UG10338

i.MX Machine Learning User Guide for Android

Rev. 1.4 — 3 November 2026 **User guide**

Document information

Information	Content
Keywords	Android, i.MX, Machine Learning
Abstract	This document describes how to deploy part of the eIQ Core into the NXP Android BSP and how to develop Machine Learning (ML) applications based on NXP eIQ. It particularly describes the deployment of the TensorFlow Lite (TFLite) inference engine and related delegates for on-chip ML accelerators (NPU and GPU) available on the i.MX 8M Plus and i.MX 95/952/937 applications processors.

1 Introduction

This document describes how to deploy part of the eIQ Core into the NXP Android BSP and how to develop Machine Learning (ML) applications based on NXP eIQ. It particularly describes the deployment of the TensorFlow Lite (TFLite) inference engine and related delegates for on-chip ML accelerators (NPU and GPU) available on the i.MX 8M Plus and i.MX 95/952/937 applications processors.

It aims to provide ways to use the ML hardware accelerators and this achieves comparable ML performance as that available on the NXP Yocto Linux platform. The following hardware accelerators are addressed in this document:

- NPU using the VX Delegate on the i.MX 8M Plus platform
- eIQ Neutron NPU using the Neutron Delegate on the i.MX 95/952/937 platforms

To demonstrate the way to use NPU accelerators on NXP devices, we have pre-installed three applications for classification using MobileNet V1, model benchmarking, and an end-to-end classification demo. This document first describes the usages of these applications in prebuilt images. NXP TFLite libraries are also provided for the Android platform. Users can develop their own ML applications based on NXP TFLite. This document then describes how to use the NXP TFLite libraries to develop ML applications, and provides the steps to build eIQ Core packages from the source code if users need to add customized features.

The requirements to follow this document are as follows:

- i.MX 8M Plus or i.MX 95/952/937 Evaluation Kit
- Build Host machine with the Linux OS and +450 GB free disk space, +16 GB RAM
- Debug/Deployment Host machine to push/pull data to/from the i.MX target: Linux or Windows machine

Note: *This document describes the deployment on the Android 17.0.0_1.0.0 BSP and eIQ Core LF6.18.37_2.1.0 releases.*

2 NNAPI Delegate and dedicated delegate

The standard way to access the Machine Learning (ML) accelerators on Android is through the Neural Networks API (NNAPI), such as by using the NNAPI Delegate in the TensorFlow Lite inference engine.

The NNAPI is a software layer defined by the Android OS that facilitates the execution of compute-intensive operations by coordinating the available hardware, such as CPU, GPU, and other ML hardware accelerators. It is a part of the Android public API since level 27 and adds new features with every Android release. The NNAPI can be used by different inference engines, for example, through the NNAPI Delegate in TensorFlow Lite or NNAPI provider in ONNX Runtime.

As the NNAPI is defined by the Android OS, discrepancies between NNAPI specifications, the supported capabilities of the acceleration hardware (provided by device manufacturers), and the inference engines (developed by third-party organizations) are common. For example, on i.MX platforms, there is strong alignment between TensorFlow Lite operator definitions and the operators supported by the NPU. On the other hand, NNAPI needs to support multiple hardware backends (CPU, GPU, custom accelerators) from multiple manufacturers, as well as support multiple inference engines. This broad scope necessitates a best-effort approach in NNAPI's operator definitions to serve all stakeholders. As a result, the NNAPI Delegate may assign operators that could otherwise run on the NPU to the CPU, which can negatively impact overall inference performance.

Compared to NNAPI, dedicated delegates for particular hardware accelerators, such as VX Delegate for i.MX 8M Plus and Neutron Delegate for i.MX 95/952/937 platforms, are fully aligned with NPU capabilities. The NPU delegates (VX Delegate and Neutron Delegate) are key components of the eIQ software enablement stack to apply the hardware acceleration on the ML workload. They are already supported in the NXP Embedded Linux

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

for i.MX Applications Processors, with proven acceleration of embedded ML models. This document describes how to use the delegates in TensorFlow Android applications.

3 Pre-installed ML applications

There are three pre-installed Machine Learning (ML) applications that are built on the NXP eIQ core: label image, `benchmark_model`, and `TfliteCameraDemo`. Users need to flash the pre-built NXP Android images and boot up the Android OS, and then follow the steps below to run these applications.

3.1 Preparation

3.1.1 Installing the ADB

Android Debug Bridge (ADB), which is included in the `platform-tools` package, is used to interact with the i.MX application processors running the Android OS. `platform-tools` must be present on the host connected to the target platform.

For more information, see SDK Platform Tools release notes.

3.1.2 Downloading the MobileNet v1 model and test files

Download and extract the float point and quantized Mobilenet V1 models below:

- mobilenet_v1_1.0_224_quant.tgz
- mobilenet_v1_1.0_224.tgz

Download the label for Mobilenet V1 from https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt.

Download the sample image from https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/examples/label_image/testdata/grace_hopper.bmp.

3.1.3 Installing eIQ Neutron SDK

The eIQ Neutron NPU is integrated into the i.MX 95/952/937 SoCs. To run the models on the eIQ Neutron, convert the models into Neutron Graph by using `neutron-converter`, which is part of eIQ Neutron SDK.

The eIQ Neutron SDK 3.2.1 is used for Android 17.0.0_1.0.0. The download link and user guide can be found on eIQ® Toolkit for End-to-End Model Development and Deployment. See Section "Neutron converter" in the user guide for the usage and supported models.

3.2 Label Image

Label Image is a classic TensorFlow Lite (TFLite) application that shows how to perform image classification using a pre-trained TFLite model. TensorFlow Lite provides C++ and Python implementations of the Label Image. In addition, the NXP eIQ Core provides Java implementation for Android OS.

- Push the model and test data to the Android OS:

```
$ adb push mobilenet_v1_1.0_224_quant.tflite grace_hopper.bmp /data/data/
org.tensorflow.lite.label_image/
$ adb push ImageNetLabels.txt /data/data/org.tensorflow.lite.label_image/labels.txt
```

- Run the Label Image on the CPU:

```
$ adb shell logcat -c
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
$ adb shell am start -S -n org.tensorflow.lite.label_image/.LabelImageActivity \
    --es graph "/data/data/org.tensorflow.lite.label_image/
mobilenet_v1_1.0_224_quant.tflite " \
    --es label "/data/data/org.tensorflow.lite.label_image/labels.txt" \
    --es image "/data/data/org.tensorflow.lite.label_image/grace_hopper.bmp"
$ adb shell logcat | grep "LabelImage"
```

The following output is expected.

```
09-12 05:26:42.064  3643  3643 I LabelImageActivity: Label image application starts.
09-12 05:26:42.064  3643  3643 I LabelImageActivity: Running TensorFlow Lite classification inference
09-12 05:26:42.064  3643  3643 D LabelImage: Init label image object
09-12 05:26:42.065  3643  3643 D LabelImage: Local model /data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
09-12 05:26:42.065  3643  3643 D LabelImage: Create interpreter
09-12 05:26:42.096  3643  3643 D LabelImage: Load label file /data/local/tmp/labels.txt
09-12 05:26:42.097  3643  3643 D LabelImage: Allocate input and output memory
09-12 05:26:42.098  3643  3643 D LabelImage: Inference image /data/local/tmp/grace_hopper.bmp
09-12 05:26:42.098  3643  3643 D LabelImage: Preprocess image
09-12 05:26:42.169  3643  3643 D LabelImage: Inference starts starts
09-12 05:26:42.329  3643  3643 D LabelImage: Get output tensor
09-12 05:26:42.329  3643  3643 D LabelImage: There are 1001 numbers.
09-12 05:26:42.331  3643  3643 I LabelImage: Top-1: military uniform, score 0.80
09-12 05:26:42.331  3643  3643 I LabelImage: Top-2: Windsor tie, score 0.09
09-12 05:26:42.331  3643  3643 I LabelImage: Top-3: bow tie, score 0.02
09-12 05:26:42.331  3643  3643 I LabelImage: Top-4: bulletproof vest, score 0.01
09-12 05:26:42.331  3643  3643 I LabelImage: Top-5: mortarboard, score 0.01
```

