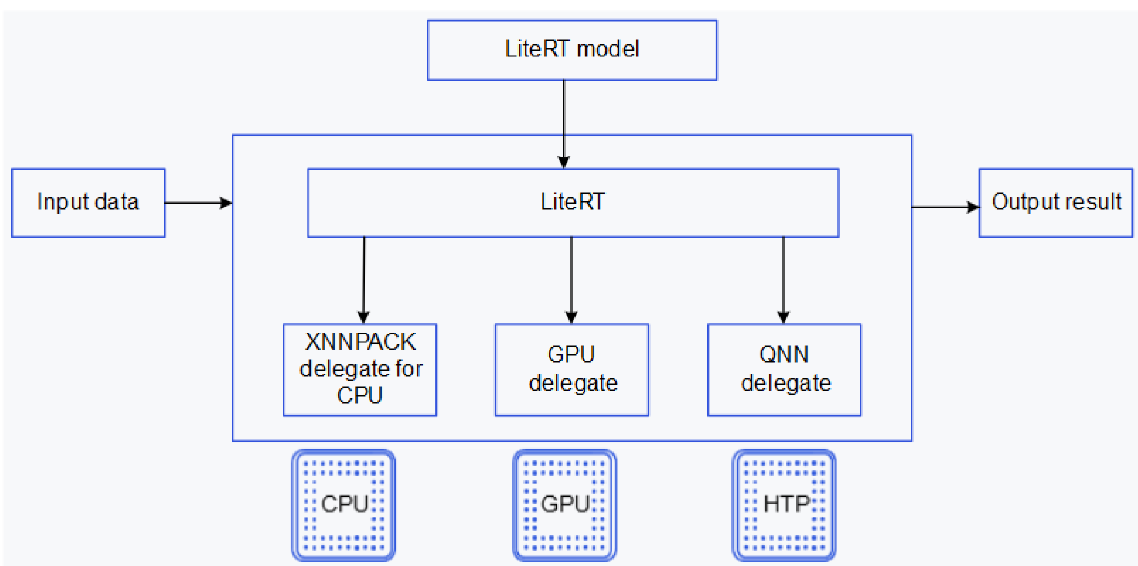


Figure: LiteRT architecture

3.1 LiteRT on-device inference

The LiteRT on-device inference loads the model into an interpreter, which parses the model and uses a delegate to run it.

The process includes the following:

1. The inference loads the LiteRT model into a LiteRT interpreter interface, which parses the model to identify the neural network operators present in it.
2. The interpreter interface is further configured to run the model using a delegate.
3. The interpreter invokes a model inference on the provided inputs and saves the corresponding outputs into the buffers provided to the interpreter interface.

Qualcomm supports running LiteRT models on the following accelerators using delegates:

- CPU
- Adreno GPU
- Hexagon Tensor Processor

The following table lists the delegates and their accelerators.

Table: Supported delegates and accelerators

Delegate	Acceleration
XNNPACK delegate	CPU
GPU delegate	GPU
Qualcomm® AI Engine direct delegate (Qualcomm® Neural Network (QNN) delegate)	CPU, GPU, and Hexagon Tensor Processor

3.2 Delegates for LiteRT

Delegates allow you to offload the LiteRT graph execution to hardware accelerators, such as CPU, GPU, and the Hexagon Tensor Processor.

Currently, LiteRT supports the following delegates:

XNNPACK delegate for CPU

The XNNPACK delegate uses the XNNPACK library to speed up LiteRT models efficiently on CPUs.

XNNPACK is an open-source library from Google, which does the following:

- Provides an optimized implementation of neural network operators for Arm CPUs
- Uses low-level CPU instructions, such as the Arm® Neon™ instruction set, to optimize operators for efficient execution

The XNNPACK delegate can run models in both 32-bit floating-point and INT8 formats. For more information, see XNNPACK back-end for TensorFlow Lite.

GPU delegate

The GPU open-source delegate accelerates LiteRT models on various vendor-specific GPUs, including the Adreno GPU.

LiteRT can use the GPU delegate to improve the parallel-processing power of GPUs, which makes inferencing faster. The GPU delegate uses OpenCL kernels to run neural network operations within a LiteRT model execution graph on the GPU.

The default cross-compilation of the GPU delegate includes the LiteRT library, optimizing the execution of the following LiteRT models on the Adreno GPU:

- 16-bit floating-point
- 32-bit floating-point

For more information, see GPU delegates for LiteRT.

QNN delegate

The QNN delegate is a proprietary delegate designed for vendor-specific hardware acceleration to speed up LiteRT models. It's based on the external delegate interface of LiteRT.

You can use the QNN delegate to offload parts or the entire LiteRT model to specialized Qualcomm hardware, such as the Adreno GPU and the Hexagon Tensor Processor.

This delegate improves model execution performance and power efficiency by reducing the CPU workload. It also uses the existing Qualcomm AI Engine direct APIs and available back ends to speed up models. For more information about these APIs, see Qualcomm AI Engine direct SDK.

The QNN delegate can run models in both 32-bit floating-point precision and INT8 precision on the available hardware.

You can build applications using the following interfaces:

- Qualcomm AI Engine direct delegate interface
- LiteRT external delegate interface

You can access both the interfaces when using a standalone LiteRT application. However, if you deploy your LiteRT models using the Qualcomm IM SDK, the qtimltflite GStreamer plug-in for Qualcomm TensorFlow Lite uses the QNN delegate. For more information, see Leverage external delegate.

