AN14411

Enabling eIQ Core NPU Delegates for i.MX Android Applications

Rev. 1.0 — 23 September 2024

Application note

Document information

Information	Content
Keywords	AN14411, i.MX 8MP and i.MX 95 application processors, Neural Networks API (NNAPI), Android OS, Android Native Development Kit (NDK), board support package (BSP), eIQ Core NPU Delegates, i.MX Android applications
Abstract	This document describes the deployment of the TensorFlow Lite inference engine and related delegates for on-chip Machine learning (ML) accelerators (NPU and GPU) available on i.MX 8MP and i.MX 95 application processors.

1 Introduction

This application note describes how to deploy part of the eIQ® Core into the NXP Android board support package (BSP). It specifically describes the deployment of the TensorFlow Lite inference engine and related delegates for on-chip Machine learning (ML) accelerators (NPU and GPU) available on i.MX 8MP and i.MX 95 application processors.

The primary objective is to enable the use of ML hardware accelerators to achieve performance comparable to that on NXP Yocto Linux. The following hardware accelerators are addressed in this application note:

- NPU on the i.MX 8MP platform using the VX Delegate,
- eIQ Neutron NPU using Neutron Delegate on the i.MX 95 platform.

This document is divided as follows:

Section 2 "NNAPI versus dedicated delegates" compares the dedicated delegates with the standard Android mean for accessing ML HW Accelerators.

Section 3 "Building the Android platform" and Section 4 "Building eIQ Core using the NDK" describe the steps to:

- Configure Android BSP
- Build the eIQ Core packages using Android's Native Development Kit (NDK)
- Package the eIQ Core components into the Android Application Package and its deployment on the platform.

Section 5 "App installation and execution" shows a demo application running the inference on the HW accelerator.

The requirements to follow this application note are:

- i.MX 8M Plus or i.MX 95 Evaluation Kit
- Build Host machine with 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 Application Note describes the deployment on **Android 14.0.0_2.0.0 BSP** and **eIQ Core LF 6.6.23_2.0.0** releases.*

2 NNAPI versus dedicated delegates

Android provides the Neural Networks API (NNAPI) to access the ML accelerators. For instance, the NNAPI Delegate in the TensorFlow Lite inference engine uses this API.

The NNAPI is a software layer defined by Android OS that enables execution of compute-intensive operations. For this purpose, it uses the available hardware such as the CPU, GPU, and other ML HW Accelerators. It is a part of the Android public API since level 27 and adds new features with every Android release. Different inference engines can use the NNAPI. For example, the API can be accessed through the NNAPI delegate in TensorFlow Lite or NNAPI provider in ONNX Runtime.

As Android governs the NNAPI definition, misalignments between NNAPI, the supported capabilities of the acceleration hardware (device manufacturer), and the inference engines (third-party organization) are expected. For instance, in i.MX platforms, TensorFlow Lite operator definition and NPU supported operators are heavily aligned. On the other hand, NNAPI must support multiple hardware backends (CPU, GPU, and custom accelerators) from multiple manufacturers and multiple inference engines. These requirements force NNAPI to allocate best effort definitions for all stakeholders. Therefore, the NNAPI delegate might allocate operators that could run on NPU to CPU, which impacts overall inference performance.

Compared to NNAPI, the dedicated delegates for specific HW accelerators such as VX Delegate for i.MX 8MP and Neutron Delegate for i.MX 95 platforms, are fully aligned with NPU capabilities. The NPU delegates (VX Delegate and Neutron Delegate) are key components of the eIQ SW enablement stack to apply the HW acceleration on the ML workload. They are already supported in NXP Embedded Linux for i.MX Applications Processors, with proven acceleration of embedded ML models. The following sections of this document explain how to use the NPU delegates in TensorFlow Lite Android applications.

3 Building the Android platform

The details of how to build the Android platform for i.MX, can be found in https://www.nxp.com/docs/en/user-guide/ANDROID_USERS_GUIDE.pdf. The chapter aims to describe how to modify Android sources to support eIQ Core.

For other documents related to Android platform, refer https://www.nxp.com/design/design-center/software/embedded-software/i-mx-software/android-os-for-i-mx-applications-processors:IMXANDROID#documentation

3.1 Android setup

This section describes the procedure to build the Android platform.

- Download the Android 14.0.0_2.0.0 release from the URL https://www.nxp.com/design/design-center/software/embedded-software/i-mx-software/android-os-for-i-mx-applications-processors:IMXANDROID and unzip it in a convenient directory.

```
$ tar -xzvf imx-android-14.0.0_2.0.0.tar.gz
```

- Install the repo utility using the command below (perform this step only once):

```
$ mkdir ~/bin
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
$ export PATH=${PATH}:~/bin
```

- Set up the i.MX Android root path using the command below:

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

After sourcing the setup script, the working directory points to the root where all required sources have been unrolled (from now onwards referred as `ANDROID_ROOT`). This working directory contains the following relevant resources:

- `${ANDROID_ROOT}/vendor/nxp/fsl-proprietary/gpu-viv:` contains GPU/NPU drivers.
- `${ANDROID_ROOT}/vendor/nxp/fsl-proprietary/include`: contains GPU/NPU drivers headers.
- `${ANDROID_ROOT}/device/nxp/`: contains Android build and configuration files for NXP platforms. These include primarily `Kati.mk` files and configurations for specific devices, including Security-Enhanced Linux (SELinux) policy descriptions.