Figure 1. Expected output

- Run the Label Image on the NPU:
  The eIQ TFLite runs the model on the NPU through an external delegate. To enable operator execution on the NPU, specify the delegate library path to the `ext_delegate` argument.
  - For i.MX 8M Plus:

```
$ adb shell am start -S -n org.tensorflow.lite.label_image/.LabelImageActivity \
    --es graph "/data/data/org.tensorflow.lite.label_image/
mobilenet_v1_1.0_224_quant.tflite " \
    --es label "/data/data/org.tensorflow.lite.label_image/labels.txt" \
    --es image "/data/data/org.tensorflow.lite.label_image/grace_hopper.bmp" \
    --es ext_delegate "/vendor/lib64/libvx_delegate.so"
```

  - For i.MX 95/952/937:
    Convert the TFLite model to Neutron Graph, and push the converted model to `/data/data/org.tensorflow.lite.label_image` on the Android OS.
    Since `mobilenet_v1_1.0_224_quant.tflite` has asymmetric uint8 weights and Neutron only supports symmetric int8 weights, add the `--convert-inputs-uint8-to-int8 --convert-outputs-uint8-to-int8` parameter to `neutron-converter`. If your model has symmetric int8 weight only, you do not need to add such a parameter.

```
$ ./neutron-converter --target imx95 \
--input mobilenet_v1_1.0_224_quant.tflite \
--output mobilenet_v1_1.0_224_quant_neutron.tflite \
--convert-inputs-uint8-to-int8 \
--convert-outputs-uint8-to-int8
$ adb push mobilenet_v1_1.0_224_quant_neutron.tflite /data/data/
org.tensorflow.lite.label_image
$ adb shell logcat -c
$ adb shell am start -S -n org.tensorflow.lite.label_image/.LabelImageActivity \
    --es graph "/data/data/org.tensorflow.lite.label_image/
mobilenet_v1_1.0_224_quant_neutron.tflite " \
    --es label "/data/data/org.tensorflow.lite.label_image/labels.txt" \
    --es image "/data/data/org.tensorflow.lite.label_image/grace_hopper.bmp" \
    --es ext_delegate "/vendor/lib64/libneutron_delegate.so"
$ adb shell logcat | grep "LabelImage"
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

3.3 Benchmark Model

The `benchmark_model` evaluates the performance of a selected model and supports the use of different delegates. It is useful to compare the performance difference among the delegates.

The Android application for Benchmark Model (`benchmark_model.apk`), which implements an Android activity in Java, uses the JNI to run the inference in the native code. The native code contains the logic to include external delegates (such as the VX or Neutron Delegate).

Although you can run `benchmark_model` with multiple threads, there is a limitation on the Android OS that the application process is running on a single core. Therefore, it is not beneficial to run the `benchmark_model` with multiple threads. To measure a better performance with multiple threads, use C++ implementation of `benchmark_model`. See Section 5.3 to compile the `benchmark_model` binary from the C++ source code.

- Run `benchmark_model` on the CPU:

```
$ adb shell logcat -c
$ adb shell am start -S -n
 org.tensorflow.lite.benchmark/.BenchmarkModelActivity \
    --es args '" --graph=/data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite \
    --num_threads=1 "'
$ adb shell logcat | grep tflite
```

```
09-11 18:39:09.734  2859  2859 I tflite_BenchmarkModelActivity: Running TensorFlow Lite benchmark with args:  --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite --num_threads=1
09-11 18:39:09.756  2859  2859 I tflite  : Log parameter values verbosely: [0]
09-11 18:39:09.757  2859  2859 I tflite  : Num threads: [1]
09-11 18:39:09.757  2859  2859 I tflite  : Graph: [/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite]
09-11 18:39:09.757  2859  2859 I tflite  : Signature to run: []
09-11 18:39:09.758  2859  2859 I tflite  : #threads used for CPU inference: [1]
09-11 18:39:09.761  2859  2859 I tflite  : Loaded model /data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
09-11 18:39:09.762  2859  2859 I tflite  : Initialized TensorFlow Lite runtime.
09-11 18:39:09.766  2859  2859 I tflite  : Created TensorFlow Lite XNNPACK delegate for CPU.
09-11 18:39:09.768  2859  2859 I tflite  : The input model file size (MB): 4.27635
09-11 18:39:09.768  2859  2859 I tflite  : Initialized session in 7.422ms.
09-11 18:39:09.782  2859  2859 I tflite  : Running benchmark for at least 1 iterations and at least 0.5 seconds but terminate if exceeding 150 seconds.
09-11 18:39:10.395  2859  2859 I tflite  : count=4 first=159672 curr=150514 min=150514 max=159672 avg=153165 std=3779 p5=150514 median=151648 p95=159672
09-11 18:39:10.395  2859  2859 I tflite  : Running benchmark for at least 50 iterations and at least 1 seconds but terminate if exceeding 150 seconds.
09-11 18:39:17.936  2859  2859 I tflite  : count=50 first=150900 curr=150206 min=150034 max=154104 avg=150747 std=727 p5=150210 median=150596 p95=151914
09-11 18:39:17.936  2859  2859 I tflite  : Inference timings in us: Init: 7422, First inference: 159672, Warmup (avg): 153165, Inference (avg): 150747
09-11 18:39:17.936  2859  2859 I tflite  : 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.
09-11 18:39:17.937  2859  2859 I tflite  : Memory footprint delta from the start of the tool (MB): init=4.74609 overall=14.3008
```

Figure 2. Running `benchmark_model` on the CPU

- Run `benchmark_model` on the NPU:
  - For i.MX 8M Plus:

```
delegate=libvx_delegate.so
model=mobilenet_v1_1.0_224_quant.tflite
```

  - For i.MX 95/952/937:

```
delegate=libneutron_delegate.so
model=mobilenet_v1_1.0_224_quant_neutron.tflite
```

  - For i.MX 8M Plus and i.MX 95/952/937:

```
$ adb shell am start -S -n
 org.tensorflow.lite.benchmark/.BenchmarkModelActivity --es args '" \
    --graph=/data/data/org.tensorflow.lite.benchmark/${model} \
    --external_delegate_path=/vendor/lib64/${delegate} "'
```

- Run `benchmark_model` on the GPU:
  - For i.MX 8M Plus:

```
$ adb shell setprop vendor.USE_GPU_INFERENCE 1
$ adb shell am start -S -n
 org.tensorflow.lite.benchmark/.BenchmarkModelActivity --es args '" \
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
    --graph=/data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite \
    --external_delegate_path=/vendor/lib64/libvx_delegate.so "'
```

– For i.MX 95/952/937, you can choose to run with either the OpenGL library or OpenCL library through the `gpu_backend` parameter:

```
$ adb shell am start -S -n
 org.tensorflow.lite.benchmark/.BenchmarkModelActivity --es args '" \
    --graph=/data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite \
    --use_gpu=true --gpu_backend=cl"'
Execution log:
tflite_BenchmarkModelActivity: Running TensorFlow Lite benchmark
 with args:  --graph=/data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite --use_gpu=true --gpu_backend=cl
tflite  : Log parameter values verbosely: [0]
tflite  : Graph: [/data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite]
tflite  : Signature to run: []
tflite  : Use gpu: [1]
tflite  : GPU backend: [cl]
tflite  : Loaded model /data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite
tflite  : Initialized TensorFlow Lite runtime.
tflite  : Created TensorFlow Lite delegate for GPU.
tflite  : GPU delegate created.
tflite  : Loaded OpenCL library with dlopen.
tflite  : Replacing 31 out of 31 node(s) with delegate (TfLiteGpuDelegateV2)
 node, yielding 1 partitions for subgraph 0.
tflite  : Initialized OpenCL-based API.
tflite  : Created 1 GPU delegate kernels.
tflite  : Explicitly applied GPU delegate, and the model graph will be
 completely executed by the delegate.
tflite  : The input model file size (MB): 4.27635
tflite  : Initialized session in 1539.39ms.
tflite  : Running benchmark for at least 1 iterations and at least 0.5
 seconds but terminate if exceeding 150 seconds.
tflite  : count=11 first=47749 curr=45925 min=45872 max=47749 avg=46098.3
 std=525 p5=45872 median=45907 p95=47749
tflite  : Running benchmark for at least 50 iterations and at least 1 seconds
 but terminate if exceeding 150 seconds.
tflite  : count=50 first=46064 curr=24565 min=24520 max=46064 avg=25220.2
 std=3236 p5=24535 median=24592 p95=25557
tflite  : Inference timings in us: Init: 1539387, First inference: 47749,
 Warmup (avg): 46098.3, Inference (avg): 25220.2
tflite  : 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.
tflite  : Memory footprint delta from the start of the tool (MB):
 init=109.492 overall=109.492

$ adb shell am start -S -n
 org.tensorflow.lite.benchmark/.BenchmarkModelActivity --es args '" \
    --graph=/data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite \
    --use_gpu=true --gpu_backend=gl"'
Execution log:
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
tflite_BenchmarkModelActivity: Running TensorFlow Lite benchmark
 with args:  --graph=/data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite --use_gpu=true --gpu_backend=gl
tflite  : Log parameter values verbosely: [0]
tflite  : Graph: [/data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite]
tflite  : Signature to run: []
tflite  : Use gpu: [1]
tflite  : GPU backend: [gl]
tflite  : Loaded model /data/data/org.tensorflow.lite.benchmark/
mobilenet_v1_1.0_224_quant.tflite
tflite  : Initialized TensorFlow Lite runtime.
tflite  : Created TensorFlow Lite delegate for GPU.
tflite  : GPU delegate created.
tflite  : Loaded OpenCL library with dlopen.
tflite  : Replacing 31 out of 31 node(s) with delegate (TfLiteGpuDelegateV2)
 node, yielding 1 partitions for subgraph 0.
tflite  : Initialized OpenGL-based API.
tflite  : Created 1 GPU delegate kernels.
tflite  : Explicitly applied GPU delegate, and the model graph will be
 completely executed by the delegate.
tflite  : The input model file size (MB): 4.27635
tflite  : Initialized session in 1728.02ms.
tflite  : Running benchmark for at least 1 iterations and at least 0.5
 seconds but terminate if exceeding 150 seconds.
tflite  : count=4 first=148819 curr=73520 min=73520 max=148819 avg=125950
 std=30684 p5=73520 median=146084 p95=148819
tflite  : Running benchmark for at least 50 iterations and at least 1 seconds
 but terminate if exceeding 150 seconds.
tflite  : count=50 first=74249 curr=75852 min=73382 max=76567 avg=74403.9
 std=979 p5=73535 median=73917 p95=76239
tflite  : Inference timings in us: Init: 1728018, First inference: 148819,
 Warmup (avg): 125950, Inference (avg): 74403.9
tflite  : 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.
tflite  : Memory footprint delta from the start of the tool (MB):
 init=129.609 overall=129.609
```

3.4 TFLite Camera Demo

TFLite Camera Demo is a sample Android GUI application that performs real-time image classification using TensorFlow Lite and the device's camera. It supports multiple inference accelerators on the NXP devices. For i.MX 8M Plus, it supports CPU, NNAPI, and NPU. For i.MX 95/952/937, it supports CPU, GPU, and NPU.

To run the application, click **TFLite Camera Demo** on the application panel. If the application is running in landscape mode, run `adb shell settings put system user_rotation 0` to change it to portrait mode.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

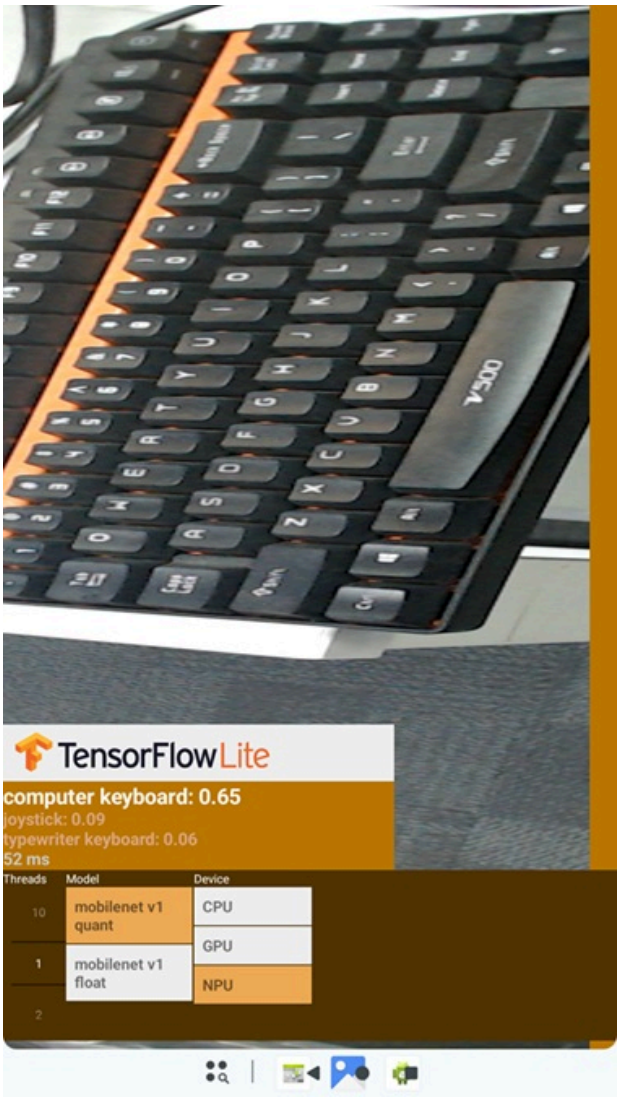


Figure 3. TFLite Camera Demo

4 Building Android applications based on the NXP eIQ Core

The NXP eIQ Core provides TensorFlow Lite (TFLite) run-time libraries for the Android OS in the AAR format. Users can build their own TFLite applications for the Android OS based on these libraries.

This section takes the TFLite Camera Demo as an example to illustrate how to develop Android applications with the eIQ Core TFLite libraries for the Android OS.

4.1 eIQ TFLite libraries for the Android OS

There are 5 TFLite libraries for the Android OS under `vendor/nxp/neutron-software-stack/Android/TfLiteLib` of the NXP Android BSP release package:

- `tensorflow-lite-api.aar`: contains the Java API layer of TFLite Runtime.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- `tensorflow-lite.aar`: contains the implementation of TFLite Runtime, including Interpreter class and other essential APIs for running inference. It also contains the native library (`libtensorflowlite_jni.so`) and header files.
- `tensorflow-lite-gpu-api.aar`: contains the Java API layer for the GPU delegate.
- `tensorflow-lite-gpu.aar`: contains Java APIs for using a GPU delegate (GpuDelegate class) and the native GPU delegate library (`libtensorflowlite_gpu_jni.so`).
- `tensorflow-lite-external-delegate.aar`: contains Java APIs for using an external delegate (ExternalDelegate class) and native external delegate library (`libtensorflowlite_external_delegate_jni.so`).

4.2 Creating a TFLite interpreter with the NPU delegate

The Interpreter class in TensorFlow Lite (TFLite) is the core API used to run the inference with a TFLite model on the Android OS. It provides a simple interface to load a `.tflite` model, allocate tensors, and run predictions with specific hardware accelerators through the interfaces known as 'delegate'. The TFLite includes a few built-in delegates, such as the following:

- XNNPack delegate, which is a highly optimized library for the NN inference on the CPU.
- GPUDelegate, which enables the NN inference on the GPU instead of the CPU.

In addition to built-in delegates, TFLite also supports third-party delegates and calls these delegates through the ExternalDelegate class. The following code snippet shows how to create an external delegate instance for the NPU on NXP devices and create an interpreter instance to invoke the inference.

