UG10166

i.MX Machine Learning User's Guide

Rev. LF6.12.49_2.2.0 — 23 January 2026 **User guide**

Document information

Information	Content
Keywords	i.MX, Linux, Machine Learning, UG10166, LF6.12.49_2.2.0
Abstract	The NXP eIQ Machine Learning Software Development Environment (hereinafter referred to as "NXP eIQ") provides a set of libraries and development tools for machine learning applications targeting NXP microcontrollers and applications processors.

1 Software Stack Introduction

The NXP eIQ Machine Learning Software Development Environment (hereinafter referred to as "NXP eIQ") provides a set of libraries and development tools for machine learning applications targeting NXP microcontrollers and application processors. The NXP eIQ is contained in the *meta-imx/meta-ml* Yocto layer. See also the *i.MX Yocto Project User's Guide* (UG10164) for more information.

The following four inference engines are currently supported in the NXP eIQ software stack: TensorFlow Lite, ONNX Runtime, PyTorch, and OpenCV. The following figure shows the supported eIQ inference engines across the computing units.

NXP eIQ inference engines and libraries	eIQ inference engine deployment										
	PyTorch	ONNX RUNTIME	TensorFlowLite	OpenCV	LiteRT	ONNX RUNTIME	TensorFlowLite	LiteRT	ONNX RUNTIME	TensorFlowLite	LiteRT
Compute engines	Cortex-A					GPU			NPU		
i.MX 8M Plus	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
i.MX 8QuadMax	✓	✓	✓	✓	✓	✓	✓	✓	NA	NA	NA
i.MX 8QuadXPlus	✓	✓	✓	✓	✓	✓	✓	✓	NA	NA	NA
i.MX 8M Quad, Nano	✓	✓	✓	✓	✓	✓	✓	✓	NA	NA	NA
i.MX 8M Mini, 8ULP	✓	✓	✓	✓	✓	NA	NA	NA	NA	NA	NA
i.MX 91	✓	✓	✓	✓	✓	NA	NA	NA	NA	NA	NA
i.MX 93	✓	✓	✓	✓	✓	NA	NA	NA	NA	✓	✓
i.MX 943	✓	✓	✓	✓	✓	NA	NA	NA	NA	✓	✓
i.MX 95	✓	✓	✓	✓	✓	NA	✓	✓	✓	✓	✓

✓ Supported NA (Not applicable)

aaa-056272

Figure 1. NXP eIQ supported compute vs. inference engines

The NXP eIQ inference engines support multi-threaded execution on Cortex-A cores. Additionally, TensorFlow Lite also supports acceleration on the GPU or NPU. Generally, the NXP eIQ is prepared to support the following key application domains:

- **Vision**
  - Multi-camera observation
  - Active object recognition
  - Gesture control
- **Voice**
  - Voice processing
  - Home entertainment
- **Sound**
  - Smart sense and control
  - Visual inspection

  - Sound monitoring
- LLM
  - Text Generation
  - ASR

2 TensorFlow Lite

TensorFlow Lite, recently renamed to LiteRT, is an open-source software library focused on running machine learning models on mobile and embedded devices (available at http://www.tensorflow.org/lite). It enables on-device machine learning inference with low latency and small binary size. TensorFlow Lite also supports hardware acceleration:

- Using the VX Delegate on i.MX 8 series.
- Using the Ethos-U Delegate on i.MX 93.
- Using the Neutron Delegate for Neutron-S devices.
  Note: *Neutron-S devices are i.MX 9 SoC with Neutron-S inside. Currently, it is i.MX 95 and i.MX 943.*
- Using the GPU delegate for i.MX 95 Mali GPU.

The TensorFlow Lite source code for this Yocto Linux release is available at this repository, branch lf-6.12.49_2.2.0. This repository is a fork of the mainline https://github.com/tensorflow/tensorflow, and it is optimized for NXP i.MX 8 and i.MX 9 platforms.

Features:

- TensorFlow Lite v2.19.0
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores
- Parallel computation using GPU/NPU hardware acceleration (on shader or convolution units)
- C++ and Python API (supported Python version 3)
- Per-tensor and Per-channel quantized models support

2.1 TensorFlow Lite software stack

The TensorFlow Lite software stack is shown in the following picture. The TensorFlow Lite supports computation on the following hardware units:

- CPU Arm Cortex-A cores
- GPU/NPU hardware accelerator using the VX Delegate on i.MX 8 Series. See Section 7.1 for details.
- NPU hardware acceleration using Ethos-U Delegate on i.MX 93 NPU. See Section 7.2 for details.
- NPU hardware acceleration using Neutron Delegate on i.MX 9 series with Neutron NPU. See Section 7.4 for details.
- GPU hardware acceleration using the GPU Delegate on i.MX 95 GPU

See Section 1 for some details about supporting of computation on GPU/NPU hardware accelerator on different hardware platforms.

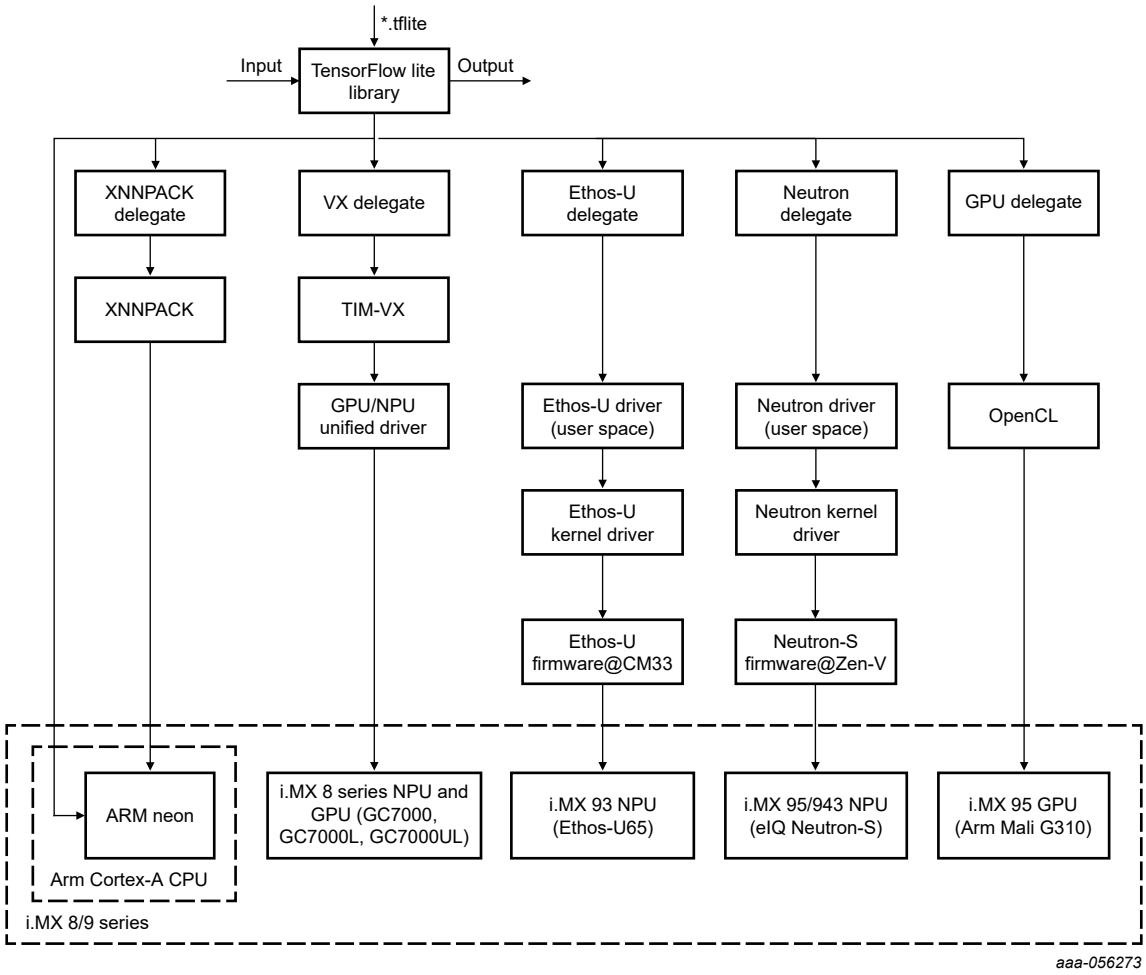

Figure 2. TensorFlow Lite software stack

Note:

The first execution of the model inference using the delegate takes longer, because of the time required for computational graph compilation and initialization for the hardware accelerator. The following iterations perform much faster. The computational graph is the representation of the operations and their dependencies to perform computation specified by the model. The computation graph is built during the model parsing phase. See Section 7 for details.

The VX Delegate implementations use the OpenVX library for computational graph execution on the GPU/NPU hardware accelerator. Therefore, OpenVX library support must be available for the selected device to be able to use the acceleration. For more details on the OpenVX library availability, see the i.MX Graphics User's Guide (UG10159).

Refer to the i.MX Graphics User's Guide (UG10159) for list GPUs with OpenVX support. Note that the GC7000 Lite and GC7000 Ultra Lite GPUs does not support full OpenVX however still capable to run ML workload.

2.2 Inference backends and delegates

Inference backend is the compute engine that enables efficient execution of machine learning models on edge devices. Tensorflow Lite comes with the options to enable different backends through the delegate mechanism.

2.2.1 Built-in kernels

Default inference backend is the CPU with reference kernels from TensorFlow Lite implementation. Built-in kernels provide full support for TensorFlow Lite operator set.

The built-in kernels are built with RUY matrix multiplication library enabled, which increases the performance of the kernels for floating point and quantized operations.

2.2.2 XNNPACK Delegate

XNNPACK library is a highly optimized library of floating-point quantize neural network inference operators for ARM, WebAssembly, and x86 platforms. The XNNPACK library is available through XNNPACK delegate in TensorFlow Lite. The XNNPACK delegate computation is performed on the CPU.

It provides optimized implementation for a subset of TensorFlow Lite operator set. In general, it provides better performance than the built-in kernels.

Note: *The models are executed via the XNNPACK Delegate by default. XNNPack now also supports quantized operators (since 2021).*

2.2.3 VX Delegate

VX Delegate enables accelerating the inference on on-chip hardware accelerator on i.MX 8 series. The VX Delegate directly uses the hardware accelerator driver (OpenVX with extension) to fully utilize the accelerator capabilities.

The VX Delegate is available as *external delegate*[1]. The corresponding library is available in `/usr/lib/libvx_delegate.so`.

VX Delegate is supported in both C++ and Python API. For using VX Delegate (or any external delegate), see the external_delegate_provider implementation in C++ and/or label_image.py for Python. List of supported operators are available in op_status.md.

2.2.4 Ethos-U Delegate

Ethos-U Delegate is an external delegate on i.MX 93 Linux platforms. It enables accelerating the inference on the on-chip hardware accelerator. The Ethos-U Delegate directly uses the hardware accelerator driver (Ethos-U driver stack) to fully utilize the accelerator capabilities.

The Ethos-U Delegate is available as *external delegate*. The corresponding library is available in `/usr/lib/libethosu_delegate.so`.

Ethos-U Delegate is supported in both C++ and Python API. For using Ethos-U Delegate (or any external delegate), see the `external_delegate_provider` implementation in C++ and/or `label_image.py` for Python. List of supported operators are available in SUPPORTED_OPS.md.

2.2.5 Neutron Delegate

Neutron Delegate is an external delegate on i.MX 9 series Linux platform containing Neutron-S NPU. It captures the operators and aggregates them as a neutron graph node, which can be directly offloaded and accelerated by the Neutron-S NPU.

The delegate library is available in `/usr/lib/libneutron_delegate.so`. It can be used in both C++ and Python API environments. For using Neutron Delegate, see the `external_delegate_provider` implementation in C++ and/or `label_image.py` for Python usage.

1 An external delegate is a special Tensorflow Lite delegate that is simply initialized from loading a dynamic library which encapsulates an actual TensorFlow Lite delegate implementation

Note:

For the offline compilation, the model should be converted through the eIQ toolkit first. In the converted model, the neutronGraph node is already generated. The neutron-delegate only captures the neutronGraph node and offloads the work to Neutron-S.

2.2.6 GPU Delegate

GPU Delegate is an internal TensorFlow Lite delegate enabled on the i.MX 95 platform to leverage inferences through the Arm Mali G310 GPU.

The GPU Delegate supports OpenCL as the main backend, and applies a set of optimizations such as accuracy lowering for performance (FP32 to FP16 lowering) and allows quantize model execution. Further information about the GPU delegate can be found in https://www.tensorflow.org/lite/performance/gpu.

GPU delegate is supported in C++ API. Refer to `tensorflow-imx/tensorflow/lite/tools/delegates/gpu_delegate_provider.cc` under the branch `lf-6.12.49_2.2.0` on the GitHub page https://github.com/nxp-imx/tensorflow-imx.

Note:

- *Even if the GPU Delegate supports quantized models, performance might be degraded when compared to other delegates such as XNNPACK using 6 cores.*
- *As the GPU delegate dynamically loads OpenCL, `libOpenCL.so` is expected to be present in the system. If `libOpenCL.so` is missing but `libOpenCL.so.1` exists, issues can be solved by adding a symbolic link:*

```
ln -s /usr/lib/libOpenCL.so.1 /usr/lib/libOpenCL.so
```

2.3 Delivery package

The TensorFlow Lite is available using Yocto Project recipes.

The TensorFlow Lite delivery package contains:

- TensorFlow Lite shared libraries
- TensorFlow Lite header files
- Python Module for TensorFlow Lite
- Image classification example application for C++ (`label_image`) and for Python (`label_image.py`)
- TensorFlow Lite benchmark application (`benchmark_model`)
- TensorFlow Lite evaluation tools (`coco_object_detection_run_eval`, `imagenet_image_classification_run_eval`, `inference_diff_run_eval`), see TensorFlow Lite Delegates for details.

For application development, the TensorFlow Lite shared libraries and header files are available in the SDK. See Section 2.5 for more details.

There are following delegates available in the TensorFlow Lite delivery package:

- XNNPACK Delegate
- VX Delegate
- Ethos-U Delegate
- Neutron Delegate
- GPU Delegate

2.4 Build details

TensorFlow Lite uses CMake build system for compilation. Notable remarks to package build are:

- RUY matrix multiplication library is enabled (`TFLITE_ENABLE_RUY=On`). RUY matrix multiplication library offers better performance compared to kernels build with Eigen and GEMLOWP.
- XNNPACK Delegate support (`TFLITE_ENABLE_XNNPACK=On`)
- External Delegate support (`TFLITE_ENABLE_EXTERNAL_DELEGATE=On`)
- (i.MX 95) GPU Delegate support (`TFLITE_ENABLE_GPU=On`)
- The runtime library is built and provided as a shared library (`TFLITE_BUILD_SHARED_LIB=On`). If static linking of the TensorFlow Lite library to the application is preferred, keep this switch in off state (default settings). This might be convenient if the application is built with CMake as described in the Section Section 2.5.1.
- The package is compiled with the default -O2 optimization level. Some CPU kernels, such as RESIZE_BILINEAR, are known to perform better with -O3 optimization level. However, some performs better with -O2, such as ARG_MAX. We recommend to adjust the optimization level, based on the application needs.

Yocto project builds the TensorFlow Lite with these settings. The build configuration can be changed by either updating the TensorFlow Lite Yocto recipe in the meta-imx layer (located in `meta-imx/meta-ml/recipes-libraries/tensorflow-lite/`), or building the TensorFlow Lite from source code using the CMake and the Yocto SDK.

2.5 Application development

This section describes how to use TensorFlow Lite C++ API in the application development.

To start with TensorFlow Lite C++ application development, a Yocto SDK must be generated firstly. See the *i.MX Yocto Project User's Guide* (UG10164) for detailed information on how to generate Yocto SDK environment for cross-compiling.

To build an application that uses the TensorFlow Lite, use the following options:

- Create a CMake project that uses TensorFlow Lite (CMake superbuild pattern).
- Use the Yocto SDK precompiled libraries.

The CMake configuration file of TensorFlow Lite is under `tensorflow/lite/CMakeLists.txt` from the root repository.

2.5.1 Create CMake project which uses TensorFlow Lite

The recommended way is to create a CMake project, which uses TensorFlow Lite as described in Build TensorFlow Lite with CMake. CMake takes care of dependencies preparation, including download, configure and build steps.