3.2 Access to additional native libraries

Starting from Android 7 onwards, AOSP (Android Open Source Project) allows to provide access to additional native libraries accessible to applications.

To support this feature, the libraries are put into specific library folders and explicitly listed in a .txt file.

This feature is relevant as both VX Delegate and Neutron Delegate have dynamic dependencies on TIM-VX and Neutron Driver libraries, respectively.

The .txt file is not generated by default. Therefore, it must be added into the vendor partition of the Android image by using the below steps:

1. Create an empty *public.libraries.txt* file in the device config directory:

`${ANDROID_ROOT}/device/nxp/<family>/<board>`

2. Add the following shared object names listed to the `public.libraries.txt` (each one in a line):
   - For i.MX 8MP: (*${ANDROID_ROOT}/device/nxp/imx8m/evk_8mp/public.libraries.txt*):
     ```
     libtim-vx.so
     ```
   - For i.MX 95: (*${ANDROID_ROOT}/device/nxp/imx9/evk_95/public.libraries.txt*):
     ```
     libNeutronDriver.so
     ```

   Additionally, to enable the GPU Delegate for the i.MX 95 target, include the *libOpenCL.so* filename to the `public.libraries.txt` file.
3. To copy the updated `public.libraries.txt` file to the target device image, add it in the list of artifacts. The patch below is for the i.MX 8MP platform. For i.MX 95 platform, modify the `imx9/evk_95/evk_95.mk` file in the same manner.
   ```
   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)
   ```

3.3 Configuring SE Linux labels for native libraries

Android uses SELinux to enforce mandatory access control over all processes. SELinux works in 2 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. These restrictions are relevant to the VX Delegate enablement for i.MX 8MP.[1]

To update the policy and therefore enable applications using VX Delegate to run in restrictive mode, a set of vendor shared libraries must be labeled as `vendor_app_file`:

For the i.MX 8MP, 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
 /vendor/lib(64)?/hw/gralloc_viv\.imx\.so      u:object_r:same_process_hal_file:s0
```

For the i.MX 95 target, add only the `libNeutronDriver.so` to the file context:

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

3.4 Building the Android image

For details about how to build the Android image, check the Android User's Guide. Ensure to prepare the build environment for U-Boot and Linux kernel as described in Chapter 3.2. of the *Android User's Guide*. After the environment is prepared, build the image using the below commands. These commands show an example for the User debug mode:

1. Set up the environment variables using the commands below:

```
$ cd ${ANDROID_ROOT}
$ source build/envsetup.sh
$ export CLANG_PATH=<Path_to_Android_toolchain>/prebuilt-android-clang
$ export ARMGCC_DIR=<Path_to_Android_toolchain>/arm-gnu-toolchain-12.3.rel1-
x86_64-arm-none-eabi
$ export AARCH32_GCC_CROSS_COMPILE=<Path_to_Android_toolchain>/arm-gnu-
toolchain-12.3.rel1-x86_64-arm-none-eabi/bin/arm-none-eabi-
$ export AARCH64_GCC_CROSS_COMPILE=<Path_to_Android_toolchain>/arm-gnu-
toolchain-12.3.rel1-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-
```

1 https://developer.android.com/about/versions/nougat/android-7.0-changes.html#ndk.