```
import org.tensorflow.lite.Interpreter;
import org.tensorflow.lite.DataType;
import org.tensorflow.lite.external.ExternalDelegate;

Interpreter.Options options = new Interpreter.Options();
options.setNumThreads(numThreads);
// Enable XNNPack so that unsupported operators of NPU can be offloaded to
 XNNPack delegate
options.setUseXNNPACK(true);
// delegate is the path to NPU delegate, it is /vendor/lib64/libvx_delegate.so
 on i.MX 8M Plus and /vendor/lib64/libneutron_delegate.so on i.MX 95/952/937
ExternalDelegate.Options extDelegateOptions = new
 ExternalDelegate.Options(delegate);
ExternalDelegate extDelegate = new ExternalDelegate(extDelegateOptions);
options.addDelegate(extDelegate);
// The model is same as what we used in chapter 3
Interpreter interpreter = new Interpreter(model_path, options);
interpreter.allocateTensors();
// Create inputData and outputData before run inference
interpreter.run(inputData, outputData);
```

4.3 Adding eIQ TFLite libraries in the build file

4.3.1 Gradle build

Perform the following steps:

1. Copy the libraries to the application directory:

```
$ mkdir <app_dir>/libs
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
$ cp <imx_android>/vendor/nxp/neutron-software-stack/Android/TfLiteLib/*
 <app_dir>/libs
```

2. Edit `build.gradle` for the application. Include libraries in the application.

```
repositories {
    …
    flatDir {
        dirs 'libs'
    }
}
dependencies {
    …
    // Use NXP eIQ TensorFlowLite library
    implementation(name: 'tensorflow-lite', ext: 'aar')
    implementation(name: 'tensorflow-lite-api', ext: 'aar')
    implementation(name: 'tensorflow-lite-gpu', ext: 'aar')
    implementation(name: 'tensorflow-lite-gpu-api', ext: 'aar')
    implementation(name: 'tensorflow-lite-external-delegate', ext: 'aar')
}
```

4.3.2 Bazel build

Perform the following steps:

1. Copy the libraries to the application directory:

```
$ mkdir <app_dir>/libs
$ cp <imx_android>/vendor/nxp/neutron-software-stack/Android/TfLiteLib/*
 <app_dir>/libs
```

2. Edit `BUILD` for the application to add the following dependency:

```
aar_import(
    name = "tensorflow_lite_api",
    aar = "libs/tensorflow-lite-api.aar"
)

aar_import(
    name = "tensorflow_lite",
    aar = "libs/tensorflow-lite.aar",
    deps = [":tensorflow_lite_api"],
)
aar_import(
    name = "tensorflow_lite_gpu",
    aar = "libs/tensorflow-lite-gpu.aar",
    deps = [":tensorflow_lite_api"],
)

aar_import(
    name = "tensorflow_lite_gpu_api",
    aar = "libs/tensorflow-lite-gpu-api.aar"
)

aar_import(
    name = "tensorflow_lite_external_delegate",
    aar = "libs/tensorflow-lite-external-delegate.aar",
    deps = [":tensorflow_lite_api"],
)
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
android_binary(
    name = "APPName",
    srcs = glob(["src/**/*.java"]),
    custom_package = "org.tensorflow.lite. APPName ",
    manifest = "AndroidManifest.xml",
    tags = ["manual"],
    deps = [
      ":tensorflow_lite",
      ":tensorflow_lite_api",
      ":tensorflow_lite_gpu",
      ":tensorflow_lite_gpu_api",
      ":tensorflow_lite_external_delegate"
    ]
)
```

4.4 Building the TFLite Camera Demo based on the eIQ core

The TFLite libraries for the Android OS are included in the Android BSP package. Download the Android 17.0.0_1.0.0 release package and unzip it in a directory.

```
$ tar -xzvf imx-android-17.0.0_1.0.0.tar.gz -C android-16.0.0
```

Set up the i.MX Android root path:

```
$ source <Path_to_unziped_package>/imx_android_setup.sh
$ export MY_ANDROID=`pwd`
```

After sourcing the setup script, the working directory points to the root where all the required sources have been extracted (will be referred to as `MY_ANDROID` from here on). This working directory contains the following relevant resources:

- `${MY_ANDROID}/vendor/nxp/fsl-proprietary/gpu-viv`: contains GPU/NPU drivers.
- `${MY_ANDROID}/vendor/nxp/fsl-proprietary/include`: contains GPU/NPU driver headers.
- `${MY_ANDROID}/vendor/nxp/neutron-software-stack/Android/TfLiteLib`: contains Android libraries in the AAR format.
- `${MY_ANDROID}/device/nxp/`: contains Android build and configuration files for NXP platforms, primarily Kati `.mk` files, and configurations for specific devices, including Security-Enhanced Linux (SELinux) policy descriptions.

5 Building and updating the pre-installed application from sources

This section describes how to build and update the pre-installed TFLite applications for the Android OS using NXP Tensorflow and Android sources.

5.1 Preparation

1. Prepare the Ubuntu 22.04 host with desktop and install JDK 1.8.0.

   ```
   $ sudo apt install openjdk-8-jdk
   ```

   Make sure that the Java in `PATH` points to Java 1.8.0.

   ```
   $ java -version
   openjdk version "1.8.0_462"
   ```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
OpenJDK Runtime Environment (build 1.8.0_462-8u462-ga~us1-0ubuntu2~22.04.2-
b08)
OpenJDK 64-Bit Server VM (build 25.462-b08, mixed mode)
```

2. Bazel 6.5.0 is used to build the Label Image and Benchmark Model.

```
$ sudo apt install bazel-6.5.0
```

3. Clone the NXP TensorFlow repository.

```
$ mkdir imx-eiq-core-android && cd imx-eiq-core-android && export
 EIQ_CORE_ROOT=`pwd`
$ git clone --recurse-submodules https://github.com/nxp-imx/tensorflow-
imx.git --branch lf-6.18.37_2.1.0
```

4. Configure the Android Studio to download NDK and build tools.
   a. Download Android Studio 4.1.3.

```
$ wget https://redirector.gvt1.com/edgedl/android/studio/ide-zips/4.1.3.0/
android-studio-ide-201.7199119-linux.tar.gz
$ tar xvfz android-studio-ide-201.7199119-linux.tar.gz -C $EIQ_CORE_ROOT
$ mkdir ${EIQ_CORE_ROOT}/android-sdk
```

   b. Open Ubuntu desktop and navigate to `${EIQ_CORE_ROOT}/android-studio/bin` and run `studio.sh`.
   c. Change the SDK location to `${EIQ_CORE_ROOT}/android-sdk`.