To demonstrate this build option, there is a minimal example project available in `tensorflow/lite/examples/minimal`. To build it:

1. Build the Native flat compiler for TensorFlow Lite.

```
cmake -S tensorflow/lite/tools/cmake/native_tools/flatbuffers \
      -B native-tools \
      -DCMAKE_INSTALL_PREFIX=native-tools
cmake --build native-tools -- -j 4 --keep-going
cmake --install native-tools
```

2. Set up the Yocto SDK as described above. To activate this Yocto SDK environment on your host, use the following command:

```
$ source <Yocto_SDK_install_folder>/environment-setup-aarch64-poky-linux
```

3. Configure the project using CMake:

```
$ mkdir build-minimal-example; cd build-minimal-example
$ cmake -DCMAKE_TOOLCHAIN_FILE=${OE_CMAKE_TOOLCHAIN_FILE} -
DTFLITE_ENABLE_XNNPACK=on \
-DTFLITE_ENABLE_RUY=on \
-DTFLITE_HOST_TOOLS_DIR=native-tools
../tensorflow/lite/examples/minimal
```

4. Build the project:

```
$ cmake --build . -j4
```

5. The minimal example is available in the build directory:

```
$ file minimal
minimal: ELF 64-bit LSB shared object, ARM aarch64, version 1 (GNU/
Linux), dynamically linked, interpreter /lib/ld-linux-aarch64.so.1,
 BuildID[sha1]=4a928894439e0b33217ea28790378690ab4ce7cd, for GNU/Linux
 3.14.0, with debug_info, not stripped
```

6. Optionally you can strip the final binary:

```
$ $STRIP --remove-section=.comment --remove-section=.note --strip-unneeded
 <file>
```

This build option has several advantages:

- Automatic dependency resolution based on configure options
- Option to choose between static or dynamic linking (`TFLITE_BUILD_SHARED_LIB=on/off`)
- Building the whole project (including its dependencies) in the Debug mode (`CMAKE_BUILD_TYPE=Debug/Release/…`), for enhanced debugging experience

2.5.2 Using Yocto SDK precompiled libraries

Another option is to use the precompiled binaries and header files which are directly available in the Yocto SDK. The TensorFlow Lite artifacts are in the Yocto SDK as follows:

- TensorFlow Lite shared library (libtensorflow-lite.so) in `/usr/lib`
- TensorFlow Lite header files in `/usr/include`

Note: *Not all TensorFlow Lite dependencies are installed in the Yocto SDK and it is necessary to download and optionally build them manually. For the required versions see the `tensorflow/lite/tools/cmake/modules/` folder.*

To build the image classification demo (label_image), located in `tensorflow/lite/examples/label_image/`, follow these steps:

1. Create build directory:

```
$ mkdir build-manual
$ cd build-manual
```

2. Download the Abseil library dependency:

```
$ wget https://github.com/abseil/abseil-cpp/
archive/997aaf3a28308eba1b9156aa35ab7bca9688e9f6.tar.gz -O abseil-cpp.tar.gz
$ tar -xzf abseil-cpp.tar.gz
$ mv abseil-cpp-997aaf3a28308eba1b9156aa35ab7bca9688e9f6 abseil-cpp
```

3. Build the label_image example:

```
$ $CC ../tensorflow/lite/examples/label_image/label_image.cc ../tensorflow/
lite/examples/label_image/bitmap_helpers.cc ../tensorflow/lite/tools/
evaluation/utils.cc ../tensorflow/lite/tools/delegates/delegate_provider.cc -
Iabseil-cpp -O2 -ltensorflow-lite -lstdc++ -lpthread -lm -ldl -lrt -I../
```

2.6 Enabling TensorFlow Operators in TensorFlow Lite Runtime

The TensorFlow Lite Operator Set counts more than a hundred of frequently used operators and layers, and majority of Machine Learning models can fit into it. Still the TensorFlow Lite Operator Set is only a subset of TensorFlow Operator Set, so not every model is convertible.

To tackle this limitation, TensorFlow offers an option to use TensorFlow operators inside the TensorFlow Lite runtime. See https://www.tensorflow.org/lite/guide/ops_select. It shows how this feature can be used with NXP i.MX devices with Yocto Linux platform.

2.6.1 TensorFlow and TensorFlow Lite Operator Set

If the model is not convertible within the standard TensorFlow Lite Operator Set, the TensorFlow Lite converter raises an error, indicating particular operator is not available in TensorFlow Lite, for example:

```
Some ops are not supported by the native TFLite runtime, you can enable TF
 kernels fallback using TF Select. See instructions: https://www.tensorflow.org/
lite/guide/ops_select
TF Select ops: Roll
Details: tf.Roll(tensor<?x10xf32>, tensor<i32>, tensor<i32>) -> (tensor<?
x10xf32>) : {device = ""}
```

To Convert such a model, the **Select TensorFlow Operators** feature needs to be enabled in the Converter:

```
converter.target_spec.supported_ops = [
     tf.lite.OpsSet.SELECT_TF_OPS  # enable TensorFlow ops.
]
```

When the model is converted with `SELECT_TF_OPS` enabled, TensorFlow operators are transformed into Flex operators, which are supported by TensorFlow Lite through the Flex Delegate. The Flex Delegate is the TensorFlow Lite counterpart for the **Select TensorFlow Operators** feature and bridges the TensorFlow Lite and TensorFlow runtimes.

2.6.2 Building the TensorFlow Lite Library with the Flex Delegate for i.MX Linux platforms

The library can be built using `bazel` on any supported host or inside the Docker container. It is recommend to use Docker because the environment is ready for TensorFlow compilation. Compilation outside of Docker might fail for multiple reasons. This document focuses on building the TensorFlow Lite Library with Flex Delegate inside the TensorFlow's Docker image.

Note:

To build the library outside the Docker image, the `bazel` build system needs to be installed on the machine. The TensorFlow requires an exact version of `bazel`, which is specific to particular TensorFlow version. Therefore, use `bazelisk` to handle the `bazel` version management. Find the `bazelisk` tool on its GitHub space https://github.com/bazelbuild/bazelisk, with prebuilt executables for multiple platforms available.

It is recommended to have at least 32 GB RAM to build the Flex Delegate, and ensure that enough inodes are available in build-related directories, such as `/tmp`.

2.6.2.1 Checking out the TensorFlow repository

To build the TensorFlow Lite library, check out the TensorFlow sources:

Clone the `tensorflow-imx` repository from https://github.com/nxp-imx/tensorflow-imx and check out the corresponding release branch:

```
$ git clone https://github.com/nxp-imx/tensorflow-imx.git -b lf-6.12.49_2.2.0
$ cd tensorflow-imx
```

2.6.2.2 Setting up Docker VM

For more details about the Docker VM setup for TensorFlow, see https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/tf_sig_build_dockerfiles.

Note:

Depending on the host, the Docker may require administrative privileges to run (e.g., `sudo` in Linux). Alternatively, the Docker Daemon can run as a non-root user (Rootless mode), as described here https://docs.docker.com/engine/security/rootless/.

1. Download the `tensorflow/build:2.18-python3.12` Docker image. This image is aligned with the upstream release and contains all the required tools to build Flex Delegate related components.

   ```
   $ docker pull tensorflow/build:2.18-python3.12
   ```

2. Run the Docker VM. During the build process, Bazel downloads various packages from the Internet. Therefore, Internet access inside the instantiated container is required. In case of conflict, a minimal setup is to initialize `http_proxy` and `https_proxy` environmental variables when launching the Docker image. Particular steps depend on the host configuration.

   ```
   $ docker run  -e "http_proxy=<your-http-proxy>" \
                 -e "https_proxy=<your-https-proxy>" \
                 -e "no_proxy=localhost,127.0.0.1" \
                 -it -w /tensorflow -v /<path-to-tensorflow-sources>:/tensorflow
    \
                 -e HOST_PERMS="\\((id -u):\\)(id -g)" \
                 tensorflow/build:2.18-python3.12
   ```

2.6.2.3 Building the TensorFlow Lite with Flex Delegate

The main CPUs in NXP i.MX 8 and i.MX 9 families are Arm based, implementing AArch64 ISA, and the platform OS is based in the Embedded Linux BSP release. To build TensorFlow Lite with Flex Delegate, the build system supports the described environment combination through the `elinux_aarch64` option.

1. Configure the project using the configure script:

   ```
   $ ./configure
   ```

   Note: *More details about the configuration can be found in `build_benchmark_flex_test.sh` and `build_elinux_flex_libs_from_models.sh` located in the `tensorflow/lite/tools` directory.*
   The Flex Delegate sources and `bazel` build recipes are located in `/tensorflow/lite/delegates/flex`. There are two libraries defined:
   - `tensorflowlite_flex_[full|reduced]` – TensorFlow Lite Flex Delegate shared library (`libtensorflowlite_flex.so`)
   - `delegate_[full|reduced]` – special target to be used for static linking of the TensorFlow Lite Flex Delegate. Similar to object library concept in CMake.

Additionally Bazel targets for building different variants of the `benchmark_model` binary are provided in the `/tensorflow/lite/delegates/flex/test/BUILD`, for evaluation purposes:

```
benchmark_model_plus_flex_[dynamic]_[full|reduced]
```

2. Build the `benchmark_model_plus_flex` target with a full TensorFlow Operator Set:

```
$ bazel --output_base=/tensorflow/docker-build/ build --config=monolithic
 --config=elinux_aarch64 -c opt  --cxxopt='--std=c++17' --
host_crosstool_top=@bazel_tools//tools/cpp:toolchain //tensorflow/lite/
delegates/flex/test:benchmark_model_plus_flex_full
```

Note: *To preserve the Bazel's cache, use the `--output_base` switch to override the default output base. For the build outside the Docker, this switch can be omitted. The directory shall be available prior to running Bazel build.*

The output is the `benchmark_model_plus_flex` binary with statically linked Flex Delegate. This can be directly used on NXP MPU platforms.

The `-c` (or `--compilation_mode`) affects code generation option. It can be set to `fastbuild`, `dbg` or `opt`. See https://bazel.build/docs/user-manual#build-semantics. To build the Flex Delegate for debugging purposes, use the `-c dbg` option.

The following table lists the `benchmark_model_plus_flex_*` build configurations.

Table 1. benchmark_model build configurations

Operator set	Static linkage	Dynamic linkage
Full Flex Delegate	`benchmark_model_plus_flex_full`	`benchmark_model_plus_flex_dynamic_full`
Reduced Flex Delegate	`benchmark_model_plus_flex_reduced`	`benchmark_model_plus_flex_dynamic_reduced`

2.6.3 Reducing the size of the Flex Delegate library

The previous section describes how to build a TensorFlow Lite Library with a complete TensorFlow Operator Set. The approach is useful for quick evaluation, but for practical use, it generates an oversized binary. Moreover, typically only a small subset of TensorFlow operators are required.

To minimize the size, there is a model-dependent build option which extracts the required operators from the models, and selectively includes them in the deployed TensorFlow Lite library. For more details, see https://www.tensorflow.org/lite/guide/reduce_binary_size.

For example, there are targets with the `_reduced` suffix, which builds the TensorFlow Lite library for the `/tensorflow/lite/delegates/flex/test/simple_flex_model_int8.tflite` example model. The model contains a single TensorFlow operation: `tf.roll()`.

The reduced size binaries are built in the following process:

1. Generate the headers and listings for the Flex operators contained in the provided TensorFlow Lite models. This is managed by the rule `fetch_flex_files`. Note that the command is "run" and executed natively in the host platform (lack of the `elinux_aarch64` option).
2. Build the target Flex delegate (using the rules contained in Table 1).

```
$ bazel --output_base=/tensorflow/docker-build/ run //tensorflow/lite/delegates/
flex/test:fetch_flex_files
$ bazel --output_base=/tensorflow/docker-build/ build --config=monolithic
 --config=elinux_aarch64 -c opt  --cxxopt='--std=c++17' --
host_crosstool_top=@bazel_tools//tools/cpp:toolchain
```

```
//tensorflow/lite/delegates/flex/test:benchmark_model_plus_flex_reduced
```

To build the TensorFlow Lite library for custom model, use a bazel function `tflite_flex_cc_library` (for static library) or `tflite_flex_shared_library` (for shared library), and list the models into the `models` attribute. Remember regenerating the headers and listings through the `fetch_flex_files` rule for each additional model containing Flex operators. The `kernel_headers` attribute uses the files generated by the `fetch_flex_files` rule through the `load_flex_kernel_header` rule. For more details, see the `tensorflow/lite/delegates/flex/test/BUILD` file.

```
tflite_flex_cc_library(
    name = "delegate_reduced",
    models = [
        "simple_flex_model_int8.tflite",
    ],
    visibility = ["//visibility:public"], kernel_headers = loaded_headers,
)
```

This library can be used inside the bazel to link to a custom TensorFlow Lite binary, like this:

```
tf_cc_binary(
    name = "benchmark_model_plus_flex_reduced",
    srcs = [
        "//tensorflow/lite/tools/benchmark:benchmark_plus_flex_main.cc",
    ],
    copts = tflite_copts() + tflite_copts_warnings(),
    linkopts = tflite_linkopts(),
    deps = [
        ":delegate_reduced",
        "//tensorflow/lite/tools/benchmark:benchmark_tflite_model_lib",
        "//tensorflow/lite/testing:init_tensorflow",
        "//tensorflow/lite/tools:logging",
    ],
)
```

Note: *To ease the build process for Reduced Size Flex Delegate containing user models, refer to `build_elinux_flex_libs_from_models.sh` located in the `tensorflow/lite/tools` directory.*

2.6.4 Flex Delegate deployment on NXP i.MX Linux platform

For the statically linked binary (in this usecase, `benchmark_model_plus_flex_[full|reduced]`), copy the binary to target the device rootfs.

For the dynamically linked binary (in this usecase, `benchmark_model_plus_flex_dynamic_[full|reduced]`), copy both `libtensorflowlite_flex.so` and the binary to target the device rootfs. The `libtensorflowlite_flex.so` should be copied to `/usr/lib/`, or alternatively the `LD_LIBRARY_PATH` should be set, to load the library by the dynamic linker or loader.

1. Copy the `simple_flex_model_int8.tflite` example model on the i.MX platform, e.g., to `/usr/bin/tensorflow-lite-2.19.0/examples/`.

   ```
   $ scp tensorflow/lite/delegates/flex/test/simple_flex_model_int8.tflite
    root@<imx-board>:/usr/bin/tensorflow-lite-2.19.0/examples/
   ```

2. Run the example application benchmark model:

   ```
   $ ./benchmark_model_plus_flex_dynamic_full --graph=/usr/bin/tensorflow-
   lite-2.19.0/examples/simple_flex_model_int8.tflite
   ```

With `--enable_op_profiling=true`, the FlexDelegate invocation is displayed:

```
============================== Summary by node type
 ==============================
           [Node type]    [count]    [avg ms]      [avg %]     [cdf %]   [mem
 KB]   [times called]
 TfLiteXNNPackDelegate          3       2.821      81.250%     81.250%
 0.000            3
    TfLiteFlexDelegate          1       0.640      18.433%     99.683%
 0.000            1
               RESHAPE          1       0.008       0.230%     99.914%
 0.000            1
               SOFTMAX          1       0.003       0.086%    100.000%
 0.000            1
```

2.6.5 Using hardware accelerators

The TensorFlow Operators are not part of the TensorFlow Lite Operators Set, so the hardware accelerator on i.MX platforms does not support these operators, though the acceleration of the TensorFlow Lite operators present in the model is supported.

For the hardware Acceleration on i.MX8 Linux platforms, use the VX Delegate (external delegate). The `benchmark_model_plus_flex` already includes support for external delegates, so the `–external_delegate_path` CLI option can be used for inference acceleration:

```
$ ./benchmark_model_plus_flex_dynamic_full --graph=/usr/bin/tensorflow-
lite-2.15.0/examples/simple_flex_model_int8.tflite --enable_op_profiling=true --
external_delegate_path=/usr/lib/libvx_delegate.so
```

For the hardware acceleration on the i.MX 9 Linux platform, use the Ethos-U Delegate for i.MX 93 or Neutron Delegate for i.MX 95.

Alternatively, convert the model with the Arm Vela Compiler as described in Section 7.2.3, and also use the Ethos-U Delegate.

```
$ vela /usr/bin/tensorflow-lite-2.19.0/examples/simple_flex_model_int8.tflite
```

Run `benchmark_model_plus_flex*` with the Ethos-u Delegate.

```
$ ./benchmark_model_plus_flex_dynamic_full --graph=/usr/bin/tensorflow-
lite-2.19.0/examples/simple_flex_model_int8.tflite --enable_op_profiling=true --
external_delegate_path=/usr/lib/libethosu_delegate.so
```

2.6.6 Flex Delegate limitations

The Flex Delegate has the following limitations:

CPU support only for TensorFlow Operators

Flex Delegate operators are not supported on the hardware accelerators of i.MX platforms. The TensorFlow Operators fall back to CPU. The acceleration of supported TensorFlow Lite Ops in the model is not impacted. The model can freely combine TensorFlow Lite and TensorFlow Operators. Supported TensorFlow Lite operators of the model will be accelerated.