The following figure shows the directory structure of QNN delegate libraries from the Qualcomm AI Engine direct SDK.

```
<QNN_SDK_ROOT>
├── bin
│   ├── aarch64-android
│   │   └── qtld-net-run
│   ├── aarch64-oe-linux-gcc9.3
│   │   └── qtld-net-run
│   ├── aarch64-oe-linux-gcc11.2
│   │   └── qtld-net-run
│   └── aarch64-ubuntu-gcc9.4
│       └── qtld-net-run
├── docs
├── examples
├── include
│   └── QNN
│       └── TFLiteDelegate
│           └── QnnTFLiteDelegate.h
├── lib
│   ├── aarch64-android
│   │   └── libQnnTFLiteDelegate.so
│   ├── aarch64-oe-linux-gcc9.3
│   │   └── libQnnTFLiteDelegate.so
│   ├── aarch64-oe-linux-gcc11.2
│   │   └── libQnnTFLiteDelegate.so
│   ├── aarch64-ubuntu-gcc9.4
│   │   └── libQnnTFLiteDelegate.so
│   └── android
│       └── qtld-release.aar
├── LICENSE.pdf
├── QNN_TFLITE_DELEGATE_NOTICE.txt
├── QNN_TFLITE_DELEGATE_README.txt
└── QNN_TFLITE_DELEGATE_ReleaseNotes.txt
```

Figure: QNN delegate directory structure

Qualcomm AI Engine direct delegate interface

The Qualcomm AI Engine direct delegate interface, also known as the QNN delegate, provides the `QnnTFLiteDelegate.h` header as an interface. You can include this header in your application before linking it to the QNN delegate library.

You can find the compatible QNN delegate library and QNN libraries in the `aarch64-oe-linux-gcc11.2 cross-compiler` toolchain triplet directory.

Table: QNN delegate acceleration support

Back end name	Back end description	Target and library names	Library description
CPU	Back end for Arm CPU acceleration	• `aarch64-oe-linux-gcc11.2` – `libQnnCpu.so`	`libQnnCpu.so`: CPU back-end library
GPU	Back end for the Adreno GPU hardware accelerator	• `aarch64-oe-linux-gcc11.2` – `libQnnGpu.so`	`libQnnGpu.so`: GPU back-end library
Hexagon Tensor Processor	Back end for the Hexagon Tensor Processor hardware accelerator	• `aarch64-oe-linux-gcc11.2` – `libQnnHtp.so` – `libQnnHtpPrepare.so` – `libQnnHtpV68Stub.so` • `hexagon-v68` – `libQnnHtpV68Skel.so`	• `libQnnHtp.so`: Library used for communication between the Arm CPU and the Hexagon Tensor Processor. • `libQnnHtpPrepare.so`: Hexagon Tensor Processor library running on the CPU side. This library prepares the graph and optimizes the execution graph at runtime. • `libQnnHtpV68Stub.so`: Hexagon Tensor Processor proxy library on the CPU side, communicating with the Skel library loaded on the Hexagon Tensor Processor. • `libQnnHtpV68Skel.so`: Hexagon Tensor Processor native library with optimized kernel/operator implementation for the Hexagon Tensor Processor.

Note: Select the appropriate libraries based on the Hexagon Tensor Processor version of the Qualcomm Linux development kit. The versions are as follows:

- QCS6490/QCS5430: Hexagon Tensor Processor v68
- QCS9075: Hexagon Tensor Processor v73
- QCS8275: Hexagon Tensor Processor v75

LiteRT external delegate interface

To run LiteRT models using an external delegate interface, the application must load the `libQnnTFLiteDelegate.so` QNN delegate library. The C/C++ application and the `libQnnTFLiteDelegate.so` delegate library have no dependency on each other. Therefore, if the delegate changes, you don't have to recompile the application.

To use the C++ API and run inference with LiteRT on Linux, see Run inference using C++.

The QNN delegate provides acceleration on the Hexagon Tensor Processor, GPU, and CPU. To customize where and how to run models using the QNN delegate with the external delegate interface, you must provide more external delegate options. For instructions, see External delegate options for QNN delegate.

3.3 Next steps

- Get started
- Run LiteRT sample applications

4 LiteRT developer workflow

You can use an existing LiteRT model by downloading it from the open-source community. Or you can convert a TensorFlow or Keras model to the LiteRT format using specific tools. After converting the model, you can run inference on a device and develop a custom application for the LiteRT model.

The following figure shows the tasks involved in using existing models, converting and quantizing models, and creating an application to run inference.

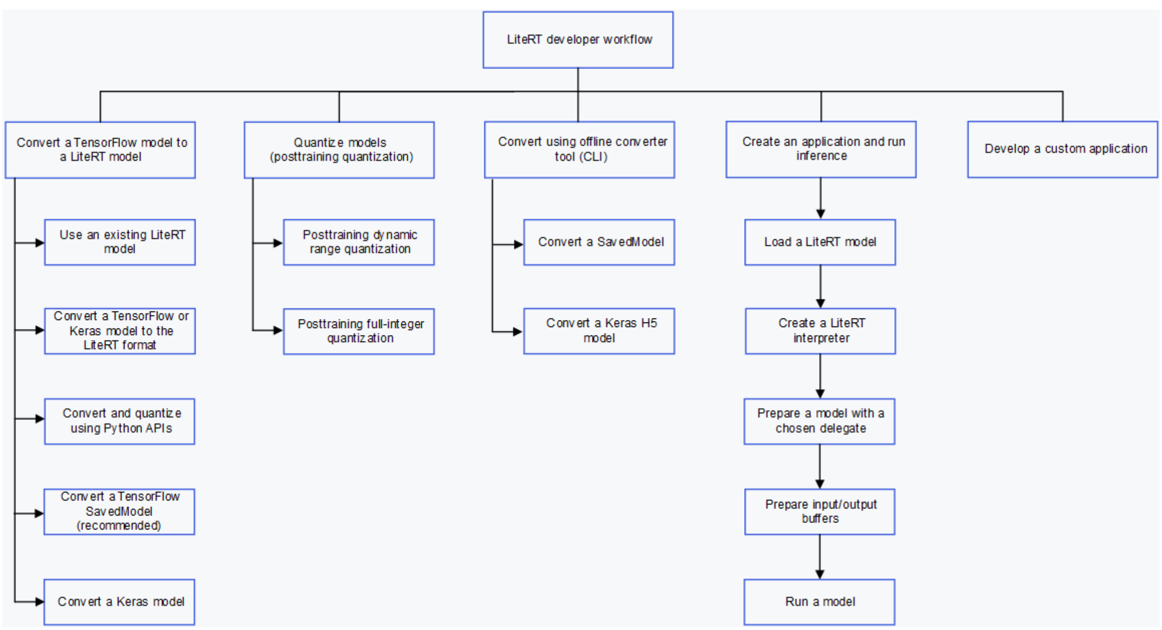


Figure: LiteRT developer workflow

Note: If you are using LiteRT models from Qualcomm AI Hub or other sources, you may skip the tasks described in the LiteRT developer workflow.