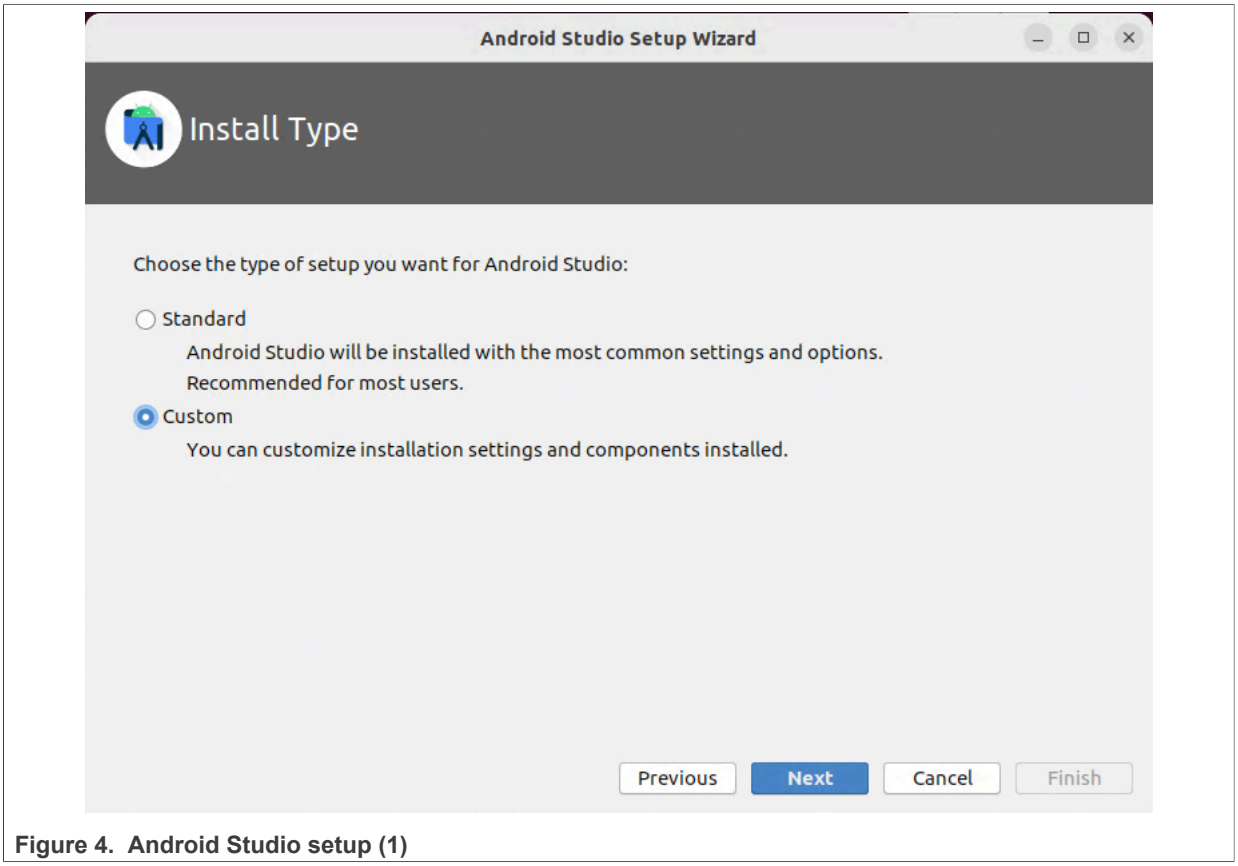


Figure 4. Android Studio setup (1)

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

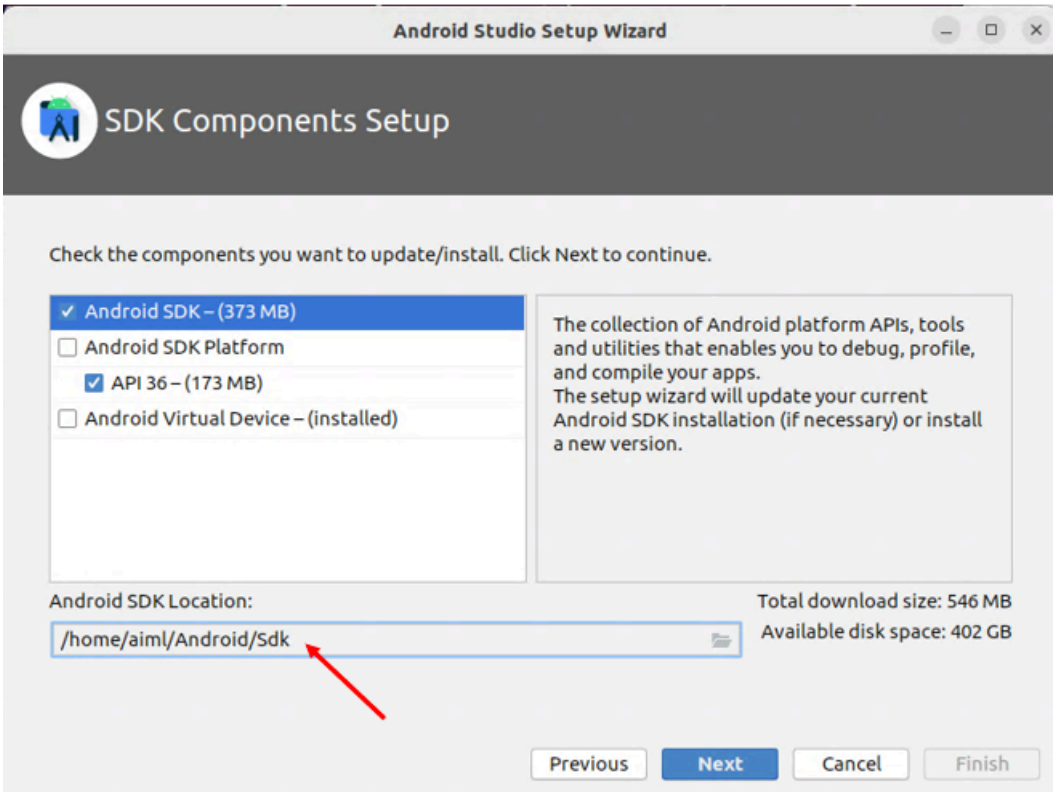


Figure 5. Android Studio setup (2)

d. Select **SDK Manager** from **Configure**.