2.7 Running image classification example

A Yocto Linux BSP image with machine learning layer included by default contains a simple pre-installed example called ‘label_image’ usable with image classification models. The example binary file is located at:

```
/usr/bin/tensorflow-lite-2.19.0/examples
```

Demo instructions:

To run the example with mobilenet model on the CPU, use the following command:

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

The output of a successful classification on the i.MX 8MPlus SoC for the 'grace_hopper.bmp' input image is as follows:

```
Loaded model mobilenet_v1_1.0_224_quant.tflite
resolved reporter
invoked
average time: 39.271 ms
0.780392: 653 military uniform
0.105882: 907 Windsor tie
0.0156863: 458 bow tie
0.0117647: 466 bulletproof vest
0.00784314: 835 suit
```

Note: *For floating point layers, the TensorFlow Lite uses XNNPACK delegated by default.*

2.7.1 Running the example on the i.MX 8 platform hardware accelerator

To run the example application on the CPU without using the XNNPACK delegate, use the `--use_xnnpack=false` switch.

To run the example with the same model on the GPU/NPU hardware accelerator, add `--external_delegate_path=/usr/lib/libvx_delegate.so` (for VX Delegate) command line argument. To differentiate between the 3D GPU and the NPU, use the `USE_GPU_INFERENCE` environmental variable. For example, to run the model accelerated on the NPU hardware using VX Delegate, use this command:

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

The output of the NPU acceleration on the i.MX 8MPlus processor is as follows:

```
INFO: Loaded model ./mobilenet_v1_1.0_224_quant.tflite
INFO: resolved reporter
Vx delegate: allowed_builtin_code set to 0.
Vx delegate: error_during_init set to 0.
Vx delegate: error_during_prepare set to 0.
Vx delegate: error_during_invoke set to 0.
EXTERNAL delegate created.
INFO: Applied EXTERNAL delegate.
W [HandleLayoutInfer:257]Op 18: default layout inference pass.
INFO: invoked
INFO: average time: 2.567 ms
INFO: 0.768627: 653 military uniform
```

```
INFO: 0.105882: 907 Windsor tie
INFO: 0.0196078: 458 bow tie
INFO: 0.0117647: 466 bulletproof vest
INFO: 0.00784314: 835 suit
```

2.7.2 Running the example on the i.MX 93 platform with Ethos-U

To use the hardware acceleration on i.MX 93, convert the model using the Vela compiler first, and run the model with the Ethos-U delegate. Alternatively, directly run the model with the Ethos-U delegate. The model is then converted in the delegate. For details, see Section 7.2.3.

To run the example with the model on the NPU hardware accelerator, add the `--external_delegate_path=/usr/lib/libethosu_delegate.so` command line argument. For example, to run the model accelerated on the NPU hardware using Ethos-U Delegate, use this command:

```
$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite
-i grace_hopper.bmp -l labels.txt --external_delegate_path=/usr/lib/
libethosu_delegate.so
```

2.7.3 Running the example on the i.MX 9 platform with Neutron-S

To use the hardware Acceleration on i.MX 9 with Neutron-S NPU, convert the model using the neutron-converter using the eIQ Toolkit. For details, see the eIQ Toolkit documentation.

- Run the example with option:

```
--external_delegate_path=/usr/lib/libneutron_delegate.so
```

- Run the Python example with option:

```
-e /usr/lib/libneutron_delegate.so
```

2.7.4 Running the Python example

Alternatively, the example using the TensorFlow Lite interpreter-only Python API can be run. The example file is located at:

```
/usr/bin/tensorflow-lite-2.19.0/examples
```

To run the example using the predefined command line arguments, use the following command:

```
$ python3 label_image.py
```

The output should be as follows:

```
Warm-up time: 159.1 ms
Inference time: 156.5 ms
0.878431: military uniform
0.027451: Windsor tie
0.011765: mortarboard
0.011765: bulletproof vest
0.007843: sax
```

The Python example supports external delegates also. The switch `--ext_delegate <PATH>` and `--ext_delegate_options <EXT_DELEGATE_OPTIONS>`, can be used to specify the external delegate library and optionally its arguments.

For example, to run the model accelerated on the NPU hardware using Neutron Delegate on i.MX 95, run this command:

```
$ python3 label_image.py --ext_delegate /usr/lib/libneutron_delegate.so
```

The output should be as follows:

```
Loading external delegate from /usr/lib/libneutron_delegate.so with args: {}
INFO: NeutronDelegate delegate: 29 nodes delegated out of 31 nodes with 1
 partitions.

Warm-up time: 1.9 ms
Inference time: 1.7 ms

0.850980: military uniform
0.058824: Windsor tie
0.011765: bulletproof vest
0.007843: bow tie
0.007843: mortarboard
```

2.7.5 Running the example on the i.MX 95 platform using GPU

To run `label_image` using the GPU, execute the following command:

```
$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite -i grace_hopper.bmp -l
 labels.txt --use_gpu=true --gpu_backend=cl --gpu_precision_loss_allowed=true --
gpu_experimental_enable_quant=true --gpu_inference_for_sustained_speed=false
```

2.8 Running benchmark applications

A Yocto Linux BSP image with machine learning layer included by default contains a pre-installed benchmarking application. It performs a simple TensorFlow Lite model inference and prints benchmarking information. The application binary file is located at:

```
/usr/bin/tensorflow-lite-2.19.0/examples
```

Benchmarking instructions are as follows:

To run the benchmark with computation on CPU, use the following command:

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite
```

You can optionally specify the number of threads with the `--num_threads=X` parameter to run the inference on multiple cores. For highest performance, set X to the number of cores available.

The output of the benchmarking application should be similar to:

```
STARTING!
Log parameter values verbosely: [0]
Graph: [mobilenet_v1_1.0_224_quant.tflite]
Loaded model mobilenet_v1_1.0_224_quant.tflite
Going to apply 0 delegates one after another.
The input model file size (MB): 4.27635
Initialized session in 3.051ms.
```

```
Running benchmark for at least 1 iterations and at least 0.5 seconds but
 terminate if exceeding 150 seconds.
count=4 first=160408 curr=155384 min=155384 max=160408 avg=156869 std=2076
Running benchmark for at least 50 iterations and at least 1 seconds but
 terminate if exceeding 150 seconds.
count=50 first=155586 curr=155424 min=155274 max=155622 avg=155443 std=81
Inference timings in us: Init: 3051, First inference: 160408, Warmup (avg):
 156869, Inference (avg): 155443
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.
Peak memory footprint (MB): init=4.49219 overall=10.6133
```

To run the inference without the XNNPACK delegate, add the `--use_xnnpack=false` switch.

To run the inference using the GPU/NPU hardware accelerator, use the `--external_delegate_path` switch:

- For VX Delegate on i.MX 8: `--external_delegate_path=/usr/lib/libvx_delegate.so`
- For Ethos-U Delegate on i.MX 93: `--external_delegate_path=/usr/lib/libethosu_delegate.so`
- For Neutron Delegate on i.MX 95: `--external_delegate_path=/usr/lib/libneutron_delegate.so`

To run the inference using the Arm Mali G310 GPU using the GPU Delegate on i.MX 95:

```
--use_gpu=true --gpu_backend=cl --gpu_precision_loss_allowed=true --
gpu_experimental_enable_quant=true --gpu_inference_for_sustained_speed=false
```

The output with GPU/NPU module acceleration enabled (for VX Delegate) should be similar to:

```
STARTING!
Log parameter values verbosely: [0]
Graph: [mobilenet_v1_1.0_224_quant.tflite]
External delegate path: [/usr/lib/libvx_delegate.so]
Loaded model mobilenet_v1_1.0_224_quant.tflite
Vx delegate: allowed_builtin_code set to 0.
Vx delegate: error_during_init set to 0.
Vx delegate: error_during_prepare set to 0.
Vx delegate: error_during_invoke set to 0.
EXTERNAL delegate created.
Going to apply 1 delegates one after another.
Explicitly applied EXTERNAL delegate, and the model graph will be completely
 executed by the delegate.
The input model file size (MB): 4.27635
Initialized session in 13.437ms.
Running benchmark for at least 1 iterations and at least 0.5 seconds but
 terminate if exceeding 150 seconds.
W [HandleLayoutInfer:257]Op 18: default layout inference pass.
count=1 curr=4586473
Running benchmark for at least 50 iterations and at least 1 seconds but
 terminate if exceeding 150 seconds.
count=398 first=2541 curr=2419 min=2419 max=2549 avg=2467.87 std=13
Inference timings in us: Init: 13437, First inference: 4586473, Warmup (avg):
 4.58647e+06, Inference (avg): 2467.87
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.
Peak memory footprint (MB): init=7.24609 overall=34.0117
```

The delegates are not required to support the full set of operators defined by the TensorFlow Lite runtime. If the model contains such an operation, which is not supported by the particular delegate, this operation execution falls back to CPU using the TensorFlow Lite reference kernels. This way the computational graph represented by the model gets divided into segments and each segment is executed . The graph segmentation or also called graph partitioning is the process, where the computational graph defined by the model is divided into smaller segments (or partitions) and each of them is executed via the delegate or on the CPU using reference kernels (CPU fallback), based on operation supported by the delegate.

The benchmark application is also useful to check the optional segmentation of the models if accelerated on GPU/NPU hardware accelerator. For this purpose, the combination of the `--enable_op_profiling=true` and `--max_delegated_partitions=<big number>` (e.g., 1000) options can be used.

Which generates detailed profiling information, such as:

```
Profiling Info for Benchmark Initialization:
=============================== Run Order ===============================
[node type]                  [start]   [first]   [avg ms]       [%]       [cdf%]
ModifyGraphWithDelegate        0.000     4.597      4.597   95.791%      95.791%
AllocateTensors                4.528     0.198      0.101    4.209%     100.000%
========================= Top by Computation Time =========================
[node type]                  [start]   [first]   [avg ms]       [%]       [cdf%]
ModifyGraphWithDelegate        0.000     4.597      4.597   95.791%      95.791%
AllocateTensors                4.528     0.198      0.101    4.209%     100.000%
Number of nodes executed: 2
============================ Summary by node type =============================
            [Node type] [count][avg ms] [avg %] [cdf %] [mem KB] [times called]
ModifyGraphWithDelegate       1   4.597 95.791% 95.791%  684.000             1
AllocateTensors               1   0.202  4.209% 100.000%   0.000             2
Timings (microseconds): count=1 curr=4799
Memory (bytes): count=0
2 nodes observed
Operator-wise Profiling Info for Regular Benchmark Runs:
=============================== Run Order ===============================
        [node type]    [start]    [first]    [avg ms]          [%]        [cdf%]
Vx Delegate      0.000     14.890      14.894     11.349%       11.349%
    RESIZE_BILINEAR     14.896      1.331       1.331      1.014%       12.363%
Vx Delegate     16.227      2.944       2.909      2.216%       14.579%
    RESIZE_BILINEAR     19.137      0.279       0.277      0.211%       14.790%
    RESIZE_BILINEAR     19.415     44.316      44.496     33.905%       48.695%
            ARG_MAX     63.912     67.438      67.332     51.305%      100.000%
========================= Top by Computation Time =========================
        [node type]     [start]   [first]   [avg ms]          [%]        [cdf%]
            ARG_MAX      63.912    67.438     67.332      51.305%       51.305%
    RESIZE_BILINEAR      19.415    44.316     44.496      33.905%       85.210%
Vx Delegate       0.000    14.890     14.894      11.349%       96.559%
Vx Delegate      16.227     2.944      2.909       2.216%       98.775%
    RESIZE_BILINEAR      14.896     1.331      1.331       1.014%       99.789%
    RESIZE_BILINEAR      19.137     0.279      0.277       0.211%      100.000%
Number of nodes executed: 6
============================ Summary by node type =============================
        [Node type] [count] [avg ms]  [avg %]   [cdf %] [mem KB] [times called]
            ARG_MAX       1   67.332  51.306%   51.306%    0.000             1
    RESIZE_BILINEAR       3   46.102  35.129%   86.435%    0.000             3
Vx Delegate       2   17.802  13.565%  100.000%    0.000             2
Timings (microseconds): count=8 first=131198 curr=130580 min=130580 max=132766 avg=131238
 std=616
Memory (bytes): count=0
6 nodes observed
```

Based on section “Number of nodes executed” in the output, it can be determined which part of the computation graph was executed on GPU/NPU hardware accelerator. Every node except Vx Delegate falls back to CPU. In the example above, the ARG_MAX and RESIZE_BILINEAR nodes fall back to CPU.

2.9 Post training quantization using TensorFlow Lite converter

TensorFlow offers several methods for model quantization:

- Post training quantization with TensorFlow Lite Converter
- Quantization aware training using Model Optimization Toolkits and TensorFlow Lite Converter
- Various other methods available in previous TensorFlow releases

Note:

The model quantization is also supported by the "eIQ Toolkit". See also eIQ Toolkit User's Guide (EIQTUG).

Covering all of them is beyond the scope of this documentation. This section describes the approach for the post training quantization using the TensorFlow Lite Converter.

The Converter is available as a part of standard TensorFlow desktop installation. It is used to convert and optionally quantize TensorFlow models into TensorFlow Lite model format (`*.tflite`). There are two options how to use the tool:

- The Python API (recommended)
- Command line script

The post training quantization using the Python API is described in this chapter. The documentation useful for model conversion and quantization is available here:

- Python API documentation: https://www.tensorflow.org/versions/r2.15/api_docs/python/tf/lite/TFLiteConverter
- Guide for model conversion: www.tensorflow.org/lite/convert
- Guide for model quantization: https://www.tensorflow.org/lite/performance/post_training_quantization
- Guide for model optimization: https://www.tensorflow.org/model_optimization

Note:

The guides on TensorFlow page usually covers the most up to date version of TensorFlow, which might be different from the version available in the NXP eIQ. To see what features are available, check the corresponding API for the specific version of the TensorFlow or TensorFlow Lite.

The current version of the TensorFlow Lite available in the NXP eIQ is 2.15.0. It is recommended to use the TensorFlow Lite converter from corresponding TensorFlow version. The TensorFlow Lite runtime should be compatible with models generated by previous version of TensorFlow Lite Converter, however this backward compatibility is not guaranteed. Usage of successive version of TensorFlow Lite converter shall be avoided.

The 2.15.0 version of the converter has the following properties:

- In the post training quantization regime, the per-channel quantization is the only option. The per-tensor quantization is available only in connection with quantization aware training.
- Input and output tensors quantization is supported by setting the required data type in `inference_input_type` and `inference_output_type`.
- TOCO or MLIR based conversions are available. This is controlled by the `experimental_new_converter` attribute. As TOCO is becoming obsolete, MLIR-based conversion is already set by default in the 2.15.0 version of the converter.
  MLIR converter uses dynamic tensor shapes, what means the batch size of the input tensor is unspecified. Dynamic tensor shapes are not supported by the GPU and NPU hardware accelerators and this shall be turned off. Standard installation of TensorFlow does not provide API to control the dynamic tensor shape feature, but can be deactivated in the tensorflow installation, as follows. Locate the `<python-install-dir>/site-packages/tensorflow/lite/python/lite.py` file and change the private method `TFLiteConverterBase._is_unknown_shapes_allowed(self)` to return False value, as follows:

```
def _is_unknown_shapes_allowed(self):
# Unknown dimensions are only allowed with the new converter.
# Return self.experimental_new_converter
```

```
# Disable unknown dimensions support.
return False
```

Note:
MLIR is a new NN compiler used by TensorFlow, which supports quantization. Before MLIR, quantization was performed by TOCO (or TOCO Converter), which is now obsolete. See https://www.tensorflow.org/api_docs/python/tf/compat/v1/lite/TocoConverter. For details about MLIR, see https://www.tensorflow.org/mlir.

Note:

Do not use the dynamic range method for models being run on NN accelerators (GPU or NPU). It converts only the weights to 8-bit integers, but retains the activations in fp32, which results in the inference running in fp32 with an additional overhead for data conversion. In fact, the inference is even slower compared to a fp32 model, because the conversion is done on the fly.

For the full-integer post training quantization, a representative dataset is needed. The proper choice of samples in representative dataset highly influences the accuracy of the final quantized model. The best practices for creating the representative dataset are:

- Use train samples for which the original floating points model has very good accuracy, based on metrics the model used (e.g., SoftMax score for classification models, IOU for object detection models, etc.).
- There shall be enough samples in representative dataset.
- The size of representative dataset and the specific samples available in it are considered as hyperparameters to tune, with respect of the required model accuracy.

2.10 TensorFlow Lite for Microcontrollers on Xtensa HiFi4 core

TensorFlow Lite for Microcontrollers (TFLM) is a lightweight re-implementation of the TensorFlow Lite library for microcontroller CPU cores and NN accelerators (like the Xtensa HiFi4 core on i.MX 8ULP or Arm Ethos-U on i.MX 93). Compared to TensorFlow Lite, it uses less memory, has no C/C++ library dependencies and uses only static memory allocation. On the other hand, the list of supported operators is more limited and optimized kernels are available only for Cortex-M and Xtensa cores or the ARM Ethos-U accelerator. The main purpose of TFLM on the i.MX platform is low-power applications.

To use TFLM on the Xtensa HiFi4 core, the DSP firmware has to be rebuilt with the TFLM library and a TensorFlow Lite model included. As the Xtensa HiFi4 core is also used for audio encoding/decoding, the TFLM library has to be wrapped into an Xtensa Audio Framework (XAF) component to allow simultaneous audio and model inference execution. Moreover, the XAF client/server protocol implements input and output buffer passing to and from the CPU core via the Linux XAF API. The DSP firmware and usage example source codes are available at https://github.com/NXP/imx-audio-framework. See the DSP User's Guide in the docs subfolder for information on toolchain setup and build instructions.

To build the DSP firmware with the TFLM library (after the toolchain is installed), use the following Makefile options:

```
make PLATF=imx8ulp TFLM=1 DSP_FIRMWARE
```

The command produces a hifi4_tflm_imx8ulp.bin file which has to be copied to the `/lib/firmware/imx/dsp` folder of the Yocto Linux BSP image.

To build the TFLM usage example for Linux, use the following Makefile options:

```
make PLATF=imx8ulp TFLM=1 UNIT_TEST
```

The command compiles the `unit_test/src/dsp_tflm_test.c` source file and produces a `dsp_tflm_test.out` binary executable file which demonstrates a simple keyword detection application processing a built-in static audio buffer with "yes" and "no" speech command data samples.

By default, TFLM included in the DSP firmware is compiled with reference kernel implementations due to licensing. To improve the library performance on Xtensa HiFi4 cores, the library has to be built with proprietary licensed optimized kernel implementations provided by Cadence at https://github.com/foss-xtensa/nnlib-hifi4 (see the license file in the GitHub repository). Add the `OPTIMIZED_KERNEL_DIR=xtensa` option into the `dsp_framework/tensorflow_lite_micro.inc` file to automatically download the Cadence library and build TFLM with the optimized kernels:

```
cd $(SRC_DIR)/tflite-micro && make -f tensorflow/lite/micro/tools/make/
Makefile TARGET=xtensa TARGET_ARCH=hifi4 OPTIMIZED_KERNEL_DIR=xtensa
 XTENSA_USE_LIBC=true microlite
```

A DSP firmware file with the same name as previously is produced, which has to be copied to the `/lib/firmware/imx/dsp` folder of the Yocto Linux BSP image.

3 ONNX Runtime

ONNX Runtime is an open-source inference engine to run ONNX models, which enables the acceleration of machine learning models across all of your deployment targets using a single set of API. Source codes are available at https://github.com/nxp-imx/onnxruntime-imx.

Note:

- *For the full list of the CPU supported operators, see the 'operator kernels' documentation section: OperatorKernels.*
- *If you encounter an error indicating that the hash value of Eigen has changed during the ONNX Runtime build, see the patch available here: onnxruntime.*

Features:

- ONNX Runtime 1.23.2.
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores provided by the CPU execution provider.
- Neutron NPU Execution provider on i.MX 95 as an experimental feature.
- VSINPU execution provider on i.MX 8 series NPU/GPU.
- C++ and Python API (supported Python version 3).
- ONNX Runtime 1.23.2 supports ONNX 1.18 and Opset version 23.
- Integrated Arm KleidiAI into ONNX Runtime/MLAS for enhanced performance on Arm architectures.
- Added support for MatMulNBits for CPU Execution provider and Neutron Execution provider, enabling matrix multiplication with weights quantized to 8 bits and 4 bits.
- Added support for CNN models for the Neutron Execution Provider.

3.1 ONNX Runtime software stack

The following figure shows the ONNX Runtime software stack.

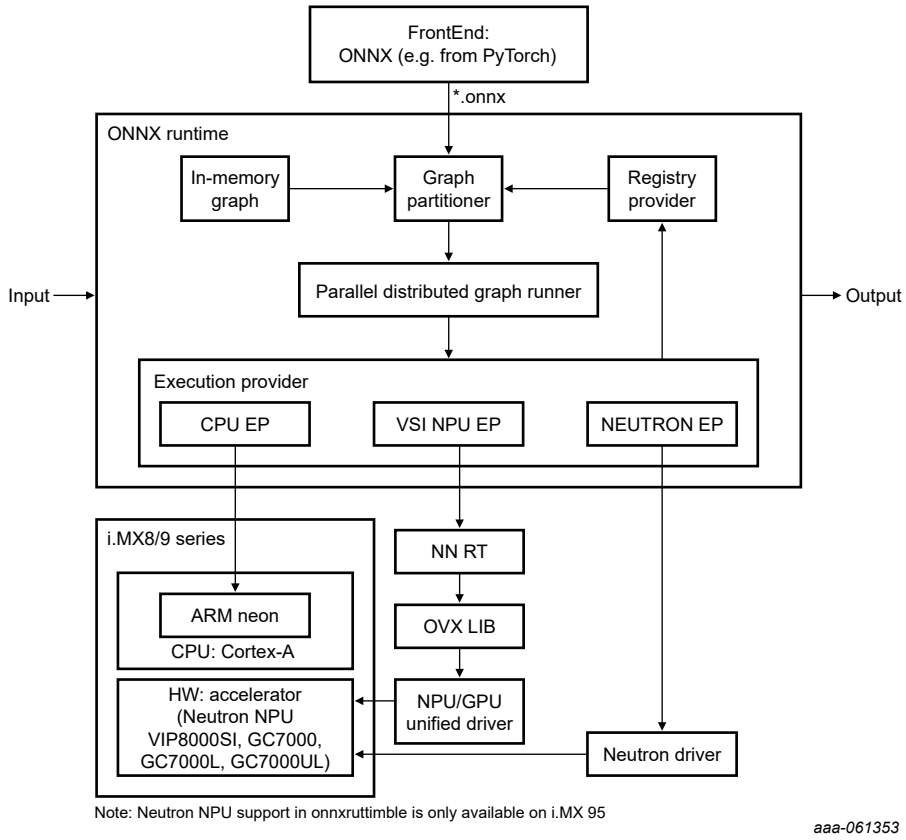

Figure 3. ONNX Runtime software stack

The following are the core components of ONNX Runtime:

Inference Session

The Inference Session is the central class that manages the entire execution pipeline. It loads models, applies optimizations, partitions the computation graph across available execution providers, and executes inference requests.

Key responsibilities:

- Loading ONNX models from files, memory, or streams
- Registering and managing execution providers
- Applying graph optimizations
- Managing memory allocation
- Executing inference requests
- Handling input/output binding

Graph and Model Representation

ONNX Runtime represents the machine learning model using several key abstractions:

- **Model**: Contains the computational graph and metadata.
- **Graph**: Represents the computation as a directed graph of operations.

Execution Providers

Execution Providers (EPs) are responsible for executing portions of the computation graph on specific hardware. ONNX Runtime partitions the graph based on the capabilities of the registered execution providers.

Key execution providers include:

- **CPU**: Default provider that runs on Arm Cortex-A cores.
- **Neutron**: Execute operations (Matmul operations) on Neutron NPU.
- **VSINPU**: Execute operations on VSINPU.

Note: *For Neutron NPU, now it is an experimental feature for the LLM model support and is only on the i.MX 95 NPU.*

3.2 ONNX model test

ONNX Runtime supports running various ONNX models, including CNNs and Large Language Models (LLMs).

ONNX models are available at https://github.com/onnx/models and consist of models and sample test data. Because some models require a lot of disk space, it is recommended to store the ONNX test files on a larger partition, as described in the SD card image flashing section.

The following sections provide examples of how to run CNN and LLM models in the target environment.

3.2.1 Running a CNN model

Note: *CNN model support with Neutron EP is an experimental feature on i.MX 95.*

Here is an example with the steps required to run the MobileNet version 2 test:

1. Download and unpack the MobileNet version 2 test archive to a folder, for example, to `/home/root`:

```
$ cd /home/root
$ wget https://github.com/onnx/models/raw/refs/heads/jfowers/bringup/archive/
vision/classification/mobilenet/model/mobilenetv2-12.tar.gz
$ tar -xzvf mobilenetv2-12.tar.gz
$ ls ./mobilenetv2-12
mobilenetv2-12.onnx test_data_set_0
```

2. Obtain `label_image_onnx.py` from `/usr/bin/onnxruntime-1.23.2`, and obtain `labels.txt` and `grace_hopper.bmp` from `/usr/bin/tensorflow-lite-2.19.0/examples`.
3. Run the `label_image_onnx.py` script with the `mobilenetv2-12.onnx` model, specifying the CPU execution provider.

```
$ python3 label_image_onnx.py -e cpu -l labels.txt -i grace_hopper.bmp -m
 mobilenetv2-12/mobilenetv2-12.onnx
0.439194: military uniform
0.143580: suit
0.111152: bow tie
0.094492: mortarboard
0.028997: Windsor tie
```

4. Run the `label_image_onnx.py` script with the `mobilenetv2-12.onnx` model, specifying the VsiNPU execution provider.

```
$ python3 label_image_onnx.py -e vsinpu -l labels.txt -i grace_hopper.bmp -m
 mobilenetv2-12/mobilenetv2-12.onnx
0.439194: military uniform
0.143581: suit
0.111152: bow tie
0.094492: mortarboard
0.028997: Windsor tie
```

5. Run the `label_image_onnx.py` script with the `mobilenetv2-12_neutron.onnx` model, specifying the Neutron execution provider.

ONNX Runtime now supports executing CNN models on the Neutron NPU as an **experimental feature**. To enable this, the model must be quantized and converted with the `onnx2neutron` tool (see README of the `onnx2neutron` tool from the eIQ Toolkit).
Then, the quantized Neutron model `mobilenetv2-12_neutron.onnx` is converted from `mobilenetv2-12.onnx`.

```
$ python3 label_image_onnx.py -e neutron -l labels.txt -i grace_hopper.bmp -m
 mobilenetv2-12_neutron.onnx
0.563028: military uniform
0.133023: suit
0.090539: mortarboard
0.055971: bow tie
0.034602: academic gown
```

3.2.2 Running an LLM model

ONNX Runtime now supports running Quantized Large Language Models (LLMs) using the CPU and Neutron NPU Execution provider. This enhancement enables efficient inference for LLMs with 8-bit and 4-bit quantization on selected i.MX SoCs.

ONNX Runtime now supports execution of Large Language Models (LLMs) using quantized formats on CPU by default. Building on this, it aslo supports running Quantized LLMs using the Neutron AI engine. This enhancement enables efficient inference for LLMs with 8-bit and 4-bit quantization on selected i.MX SoCs by offloading key operator kernels to the Neutron AI engine, and delivering improved performance and power efficiency for edge AI deployments.

This hybrid execution model allows developers to leverage the full flexibility of ONNX Runtime for LLM inference, while benefiting from hardware acceleration on supported operators.

Here is an example with steps required to run an LLM model on i.MX 943 with CPU as an execution provider:

1. Download and unpack the LLM compressed files to a folder, for example, to `/home/root`:

```
$ mkdir qwen2_5_0_5B_W4GS32
$ tar -xzvf qwen2_5_0_5B_W4GS32.tar.gz -C qwen2_5_0_5B_W4GS32
$ ls ./qwen2_5_0_5B_W4GS32
added_tokens.json  config.json  model_dyn.onnx  model_dyn.onnx.data
 special_tokens_map.json  tokenizer.json  tokenizer_config.json
```

2. To perform inference on an LLM model, install additional packages:

```
$ python3 -m pip install onnx
$ python3 -m pip install transformers
$ python3 -m pip install sentencepiece
```

3. Run the LLM model with CPU as the execution provider as follows:

```
$ python3 llm_chat_randominput.py -m qwen2_5_0_5B_W4GS32 -e cpu -p "Once upon
 a time"
```

If no error occurs, the generated text and the output text from the execution provider is printed as follows:

- Generated text: Once upon a time, a man was walking in a street. He saw a man who was sitting on a bench. The man was sitting on the bench, and he was looking at the man sitting on the bench.
- cpu_output: Once upon a time, a man was walking in a street. He saw a man who was sitting on a bench. The man was sitting on the bench, and he was looking at the man sitting on the bench.

To run the LLM model on i.MX 95 with Neutron as the execution provider:

1. Use the `imx95-19x19/15x5-evk-neutron.dtb` DTB to boot up the board.

2. (Optional) To speed up the preparation phase for the int4 LLM model, an offline convertion tool can be used.

```
$python3 /usr/lib/python3.13/site-packages/onnxruntime/tools/
convert_ort_models_to_neutron.py -i model.onnx
```

After that, a new model named `model_neutron.onnx` is generated. Use this one instead of the original model.

3. Run the following command:

```
$ python3 llm_chat_randominput.py -m qwen2_5_0_5B_W4GS32 -e neutron -p "Once
 upon a time"
```

Note: *Use `python3 llm_chat_randominput.py -h` for the full list of options available.*

`llm_chat_randominput.py` script:

```
# Copyright (c) Facebook, Inc. and Microsoft Corporation.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#     http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import onnx
import argparse
import time
import onnxruntime as ort
import numpy as np
from transformers import AutoTokenizer, AutoConfig
import os
import json
provider_map = {'cpu':"CPUExecutionProvider",
                'neutron':"NeutronExecutionProvider"}

def validate_model_files(model_path):
    onnx_file = None
    for file in os.listdir(model_path):
        if file.endswith(".onnx"):
            return os.path.join(model_path, file)
    if not onnx_file:
        return "Error: .onnx file is missing in the model path."

def run_inference_with_decoding(model, model_path, provider, prompt,
 new_tokens, num_layers, num_heads, head_dim):
    """Run model with random inputs and decode the output text"""
    session_options = ort.SessionOptions()
    try:
        session = ort.InferenceSession(model,
 providers=[provider_map[provider]], sess_options=session_options)
        tokenizer = AutoTokenizer.from_pretrained(model_path)
        config = AutoConfig.from_pretrained(model_path)
        is_gemma = "gemma" in config.model_type.lower()
        prompt_ids = tokenizer.encode(prompt, return_tensors="np")  # Shape:
 [1, seq_len]
        batch_size = 1
```

```
        input_ids = prompt_ids
        seq_length = input_ids.shape[1]
        position_ids = np.arange(seq_length, dtype=np.int64).reshape(1,-1)
        attention_mask = np.ones((batch_size, seq_length), dtype=np.int64)
        inputs = {
            'input_ids': input_ids,
            'attention_mask': attention_mask
        }
        if is_gemma:
            inputs["position_ids"]=position_ids
        for i in range(num_layers):
            inputs[f'past_key_values.{i}.key'] = np.zeros((batch_size,
num_heads, 0, head_dim), dtype=np.float32)
            inputs[f'past_key_values.{i}.value'] = np.zeros((batch_size,
num_heads, 0, head_dim), dtype=np.float32)
        all_generated_ids = input_ids[0].tolist()
        for i in range(new_tokens):
            outputs = session.run(None, inputs)
            logits = outputs[0] #[batch_size, seq_len, vocab_size]
            last_token_logits = logits[0, -1, :] # Get the last token's
logits and find the most likely next token
            next_token_id = np.argmax(last_token_logits)
            all_generated_ids.append(int(next_token_id)) # Add to generated
text
            if next_token_id == tokenizer.eos_token_id: # Check if we've hit
the end of sequence token
                break
            # Update inputs for next iteration
            input_ids = np.array([[next_token_id]], dtype=np.int64)
            attention_mask = np.ones((batch_size, seq_length + i + 1),
dtype=np.int64)
            position_ids = np.array([[seq_length + i]], dtype=np.int64)
            inputs = {
                'input_ids': input_ids,
                'attention_mask': attention_mask
            }
            if is_gemma:
                inputs["position_ids"]=position_ids
            # Extract KV cache from outputs
            output_idx = 1  # Start index for KV cache outputs
            for j in range(num_layers):
                inputs[f'past_key_values.{j}.key'] = outputs[output_idx]
                output_idx += 1
                inputs[f'past_key_values.{j}.value'] = outputs[output_idx]
                output_idx += 1
        # Decode the full generated sequence
        generated_text = tokenizer.decode(all_generated_ids,
skip_special_tokens=True)
        print(f"\nGenerated text: {generated_text}")
        return generated_text
    except Exception as e:
        print(f"Error: {e}")
        import traceback
        traceback.print_exc()
        return None

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--model_path","-m", type=str, required=True,
help="Input your Onnx model path.")
```