Running a LiteRT model on Qualcomm-specific hardware involves the following tasks.

4.1 Convert a TensorFlow model to a LiteRT model

You can convert TensorFlow models to LiteRT models and optimize them for on-device inference. For more information about LiteRT model conversion, see Model conversion overview.

LiteRT model conversion supports converting models to the following formats:

- 32-bit floating-point precision
- 16-bit floating-point precision
- UINT8/INT8 precision (quantizing models)

Use an existing LiteRT model

You can deploy an existing LiteRT model from the open-source community using LiteRT.

Qualcomm AI Hub publishes LiteRT models optimized for the Qualcomm Linux development kit. For LiteRT models from Qualcomm, see Qualcomm AI Hub.

To download an optimized model from Qualcomm AI Hub, do the following:

1. Go to Qualcomm AI Hub IOT models page.
2. From the left pane, filter the available models by selecting a chipset.
3. Select a model.
4. On the next page, select the *TorchScript* > *TFLite* path.
5. Select *Download model*.

Note: The downloaded model is pre-optimized and ready for deployment.

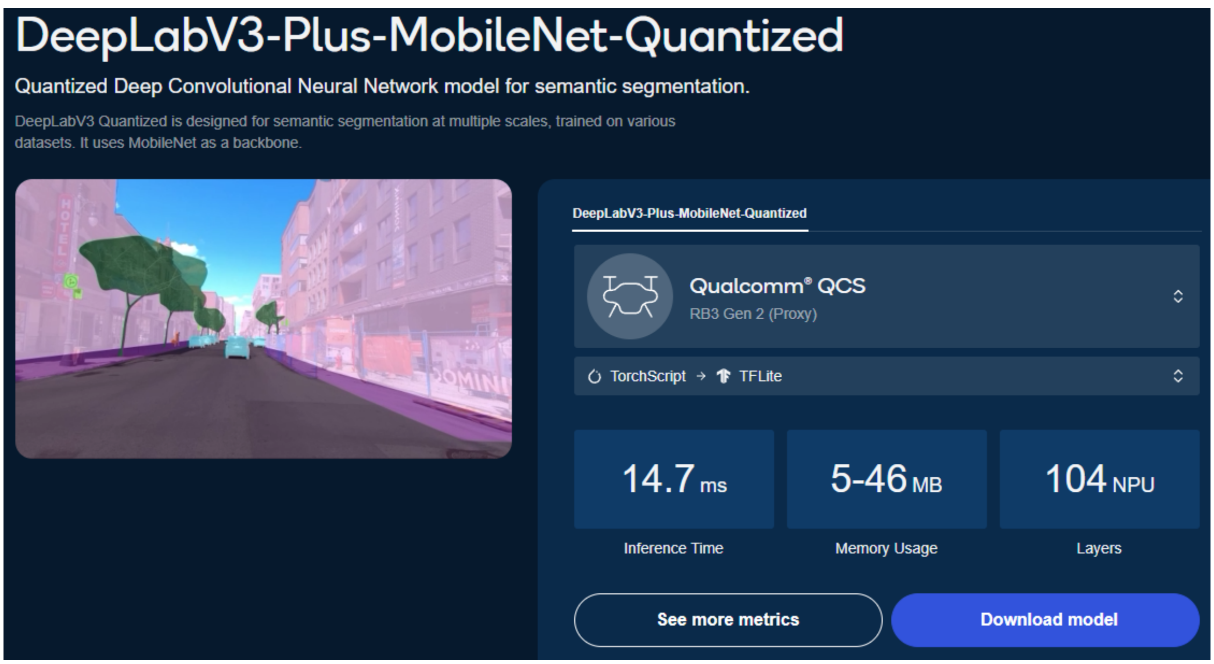


Figure: Optimized LiteRT model on Qualcomm AI Hub

Convert a TensorFlow or Keras model to the LiteRT format

The TensorFlow framework provides both Python APIs and a command-line interface (CLI) tool to convert a TensorFlow or Keras model to the LiteRT format.

Table: TensorFlow model conversion methods

Conversion method	Description
Python APIs	Converts, optimizes, and quantizes models to the LiteRT format
CLI tool	Converts models to the LiteRT format, but is suitable for basic model conversion only

Note: The TensorFlow to LiteRT Python APIs offer more flexibility to convert, optimize, and quantize models to suit your requirements.

Convert and quantize using Python APIs

TensorFlow provides the following APIs to convert a TensorFlow SavedModel or a Keras model to a LiteRT model.

Table: TensorFlow Python APIs to convert models

API	Description
tf.lite.TFLiteConverter.from_saved_model() (recommended)	Converts a TensorFlow SavedModel
tf.lite.TFLiteConverter.from_keras_model()	Converts a Keras model

Recommended: Convert a TensorFlow SavedModel

The following example shows how to convert a TensorFlow model saved in the saved_model format to a LiteRT model:

```
import tensorflow as tf

# Convert the model
saved_model_dir = "/path/to/tf/model/in/saved_model/format"
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_model = converter.convert()

# Save the model
with open("model.tflite", "wb") as f:
    f.write(tflite_model)
```

Note: The converted LiteRT model isn't quantized, and its data is in 32-bit floating-point precision.