```
$ export PATH=<Path_to_Android_toolchain>/prebuilt-android-kernel-build-
tools/linux-x86/bin:$PATH
```

2. Run the lunch command:[2]:
   a. For i.MX 8MP:
      ```
      $ lunch evk_8mp-trunk_staging-userdebug
      ```
   b. For i.MX 95:
      ```
      $ lunch evk_95-trunk_staging-userdebug
      ```
3. Trigger the build:
   ```
   $ ./imx-make.sh -j16
   ```
   The output Android image can be found in `${ANDROID_ROOT}/out/target/product/evk_8mp` (for the i.MX8MP) or `${ANDROID_ROOT}/out/target/product/evk_95` (for the i.MX 95). Now, the image is ready for VX Delegate or Neutron Delegate deployment. Download the image with UUU into a target device. See the Android Quick Start Guide (AQSUG) for a detailed description of UUU. See Section 7 "References".

2 For details of the 'lunch' command, refer to the URL:https://source.android.com/docs/setup/build/building.

4 Building eIQ Core using the NDK

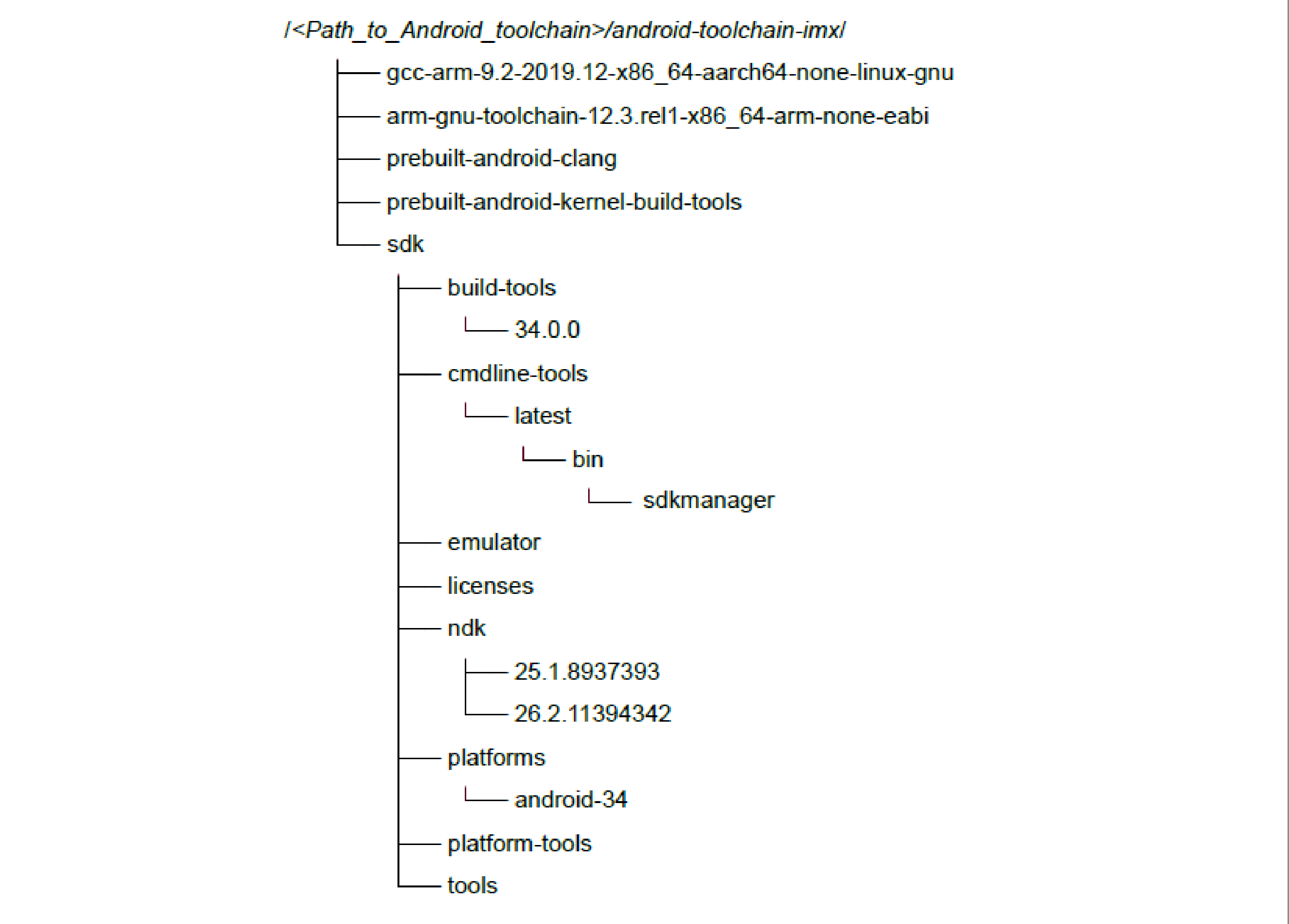
This chapter describes the process for building eIQ core components for Android. The source code for the software components is fetched from NXP's GitHub repo. It requires applying a set of patches to enable compilation for Android and building the components with *CMake* and Android's NDK.

The NDK is a set of tools that enables the use of C/C++ (native) code within Android OS and exposes public platform libraries to manage activities and operate low level components. NDK will be used to build the ML inference engines, as most of the existing code is written in C or C++. Focusing on native code through the NDK, enables a separation of concerns between the Java/Kotlin code used in typical Android applications and the inference runtime. The Java Native Interface (JNI) acts as a bridge between these two spaces. More information can be found in https://developer.android.com/training/articles/perf-jni.

4.1 Tools setup

Along with the NDK, there are a few other Android tools required to build the components (for example, `build-tools` and `platforms`). To manage to download the proper tool version and environment setup, Android provides the ***sdkmanager*** tool as part of the `command-line` toolset.

These Android tools have some restrictions related to directory placement. Figure 1 shows a valid directory structure (including AOSP tools):



```
/<Path_to_Android_toolchain>/android-toolchain-imx/
    ├── gcc-arm-9.2-2019.12-x86_64-aarch64-none-linux-gnu
    ├── arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi
    ├── prebuilt-android-clang
    ├── prebuilt-android-kernel-build-tools
    └── sdk
         ├── build-tools
         │     └── 34.0.0
         ├── cmdline-tools
         │     └── latest
         │           └── bin
         │                 └── sdkmanager
         ├── emulator
         ├── licenses
         ├── ndk
         │     ├── 25.1.8937393
         │     └── 26.2.11394342
         ├── platforms
         │     └── android-34
         ├── platform-tools
         └── tools
```

Figure 1. Directory structure for the tools setup

Download the command-line tools for Linux from the URL: https://developer.android.com/studio (lower part of the webpage). Unzip into a convenient directory (for example: check previous structure listing). To use this tool, ensure that the host system has a valid JDK installed (https://www.oracle.com/java/technologies/downloads/#jdk21-linux).

Use *sdkmanager* to install the required packages. Use NDK version 26c, *build-tools* version 34.0.0, and *platforms* version 34. The 25b NDK must be used for TensorFlow 2.15.0, because it does not support the newest 26 NDK. All available packages can be listed using the command, `sdkmanager -list`.