```
    parser.add_argument("--execution_provider","-e",
 choices=['cpu','neutron'], type=str, default="cpu")
    parser.add_argument("--prompt","-p", type=str, default="Once upon a
 time")
    parser.add_argument("--new_output_tokens","-t", type=int, default=40)
    args = parser.parse_args()
    model = validate_model_files(args.model_path)
    config_path = os.path.join(args.model_path, "config.json")
    config = {}
    if os.path.isfile(config_path):
        with open(config_path) as f:
            config = json.load(f)
    num_layers = config.get("num_hidden_layers",16)
    num_heads = config.get("num_key_value_heads",8)
    head_dim = config.get("head_dim",64)
    results = run_inference_with_decoding(model,
 args.model_path,args.execution_provider,args.prompt, args.new_output_tokens,
 num_layers, num_heads, head_dim)
    print(f'{args.execution_provider}_output: {results}')

if __name__ == "__main__":
    main()
```

The following table provides the current list of supported LLM models.

Table 2. Current list of supported LLM models

LLM Model	Quantization	Download link
Danube_500m	INT4 Group Size Quantization - Group Size 32	-
Gemma3_1B	INT4 Group Size Quantization - Group Size 32	-
Llama_3_2_1B	INT4 Group Size Quantization - Group Size 32 & 128, INT4 Per Channel Quantization	-
Tinyllama_1B	INT4 Group Size Quantization - Group Size 32	-

Table 3. Audio models

LLM Model	Quantization	Download link
Whisper-small	INT4 Group Size Quantization - Group Size 32, INT8 per-channel quantization	-
Whisper-medium	INT4 Group Size Quantization - Group Size 32 INT8 per-channel quantization	-

3.3 ONNX performance test

To run model benchmarks, ONNX Runtime provides a tool that measures performance. The tool named `onnxruntime_perf_test` is installed in `/usr/bin/onnxruntime-1.23.2`. To run it, the user must provide an *.onnx* model file. If no test data is available, the `-I` option can be used to generate dummy input data based on the model's input shape.

1. Benchmark a SqueezeNet model for a single iteration with the VsiNPU execution provider. Run the following command:

```
$/usr/bin/onnxruntime-1.23.2/onnxruntime_perf_test /usr/bin/
onnxruntime-1.23.2/squeezenet/model.onnx -r 1 -e vsinpu
Session creation time cost: 0.219617 s
```

```
First inference time cost: 429 ms
Total inference time cost: 0.429332 s
Total inference requests: 1
Average inference time cost: 429.332 ms
Total inference run time: 0.429347 s
Number of inferences per second: 2.32912
Avg CPU usage: 0 %
Peak working set size: 83386368 bytes
Avg CPU usage:0
Peak working set size:83386368
Runs:1
Min Latency: 0.429332 s
Max Latency: 0.429332 s
P50 Latency: 0.429332 s
P90 Latency: 0.429332 s
P95 Latency: 0.429332 s
P99 Latency: 0.429332 s
P999 Latency: 0.429332 s
```

2. Benchmark a mobilenetv2-12 model for a single iteration with the Neutron execution provider. Run the following command:

```
/usr/bin/onnxruntime-1.23.2/onnxruntime_perf_test mobilenetv2-12_neutron.onnx
 -I -r 1 -e neutron
Expected output:
Session creation time cost: 0.201846 s
First inference time cost: 2 ms
Total inference time cost: 0.00161538 s
Total inference requests: 1
Average inference time cost: 1.61538 ms
Total inference run time: 0.0016315 s
Number of inferences per second: 612.931
Avg CPU usage: -1 %
Peak working set size: 42250240 bytes
Avg CPU usage:-1
Peak working set size:42250240
Runs:1
Min Latency: 0.00161538 s
Max Latency: 0.00161538 s
P50 Latency: 0.00161538 s
P90 Latency: 0.00161538 s
P95 Latency: 0.00161538 s
P99 Latency: 0.00161538 s
P999 Latency: 0.00161538 sP95 Latency: 0.00161538 s
P99 Latency: 0.00161538 s
P999 Latency: 0.00161538 sP90 Latency: 0.00165035 s
```

4 PyTorch

PyTorch is a scientific computing package based on Python that facilitates building deep learning projects using power of Graphics Processing Units (GPUs).

Features:

- PyTorch 2.3.0
- Python version 3 supported
- Deep neural networks built on a tape-based autograd system

Note:

Only the CPU is supported. By default, the PyTorch runtime is running with floating point model. To enable quantized model, the quantized engine should be specified explicitly as follows:

```
torch.backends.quantized.engine = 'qnnpack'
```

4.1 Installing PyTorch

PyTorch is available on the PyPI registry. To install PyTorch on the BSP, run the following command:

```
$ pip install "torch>=2.6.0" "torchvision>=0.21.0"
```

4.2 Running image classification example

There is an example located in the examples folder, which requires urllib, PIL, and maybe some other Python3 modules depending on your image. You may install the missing modules using pip3.

```
$ cd /usr/bin/pytorch/examples
```

To run the example with inference computation on the CPU, use the following command. There are no arguments and the resources will be downloaded automatically by the script:

```
$ python3 pytorch_mobilenetv2.py
```

The output should be similar as follows:

```
File does not exist, download it from
https://download.pytorch.org/models/mobilenet_v2-b0353104.pth
... 100.00%, downloaded size: 13.55 MB
File does not exist, download it from
https://raw.githubusercontent.com/Lasagne/Recipes/master/examples/resnet50/
imagenet_classes.txt
... 100.00%, downloaded size: 0.02 MB
File does not exist, download it from
https://s3.amazonaws.com/model-server/inputs/kitten.jpg
... 100.00%, downloaded size: 0.11 MB
('tabby, tabby cat', 46.34805679321289)
('Egyptian cat', 15.802854537963867)
('lynx, catamount', 1.1611212491989136)
('lynx, catamount', 1.1611212491989136)
('tiger, Panthera tigris', 0.20774540305137634)
```

5 TVM

Apache TVM is an open source machine learning compiler framework for CPUs, GPUs, and NPUs. It aims to enable machine learning engineers to optimize and run computations efficiently on any hardware backend.

Features:

- TVM 0.7.0
- Compilation of deep learning models into minimum deployable modules
- Infrastructure to automatic generate and optimize models on more backend with better performance
- Support for i.MX 8M Plus platforms with OpenVX library
- TVM builder supported for Ubuntu 18.04, x86_64 platform

Note:

For more detailed information, see TVM Documentation.

5.1 TVM software workflow

The pre-trained model will be transformed into the Relay IR and passed through to the TVM model optimizations like constant-folding, memory planning, and finally passed to a codegen phase. In this phase, the operators supported by the target device are transformed as intrinsic calls into the offloading library which connects the model accelerator devices such as GPU/NPU.

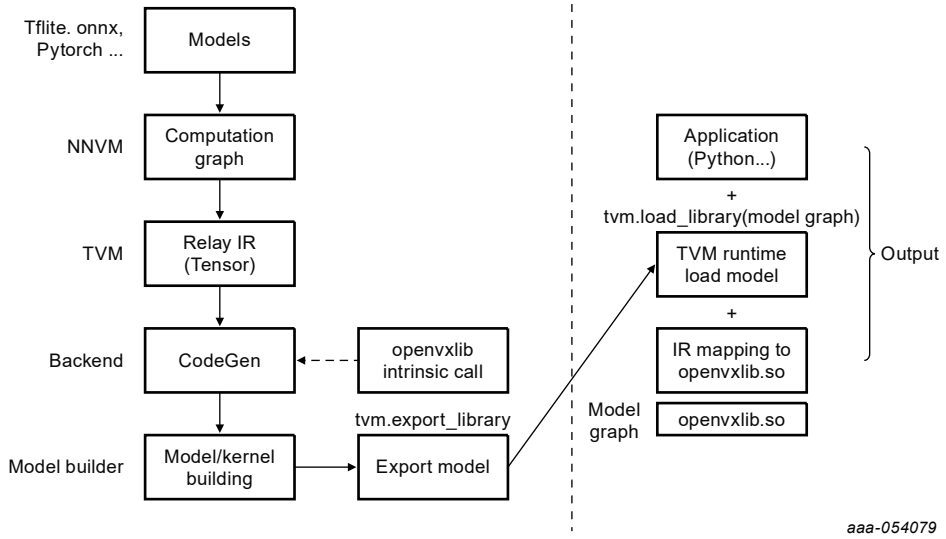

Figure 4. TVM software workflow

5.2 Getting started

5.2.1 Running example with RPC verification

TVM provides the Remote Procedure Call (RPC) capability to run a model on the remote device.

User can run examples at `tests/python/contrib/test_vsi_npu` with RPC verification. The model running result on device will be verified against the result on host with same input.

- Launch the RPC server on the device

```
$ python3 -m tvm.exec.rpc_server --host 0.0.0.0 --port=9090
```

- Export the system variables:

```
$ export TVM_HOME=/path/to/tvm
$ export PYTHONPATH=$TVM_HOME/python
```

- Run the specified models on the host PC:

```
$ python3 tests/python/contrib/test_vsi_npu/test_rpc_tflite_models.py -i
 {device_ip} -m mobilenet_v2_1.0_224_quant
```

- Run all supported TensorFlow Lite models on the host PC:

```
$ python3 tests/python/contrib/test_vsi_npu/test_rpc_tflite_models.py -i
 {device_ip}
```

Note: *This test will download the model automatically, please be sure the network can access the public internet. Example scripts may import additional Python libraries. Please check scripts and make sure they are installed correctly.*

To test `pytorch/onnx/keras` model, additional python packages needs to be installed on the host PC:

```
$ python3 -m pip install torch==1.7.0 torchvision==0.8.1
$ python3 -m pip install onnx=1.8.1 onnxruntime==1.8.1
$ python3 -m pip install tensorflow==2.5.0
```

5.2.2 Running example individually on device

In this mode, the model is compiled on the host offline and saved as model.so. Please refer `tests/python/contrib/test_vsi_npu/compile_tflite_models.py` to compile a TensorFlow Lite model on the host.

Below script snippet shows how to load and run a compiled model at the device:

```
ctx = tvm.cpu(0)
# load the compiled model
lib = tvm.runtime.load_module(args.model)
m = graph_runtime.GraphModule(lib["default"](ctx))
# set inputs
data = get_img_data(args.image, (args.input_size, args.input_size),
 args.data_type)
m.set_input(args.input_tensor, data)
# execute the model
m.run()
# get outputs
tvm_output = m.get_output(0)
```

Please refer `tests/python/contrib/test_vsi_npu/label_image.py` to a complete label image example with pre-processing of image decoding and post-processing to generate label.

5.3 How to build TVM stack on host

Conceptually, TVM can be split into two parts:

- TVM build stack: compiles the deep learning model at host
- TVM runtime: loads and interprets the model at device

This build stack is using the LLVM to cross-compile the generated source as a deployable dynamic library for device. Please, follow the LLVM Doc to install LLVM on the host. If installed successfully, llvm-config should be found under `/usr/bin`.

To build the tvm, please be sure below dependence packages installed on the host:

- cmake
- python3-dev
- build-essential
- llvm-dev
- g++-aarch64-linux-gnu

- libedit-dev
- libxml2-dev
- python3-numpy
- python3-attrs
- python3-tflite

For Ubuntu 18.04, the user could use below commands to install all dependences:

```
$ sudo apt-get update
$ sudo apt-get install -y python3 python3-dev python3-setuptools
$ sudo apt-get install -y cmake llvm llvm-dev g++-aarch64-linux-gnu gcc-aarch64-
linux-gnu
$ sudo apt-get install -y libtinfo-dev zlib1g-dev build-essential libedit-dev
 libxml2-dev
$ python3 -m pip install numpy decorator scipy attrs six tflite
```

Follow below instructions to build TVM stack on the host:

```
$ export TOP_DIR=`pwd`
$ git clone --recursive https://github.com/nxp-imx/eiq-tvm-imx/ tvm-host
$ cd tvm-host
$ mkdir build
$ cp cmake/config.cmake build
$ cd build
$ sed -i 's/USE_LLVM\ OFF/USE_LLVM\ \/usr\/bin\/llvm-config/' config.cmake
$ cmake ..
$ make tvm -j4 # make tvm build stack
```

5.4 Supported models

The following models are verified with TVM.

Table 4. TVM models ZOO

Model	float32	int8	Input size
mobilenet_v1_0.25_128	mobilenet_v1_0.25_128	mobilenet_v1_0.25_128_quant	128
mobilenet_v1_0.25_224	mobilenet_v1_0.25_224	mobilenet_v1_0.25_224_quant	224
mobilenet_v1_0.5_128	mobilenet_v1_0.5_128	mobilenet_v1_0.5_128_quant	128
mobilenet_v1_0.5_224	mobilenet_v1_0.5_224	mobilenet_v1_0.5_224_quant	224
mobilenet_v1_0.75_128	mobilenet_v1_0.75_128	mobilenet_v1_0.75_128_quant	128
mobilenet_v1_0.75_224	mobilenet_v1_0.75_224	mobilenet_v1_0.75_224_quant	224
mobilenet_v1_1.0_128	mobilenet_v1_1.0_128	mobilenet_v1_1.0_128_quant	128
mobilenet_v1_1.0_224	mobilenet_v1_1.0_224	mobilenet_v1_1.0_224_quant	224

Table 4. TVM models ZOO...*continued*

Model	float32	int8	Input size
mobilenet_v2_1.0_224	mobilenet_v2_1.0_224	mobilenet_v2_1.0_224_quant	224
inception_v1	N/A	inception_v1_224_quant	224
inception_v2	N/A	inception_v2_224_quant	224
inception_v3	inception_v3	inception_v3_quant	299
inception_v4	inception_v4	inception_v4_299_quant	299
deeplab_v3_257_mv_gpu	deeplab_v3_256_mv_gpu	N/A	257
deeplab_v3_mnv2_pascal	N/A	deeplab_v3_mnv2_pascal	513
ssdlite_mobiledet	ssdlite_mobiledet_cpu_320x320_coco	N/A	320

6 LiteRT (Experimental)

LiteRT (short for Lite Runtime) version v1.2.0, formerly known as TensorFlow Lite, is Google's high-performance runtime for on-device AI. You can find ready-to-run LiteRT models for a wide range of ML/AI tasks, or convert and run TensorFlow, PyTorch, and JAX models to the TFLite format using the AI Edge conversion and optimization tools.

6.1 Migrating to LiteRT from TensorFlow Lite

Applications that use TensorFlow Lite libraries will continue to function, but all new active development and updates will only be included in LiteRT packages. The LiteRT APIs contain the same method names as the TensorFlow Lite APIs, so migrating to LiteRT does not require detailed code changes.

All new development for Google's high-performance runtime for on-device AI will be exclusively on LiteRT. Applications that use TensorFlow Lite packages will continue to function, but all new updates will only be included in LiteRT packages. The LiteRT APIs contain the same method names as the TensorFlow Lite APIs, so migrating to LiteRT does not require detailed code changes.

For package name changes, to migrate Python code using Tensorflow Lite, replace the PIP package from `tflite-runtime` to `ai-edge-litert`.

Import LiteRT with the following:

```
from ai_edge_litert import interpreter as tflite
interpreter = tflite.Interpreter(model_path=args.model_file)
```

Note: *Only python API is supported in this release.*

6.2 Running the example

In this release, only python API is supported for LiteRT. We can modify the Tensorflow Lite Python example to run it with LiteRT.

Modify the TensorFlow Lite example `label_image.py` for LiteRT:

```
$ cd /usr/bin/tensorflow-lite-2.19.0/examples
$ vi label_image.py
---------import tflite_runtime.interpreter as tflite
```

```
++++++from ai_edge_litert import interpreter as tflite
```

Run the example on the CPU:

```
$ python3 label_image.py -m mobilenet_v1_1.0_224_quant_vela.tflite
-i grace_hopper.bmp -l labels.txt
```

6.3 Running the example on the i.MX 8 platform hardware accelerator

To run the Python example with the same model on the GPU/NPU hardware accelerator, add `-e /usr/lib/liblitert_vx_delegate.so` (for VX Delegate) command line argument. To differentiate between the 3D GPU and the NPU, use the `USE_GPU_INFERENCE` environmental variable. For example, to run the model accelerated on the NPU hardware using VX Delegate, run this command:

```
$ USE_GPU_INFERENCE=0 python3 label_image.py  -m
 mobilenet_v1_1.0_224_quant.tflite
-i grace_hopper.bmp -l labels.txt -e /usr/lib/liblitert_vx_delegate.so
```

6.4 Running the example on the i.MX 93 platform with Ethos-U

To use the hardware acceleration on i.MX 93, convert the model using the Vela compiler first, and run the model with the Ethos-U delegate. Alternatively, directly run the model with the Ethos-U delegate. The model is then converted in the delegate. For details, see Section 7.2.3.

To run the Python example with the model on the NPU hardware accelerator, add the `-e /usr/lib/liblitert_ethosu_delegate.so` command line argument. For example, to run the model accelerated on the NPU hardware using Ethos-U Delegate, run this command:

```
$ python3 label_image.py -m mobilenet_v1_1.0_224_quant_vela.tflite
-i grace_hopper.bmp -l labels.txt -e /usr/lib/liblitert_ethosu_delegate.so
```

6.5 Running the example on the i.MX 9 platform with Neutron-S

To use the hardware Acceleration on i.MX 9 with Neutron-S NPU, convert the model using the neutron converter using the eIQ Toolkit. For details, see the eIQ Toolkit documentation. To run the Python example with the model on the NPU hardware accelerator, add the `-e /usr/lib/liblitert_neutron_delegate.so` command line argument. For example, to run the model accelerated on the NPU hardware using Neutron Delegate, run this command:

```
$ python3 label_image.py -m mobilenet_v1_1.0_224_quant_converted.tflite
-i grace_hopper.bmp -l labels.txt -e /usr/lib/liblitert_neutron_delegate.so
```

7 NN Execution on Hardware Accelerators

7.1 Hardware acceleration on i.MX 8 Series

7.1.1 Hardware accelerator description

The i.MX 8 class devices are deployed with two kind of NN accelerators (see also the figure below):

- Neural Processing Unit (NPU)
- Graphics Processing Unit (GPU)

Neural processing unit is optimized for fixed point arithmetic, in 8-bit and 16-bit width. For optimal performance on the NPU, quantized models shall be used.

Graphics processing unit is optimized for fixed point arithmetic and half precision floating point arithmetic. For optimal performance on the GPU, quantized models or floating-point models with half precision shall be used.

Note:

The TensorFlow Lite framework enables to compute the floating-point models directly in 16-bit half precision arithmetic.

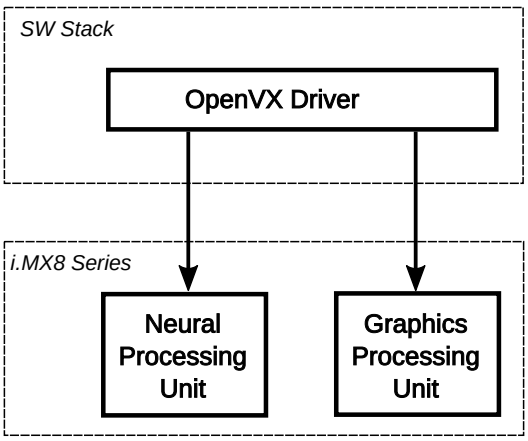

Figure 5. NN accelerator SW stack

Interface to NPU/GPU HW accelerator is provided via the OpenVX v1.3 with NN Extensions. OpenVX is an open, royalty-free standard for cross platform acceleration of computer vision applications. It provides:[2]

- A library of predefined and customizable vision functions
- A graph-based execution model to combine function enabling both task and data independent execution
- A set of memory objects that abstract the physical memory

Open VX defines a C-application programming interface for building, verifying and coordinating graph execution and accessing memory objects. More information about OpenVX can be find on the OpenVX home page.

Note:

In the current OpenVX driver implementation, the maximum number of nodes supported in OpenVX graph is 2048.

7.1.2 Profiling on hardware accelerators

This section describes how to enable profiler on the GPU/NPU, and how to capture logs.

1. Stop the EVK board in the U-Boot by pressing **Enter**.
2. Update mmcargs by adding `galcore.showArgs=1` and `galcore.gpuProfiler=1`.

```
u-boot=> editenv mmcargs
edit: setenv bootargs ${jh_clk} ${mcore_clk} console=${console} root=
${mmcroot} galcore.showArgs=1 galcore.gpuProfiler=1
u-boot=> boot
```

3. Boot the board and wait for the Linux OS prompt.
4. The following environment flags should be enabled before executing the application. `VIV_VX_DEBUG_LEVEL` and `VIV_VX_PROFILE` flags should always be **1** during the process of profiling.

2 OpenVX 1.3 specification: https://registry.khronos.org/OpenVX/specs/1.3/html/OpenVX_Specification_1_3.html

The CNN_PERF flag enables the driver's ability to generate per layer profile log. NN_EXT_SHOW_PERF shows the details of how compiler estimates performance and determines tiling based on it.

```
export CNN_PERF=1 NN_EXT_SHOW_PERF=1 VIV_VX_DEBUG_LEVEL=1 VIV_VX_PROFILE=1
```

5. Capture the profiler log. We use the sample ML example part of standard NXP Linux release to explain the following section.
   - TensorFlow Lite profiling
     Run the TensorFlow Lite application with GPU/NPU backend as follows:

```
$ cd /usr/bin/tensorflow-lite-2.19.0/examples $ ./label_image -
m mobilenet_v1_1.0_224_quant.tflite -t 1 -i grace_hopper.bmp -l
 labels.txt --external_delegate_path=/usr/lib/libvx_delegate.so -v 0 >
 viv_test_app_profile.log 2>&1
```

   The log captures detailed information of the execution clock cycles and DDR data transmission in each layer.
   Note:
   The average time for inference might be longer than usual, as the profiler overhead is added.

7.1.3 Hardware accelerators warmup time

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 subsequent application that runs by storing on disk the information resulted from the initial OpenVX graph processing. The following environment variables should be used for this purpose:

VIV_VX_ENABLE_CACHE_GRAPH_BINARY: flag to enable/disable OpenVX graph caching

VIV_VX_CACHE_BINARY_GRAPH_DIR: set location of the cached information on disk

For example, set these variables on the console in this way:

```
export VIV_VX_ENABLE_CACHE_GRAPH_BINARY="1"
export 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. These environment variables need to be set persistently, for example, available after reboot. Otherwise, the caching mechanism is bypassed even if the *.nb files are available.

The iterations following the graph initialization are performed many times faster. When evaluating the performance of an application running on GPU/NPU, the time should be measured separately for warmup and inference. Warmup time usually affects only the first inference run. However, depending on the machine learning model type, it might be noticeable for the first few inference runs. Some preliminary tests must be done to make a decision on what to consider warmup time. When this phase is well delimited, the subsequent inference runs can be considered as pure inference and used to compute an average for the inference phase.

Note: *OpenVX graph caching is not available on i.MX 8QuadMax platform.*

7.1.4 Switching between GPU and NPU

Some platforms are deployed with both 3D GPU and NPU hardware accelerators. Both can be used for execution of the OpenVX graph (i.e. for ML inference). To differentiate between the GPU and the NPU, there is an environmental variable USE_GPU_INFERENCE. The variable is directly read by the HW acceleration driver.

The behavior is as follows:

- If `USE_GPU_INFERENCE=1`, the graph is executed on the GPU
- Otherwise, the graph is executed on the NPU (if available)

By default, the NPU is used for OpenVX graph execution.

Example with TensorFlow Lite:

```
$ USE_GPU_INFERENCE=1 ./label_image -m mobilenet_v1_1.0_224_quant.tflite
 -i grace_hopper.bmp -l labels.txt --external_delegate_path=/usr/lib/
libvx_delegate.so
```

7.1.5 Per-tensor vs. per-channel quantization

Both per-tensor and per-channel quantizations are supported by the GPU and NPU hardware accelerators on i.MX 8 Series.

The NPU on i.MX 8M Plus is optimized for per-tensor quantization. When running per-channel quantized models, the accelerator must involve additional compute. Therefore, the performance might be lower compared to per-tensor quantized models. The additional compute might also introduce a small accuracy error. The actual impact for both accuracy and latency depends on the model used.

7.2 Hardware acceleration with Ethos-U on i.MX 93 platform

Ethos-U65 is a neural processing unit (NPU) designed to accelerate ML inference in area-constrained embedded and IoT devices from Arm. This NPU is integrated with NXP i.MX 93 processor and works in concert with the Cortex-M core and on-chip SRAM of the SoC. Currently, it provides the following main features:

- Running at 1 GHz and providing 0.5 Tops computation power (256 MAC/cycle).
- Targets quantized Convolutional Neural Networks (CNN) and supports 8 bit weights and 8/16 bit activations.
- Supports TensorFlow Lite (TFLite) inference with fallback to Cortex-A.
- Supports TensorFlow Lite Micro (TFLite-Micro) inference with fallback to Cortex-M.
- Supports inference API to offload the entire model to TFLite-Micro and NPU on Cortex-M.
- Supports TFLite API to offload the customized “ethos-u” operator to NPU on Cortex-M.
- Provides Vela model tool to optimize the model performance and memory usage for the Ethos-U65 target.

7.2.1 Ethos-U subsystem overview

This i.MX 93 machine learning system involves several HW components working collaboratively to support the acceleration of the tensor computation of an ML model: Cortex-A, Cortex-M, Messaging Unit (MU), and Ethos-U NPU.

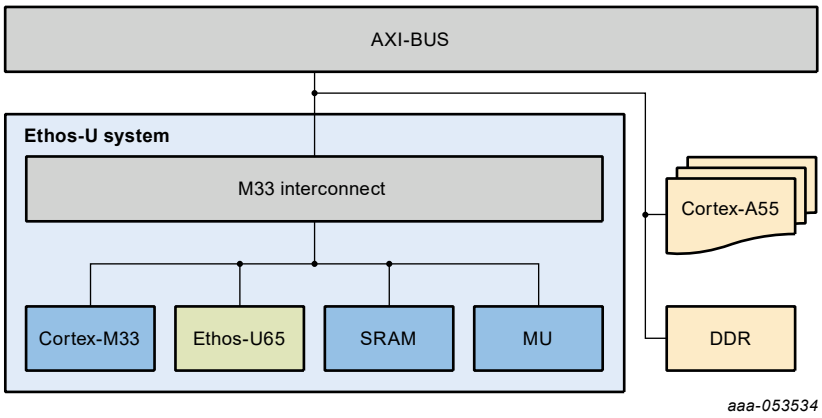

Figure 6. Ethos-U subsystem overview

The Cortex-A55 is responsible for loading the ML model, capture and pre-process the inputs with Linux OS and rich libraries. The Cortex-M is the controller of the attached Ethos-U NPU and it prepares the offloading descriptor for the NPU and triggers the NPU execution. It also provides the un-supported kernels execution for NPU. The MU is the message unit IP to facilitate the core communication between Cortex-A and Cortex-M.

7.2.2 Ethos-U software architecture

The software for Ethos-U support includes three main components, as shown in the following figure.

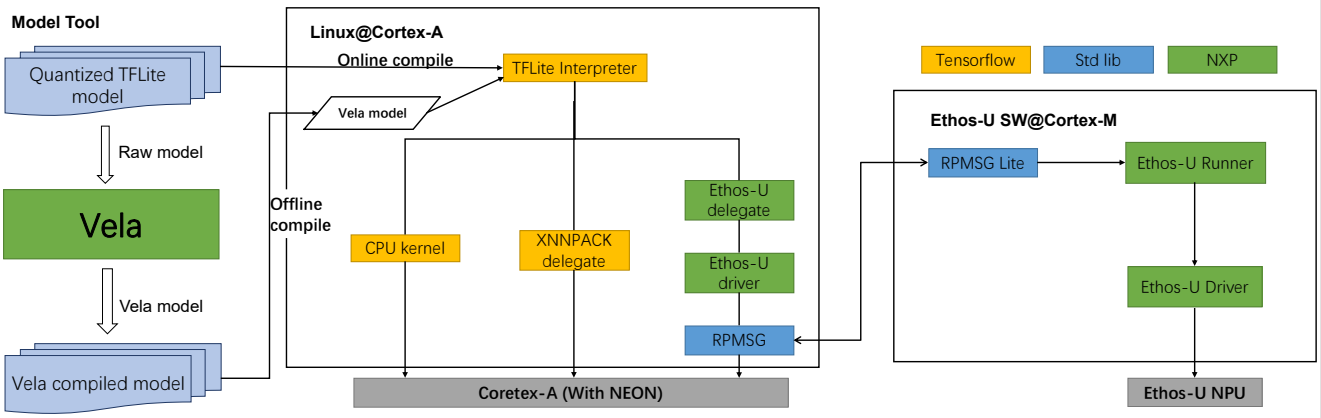

Figure 7. Ethos-U software architecture

- Vela model compiler: offline tool to compile the TensorFlow Lite model graph for Ethos-U. The compiler replaces supported operators in the model with custom “ethos-u” operator containing the command stream for Ethos-U NPU. The output of the compiler is a modified TensorFlow Lite model graph for TensorFlow Lite/ TensorFlow Lite-Micro inference engines. This is only required for Inference API.
- Cortex-A SW stack for Linux: containing MPU inference engine (TensorFlow Lite), driver library, and Linux kernel driver.
- Cortex-M SW stack: containing MCU inference engine SW (TensorFlow Lite-Micro, CMSIS-NN) and NPU driver.

The typical inference workflow is as follows:

1. Converts the TensorFlow Lite model into Vela model using the Vela model compiler and generates the optimized version for Ethos-U NPU.
   Note: *For TensorFlow Lite inference engine with Ethos-U delegate, this step is optional. The Ethos-U delegate supports both TensorFlow Lite model and Vela compiled model. Using the Ethos-U delegate*

increases the warm-up time for model execution. The model needs to be compiled at runtime. Models precompiled with Vela brings warm-up time decrease.

2. The optimized model is inferred either by:
   a. TensorFlow Lite Ethos-U delegate, which recognizes the custom "ethos-u" operator in Vela compiled model, allocates the buffer for input/output feature map (IFM/OFM) and executes the operator via Ethos-U Linux driver.
   b. TensorFlow Lite Ethos-U delegate, which recognizes the supported operators in TensorFlow Lite model, compiles the operators to "ethos-u" operator and allocates the buffer for input/output feature map (IFM/OFM) and executes the operator via Ethos-U Linux driver.
   c. Inference API, which allocates the buffer for input/output feature map and sends entire model via Ethos-U driver.
3. The Ethos-U driver composes the inference task message and sends it over RPMSG to Cortex-M.
4. The Ethos-U Runner on Cortex-M dispatches the task to TensorFlow Lite-Micro or Ethos-U driver directly according to the task type.
   a. If the task type is accelerating the “ethos-u” operator (using the TensorFlow Lite), the Runner calls the Ethos-U driver directly.
   b. If the task type is accelerating the entire model (using the Inference API), the Runner dispatches the model to TensorFlow Lite-Micro and further calls Ethos-U driver for processing.
5. After the Ethos-U driver completes the inference task, it writes the result into the OFM buffer and sends the response back to Cortex-A via RPMSG.

Note: *The model is loaded from Cortex-A and shared with Cortex-M over RPMSG.*

The Cortex-M SW is pre-built with both the model and Ethos-U operator acceleration capabilities in a single-binary firmware. This firmware is integrated into Yocto rootfs and will be loaded automatically when the user starts an inference task using the TensorFlow Lite or Inference API by opening the Ethos-U device.

7.2.3 Getting started

In the Yocto rootfs, there are several examples provided to show how to use different APIs to interact with Ethos-U NPU with an image classification inference.