Convert a Keras model

The following example shows how to convert a Keras model to a LiteRT model:

```python
import tensorflow as tf

# Create a model using high-level tf.keras.* APIs
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=1, input_shape=[1]),
    tf.keras.layers.Dense(units=16, activation='relu'),
    tf.keras.layers.Dense(units=1)
])

# compile the model
model.compile(optimizer='sgd', loss='mean_squared_error')

# train the model
model.fit(x=[-1, 0, 1], y=[-3, -1, 1], epochs=5)

# Convert the model to LiteRT
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model
with open('model.tflite', 'wb') as f:
    f.write(tflite_model)
```

Note: The converted LiteRT model isn't quantized, and its data is in 32-bit floating-point precision.

4.2 Quantize models

After converting a model to the LiteRT format, you can quantize it. Quantization in neural network models involves the following steps:

1. Quantize weights and biases: These are already part of the trained model and you can quantize them without additional information. Therefore, quantizing weights and biases is a static step.

2. Quantize activation layers: The ranges for the activation layer output depend on the input image during forward propagation. Therefore, a set of sample inputs, known as calibration or representative data sets, is necessary to quantize these layers and identify the minimum and maximum ranges.

To quantize a TensorFlow floating-point model to a quantized LiteRT model, LiteRT provides posttraining quantization techniques. For more information, see Posttraining quantization.

Posttraining quantization

LiteRT supports two types of posttraining quantizations:

- Posttraining dynamic range quantization
- Posttraining full-integer quantization

Posttraining dynamic range quantization

In posttraining dynamic range quantization, weights and biases are statically quantized from floating-point precision to fixed-point integer 8-bit precision. The activation layer ranges remain in 32-bit floating-point precision.

To reduce latencies during inference, dynamic-range operators:

- Quantize activations based on their ranges to fixed-point integer 8-bit precision
- Perform computations with 8-bit weights and activations

Note: This step only quantizes weights and doesn't need extra calibration data.

The following script converts and quantizes a TensorFlow model to a LiteRT model:

```
import tensorflow as tf
from tensorflow import keras

converter = tf.lite.TFLiteConverter.from_saved_model(exp_model_path)
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_
BUILTINS]
converter.optimizations = [tf.lite.Optimize.DEFAULT]

tflite_model = converter.convert()
save_name = 'quantized_model.tflite'

print('Saving Dynamic Quantized LiteRT model .................')

with open(save_name, 'wb') as f:
```

```
    f.write(tflite_model)
```

Posttraining full-integer quantization

In full-integer quantization, a representative data quantizes the activation layers within the model.

The following script converts and quantizes a TensorFlow model to a LiteRT model. It generates a full-integer quantized model that's more suitable for fixed-point integer hardware, such as the Hexagon Tensor Processor on the Qualcomm Linux development kit.

```
import tensorflow as tf

def representative_dataset():
    for data in dataset:
        yield {
            "image": data.image,
            "bias": data.bias,
        }
saved_model_dir = "/path/to/saved/model"

# prepare converter by loading model in saved_model format.
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Set representative dataset used for quantization.
converter.representative_dataset = representative_dataset

# For full-integer quantization, set target_spec supported_ops to
TFLITE_BUILTINS_INT8.
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_
BUILTINS_INT8]
converter.inference_input_type = tf.int8  # or tf.uint8
converter.inference_output_type = tf.int8  # or tf.uint8

# Convert model
tflite_quant_model = converter.convert()
save_name = 'quantized_model_int8.tflite'

print('Saving Quantized LiteRT model ..................')

with open(save_name, 'wb') as f:
    f.write(tflite_model)
```

Note: supported_ops in converter sets target_spec to tf.lite.OpsSet.TFLITE_BUILTINS_INT8.

4.3 Convert using offline converter tool

The TensorFlow pip package includes the tflite_convert TensorFlow Lite offline converter tool (CLI), which you can use offline for TensorFlow versions 2.x and later.

The tflite_convert tool accepts the following input in the CLI:

```
tflite_convert --help

optional arguments:
  -h, --help            show this help message and exit
  --output_file OUTPUT_FILE
                        Full filepath of the output file.
  --saved_model_dir SAVED_MODEL_DIR
                        Full path of the directory containing the
SavedModel.
  --keras_model_file KERAS_MODEL_FILE
                        Full filepath of HDF5 file containing tf.
Keras model.
  --saved_model_tag_set SAVED_MODEL_TAG_SET
                        Comma-separated set of tags identifying the
MetaGraphDef within the SavedModel to analyze. All tags must be
present. To pass in an empty
                        tag set, pass in "". (default "serve")
  --saved_model_signature_key SAVED_MODEL_SIGNATURE_KEY
                        Key identifying the SignatureDef containing
inputs and outputs. (default DEFAULT_SERVING_SIGNATURE_DEF_KEY)
  --enable_v1_converter
                        Enables the TensorFlow V1 converter in 2.0
```

Convert a SavedModel

To convert a typical TensorFlow model in the saved_model format using the tflite_convert tool, run the following command:

```
tflite_convert \
      --saved_model_dir=/tmp/mobilenet_saved_model \
      --output_file=/tmp/mobilenet.tflite \
      --saved_model_tag_set=serve \
         --saved_model_signature_key="serving_default"
```

Convert a Keras H5 model

To convert a Keras model using the tflite_convert tool, run the following command:

```
tflite_convert \
  --keras_model_file=/tmp/mobilenet_keras_model.h5 \
  --output_file=/tmp/mobilenet.tflite
```

Note: The tflite_convert tool is suitable for basic purposes only. For posttraining integer quantization, Qualcomm recommends using Python APIs.

4.4 Create an application and run inference

You can use the LiteRT C++ APIs to create an application, load a LiteRT model, and run it on hardware using delegates.

The following figure shows the steps involved in creating an application using C++ APIs to run a LiteRT model.