```
$ <path_to_sdkmanager>/sdkmanager --install "ndk;25.1.8937393"
$ <path_to_sdkmanager>/sdkmanager --install "ndk;26.2.11394342"
$ <path_to_sdkmanager>/sdkmanager --install "build-tools;34.0.0"
$ <path_to_sdkmanager>/sdkmanager --install "platforms;android-34"
```

To set up *CMake* with NDK information, the NDK includes a *CMake* toolchain file. Export a variable pointing to the `android.toolchain.cmake` file by using the command below:

```
$ export NDK_TOOLCHAIN_FILE=<PATH_TO_NDK>/build/cmake/android.toolchain.cmake
```

4.2 Fetching and preparing the code

The code for TensorFlow, and NPU Delegates (VX and Neutron) is available on https://github.com/nxp-imx:

1. Create the folder for eIQ core and clone the TensorFlow Lite repo:

   ```
   $ 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-6.6.23_2.0.0-android
   ```

2. For **i.MX 8MP only**:
   Clone the VX Delegate repo using the command below:

   ```
   $ git clone https://github.com/nxp-imx/tflite-vx-delegate-imx.git --branch
    lf-6.6.23_2.0.0-android
   ```

3. For **i.MX 95 only:**
   Clone the Neutron Delegate repo using the command below:

   ```
   $ git clone https://github.com/nxp-imx/tflite-neutron-delegate.git --branch
    lf-6.6.23_2.0.0-android
   ```

4.3 Building NPU Delegate

4.3.1 VX Delegate for i.MX 8MP

The component is built using CMake with a set of arguments. For this component, the TensorFlow source path is needed. The TIM-VX library is a part of the Android build. Ensure that the `ANDROID_ROOT` variable is set as described in the section, Section 3.1 "Android setup".

1. Export the environment variables using the commands below:

```
$ export VX_DELEGATE_PATH=${EIQ_CORE_ROOT}/tflite-vx-delegate-imx
$ export TF_PATH=${EIQ_CORE_ROOT}/tensorflow-imx
```

2. Use the '`cmake`' and '`build`' commands as shown below:

```
$ cmake -S ${VX_DELEGATE_PATH} -B ${VX_DELEGATE_PATH}/_build \
-DFETCHCONTENT_SOURCE_DIR_TENSORFLOW=${TF_PATH} \
-DCMAKE_TOOLCHAIN_FILE=${NDK_TOOLCHAIN_FILE} \
-DANDROID_ABI=arm64-v8a \
-DANDROID_PLATFORM=34 \
-DTFLITE_ENABLE_GPU=on \
-DANDROID_ROOT=${ANDROID_ROOT}
$ cmake --build ${VX_DELEGATE_PATH}/_build -- -j16
```

4.3.2 Neutron Delegate for i.MX 95

The component is build using *CMake* with a set of arguments. For this component, TensorFlow source path and Neutron Driver prebuilt library path are needed. Neutron Driver prebuilt library can be found in Android sources in the `vendor/nxp` folder.

1. Adjust the neutron driver folder structure for Android build:

```
$ mkdir -p ${ANDROID_ROOT}/vendor/nxp/neutron-software-stack/imx95/include/
neutron
$ cp file ${ANDROID_ROOT}/vendor/nxp/neutron-software-stack/imx95/include/*.h
 $_
```

2. Export the environment variables using the commands below:

```
$ export NEUTRON_DELEGATE_PATH=${EIQ_CORE_ROOT}/tflite-neutron-delegate
$ export TF_PATH=${EIQ_CORE_ROOT}/tensorflow-imx
$ export NEUTRON_DRIVER_PATH=${ANDROID_ROOT}/vendor/nxp/neutron-software-
stack/imx95
```

3. Cmake and build

```
$ cmake -S ${NEUTRON_DELEGATE_PATH} -B ${NEUTRON_DELEGATE_PATH}/_build \
-DFETCHCONTENT_SOURCE_DIR_TENSORFLOW=${TF_PATH} \
-DCMAKE_TOOLCHAIN_FILE=${NDK_TOOLCHAIN_FILE} \
-DANDROID_ABI=arm64-v8a \
-DANDROID_PLATFORM=34 \
-DNEUTRON_DRIVER_PATH=${NEUTRON_DRIVER_PATH}
$ cmake --build ${NEUTRON_DELEGATE_PATH}/_build -- -j16
```

4.4 Building the TensorFlow Lite Demo Application

This section describes how to build a demo Android application package provided by TensorFlow and containing the NPU Delegate shared library. The demo application code is in `${EIQ_CORE_ROOT}/tensorflow-imx/tensorflow/lite/tools/benchmark/android`. The *benchmark_model* measures the performance of the selected model and supports the use of different delegates. It is useful to compare the performance difference among the delegates. The application is built with Bazel (6.1.0 installation instructions for Ubuntu).

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

Follow the below steps to build the `benchmark_model.apk` with NPU Delegate:

1. Create a directory available to the `Bazel` file that creates the application, to copy the previously built eIQ components.