1. Go to the example folder and copy the `label.txt` and input picture from TensorFlow.

```
$ cd /usr/bin/ethosu/examples
$ cp ../../tensorflow-lite-2.19.0/examples/labels.txt ./
$ cp ../../tensorflow-lite-2.19.0/examples/grace_hopper.bmp ./
```

2. Compile the model for Ethos-U using Vela tool, reusing the model `mobilenet_v1_1.0_224_quant.tflite` from `/usr/bin/tensorflow-lite-2.19.0/examples/`. If running successfully, an optimized vela model `mobilenet_v1_1.0_224_quant_vela.tflite` is generated in the output folder.

```
$ vela ../../tensorflow-lite-2.19.0/examples/
mobilenet_v1_1.0_224_quant.tflite
```

3. Run the model with the Inference API (offloads the entire model to TFLite-Micro).

```
$ ./inference_runner -n ./output/mobilenet_v1_1.0_224_quant_vela.tflite -i
 grace_hopper.bmp -l labels.txt -o output.txt
```

   The following will be printed if no error occurs:

```
Send capabilities request
Capabilities:
        version_status:1
        version:{ major=0, minor=0, patch=0 }
        product:{ major=6, minor=0, patch=0 }
```

```
        architecture:{ major=1, minor=0, patch=6 }
        driver:{ major=0, minor=16, patch=0 }
        macs_per_cc:8
        cmd_stream_version:0
        custom_dma:false
Create network
Create inference
Wait for inferences
Inference status: success
Detected: military uniform, confidence:70
```

4. Run the converted model with TFLite inference engine using the Ethos-U Delegate (offload the “ethos-u” operator to Ethos-U NPU).

```
$ cd /usr/bin/tensorflow-lite-2.19.0/examples
$ ./label_image -m
../../ethosu/examples/output/mobilenet_v1_1.0_224_quant_vela.tflite
--external_delegate_path=/usr/lib/libethosu_delegate.so
```

The following is printed if no error occurs:

```
INFO: Loaded model ../../ethosu/examples/output/
mobilenet_v1_1.0_224_quant_vela.tflite
INFO: resolved reporter
Ethosu delegate: device_name set to /dev/ethosu0.
Ethosu delegate: timeout set to 60000000000.
Ethosu delegate: enable_cycle_counter set to 0.
Ethosu delegate: pmu_event0 set to 0.
Ethosu delegate: pmu_event1 set to 0.
Ethosu delegate: pmu_event2 set to 0.
Ethosu delegate: pmu_event3 set to 0.
EXTERNAL delegate created.
INFO: EthosuDelegate delegate: 1 nodes delegated out of 1 nodes with 1
 partitions.

INFO: Applied EXTERNAL delegate.
INFO: invoked
INFO: average time: 3.903 ms
INFO: 0.780392: 653 military uniform
INFO: 0.105882: 907 Windsor tie
INFO: 0.0156863: 458 bow tie
INFO: 0.0117647: 466 bulletproof vest
INFO: 0.00784314: 835 suit
```

7.2.4 Vela tool

The vela tool is used to compile a TensorFlow Lite for Microcontrollers neural network model into an optimized version that can run on an embedded system containing an Arm Ethos-U NPU. The optimized model contains TFLite Custom operators for those parts of the model that can be accelerated by the Ethos-U NPU. Parts of the model that cannot be accelerated are left unchanged and run on CPU (Cortex-A or Cortex-M) using an appropriate kernel (such as the Arm optimized CMSIS-NN kernels). After compilation, the optimized model can only be run on an Ethos-U NPU embedded system. The tool also generates performance estimates for the compiled model.

To deploy the neural network (NN) model on Ethos-U, the first step is to use Vela to compile the prepared model. To be accelerated by the Ethos-U NPU, the network operators must be quantized to either 8-bit (unsigned or signed) or 16-bit (signed).

NXP Vela is based on Arm ethos-u-vela. Compared to ethos-u-vela, NXP added more OPs support and reduced some OP constraints.

Note: *A specific version of Vela is tightly coupled with a specific version of the Ethos-U driver. The compatibility between different Vela versions is not guaranteed.*

7.2.4.1 Installing the Vela tool

The Vela tool can be run on the i.MX 93 board or Linux PC. It is already available in NXP Yocto rootfs. This section describes how to install it on the X86 Linux PC. The steps are as follows.

1. Get the vela source code.

```
$ git clone https://github.com/nxp-imx/ethos-u-vela.git
```

2. Install with python pip.

```
$ cd ethos-u-vela
$ git checkout lf-6.12.49_2.2.0
$ pip3 install .
```

3. After all the commands are successful, use `vela --help` to check if Vela is installed successfully.

```
$ vela --version
4.2.0
```

7.2.4.2 Compiling the TFLite model

After Vela is installed, the following commands can be used to compile a TFLite model to the optimized version for Ethos-U NPU. The optimized model is stored into the OUTPUT_DIR ("./output" by default). The output file has the suffix `_vela.tflite`. It is also a TFLite model . After the compilation, Vela outputs the detailed log into the console.

Note: *The Vela expects that the TFLite model is quantized already. Vela supports asymmetric quantization to 8 bit (signed and unsigned) and 16 bit (signed), as defined by TFLite. To accelerate the model operators with Ethos-U NPU, the input model to Vela has to be quantized. Non-quantized operators will fall back to CPU.*

The following provides an example for how to compile a model and shows the corresponding output log:

```
$ vela mobilenet_v1_1.0_224_pb_int8.tflite
```

Output log:

```
Network summary for mobilenet_v1_1.0_224_pb_int8
Accelerator configuration               Ethos_U65_256
System configuration                 internal-default
Memory mode                            internal-default
Accelerator clock                                 1000 MHz
Design peak SRAM bandwidth                       16.00 GB/s
Design peak DRAM bandwidth                        3.75 GB/s
Total SRAM used                                 381.08 KiB
Total DRAM used                                4293.34 KiB
CPU operators = 0 (0.0%)
NPU operators = 60 (100.0%)
Average SRAM bandwidth                            4.28 GB/s
Input   SRAM bandwidth                            7.95 MB/batch
Weight  SRAM bandwidth                           12.61 MB/batch
Output  SRAM bandwidth                            0.00 MB/batch
Total   SRAM bandwidth                           20.67 MB/batch
Total   SRAM bandwidth          per input       20.67 MB/inference (batch size 1)
Average DRAM bandwidth                            3.00 GB/s
Input   DRAM bandwidth                            5.53 MB/batch
```

```
Weight  DRAM bandwidth                          3.92 MB/batch
Output  DRAM bandwidth                          5.06 MB/batch
Total   DRAM bandwidth                         14.52 MB/batch
Total   DRAM bandwidth          per input     14.52 MB/inference (batch size 1)
Neural network macs                        572406226 MACs/batch
Network Tops/s                                  0.24 Tops/s
NPU cycles                                   3937697 cycles/batch
SRAM Access cycles                            719415 cycles/batch
DRAM Access cycles                           2984386 cycles/batch
On-chip Flash Access cycles                        0 cycles/batch
Off-chip Flash Access cycles                       0 cycles/batch
Total cycles                                 4831570 cycles/batch
Batch Inference time              4.83 ms,  206.97 inferences/s (batch size 1)
```

The following is the computational graph after the model (`mobilenet_v1_1.0_224_pb_int8.tflite`) is compiled. Here, Vela encapsulates all supported OPs into one Ethos-U OP.

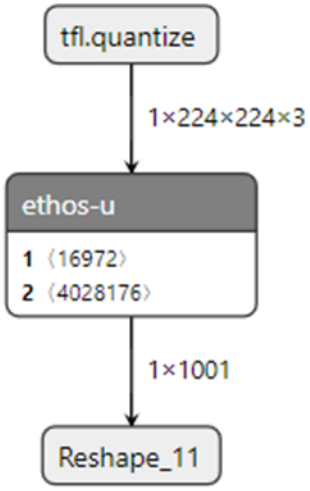

Figure 8. Computational graph

Note:

The Vela tool takes a lot of memory when converting big models, so a swap file is needed when converting big models to avoid the Out Of Memory issue. Use the following commands to add a swap file:

```
dd if=/dev/zero of=/swapfile count=2048 bs=1M
mkswap /swapfile
chmod 0600 /swapfile
swapon /swapfile
```

7.2.5 Inference with Ethos-U inference API

The Ethos-U inference API provides the methods to use the Ethos-U NPU on Linux OS without the TensorFlow Lite inference engine. It takes the compiled model and IFM/OFM as inputs, composes an inference task, and dispatches the inferences to the Cortex-M with Ethos-U.

7.2.5.1 Ethos-U driver library

The Ethos-U Driver provides a C++ APIs for dispatching the inference to the Ethos-U Linux kernel driver. The library and the corresponding header file is available on Yocto rootfs and SDK:

- `/usr/include/ethosu.hpp`
- `/usr/lib/libethosu.so`

The following is the component diagram of Ethos-U Driver library:

- The Device class represents the instance of Ethos-U unit.
- The Buffer class is used to store any data, including the model.
- The Network class represents a model instance bind to specific Device.
- The Inference class represents the inference, which is computation of the computation graph (model) on input data. Notice, the Network class is separated from the Inference class, allowing multiple inferences to share the same network.

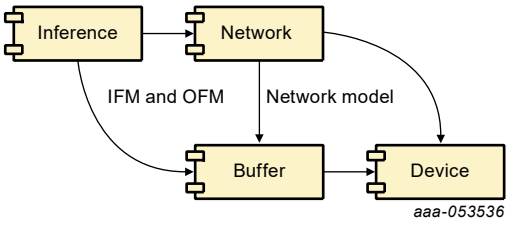

Figure 9. Component diagram of Ethos-U Driver library

The inference runner demonstrates how to dispatch inferences to the Ethos-U Linux kernel driver. All the steps described in the sequence diagram below are executed by the `inference_runner` application.

1. The Device class obtains a file descriptor handle for the device node (`/dev/ethosu<nr>`) using the `open()` system call. The Device class uses `ioctl()` system calls to manipulate with the underlying device, like buffer and network creation.
2. The Network class uses the Device and buffer handles to create a new network object. The model is stored in the Buffer that the network parses to discover the input and output shapes of the network model.
3. The Inference class uses the Network object to create an inference. The array of IFM Buffers need to be populated with data before the inference object is created.

The inference object must poll the file descriptor waiting for the inference to complete.

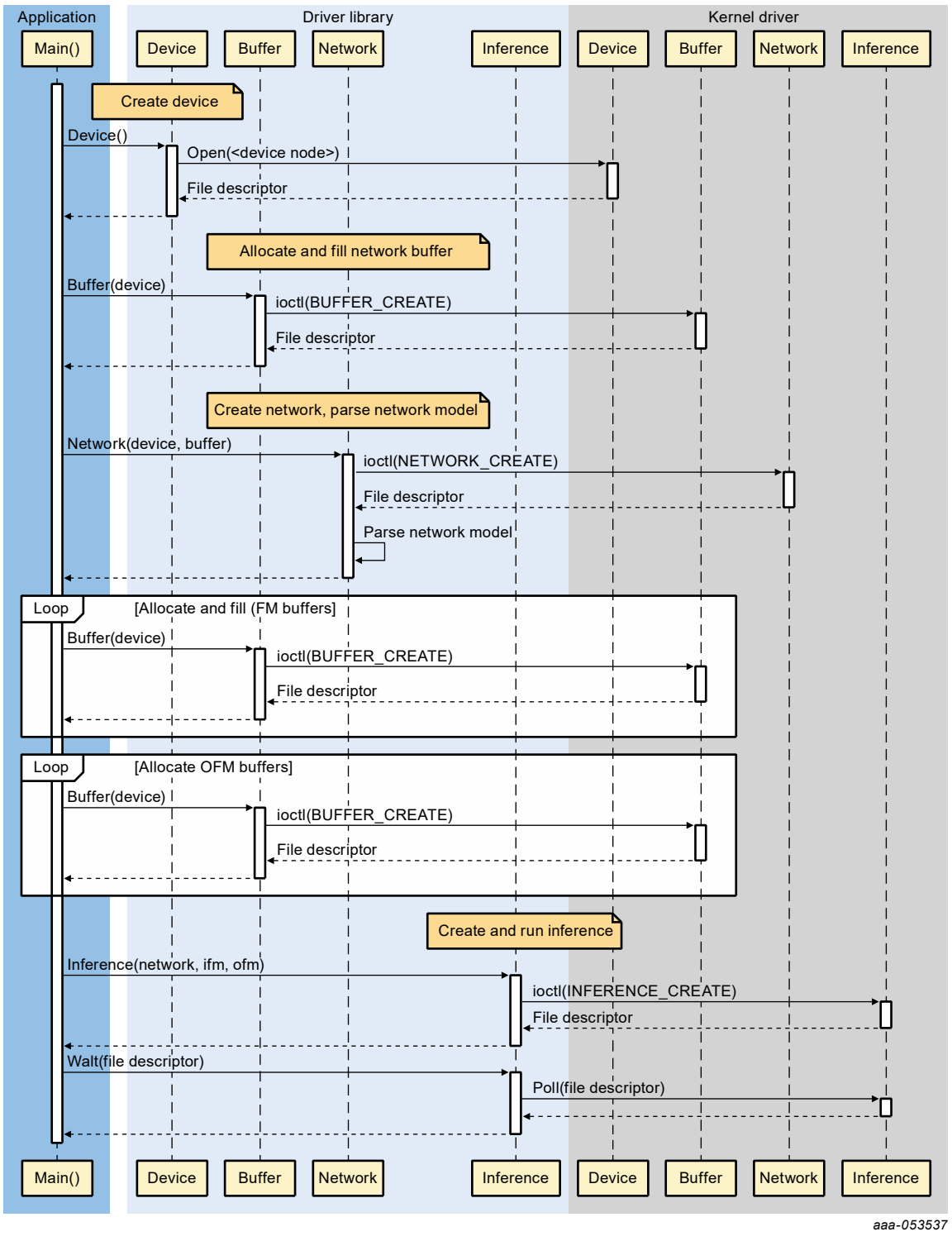

Figure 10. Sequence diagram

7.2.5.2 Ethos-U Linux kernel driver interface

The Ethos-U Linux kernel driver exposes User-space API (UAPI) for Ethos-U subsystem, and to communicate with the Cortex-M in the Ethos-U subsystem.

The communication with the Ethos-U subsystem is based on message passing in shared memory, and the Linux kernel mailbox APIs for triggering IRQs on the remote CPU, what is the Cortex-M in this case.

The address of the message queues is hard coded in the Cortex-M application, and configured in the DTB for the Ethos-U Linux kernel driver.

When the Linux kernel driver allocates dynamic memory for the Ethos-U subsystem, it must be able to map a physical address to a bus address. The DTB contains a dma-ranges, which define how to remap physical addresses to the Cortex-M address space.

7.2.5.3 Device and Buffer class

The device driver creates a device node at `/dev/ethosu<nr>` that a user space application can open and issues IOCTL requests to. This is how buffers and networks are created.

Creating a new buffer returns another file descriptor that can be memory mapped for reading and/or writing.

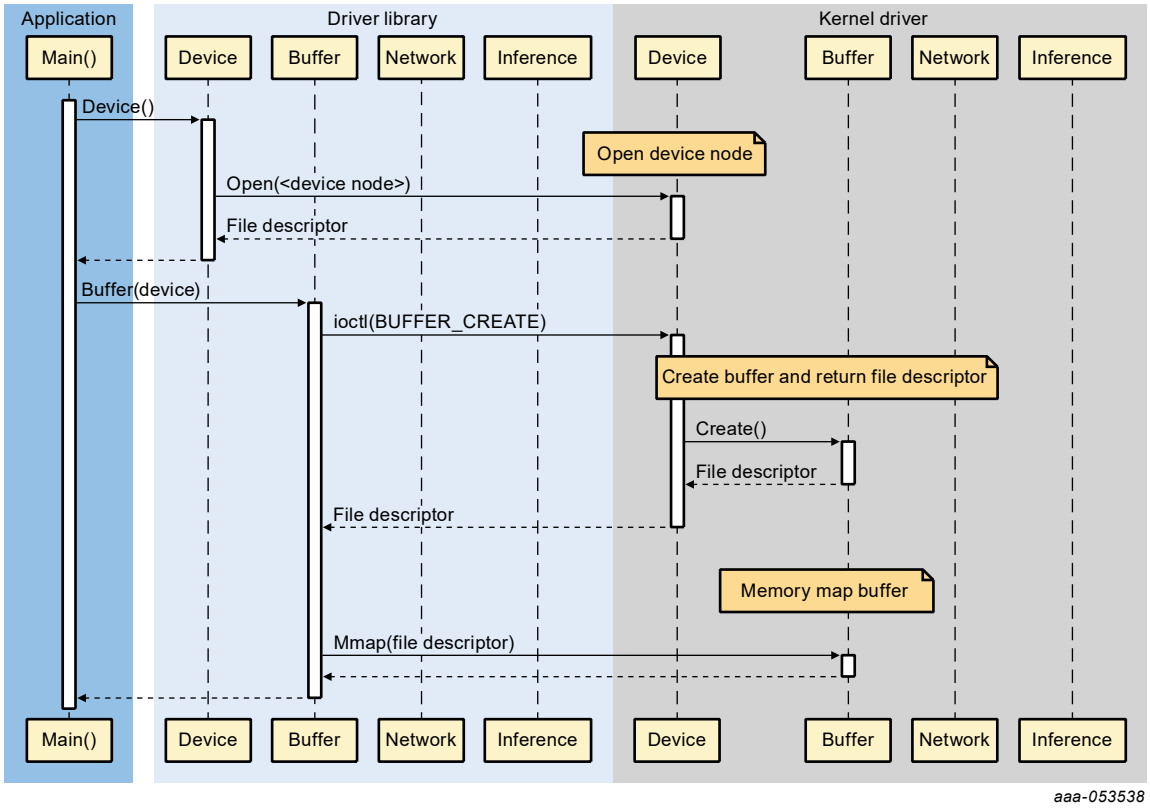

Figure 11. Device and Buffer class

7.2.5.4 Network class

Creating a network assumes that the device node has already been opened, and that a buffer has been allocated and populated with the network model.

A new network is created by issuing an IOCTL command on the device node file descriptor. A file descriptor to a buffer, containing the network model, is passed in the IOCTL data. The network class increases the reference count on the buffer, preventing the buffer from being freed before the network object has been destructed.