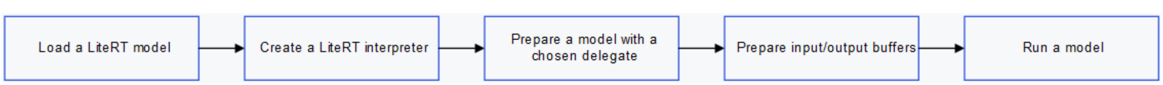


Figure: Workflow to create an application and run a LiteRT model

Load a LiteRT model

A LiteRT model is a FlatBuffers file that has information on model operators and any associated weights and biases.

The contents of the FlatBuffers file include the following:

- Tensors (input and outputs of each operation)
- Buffers (weights and biases)
- Operations that create an execution graph

The LiteRT framework provides APIs to do the following:

- Load a LiteRT model file
- Unpack all the content of the FlatBuffers file into memory

You can use the following APIs to load a LiteRT model for inference:

```
include <cstdio>
include <iostream>
include "tensorflow/lite/interpreter.h"
include "tensorflow/lite/kernels/register.h"
include "tensorflow/lite/model.h"
include "tensorflow/lite/optional_debug_tools.h"

std::unique_ptr<tflite::FlatBufferModel> model;

model = tflite::FlatBufferModel::BuildFromFile(model_name.c_str());
if (!model) {
  std::cerr << "Failed to mmap model " << model_name << std::endl;
  exit(-1);
}
```

Create a LiteRT interpreter

Using the TensorFlow C/C++ APIs, you can build an interpreter to run the model.

The interpreter interface helps you to do the following:

- Configure model execution on a chosen delegate.
- Assign the memory needed to for forward propagation.

The following example code demonstrates how you can create an interpreter. You can configure the interpreter instance to use a specific delegate and perform forward propagation.

```
//Build the interpreter with the InterpreterBuilder.
//Note: all Interpreters should be built with the InterpreterBuilder,
// which allocates memory for the Interpreter and does various set up
// tasks so that the Interpreter can read the provided model.

tflite::ops::builtin::BuiltinOpResolver resolver;
tflite::InterpreterBuilder builder(*model, resolver);
std::unique_ptr<tflite::Interpreter> interpreter;
builder(&interpreter);
if (!interpreter) {
   std::cerr << "Failed to construct interpreter on provided tflite
model" << std::endl;
}
if (interpreter->AllocateTensors() != kTfLiteOk) {
   std::cerr <<  "Failed to allocate tensors!" << std::endl;
   exit(-1);
}
```

Prepare a model with a chosen delegate

After creating an interpreter and allocating the necessary memory to run the model, prepare the model with a chosen delegate. This step creates an execution graph from the model loaded earlier and uses the underlying library to perform inference on the delegate hardware.

The following example code creates the XNNPACK delegate for running a LiteRT model on the Arm CPU. It creates the delegate by calling the `TfLiteXNNPackDelegateCreate(...)` API. You can also customize the delegate using the Delegate Options API.

```
TfLiteDelegate *delegate = NULL;
TfLiteXNNPackDelegateOptions xnnpack_options =
TfLiteXNNPackDelegateOptionsDefault();
xnnpack_options.num_threads = num_threads;

TfLiteDelegate* xnnpack_delegate =
```

```
TfLiteXNNPackDelegateCreate(&xnnpack_options);
if (interpreter->ModifyGraphWithDelegate(xnnpack_delegate) !=
kTfLiteOk) {
  // Report error and fall back to another delegate, or the default
backend
}
```

Prepare input/output buffers

When you build a standalone LiteRT application, it's essential to prepare input data, such as camera frames, for the pipeline to run LiteRT models.

Preprocessing operations, in the following cases for example, are important to ensure that inference happens correctly:

- Resizing the input image to a resolution expected by the model
- Normalization
- Mean subtraction

Run a model

To run inference on a model, you must invoke a delegate using the `Invoke()` API. Before invoking this API, create the appropriate input/output buffers and provide them to the interpreter.

After the inference is complete, you can parse the output from the output buffers of the interpreter to generate the inference results.

An example of the `Invoke()` API running a model using a delegate is as follows:

```
// Run Inference
interpreter->Invoke()
```

After the inference is complete, you can find the output tensors from the LiteRT `Invoke()` API in the output buffers of the interpreter. To perform further postprocessing on these outputs, you can parse them from the interpreter.

For a comprehensive example, see the label_image example in the TensorFlow GitHub repository.

For more information, see LiteRT documentation.

4.5 Develop a custom application

To enhance the developer experience, the Qualcomm IM SDK provides the qtimltflite GStreamer-based plug-in, which performs LiteRT model inference.

For more details, see the following:

- Qualcomm IM SDK documentation
- qtimltflite plug-in documentation
- Develop your own application

5 Run LiteRT sample applications

The LiteRT framework provides sample applications that you can use to do the following:

- Run an arbitrary LiteRT model
- Perform benchmarking

5.1 Download models and sample images

The sample applications in Get started use the label_image sample application provided by the LiteRT framework, which can run any classification models. For example, MobileNet v1, v2.

Before you begin, ensure that you have the following:

- Ubuntu 22.04 host computer
- Qualcomm Linux development kit

To use the sample applications, download the following:

- Sample model
- Corresponding file containing labels
- Sample image

The sample applications use the MobileNet v1 model, which is trained on an ImageNet data set with 1000 classes as an example. MobileNet v1 demonstrates a model trained to classify an image.

For instructions on how to download and copy the models, label files, and the sample image to the device, see Get started.

5.2 Run a LiteRT model using an available delegate

The LiteRT open-source framework provides the label_image sample application to run a LiteRT model using an available delegate. The source code for the label_image sample application is available on the TensorFlow GitHub repository.

The label_image sample application is cross-compiled along with the LiteRT library and installed on the target device.

The following example demonstrates how to run LiteRT models using the available LiteRT delegates:

Note: You can also use a delegate with the label_image sample application.