```
 $ mkdir  ${EIQ_CORE_ROOT}/tensorflow-imx/tensorflow/lite/tools/benchmark/
android/shared_libs
```

2. For **i.MX 8MP**, copy the VX Delegate library into `shared_libs` folder:

```
$ cp ${EIQ_CORE_ROOT}/tflite-vx-delegate-imx/_build/libvx_delegate.so
 ${EIQ_CORE_ROOT}/tensorflow-imx/tensorflow/lite/tools/benchmark/android/
shared_libs/
```

3. For **i.MX 95**, copy the Neutron Delegate and the Neutron Converter libraries into the `shared_libs` folder:

```
$ cp ${ EIQ_CORE_ROOT}/tflite-neutron-delegate/_build/libneutron_delegate.so
 ${EIQ_CORE_ROOT}/tensorflow-imx/tensorflow/lite/tools/benchmark/android/
shared_libs/
$ cp ${ ANDROID_ROOT}/vendor/nxp/neutron-software-stack/imx95/library/
libNeutronConverter.so ${EIQ_CORE_ROOT}/tensorflow-imx/tensorflow/lite/tools/
benchmark/android/shared_libs/
```

4. Configure the environment by running the `./configure` file. Also ensure that the following bazel environment variables are set in `.bazelrc` file that is located in the `tensorflow-imx` folder. The configuration step also asks for local python interpreter and library paths. These paths are not mandatory for following this application note but might be useful for additional features. Other required variables can be defaulted.

```
$ cd ${EIQ_CORE_ROOT}/tensorflow-imx && ./configure
build --action_env ANDROID_NDK_HOME="<PATH_TO_NDK_25.1.8937393>"
build --action_env ANDROID_NDK_VERSION="25"
build --action_env ANDROID_NDK_API_LEVEL="26"
build --action_env ANDROID_BUILD_TOOLS_VERSION="34.0.0"
build --action_env ANDROID_SDK_API_LEVEL="34"
build --action_env ANDROID_SDK_HOME="<PATH_TO_SDK>"
```

5. Finally, build the application using *Bazel* (for TensorFlow 2.15, *Bazel* 6.1.0 is expected).

```
$ cd ${EIQ_CORE_ROOT}/tensorflow-imx
$ bazel build -c opt --config=android_arm64 tensorflow/lite/tools/benchmark/
android:benchmark_model
```

If build is successful, the output is generated in the path below:

`${EIQ_CORE_ROOT}/bazel-bin/tensorflow/lite/tools/benchmark/android/benchmark_model.apk`

The *.apk* file can be unzipped to inspect if all expected components are present.

5 App installation and execution

This section describes the procedure for application installation, test modeling, execution, and results obtained.

5.1 Installation

Application installation and interaction with the i.MX application processors running Android is done using the Android Debug Bridge (ADB), which is part of the *platform-tools*. Platform tools must be present in the host connected to the target platform deploying the example package. For more information, refer to the URL:

https://developer.android.com/tools/releases/platform-tools.

With ADB, the `benchmark_model.apk` can be installed by using the following command (ensure that Android is booted up and running):

```
$ adb install -r -d -g <Path_to_the_apk>/benchmark_model.apk
```

Where:

`-r`: Reinstalls the app, keeping the data.

`-d`: Allows version code downgrade.

`-g`: Grants all permissions defined in the app manifest file.

If the installation succeeds, it is possible to inspect installation path using *adb shell* commands, which provides most of the usual Unix command-line tools. Using `ls`-based commands, the base path for the application can be found in the directory `/data/app/<gen_string_1>/org.tensorflow.lite.benchmark-<gen_string_2>/`. Inside this path, `lib/arm64` contains a delegate library - VX Delegate for i.MX 8MP or Neutron Delegate for i.MX 95.

If navigation with `ls` is restricted, `adb root` can be used to access restricted file systems. Returning to non-root mode can be done using `adb unroot`. Note that these commands affect the adb daemon (*adbd*) in the platform, meaning that theroot session might outlast the terminal lifecycle.

5.2 Test model

Download and unpack the `mobilenet_v1_1.0_224_quant` model:

```
$ wget download.tensorflow.org/models/mobilenet_v1_2018_08_02/
mobilenet_v1_1.0_224_quant.tgz
$ tar -xzvf mobilenet_v1_1.0_224_quant.tgz
```

For the i.MX 8MP target, no further conversion of the test model is required. Put the model onto the board with the **`adb`** tool:

```
$ adb push <Path_to_the_model>/mobilenet_v1_1.0_224_quant.tflite /data/local/
tmp/
```

For the i.MX 95 target, the model can be executed with the Neutron Delegate. The Neutron Delegate converts the model for the NPU.

Another option to convert the model before inference is by using the an offline conversion tool that is a part of eIQ Toolkit. For more information about the offline conversion tool, refer to the eIQ Toolkit User Guide. The user guide is a part of eIQ Toolkit installation. The installer can be found on the nxp.com.

The conversion in eIQ Toolkit can be run using the below command:

```
$ ./neutron-converter --input=mobilenet_v1_1.0_224_quant.tflite --target=imx95
```

After conversion, the model is named as: `mobilenet_v1_1.0_224_quant_converted.tflite`.

The converted model can be run only on the Neutron NPU. To run the model on CPU or GPU, use the original unconverted version.

5.3 Execution

The following ADB commands execute the application, using different backends. To check the output, the *adb logcat* command can be used in an extra terminal. The application logs are tagged with the keyword *“tflite”*.

1. **Execution with CPU (for both target platforms – i.MX 8MP and i.MX 95):**

   ```
   $ adb shell am start -S  -n
    org.tensorflow.lite.benchmark/.BenchmarkModelActivity --es args  '" \
   --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite "'
   ```

   There is a known issue with benchmark_model run on CPU with multiply threads on Android. The app performance for multiple threads significantly drops after board rebooting.
2. **Execution with Delegates on i.MX 8MP**
   - Execution with NNAPI delegate (only for i.MX 8MP):

     ```
     $ adb shell am start -S  -n
      org.tensorflow.lite.benchmark/.BenchmarkModelActivity --es args '" \
     --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite \
     --use_nnapi=true "'
     ```

   - Execution with VX Delegate (only for i.MX 8MP):

     ```
     $ adb shell am start -S  -n
      org.tensorflow.lite.benchmark/.BenchmarkModelActivity --es args '" \
     --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite \
     --external_delegate=libvx_delegate.so "'
     ```

3. **Execution with Delegates on i.MX 95**
   - Execution with GPU Delegate:

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

   - Execution with Neutron Delegate:

     ```
     $ adb shell am start -S -n
     org.tensorflow.lite.benchmark/.BenchmarkModelActivity --es args '" \
     --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite \
     --external_delegate=libneutron_delegate.so "'
     ```

5.4 Results

Filter *logcat* output for tflite (use *adb.exe logcat* | *findstr "tflite"* for Windows hosts).

5.4.1 Execution logs for i.MX 8MP

The average inference time can be compared for the mobilenet_v1_1.0_224_quant.tflite model on i.MX 8MP with Android 14 in the Table 1.

Table 1. Average inference time for the mobilenet_v1_1.0_224_quant.tflite model on i.MX 8MP with Android 14

i.MX 8MP	CPU	NNAPI	VX Delegate
Inference average, us	201151	3868	3299

1. **TensorFlow Lite XNNPACK delegate for CPU:**

```
tflite_BenchmarkModelActivity: Running TensorFlow Lite benchmark with args:
 --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
tflite: Log parameter values verbosely: [0]
tflite: Graph: [/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite]
tflite: Loaded model /data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
tflite: Initialized TensorFlow Lite runtime.
tflite: Created TensorFlow Lite XNNPACK delegate for CPU.
tflite: Replacing 29 out of 31 node(s) with delegate (TfLiteXNNPackDelegate)
 node, yielding 4 partitions for the whole graph.
tflite: The input model file size (MB): 4.27635
tflite: Initialized session in 240.783ms.
tflite: Running benchmark for at least 1 iterations and at least 0.5 seconds
 but terminate if exceeding 150 seconds.
tflite: count=3 first=228116 curr=200867 min=198790 max=228116 avg=209258
 std=13361
tflite: Running benchmark for at least 50 iterations and at least 1 seconds
 but terminate if exceeding 150 seconds.
tflite: count=50 first=200313 curr=214059 min=180116 max=236908 avg=201151
 std=9509
tflite: Inference timings in us: Init: 240783, First inference: 228116,
 Warmup (avg): 209258, Inference (avg): 201151
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=11.375
 overall=21.4922
```

2. **TensorFlow Lite delegate for NNAPI:**

```
tflite_BenchmarkModelActivity: Running TensorFlow Lite benchmark with args:
 --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite --use_nnapi=true
tflite: Log parameter values verbosely: [0]
tflite: Graph: [/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite]
tflite: Use NNAPI: [1]
tflite: NNAPI accelerators available: [nnapi-imx_sl,nnapi-reference]
tflite: Loaded model /data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
tflite: Initialized TensorFlow Lite runtime.
tflite: Created TensorFlow Lite delegate for NNAPI.
tflite: NNAPI delegate created.
tflite: NNAPI SL driver did not implement
 SL_ANeuralNetworksDiagnostic_registerCallbacks!
```

```
tflite: Replacing 31 out of 31 node(s) with delegate (TfLiteNnapiDelegate)
 node, yielding 1 partitions for the whole graph.
tflite: Explicitly applied NNAPI 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 680.547ms.
tflite: Running benchmark for at least 1 iterations and at least 0.5 seconds
 but terminate if exceeding 150 seconds.
tflite: count=1 curr=4988406
tflite: Running benchmark for at least 50 iterations and at least 1 seconds
 but terminate if exceeding 150 seconds.
tflite: count=254 first=4201 curr=3791 min=3554 max=5198 avg=3868.09 std=297
tflite: Inference timings in us: Init: 680547, First inference: 4988406,
 Warmup (avg): 4.98841e+06, Inference (avg): 3868.09
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=17.125
 overall=19.625
```

3. **EXTERNAL (Vx Delegate) delegate:**