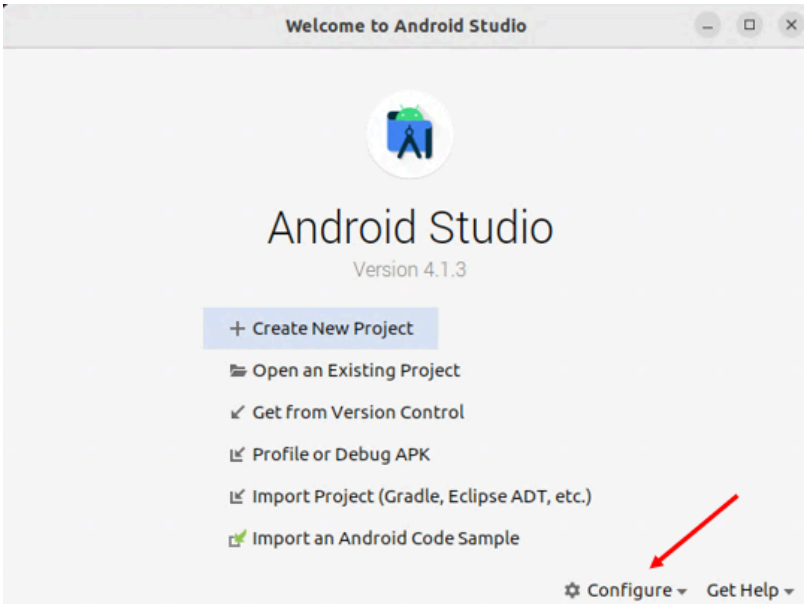


Figure 6. Android Studio configuration

i. Select **Android SDK Platform 26** and **Sources for Android 26** from **SDK Platforms**.
ii. Select **NDK 26.2.11394342**, **Build Tool 30.0.3**, and **Command-line Tool (latest)** from **SDK Tools**.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

iii. Apply the changes. The studio then downloads the SDK stuff to `${EIQ_CORE_ROOT}/android-sdk`.

5. Configure the Bazel build environment and NDK tools for TensorFlow by running `./configure`. The configuration step also asks for the local Python interpreter and library paths, which are not mandatory for following the application note but is useful for additional features. Other required variables can be defaulted.

```
$ cd ${EIQ_CORE_ROOT}/tensorflow-imx && ./configure
```

Edit `.bazelrc`, and append the following lines in the end of the file:

```
build --action_env ANDROID_NDK_HOME="${EIQ_CORE_ROOT}/android-sdk/
ndk/26.2.11394342"
build --action_env ANDROID_NDK_VERSION="26"
build --action_env ANDROID_NDK_API_LEVEL="26"
build --action_env ANDROID_BUILD_TOOLS_VERSION="30.0.3"
build --action_env ANDROID_SDK_API_LEVEL="26"
build --action_env ANDROID_SDK_HOME="${EIQ_CORE_ROOT}/android-sdk"
```

5.2 Building the Label Image

5.2.1 Building the APK

Run the following command:

```
$ bazel build -c opt --config=android_arm64 tensorflow/lite/examples/
label_image/android:label_image
```

The APK is then generated at:

- `bazel-bin/tensorflow/lite/examples/label_image/android/label_image.apk` for debugging
- `bazel-bin/tensorflow/lite/examples/label_image/android/label_image_unsigned.apk` for signing

5.2.2 Building the C++ binary

TensorFlow Lite provides the C++ version of the Label Image that can be executed in the Android shell environment with better performance. Run the following Bazel command to build it.

```
$ bazel build -c opt --config=android_arm64 tensorflow/lite/examples/
label_image:label_image
```

The binary is generated at `bazel-bin/tensorflow/lite/examples/label_image/label_image`. Run the binary on the Android OS with the following commands:

1. Follow Section 3.2 to push the model, label, and image to the Android OS.
2. Copy `label_image` to the Android OS, and then execute it in the Android shell.

```
$ adb push label_image /data/local/tmp/
$ adb shell
$ cd /data/local/tmp
```

- Run the model on the CPU:

```
$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite -l labels.txt -i
 grace_hopper.bmp
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- Run the model on the NPU on i.MX 8M Plus:

```
$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite -l labels.txt -i
 grace_hopper.bmp --external_delegate_path=/vendor/lib64/libvx_delegate.so
```

- Run the model on the NPU on i.MX 95/952/937:

```
$ ./label_image -m mobilenet_v1_1.0_224_quant_neutron.tflite -l
 labels.txt -i grace_hopper.bmp --external_delegate_path=/vendor/lib64/
libneutron_delegate.so
INFO: Loaded model mobilenet_v1_1.0_224_quant_neutron.tflite
INFO: resolved reporter
INFO: Initialized TensorFlow Lite runtime.
INFO: EXTERNAL delegate created.
INFO: NeutronDelegate delegate: 1 nodes delegated out of 4 nodes with 1
 partitions.

VERBOSE: Replacing 1 out of 4 node(s) with delegate (NeutronDelegate) node,
 yielding 3 partitions for subgraph 0.
INFO: Neutron delegate version: v1.0.0-be8bf399, zerocp enabled.
INFO: Applied EXTERNAL delegate.
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
INFO: invoked
INFO: average time: 1.491 ms
INFO: 0.729412: 653 military uniform
INFO: 0.164706: 907 Windsor tie
INFO: 0.0196078: 458 bow tie
INFO: 0.00784314: 835 suit
INFO: 0.00784314: 466 bulletproof vest
```

5.3 Building the Benchmark Model

5.3.1 Building the APK

Run the following command:

```
$ bazel build -c opt --config=android_arm64 tensorflow/lite/tools/benchmark/
android:benchmark_model
```

The APK is then generated at:

- `bazel-bin/tensorflow/lite/tools/benchmark/android/benchmark_model.apk` for debugging
- `bazel-bin/tensorflow/lite/tools/benchmark/android/benchmark_model_unsigned.apk` for signing

5.3.2 Building the C++ binary

As described in Section 3.3, the pre-installed `benchmark_model.apk` can only benchmark the single core performance. In contrast, the `benchmark_model` binary can run with multiple cores to get the best performance, especially for inference benchmark on CPUs.

```
$ bazel build -c opt --config=android_arm64 tensorflow/lite/tools/
benchmark:benchmark_model
```

The binary is generated at `bazel-bin/tensorflow/lite/tools/benchmark/benchmark_model`.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- Benchmark Model inference performance for the CPU:

```
$ adb push benchmark_model /data/local/tmp/
$ adb shell
$ cd /data/local/tmp
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite --
num_threads=<num_of_cores>
INFO: STARTING!
INFO: Log parameter values verbosely: [0]
INFO: Num threads: [6]
INFO: Graph: [mobilenet_v1_1.0_224_quant.tflite]
INFO: Signature to run: []
INFO: #threads used for CPU inference: [6]
INFO: Loaded model mobilenet_v1_1.0_224_quant.tflite
INFO: Initialized TensorFlow Lite runtime.
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
INFO: The input model file size (MB): 4.27635
INFO: Initialized session in 7.094ms.
INFO: Running benchmark for at least 1 iterations and at least 0.5 seconds but
 terminate if exceeding 150 seconds.
INFO: count=42 first=17689 curr=11871 min=11847 max=17689 avg=12103.8 std=885
 p5=11851 median=11926 p95=12221

INFO: Running benchmark for at least 50 iterations and at least 1 seconds but
 terminate if exceeding 150 seconds.
INFO: count=83 first=12945 curr=12015 min=11843 max=13839 avg=12028.5 std=275
 p5=11855 median=11979 p95=12258

INFO: Inference timings in us: Init: 7094, First inference: 17689, Warmup
 (avg): 12103.8, Inference (avg): 12028.5
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=3.64062
 overall=9.22656
```

- Benchmark Model inference performance for the NPU on i.MX 8M Plus:

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite --num_threads=4
 --external_delegate_path=/vendor/lib64/libvx_delegate.so
```

- Benchmark Model inference performance for the NPU on i.MX 95/952/937:

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant_neutron.tflite --
num_threads=6 --external_delegate_path=/vendor/lib64/libneutron_delegate.so
```

5.4 Building the TFLite Camera Demo

Android Studio+Gradle is used to build `tflitecamerademo.apk`.

1. Copy the eIQ TFLite libraries to `${EIQ_CORE_ROOT}/tensorflow-imx/tensorflow/lite/java/demo/app/libs`.
2. Open Android Studio, select **Open an Existing Project**, and then open `${EIQ_CORE_ROOT}/tensorflow-imx/tensorflow/lite/java/demo`.
3. After the project synchronization is completed, click **Build** -> **Build bundles(s)/APK(s)** -> **Build APKs**.

The APK is then generated at `./build/outputs/apk/debug/app-debug.apk`.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

5.5 Rebuilding the Android image with updated applications

To rebuild the Android image with updated applications, perform the following steps:

1. Download Android 17.0.0_1.0.0 BSP package and the *Android User's Guide* (UG10156) from Android OS for i.MX Applications Processors, and extract the Android BSP package to `${EIQ_CORE_ROOT}/android_16.0.0` (will be referred to as `MY_ANDROID` from here on).
2. Follow Section "Building Android images" in the *Android User's Guide* (UG10156) to download the GCC toolchain for AArch32 and AArch64, `clang`, `kernel-build-tools`, `rust`, and `clang-tools`.
3. Copy `label_image.apk`, `benchmark_model.apk`, and `app-debug.apk` built in Section 5.2.1, Section 5.3.1, and Section 5.4 to `${MY_ANDROID}/vendor/nxp/neutron-software-stack/Android/TfLiteApks/`. Rename `app-debug.apk` to `TfliteCameraDemo.apk`.
4. Follow the *Android User's Guide* (UG10156) to build and flash images for your device, and boot up the Android OS. The updated applications are on the Android OS. See Section 3 for the usages.

6 Installing a third-party application to the Android OS

Assume that you have built your own application based on the eIQ Core for the Android OS and you want to install it to the Android OS. However, the application cannot work with the NPU after the installation due to SELinux security policy. You need to grant the NPU libraries access permissions to user applications.

Follow Section 5.5 to set up the building environment for Android 16. The Android source code contains the following relevant resources:

- `${MY_ANDROID}/vendor/nxp/fsl-proprietary/gpu-viv`: contains GPU/NPU drivers.
- `${MY_ANDROID}/vendor/nxp/fsl-proprietary/include`: contains GPU/NPU driver headers.
- `${MY_ANDROID}/vendor/nxp/neutron-software-stack/Android/TfLiteLib`: contains Android libraries in the AAR format.
- `${MY_ANDROID}/device/nxp/`: contains Android build and configuration files for NXP platforms, primarily Kati `.mk` files, and configurations for specific devices, including Security-Enhanced Linux (SELinux) policy descriptions.

6.1 Adding additional native libraries

From Android 7 onwards, AOSP allows providing additional native libraries accessible to applications by ubicating them into specific library folders and explicitly listing them in a `.txt` file.

This is relevant as the VX Delegate has dynamic dependencies on the TIM-VX library for i.MX 8M Plus, and the Neutron Delegate also has dependencies on the Neutron Driver library.

As this file is not generated by default, add it into the vendor partition of the Android image:

1. Create an empty `public.libraries.txt` file in the device config directory `${MY_ANDROID}/device/nxp/<family>/<board>`.
2. Add the following listed shared objects to the `public.libraries.txt` (each one in a line):
   - For i.MX 8M Plus (`${MY_ANDROID}/device/nxp/imx8m/evk_8mp/public.libraries.txt`):

     ```
     libtim-vx.so
     ```

   - For i.MX 95/952/937 (`${MY_ANDROID}/device/nxp/imx9/evk_95/public.libraries.txt`):

     ```
     libNeutronDriver.so
     ```

   To allow the GPU Delegate for the i.MX 95/952/937 target, add `libOpenCL.so` to the `public.libraries.txt` file as well.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

3. Add the updated `public.libraries.txt` file in the list of artifacts to copy to the image for your device. The patch below is for the i.MX 8M Plus platform. For i.MX 95/952/937, modify the `imx9/evk_<soc>/evk_<soc>.mk` file in the same way.

```
diff --git a/imx8m/evk_8mp/evk_8mp.mk b/imx8m/evk_8mp/evk_8mp.mk
index eb5d3056..6f51d1a0 100644
--- a/imx8m/evk_8mp/evk_8mp.mk
+++ b/imx8m/evk_8mp/evk_8mp.mk
@@ -594,6 +594,9 @@ PRODUCT_COPY_FILES += \
 PRODUCT_COPY_FILES += \
     frameworks/native/data/etc/android.software.device_id_attestation.xml:
$(TARGET_COPY_OUT_VENDOR)/etc/permissions/
android.software.device_id_attestation.xml

+# public vendor libs
+PRODUCT_COPY_FILES += \
+     $(IMX_DEVICE_PATH)/public.libraries.txt:$(TARGET_COPY_OUT_VENDOR)/etc/
public.libraries.txt
+
 # Included GMS package
 ifeq ($(filter TRUE true 1,$(IMX_BUILD_32BIT_ROOTFS)
 $(IMX_BUILD_32BIT_64BIT_ROOTFS)),)
 $(call inherit-product-if-exists, vendor/partner_gms/products/
gms_64bit_only.mk)
```

6.2 Configuring SELinux labels for native libraries

Android uses Security-Enhanced Linux (SELinux) to enforce mandatory access control over all processes. SELinux works in two modes: permissive and enforcing. In both modes, permission denials are logged, but in the enforcing case, the kernel ensures that the access is not granted. Since Android 7, there were major native symbol restrictions for linking and loading, including `dlopen`-related operations, which are relevant to the VX Delegate enablement for i.MX 8M Plus and Neutron Delegate for i.MX 95/952/937.

To avoid this issue and allow applications using the VX Delegate and Neutron Delegate to run in restrictive mode, a set of vendors shared libraries need to be labeled as `vendor_app_file`:

For the i.MX 8M Plus, modify the `/imx8m/sepolicy/file_contexts b/imx8m/sepolicy/` file as shown in the patch below:

```
diff --git a/imx8m/sepolicy/file_contexts b/imx8m/sepolicy/file_contexts
index 8c160239..ba7fa202 100644
--- a/imx8m/sepolicy/file_contexts
+++ b/imx8m/sepolicy/file_contexts
@@ -39,6 +39,14 @@
 /vendor/lib(64)?/libGLSLC\.so               u:object_r:same_process_hal_file:s0
 /vendor/lib(64)?/libVSC\.so                 u:object_r:same_process_hal_file:s0
 /vendor/lib(64)?/libGAL\.so                 u:object_r:same_process_hal_file:s0

+/vendor/lib(64)?/libOpenVX\.so               u:object_r:vendor_app_file:s0
+/vendor/lib(64)?/libOpenVXU\.so              u:object_r:vendor_app_file:s0
+/vendor/lib(64)?/libarchmodelSw\.so          u:object_r:vendor_app_file:s0
+/vendor/lib(64)?/libNNArchPerf\.so           u:object_r:vendor_app_file:s0
+/vendor/lib(64)?/libNNVXCBinary-evis2\.so    u:object_r:vendor_app_file:s0
+/vendor/lib(64)?/libOvx12VXCBinary-evis2\.so u:object_r:vendor_app_file:s0
+/vendor/lib(64)?/libNNGPUBinary-evis2\.so    u:object_r:vendor_app_file:s0
+/vendor/lib(64)?/libtim-vx\.so               u:object_r:vendor_app_file:s0
+
 /vendor/lib(64)?/hw/vulkan\.imx\.so        u:object_r:same_process_hal_file:s0
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
 /vendor/lib(64)?/hw/gralloc_viv\.imx\.so   u:object_r:same_process_hal_file:s0
```

For the i.MX 95/952/937, add only `libNeutronDriver.so` to the `contextcmake bui` file:

```
+/vendor/lib(64)?/libNeutronDriver\.so
 u:object_r:vendor_app_file:s0
```

6.3 Booting the Android OS in permissive mode

To boot the Android OS in permissive mode, perform the following steps:

1. Follow the *Android User's Guide* (UG10156) to build and flash the image to the target device.
2. Connect the console through the serial port of the target. Access U-Boot and set extra bootargs, and then boot the Android OS.
3. After accessing the system prompt, check if the libraries have been added.

6.4 Installing and executing applications

With ADB, applications can be installed by using the following command (ensure that the Android OS is booted and running):

```
$ adb install -r -d -g <Path_to_the_apk>
```

Where:

- `-r`: Reinstalls the application, keeping the data.
- `-d`: Allows version code downgrade.
- `-g`: Grants all permissions defined in the application manifest file.

If the installation succeeds, you can check the installation path by using the `adb shell` commands, which provide most of the usual Unix command-line tools. Using `ls-based` commands, the base path for the application can be found in `/data/app/<gen_string_1>/<package_name>-<gen_string_2>/`. Inside this path, `lib/arm64` contains the TFLite JNI libraries.

```
-rwxr-xr-x 1 system system  226816 2010-01-01 00:00
 libtensorflowlite_external_delegate_jni.so
-rwxr-xr-x 1 system system 2519344 2010-01-01 00:00 libtensorflowlite_gpu_jni.so
-rwxr-xr-x 1 system system 4001112 2010-01-01 00:00 libtensorflowlite_jni.so
```

If navigation with `ls` is restricted, `adb root` can be used to access the restricted file systems. To return to non-root mode, use `adb unroot`. These commands affect the ADB daemon (`adbd`) in the platform, which means that the root session may exceed the terminal life cycle.

To run the command-line applications or GUI applications, see Section 3.

7 Optimization

For some cases, you may need to do extra configuration to get optimized performance.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

7.1 General

7.1.1 Using Per-Channel-Quantization (PCQ) model instead of Per-Tensor-Quantization (PTQ) model

Since TFLite 2.18, the XNNPack Delegate discards optimization for asymmetric uint8 Conv2D operators. Model performance significantly degrades on the CPU when using asymmetric uint8 Conv2D operators, such as MobileNet V1/V2 and Inception V4 model.

A workaround is to convert asymmetric uint8 Conv2D to symmetric int8 Conv2D. This conversion can be facilitated using the `tflite-optimizer` tool available in the eIQ Neutron SDK.

```
neutron_sdk/bin $ ./tflite-optimizer --input <original model path> --output
 <converted model path> --run=ConvertAsymUint8ToSymInt8
```

7.1.2 Setting the CPU to performance mode

By default, the CPU operates in `ondemand` mode, which decides how the CPU frequency should be adjusted based on system load. You need to set the CPU to `performance` mode.

```
$ echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

7.2 i.MX 8M Plus

7.2.1 PCQ model enablement for i.MX 8M Plus

To enable optimal execution of Per Channel Quantized (PCQ) models, additional configuration is required by the NPU on i.MX 8M Plus. From an `adb shell`, run the following commands:

```
$ setprop vendor.VIV_VX_ENABLE_GRAPH_TRANSFORM -pcq:1
$ setprop vendor.VIV_VX_SET_PER_CHANNEL_ENTROPY 0.35
```

You can validate the properties by using `getprop`. Without these properties, a noticeable drop in performance is expected.

7.2.2 Hardware accelerators warmup time for i.MX 8M Plus

For TensorFlow Lite, the initial execution of model inference takes longer time due to the model graph initialization needed by the GPU/NPU hardware accelerator. The initialization phase is known as warmup. To reduce this initialization time for subsequent applications running on the i.MX 8M Plus target, the information generated during the initial OpenVX graph processing can be saved to disk. To enable this optimization, the following environment variables should be set:

```
$ setprop vendor.VIV_VX_ENABLE_CACHE_GRAPH_BINARY 1
$ setprop vendor.VIV_VX_CACHE_BINARY_GRAPH_DIR `pwd`
```

When these environment variables are set, the result of the OpenVX graph compilation is saved to disk as network binary graph files (`*.nb`). During runtime, a quick hash check is performed on the network, and if it matches the hash of the corresponding `.nb` file, the graph is directly loaded into NPU memory.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

7.3 i.MX 9 series

7.3.1 Disabling DDR clock gating

On the i.MX 9 series, DDR clock gating is enabled by default to reduce power consumption during idle states. However, this feature may negatively impact TensorFlow Lite performance. To mitigate this, disable DDR clock gating in the System Manager. Assuming the console serial port is `/dev/ttyUSB2`, the corresponding System Manager console is accessible through `/dev/ttyUSB3`.

```
>$ mm 0x4e010010 0
```

This configuration will be lost after power-down.

8 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025-2026 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9 Revision history

Table 1. Revision history

Document ID	Release date	Description
UG10338 v.1.4	3 November 2026	Minor update for the android-17.0.0_1.0.0 release.
UG10338 v.1.3	24 July 2026	Minor update for the android-16.0.0_2.0.0 release.
UG10338 v.1.2	1 April 2026	Minor update for the android-16.0.0_1.4.0 release, changing eIQ Toolkit to eIQ Neutron SDK.
UG10338 v.1.1	6 January 2026	Minor update for the android-16.0.0_1.2.0 release.
UG10338 v.1.0	28 October 2025	Initial release, new document added for the i.MX android-16.0.0_1.0.0 release.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Contents

1 Introduction 2
2 NNAPI Delegate and dedicated delegate 2
3 Pre-installed ML applications 3
3.1 Preparation 3
3.1.1 Installing the ADB 3
3.1.2 Downloading the MobileNet v1 model and test files 3
3.1.3 Installing eIQ Neutron SDK 3
3.2 Label Image 3
3.3 Benchmark Model 5
3.4 TFLite Camera Demo 7
4 Building Android applications based on the NXP eIQ Core 8
4.1 eIQ TFLite libraries for the Android OS 8
4.2 Creating a TFLite interpreter with the NPU delegate 9
4.3 Adding eIQ TFLite libraries in the build file 9
4.3.1 Gradle build 9
4.3.2 Bazel build 10
4.4 Building the TFLite Camera Demo based on the eIQ core 11
5 Building and updating the pre-installed application from sources 11
5.1 Preparation 11
5.2 Building the Label Image 14
5.2.1 Building the APK 14
5.2.2 Building the C++ binary 14
5.3 Building the Benchmark Model 15
5.3.1 Building the APK 15
5.3.2 Building the C++ binary 15
5.4 Building the TFLite Camera Demo 16
5.5 Rebuilding the Android image with updated applications 17
6 Installing a third-party application to the Android OS 17
6.1 Adding additional native libraries 17
6.2 Configuring SELinux labels for native libraries 18
6.3 Booting the Android OS in permissive mode ... 19
6.4 Installing and executing applications 19
7 Optimization 19
7.1 General 20
7.1.1 Using Per-Channel-Quantization (PCQ) model instead of Per-Tensor-Quantization (PTQ) model 20
7.1.2 Setting the CPU to performance mode 20
7.2 i.MX 8M Plus 20
7.2.1 PCQ model enablement for i.MX 8M Plus 20
7.2.2 Hardware accelerators warmup time for i.MX 8M Plus 20
7.3 i.MX 9 series 21
7.3.1 Disabling DDR clock gating 21
8 Note about the source code in the document 21
9 Revision history 21
Legal information 22

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2026 NXP B.V. **All rights reserved.**