- To use the XNNPACK delegate, run the following commands:

```
ssh root@[ip-addr]
cd /etc/artifacts
```

```
label_image -l /etc/artifacts/labels.txt -i /etc/artifacts/
grace_hopper.bmp -m /etc/artifacts/mobilenet_v1_1.0_224_quant.
tflite -c 10 -p 1 --xnnpack_delegate 1
```

- To use the GPU delegate, run the following commands:

```
ssh root@[ip-addr]
cd /etc/artifacts
```

```
label_image -l /etc/artifacts/labels.txt -i /etc/artifacts/
grace_hopper.bmp -m /etc/artifacts/mobilenet_v1_1.0_224_quant.
tflite -c 10 -p 1 --gl_backend 1
```

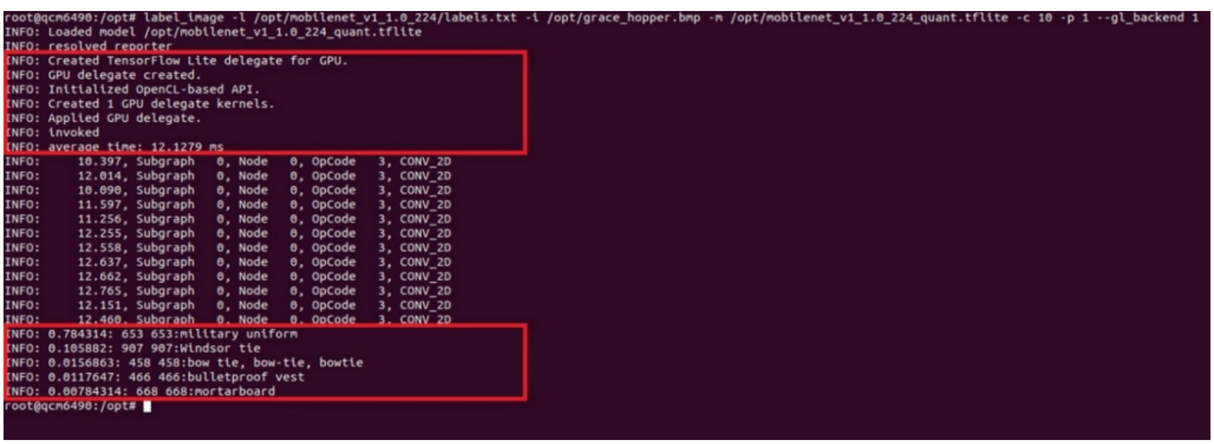


Figure: Performance statistics for GPU delegate creation

Benchmark LiteRT model performance

The open-source LiteRT provides a tool to benchmark model execution on hardware using delegates. This tool is available along with other artifacts installed on the device.

This benchmarking tool measures and calculates statistics for the following performance metrics:

- Initialization time
- Inference time of the Warm-up state
- Inference time of the Steady state
- Memory usage during initialization
- Overall memory usage

Before you begin, ensure that the downloaded models are in the `/etc/artifacts/` directory on the target device.

To perform benchmarking, do the following:

- To benchmark models using the XNNPACK delegate, run the following commands:

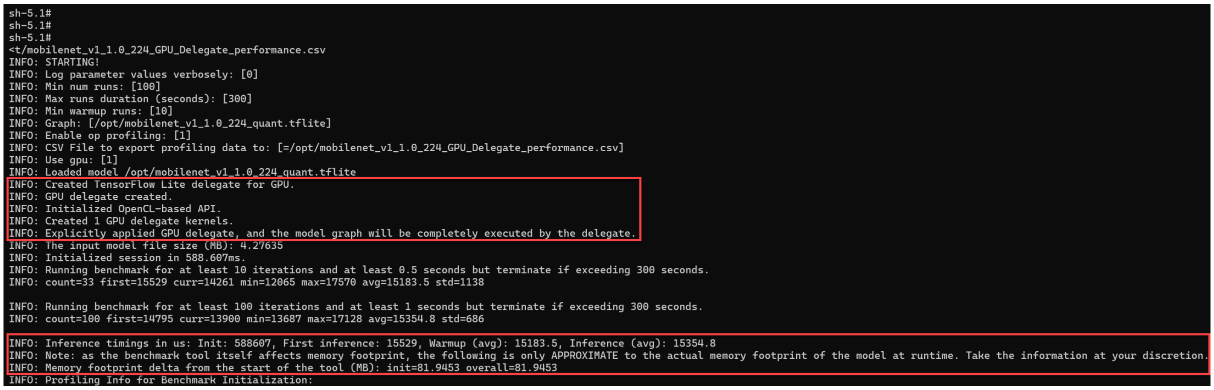
```
ssh root@[ip-addr]
cd /etc/artifacts
```

```
benchmark_model --graph=/etc/artifacts/mobilenet_v1_1.0_224_
quant.tflite --enable_op_profiling=true --use_xnnpack=true --
num_threads=4 --max_secs=300 --profiling_output_csv_file=/etc/
artifacts/mobilenet_v1_1.0_224_quant_xnnpack_performance.csv
```