```
tflite_BenchmarkModelActivity: Running TensorFlow Lite benchmark with
 args:  --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite --
use_vx_delegate=true  --external_delegate_path=/data/app/<...>/lib/arm64/
libvx_delegate.so
tflite: Unconsumed cmdline flags: --use_vx_delegate=true
tflite: Log parameter values verbosely: [0]
tflite: Graph: [/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite]
tflite: External delegate path: [/data/app/<...>/lib/arm64/libvx_delegate.so]
tflite: Loaded model /data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
tflite: Initialized TensorFlow Lite runtime.
tflite: Vx delegate: allowed_cache_mode set to 0.
tflite: Vx delegate: device num set to 0.
tflite: Vx delegate: allowed_builtin_code set to 0.
tflite: Vx delegate: error_during_init set to 0.
tflite: Vx delegate: error_during_prepare set to 0.
tflite: Vx delegate: error_during_invoke set to 0.
tflite: EXTERNAL delegate created.
tflite: Replacing 31 out of 31 node(s) with delegate (Vx Delegate) node,
 yielding 1 partitions for the whole graph.
tflite: Explicitly applied EXTERNAL 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 286.832ms.
tflite: Running benchmark for at least 1 iterations and at least 0.5 seconds
 but terminate if exceeding 150 seconds.
tflite: count=1 curr=7882997
tflite: Running benchmark for at least 50 iterations and at least 1 seconds
 but terminate if exceeding 150 seconds.
tflite: count=298 first=3963 curr=5542 min=3098 max=8828 avg=3298.88 std=473
tflite: Inference timings in us: Init: 286832, First inference: 7882997,
 Warmup (avg): 7.883e+06, Inference (avg): 3298.88
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=8.72656
 overall=114.73
```

5.4.2 Execution logs for i.MX 95 platform

Using the execution logs to compare the average inference time for the `mobilenet_v1_1.0_224_quant.tflite` model on i.MX 95 with Android 14. Table 2 shows the comparison of average inference time for the model.

Table 2. Comparison of average inference time for the mobilenet_v1_1.0_224_quant.tflite model on i.MX 95 with Android 14

i.MX 95	CPU	GPU Delegate	Neutron Delegate
Inference average, µs	66731	25341	1842

1. **TensorFlow Lite XNNPACK delegate for CPU:**

```
tflite_BenchmarkModelActivity: Running TensorFlow Lite benchmark with args:
 --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
tflite: Log parameter values verbosely: [0]
tflite: Graph: [/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite]
tflite: Loaded model /data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
tflite: Initialized TensorFlow Lite runtime.
tflite: Created TensorFlow Lite XNNPACK delegate for CPU.
tflite: Replacing 29 out of 31 node(s) with delegate (TfLiteXNNPackDelegate)
 node, yielding 4 partitions for the whole graph.
tflite: The input model file size (MB): 4.27635
tflite: Initialized session in 58.863ms.
tflite: Running benchmark for at least 1 iterations and at least 0.5 seconds
 but terminate if exceeding 150 seconds.
tflite: count=8 first=73840 curr=64899 min=64860 max=73840 avg=67422.8
 std=2774
tflite: Running benchmark for at least 50 iterations and at least 1 seconds
 but terminate if exceeding 150 seconds.
tflite: count=50 first=68311 curr=65812 min=62794 max=70290 avg=66731.2
 std=1599
tflite: Inference timings in us: Init: 58863, First inference: 73840, Warmup
 (avg): 67422.8, Inference (avg): 66731.2
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=10.875
 overall=18.2344
```

2. **TensorFlow Lite GPU delegate (OpenCL-based API):**

```
tflite_BenchmarkModelActivity: Running TensorFlow Lite benchmark with args:
 --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite --use_gpu=true
tflite: Log parameter values verbosely: [0]
tflite: Graph: [/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite]
tflite: Use gpu: [1]
tflite: Loaded model /data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
tflite: Initialized TensorFlow Lite runtime.
tflite: Created TensorFlow Lite delegate for GPU.
tflite: GPU delegate created.
tflite: Replacing 31 out of 31 node(s) with delegate (TfLiteGpuDelegateV2)
 node, yielding 1 partitions for the whole graph.
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 1473.09ms.
```

```
tflite: Running benchmark for at least 1 iterations and at least 0.5 seconds
 but terminate if exceeding 150 seconds.
tflite: count=20 first=26847 curr=25453 min=25278 max=26847 avg=25408.6
 std=331
tflite: Running benchmark for at least 50 iterations and at least 1 seconds
 but terminate if exceeding 150 seconds.
tflite: count=50 first=25441 curr=25314 min=25253 max=25630 avg=25341.4
 std=59
tflite: Inference timings in us: Init: 1473090, First inference: 26847,
 Warmup (avg): 25408.6, Inference (avg): 25341.4
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=96.3438
 overall=96.3438
```

3. **EXTERNAL (Neutron Delegate) delegate:**