- To benchmark models using the GPU delegate, run the following commands:

```
ssh root@[ip-addr]
cd /etc/artifacts/
```

```
benchmark_model --graph=/etc/artifacts/mobilenet_v1_1.0_224_
quant.tflite --enable_op_profiling=true --use_gpu=true --num_
runs=100 --warmup_runs=10 --max_secs=300 --profiling_output_csv_
file==/etc/artifacts/mobilenet_v1_1.0_224_GPU_Delegate_
performance.csv
```



```
sh-5.1#
sh-5.1#
sh-5.1#
<t/mobilenet_v1_1.0_224_GPU_Delegate_performance.csv
INFO: STARTING!
INFO: Log parameter values verbosely: [0]
INFO: Min num runs: [100]
INFO: Max runs duration (seconds): [300]
INFO: Min warmup runs: [10]
INFO: Graph: [/opt/mobilenet_v1_1.0_224_quant.tflite]
INFO: Enable op profiling: [1]
INFO: CSV File to export profiling data to: [=/opt/mobilenet_v1_1.0_224_GPU_Delegate_performance.csv]
INFO: Use gpu: [1]
INFO: Loaded model /opt/mobilenet_v1_1.0_224_quant.tflite
INFO: Created TensorFlow Lite delegate for GPU.
INFO: GPU delegate created.
INFO: Initialized OpenCL-based API.
INFO: Created 1 GPU delegate kernels.
INFO: Explicitly applied GPU delegate, and the model graph will be completely executed by the delegate.
INFO: The input model file size (MB): 4.27635
INFO: Initialized session in 588.607ms.
INFO: Running benchmark for at least 10 iterations and at least 0.5 seconds but terminate if exceeding 300 seconds.
INFO: count=33 first=15529 curr=14261 min=12065 max=17570 avg=15183.5 std=1138

INFO: Running benchmark for at least 100 iterations and at least 1 seconds but terminate if exceeding 300 seconds.
INFO: count=100 first=14795 curr=13900 min=13687 max=17128 avg=15354.8 std=686

INFO: Inference timings in us: Init: 588607, First inference: 15529, Warmup (avg): 15183.5, Inference (avg): 15354.8
INFO: Note: as the benchmark tool itself affects memory footprint, the following is only APPROXIMATE to the actual memory footprint of the model at runtime. Take the information at your discretion.
INFO: Memory footprint delta from the start of the tool (MB): init=81.9453 overall=81.9453
INFO: Profiling Info for Benchmark Initialization:
```

Figure: Benchmark model tool statistics for GPU

5.3 Run a QNN delegate using an external delegate

The QNN delegate relies on the Qualcomm AI Engine direct API and its back ends to speed up models on the Adreno GPU and the Hexagon Tensor Processor.

To run the QNN delegate using the external delegate interface, ensure that the following libraries are available on the device:

- `libQnnTFLiteDelegate.so` QNN delegate library
- Libraries from the Qualcomm AI Engine direct SDK

As part of the external delegate interface, `libQnnTFLiteDelegate.so` is available as an external delegate library to tools. After loading the delegate library, you can customize the model execution to use a specific back end through external delegate options.

For example:

- Use the `libQnnGpu.so` back-end library to run the QNN delegate on the GPU.
- Use the `libQnnHtp.so` back-end library to run models using the QNN delegate on the Hexagon Tensor Processor.

To benchmark a model on the Hexagon Tensor Processor, run the model through the QNN external delegate interface.

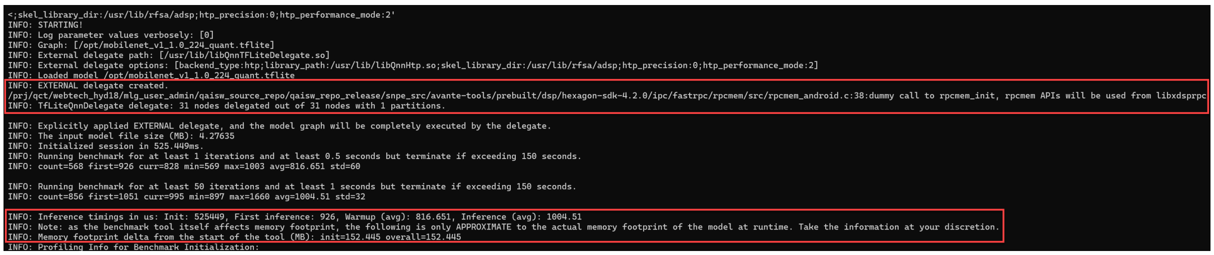
Use the following command to run inference:

```
benchmark_model --graph=/etc/artifacts/mobilenet_v1_1.0_224_quant.
tflite --external_delegate_path=/usr/lib/libQnnTFLiteDelegate.so --
external_delegate_options='backend_type:htp;library_path:/usr/lib/
libQnnHtp.so;skel_library_dir:/usr/lib/rfsa/ adsp;htp_precision:0;
htp_performance_mode:2'
```

For more details, see External delegate options for QNN delegate.

The figure highlights the following statistics presented by the benchmark_model tool.

- Successful creation of the delegate or not
- Average inference time that the model took to run on the hardware using a delegate
- Memory footprint of the model execution



```
<;skel_library_dir:/usr/lib/rfsa/adsp;htp_precision:0;htp_performance_mode:2'
INFO: STARTING!
INFO: Log parameter values verbosely: [0]
INFO: Graph: [/opt/mobilenet_v1_1.0_224_quant.tflite]
INFO: External delegate path: [/usr/lib/libQnnTFLiteDelegate.so]
INFO: External delegate options: [backend_type:htp;library_path:/usr/lib/libQnnHtp.so;skel_library_dir:/usr/lib/rfsa/adsp;htp_precision:0;htp_performance_mode:2]
INFO: Loaded model /opt/mobilenet_v1_1.0_224_quant.tflite
INFO: EXTERNAL delegate created.
/prj/qct/webtech_hyd18/mlg_user_admin/qaisw_source_repo/qaisw_repo_release/snpe_src/avante-tools/prebuilt/dsp/hexagon-sdk-4.2.0/ipc/fastrpc/rpcmem/src/rpcmem_android.c:38:dummy call to rpcmem_init, rpcmem APIs will be used from libxdsprpc
INFO: TfLiteQnnDelegate delegate: 31 nodes delegated out of 31 nodes with 1 partitions.

INFO: Explicitly applied EXTERNAL delegate, and the model graph will be completely executed by the delegate.
INFO: The input model file size (MB): 4.27635
INFO: Initialized session in 525.449ms.
INFO: Running benchmark for at least 1 iterations and at least 0.5 seconds but terminate if exceeding 150 seconds.
INFO: count=568 first=926 curr=828 min=569 max=1003 avg=816.651 std=60

INFO: Running benchmark for at least 50 iterations and at least 1 seconds but terminate if exceeding 150 seconds.
INFO: count=856 first=1051 curr=995 min=897 max=1660 avg=1004.51 std=32

INFO: Inference timings in us: Init: 525449, First inference: 926, Warmup (avg): 816.651, Inference (avg): 1004.51
INFO: Note: as the benchmark tool itself affects memory footprint, the following is only APPROXIMATE to the actual memory footprint of the model at runtime. Take the information at your discretion.
INFO: Memory footprint delta from the start of the tool (MB): init=152.445 overall=152.445
INFO: Profiling Info for Benchmark Initialization:
```

Figure: Tool statistics: benchmark_model

Known issue

The LiteRT native sample application (label_image) might crash during inferencing on the GPU or external delegate.

External delegate options for QNN delegate

The external delegate interface dynamically loads the Qualcomm AI Engine direct delegate. Therefore, it doesn't have static information about the delegate options.

The external delegate sends strings as key and value pairs to the Qualcomm AI Engine direct delegate, which parses them as options. Therefore, the application using the external delegate interface must determine the accepted key and value option strings beforehand.

The following table lists the key value option strings that are available in the Qualcomm AI Engine direct delegate.

Table: Key value option strings

Option key	Option value	Default value	Mandatory	Description
backend_type	GPU and Hexagon Tensor Processor	NA	Yes	The back-end Qualcomm AI Engine direct library used for opening and running the graph.
gpu_precision	0, 1, 2, 3	2 = Float16 for best performance	No	Precision for the GPU back end that defines the optimization levels of the graph tensors that are either input or output tensors. • 0: Obey precisions specified in the LiteRT graph • 1: Float32 • 2: Float16 • 3: Hybrid

Option key	Option value	Default value	Mandatory	Description
gpu_ performance_ mode	0, 1, 2, 3	0 = default	No	Flag to provide precision modes supported by the GPU back end. • 0: Default • 1: High • 2: Normal • 3: Low
htp_performance_ mode	0, 1, 2, 3, 4, 5, 6, 7, 8	0 = default	No	In performance_mode, the delegate votes for the provided performance level during inference and returns to a relaxed vote after the inference has completed. • 0: Default • 1: Sustained high performance • 2: Burst • 3: High performance • 4: Power saver • 5: Low-power saver • 6: High-power saver • 7: Low balance • 8: Balance
htp_pd_session	Unsigned, signed	Unsigned	No	This flag defines the PD session of the Hexagon Tensor Processor back end.
htp_precision	0	0 = quantized precision	No	Flag to provide precision modes supported by the Hexagon Tensor Processor back end. The default precision mode supports quantized networks. Only certain SoCs may support other precision modes. • 0: Quantized precision
htp_optimization_ strategy	0, 1	0 = optimize for inference	No	Flag to select the optimization strategy used by the Hexagon Tensor Processor back end. The default optimization strategy optimizes the graph for inference. • 0: Optimize for inference • 1: Optimize to prepare the model
htp_perf_ctrl_ strategy	0, 1	0 = manual	No	Flag to select the Hexagon Tensor Processor performance control strategy. Manually vote the performance while initializing the back end and release the performance the moment the back end is destroyed. • 0: Manual • 1: Auto
htp_use_conv_ hmx	0, 1	1 = enable Qualcomm® Hexagon™ Matrix eXtensions (HMX) for short depth conv2d	No	Flag to enable HMX for short depth conv2d. For more information, see the C interface or qnn_delegate.h. • 0: Don’t enable HMX for short-depth conv2d. • 1: Enable HMX for short-depth conv2d.

Option key	Option value	Default value	Mandatory	Description
htp_use_fold_relu	0, 1	0 = Don't fuse rectified linear unit (ReLU) into conv2d	No	Flag to enable integration of ReLU into conv2d. For more information, see the C interface or qnn_delegate.h. • 0: Don't enable integration of ReLU into conv2d. • 1: Enable integration of ReLU into conv2d.
library_path	<Path to the back-end library file>	Default library associated with the chosen Qualcomm AI Engine direct back end	No	Optional parameter to override the Qualcomm AI Engine direct back-end library.

6 Optional: Build LiteRT

Note: LiteRT and its libraries are built as part of the Qualcomm Linux build along with the Qualcomm Intelligent Multimedia Product (QIMP) SDK. Therefore, building Qualcomm Linux is optional. You can build Qualcomm Linux in certain scenarios, such as when you want to change the LiteRT library version.

To recompile LiteRT as part of the Qualcomm Linux build along with the QIMP SDK, see Qualcomm Linux Build Guide.

7 References

7.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
Qualcomm Linux Build Guide	80-70018-254
Qualcomm AI Engine direct	80-63442-50
RB3 Gen 2 Quick Start Guide	80-70018-253
Qualcomm Intelligent Multimedia Software Development Kit (IM SDK) Reference	80-70018-50
AI/ML Developer Workflow	80-70018-15B
Qualcomm IQ-9 Beta Evaluation Kit Quick Start Guide	80-70015-263
Qualcomm IQ-8 Beta Evaluation Kit Quick Start Guide	80-70017-263
Resources	
https://tfhub.dev/	
https://ai.google.dev/edge/litert	
https://ai.google.dev/edge/litert/models/convert	
https://ai.google.dev/edge/litert/performance/gpu	
https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/examples/label_image	
https://ai.google.dev/edge/litert/performance/implementing_delegate#option_2_leverage_external_delegate	

7.2 Acronyms and terms

Acronym or term	Definition
CLI	Command-line interface
IM	Intelligent Multimedia
QNN	Qualcomm Neural Network
QIMP	Qualcomm Intelligent Multimedia Product
ReLU	Rectified linear unit
SCP	Secure copy protocol
SDK	Software development kit
SSH	Secure shell
XNN	Xth nearest neighbor

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.