```
tflite_BenchmarkModelActivity: Running TensorFlow Lite benchmark
 with args: --graph=/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
 --external_delegate_path=/data/app/~~7hqIBPzxsbmWkxgebRwn5w==/
org.tensorflow.lite.benchmark-xCpR4AIP1XFOyYtCDvZG3A==/lib/arm64/
libneutron_delegate.so
tflite: Log parameter values verbosely: [0]
tflite: Graph: [/data/local/tmp/mobilenet_v1_1.0_224_quant.tflite]
tflite: External delegate path: [/data/app/~~7hqIBPzxsbmWkxgebRwn5w==/
org.tensorflow.lite.benchmark-xCpR4AIP1XFOyYtCDvZG3A==/lib/arm64/
libneutron_delegate.so]
tflite: Loaded model /data/local/tmp/mobilenet_v1_1.0_224_quant.tflite
tflite: Initialized TensorFlow Lite runtime.
tflite: EXTERNAL delegate created.
tflite: NeutronDelegate delegate: 30 nodes delegated out of 31 nodes with 1
 partitions.
tflite: Replacing 30 out of 31 node(s) with delegate (NeutronDelegate) node,
 yielding 2 partitions for the whole graph.
tflite: Explicitly applied EXTERNAL delegate, and the model graph will be
 partially executed by the delegate w/ 1 delegate kernels.
tflite: Created TensorFlow Lite XNNPACK delegate for CPU.
tflite: The input model file size (MB): 4.27635
tflite: Initialized session in 164623ms.
tflite: Running benchmark for at least 1 iterations and at least 0.5 seconds
 but terminate if exceeding 150 seconds.
tflite: count=267 first=2093 curr=1873 min=1665 max=2093 avg=1840.1 std=43
tflite: Running benchmark for at least 50 iterations and at least 1 seconds
 but terminate if exceeding 150 seconds.
tflite: count=532 first=1857 curr=1840 min=1760 max=2221 avg=1842 std=43
tflite: Inference timings in us: Init: 164622883, First inference: 2093,
 Warmup (avg): 1840.1, Inference (avg): 1842
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=84.1523
 overall=84.1523
```

5.5 Per Channel Quantized (PCQ) model enablement for i.MX 8MP

To enable optimal execution of per channel quantized (PCQ) models, additional configuration is required by the NPU on i.MX 8MP. From an *adb shell*, run the commands below:

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

It is possible to validate the properties by using *getprop*. Without these properties, a significant performance degradation is expected.

5.6 Hardware accelerators warmup time for i.MX 8MP

For TensorFlow Lite, the initial execution of model inference takes longer time, because of the model graph initialization needed by the GPU/NPU hardware accelerator. The initialization phase is known as warmup. This time duration can be decreased for a subsequent application that runs on the i.MX 8MP target. This is achieved by storing on disk the information resulting from the initial OpenVX graph processing. Set the following environment variables for this purpose:

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

By setting up these variables, the result of the OpenVX graph compilation is stored on disk as network binary graph files (*.nb). The runtime performs a quick hash check on the network and if it matches the *.nb file hash, it loads it into the NPU memory directly.

6 Acronyms

Table 3 lists the acronyms used in this document.

Table 3. Acronyms

Term	Description
ADB	Android Debug Bridge
API	Application Programming Interface
AOSP	Android Open Source Project
BSP	Board support package
MCU	Micro Controller Unit
ML	Machine Learning
NNAPI	Neural Networks API
NDK	Native Development Kit
NPU	Neural Processing Unit
GPU	Graphics Processing Unit
SELinux	Secure Linux
UUU	Universal Update Utility

7 References

Refer to the below documents for more information:

- *eIQ Toolkit User Guide* (*https://www.nxp.com/webapp/Download?colCode=AN13838*)
- *Android User's Guide* (*https://www.nxp.com/docs/en/user-guide/AUG.pdf*)
- *Universal Update Utility (UUU) Tool*: https://community.nxp.com/t5/Connects-Training-Material/Universal-Update-Utility-UUU-Tool/ta-p/1110422

For more resources, refer to the below URLs:

- https://www.nxp.com/design/design-center/software/eiq-ml-development-environment/eiq-toolkit-for-end-to-end-model-development-and-deployment:EIQ-TOOLKIT#documentation
- https://www.nxp.com/design/design-center/software/embedded-software/i-mx-software/android-os-for-i-mx-applications-processors:IMXANDROID#documentation
- https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-9-processors/i-mx-95-applications-processor-family-high-performance-safety-enabled-platform-with-eiq-neutron-npu:iMX95#documentation
- https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-8-applications-processors/i-mx-8m-plus-arm-cortex-a53-machine-learning-vision-multimedia-and-industrial-iot:IMX8MPLUS

Note: *Some of the documents available on these URLs might not be accessible. Contact your local NXP field applications engineer (FAE) or sales representative for obtaining this document.*

8 Note about the source code in the document

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

9 Revision history

Table 4 summarizes the revisions to this document.

Table 4. Document revision history

Document ID	Release date	Description
AN14411 v.1.0	23 September 2024	Initial public release

Legal information

Contents

Date of release: 23 September 2024
Document identifier: AN14411