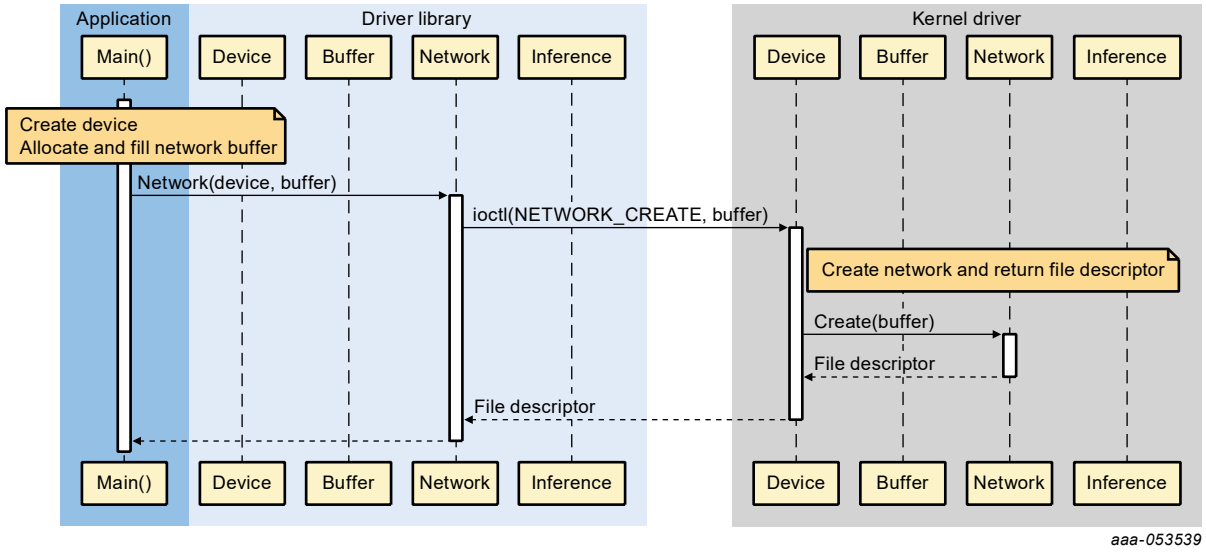

Figure 12. Network class

7.2.5.5 Inference class

Creating an inference assumes that a network has already been created, IFM buffers have been allocated and populated with data, and OFM buffers have been allocated.

A new inference is created by issuing an IOCTL command to the network file descriptor. An array of IFM and OFM buffers are passed in the IOCTL data, which reference counts will be increased.

As the inference object has been created an inference request message is sent to the Cortex-M application. The inference request message is written to a ring buffer in shared memory, cache maintenance is executed if necessary, and an IRQ is raised using the Linux mailbox APIs.

On success, a valid file handle is returned to user space. The file handle is used to read the results when the inference completes. Note this is a blocking call.

Once the inference task has finished on the Ethos-U subsystem, the message process writes an inference response message into the response queue in shared memory, executes cache maintenance if needed, and raises an IRQ.

On the Linux side the IRQ is handled and cleared. The IRQ bottom handler is a separate Linux kernel thread responsible for reading the message queue. When the inference response message is received it updates the status of the inference and unblocks any waiting user space processes.

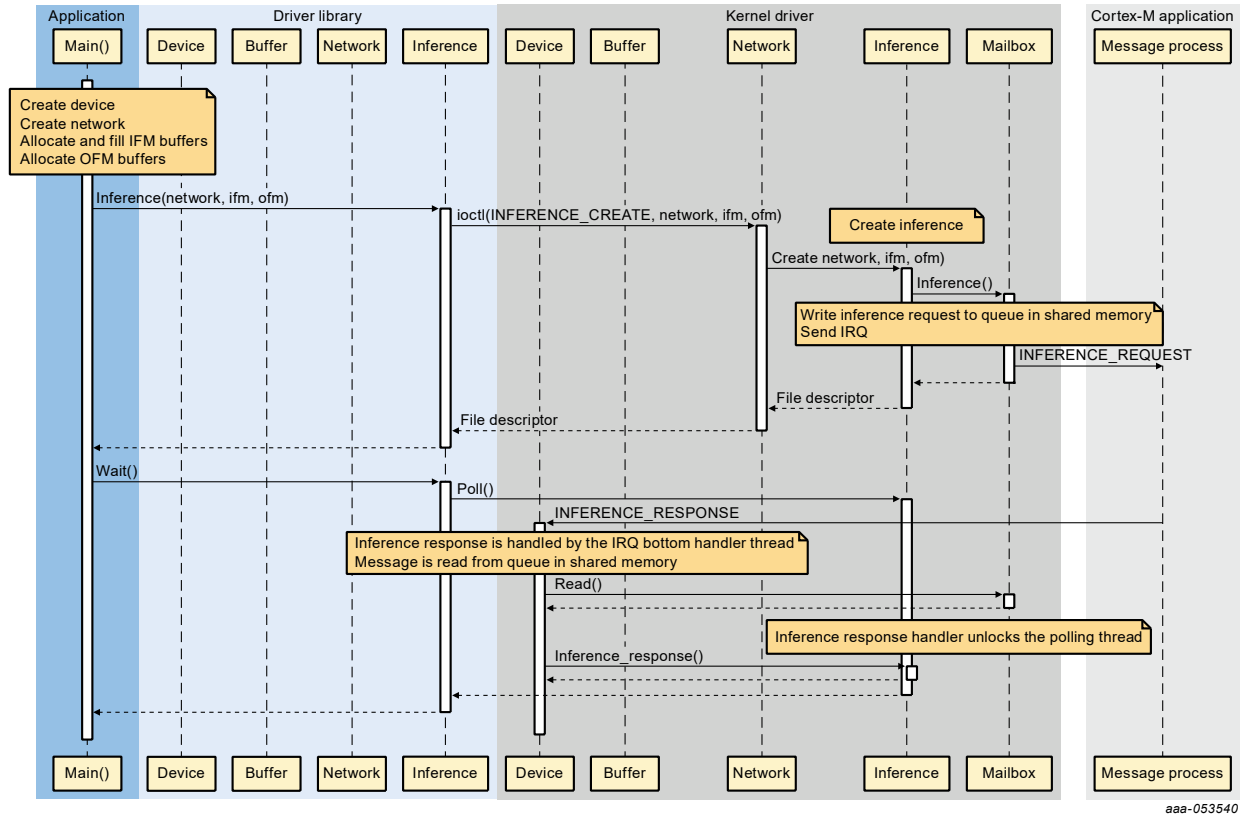

Figure 13. Inference class

7.2.5.6 How to use the inference API

The following steps show how to run a Vela model from Cortex-A:

1. Create the inference device.

```
device = Device("/dev/ethosu0")
```

2. Load the model into a buffer from the Vela model file.

```
shared_ptr<Buffer> model_buf = allocAndFill(device, vela_model);
```

3. Create the Network instance with the model buffer.

```
shared_ptr<Network> network = make_shared<Network>(device, model_buf);
```

4. Load the input feature map (IFM) from the input file (such as a picture for image classification app) into a buffer. If there are multiple inputs, create the buffers one by one and push back to a vector.

```
vector<shared_ptr<Buffer>> ifm;
ifm_size = network->getIfmDims()[0];
ifm_buf = make_shared<Buffer>(device, ifm_size);
memcpy(ifm_buf ->data(), input_data, input_size);
ifm.push_back(ifm_buf)
```

5. Create the output feature map (OFM) buffers according to the output dimensions in the model. If there are multiple outputs, create the buffer one by one and push back to a vector.

```
vector<shared_ptr<Buffer>> ofm;
ofm_size = network->getOfmDims()[0]
```

```
ofm_buf = make_shared<Buffer>(device, ofm_size);
ofm.push_back(ofm_buf);
```

6. Create an inference instance with the Network buffer, IFM buffer, and OFM buffer.

```
inf = make_shared<Inference>(net, ifm.begin(), ifm.end(), ofm.begin(),
 ofm.end());
```

7. Call `Inference->invoke()` to trigger and wait for the completion of the inference task.

```
Inf->invoke()
```

8. Access the OFM buffers to get the inference result.

```
Outputs = inf->getOfmBuffers()
```

7.2.5.7 Interpreter class

In addition to low-level APIs described above, the Ethos-U driver also provides the Interpreter class. The Interpreter handles the steps mentioned above (device, network, and buffer initialization) internally with `class Interpreter`.

Constructor:

```
Interpreter(const char *model,
const char *device = "/dev/ethosu0",
int64_t arenaSizeOfMB = 16);

model: vela model file
device: ethos-u device name, default: “/dev/ethosu0”
arenaSizeOfMB : shared DDR memory size between Cortex-A and Cortex-M, default:
 16(MB)
```

Inference blocking API:

```
void Invoke(int64_t timeoutNanos = 60000000000);
timeoutNanos: timeout for the inference, default value is 60s.
```

Input/Output tensor buffer address helper:

```
template <typename T>
T* typed_input_buffer(int index) {
    int32_t offset = network->getInputDataOffset(index);
    return (T*)(arenaBuffer->data() + offset);
}

template <typename T>
T* typed_output_buffer(int index) {
    int32_t offset = network->getOutputDataOffset(index);
    return (T*)(arenaBuffer->data() + offset);
}
```

Given the tensor index in a model, returns the tensor address and type information.

Input/Output information query:

```
std::vector<TensorInfo> GetInputInfo();
std::vector<TensorInfo> GetOutputInfo();
```

These two provides the interface to query inputs and outputs information from a model, including shape and type information (int8/uint8/f32…).

7.2.5.8 Interpreter Python wrapper

In addition to C++ API, the Ethos-U Driver also provides the Python API.

It is installed into Yocto rootfs: `/usr/lib/python3.10/site-packages/ethosu`.

Example usage:

```
import ethosu.interpreter as ethosu

# loading the vela model file into interpreter
interpreter = ethosu.Interpreter(args.vela_model_file)

# get the input and output dimensions
inputs = interpreter.get_input_details()
outputs = interpreter.get_output_details()

# resize the input according to the model input dimensions
w, h = inputs[0]['shape'][1], inputs[0]['shape'][2]
img = Image.open(args.image).resize((w, h))
data = np.expand_dims(img, axis=0)

# associcate the input data with interpreter
interpreter.set_input(0, data)

# invoke the inference, this is a blocking API, timeout is 60s
interpreter.invoke()

# get back the inference results, different models have different results.
# Check the model output dimensions and get all the outputs with index.
output_data = interpreter.get_output(0)
```

7.2.6 Inference with TensorFlow Lite

7.2.6.1 Ethos-U Delegate

See Section 2.2.4.

7.2.6.2 Delivery package

The Ethos-U support is built into shared library: `/usr/lib/libtensorflow-lite.so`. When the user loads the Vela model with TFLite API, the engine calls the Ethos-U Linux driver and dispatches the customized Ethos-U operator to Ethos-U firmware on Cortex-M.

The Ethos-U Delegate shared library: `/usr/lib/libethsou_delegate.so`.

7.2.6.3 Running image classification example

See Section 7.2.3 to try the example.

See TensorFlow Lite for how to build and use the Tensorflow Lite API with an application.

7.2.6.4 Hardware accelerators warmup time

For TensorFlow Lite, the initial execution of model inference takes longer time, because of the model graph initialization needed by the NPU hardware accelerator. The initialization phase is known as warmup. In this phase, the delegate calls the Vela tool to compile the TensorFlow Lite models.

- This time duration can be decreased for subsequent application that runs by storing on disk the information resulted from the initial Vela processing. The Ethos-U delegate option "`cache_file_path`" should be used for this purpose. For example, set this option in your application in this way:

```
# the external delegate accepts the option "cache_file_path",
ext_delegate = [tflite.load_delegate("/usr/lib/libethosu_delegate.so",
 {"cache_file_path":"your_path"})]
interpreter = tflite.Interpreter(model_path=model_file,
 experimental_delegates=ext_delegate)
```

  By setting up this option, the result of the Vela compilation is stored on disk. The runtime performs a quick check on the network. If it matches, it loads it into the NPU memory directly.
- This time duration can also be decreased by compiling the model by Vela beforehand. In this way, you should compile the model with the Vela tool and pass the Vela optimized model file to the TensorFlow Lite application.

7.2.6.5 Ethos-U performance enhancement with memory zero-copy

In addition to the memory space of TensorFlow Lite, Ethos-U also has its own memory. Therefore, there is a large number of memory copies in the inference process, which takes a lot of CPU time. To improve the performance, in Ethos-U delegate, some of TensorFlow Lite memory is mapped into the memory of the NPU to avoid memory copies.

Note: *When doing inference with the Ethos-U delegate, do not modify the TensorFlow Lite Tensor data pointers. This may cause unpredictable issues.*

7.2.7 Building and deploying the Ethos-U firmware

7.2.7.1 Getting the source

The ethos-u-core-software is part of the i.MX 93 Ethos-U NPU machine learning software package, which is an optional middleware component of MCUXpresso SDK. The ethos-u-core-software is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include Ethos-U NPU machine learning into the MCUXpresso SDK package, the ethos-u-core-software middleware component is selected in the software component selector on the SDK Builder page when building a new package. See the following figure.

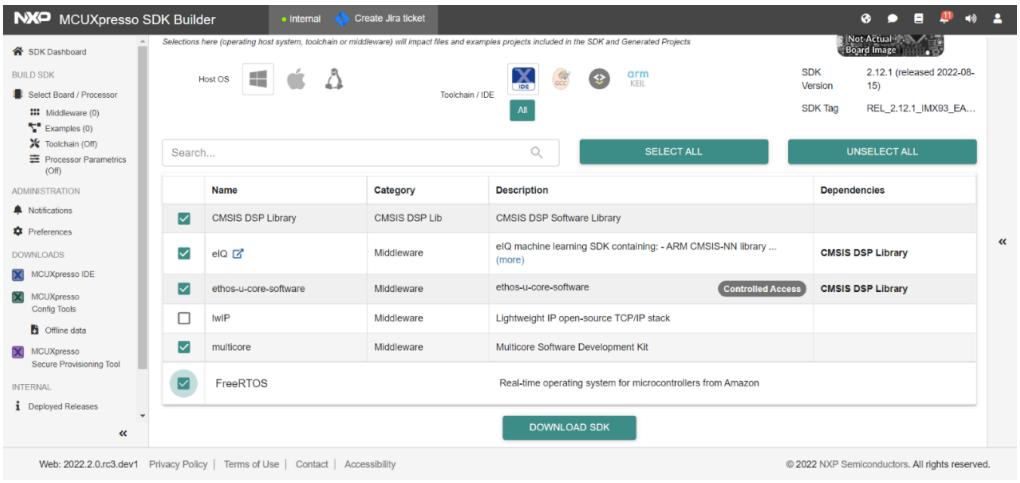

Figure 14. SDK Builder page

Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the *Getting Started with MCUXpresso SDK User's Guide* (document: MCUSDKGSUG). The package directory structure is similar as follows.

```
<MCUXpresso-SDK-root>
|-- boards
|   -- <board>
|      -- demo_apps    - Example build projects
|          -- ethosu_apps_rpmsg  - Ethos-U default firmware with rpmsg
|          -- ethosu_apps    - Ethos-U standalone app example
|
|-- middleware/ethos-u-core-software
    -- applications  - The inference process APIs
    -- boards  - The board related initialization and configuration files
    -- core_driver  - Ethos-U core driver which includes reading/writing
 registers
    -- examples    - Ethos-U example applications
        -- ethosu_apps_rpmsg  - Ethos-U default firmware with rpmsg
        -- ethosu_apps   - Ethos-U standalone app example
```

7.2.7.2 Ethos-U example applications

7.2.7.2.1 Introduction

There are two Ethos-U apps available:

- `ethosu_apps_rpmsg`: firmware for Yocto Linux BSP
- `ethosu_apps`: standalone example for Cortex-M

The `ethosu_apps_rpmsg` is the firmware for Ethos-U subsystem for Linux OS. It contains core message handling, inference request processing from Cortex-A core, NPU's registers configuration, inference execution, and inference result providing to Cortex-A core. The supported inference engine is TFLite or TFlite-Micro (if Inference API is used).

The example `ethosu_apps` is a Cortex-M standalone application that demonstrates the inference execution entirely on Cortex-M which can be used in the low power scenario with the Cortex-A sleeping. The example uses conv2d op model. There is no core message handling and only supports TFLite-Micro.

The apps are available in the `/boards/<board>/demo_apps/ethosu_apps*` folders.

7.2.7.2.2 Toolchains supported

- IAR Embedded Workbench for Arm
  When the project is opened in IAR, press the “Make” button to build the project in IAR as follows.

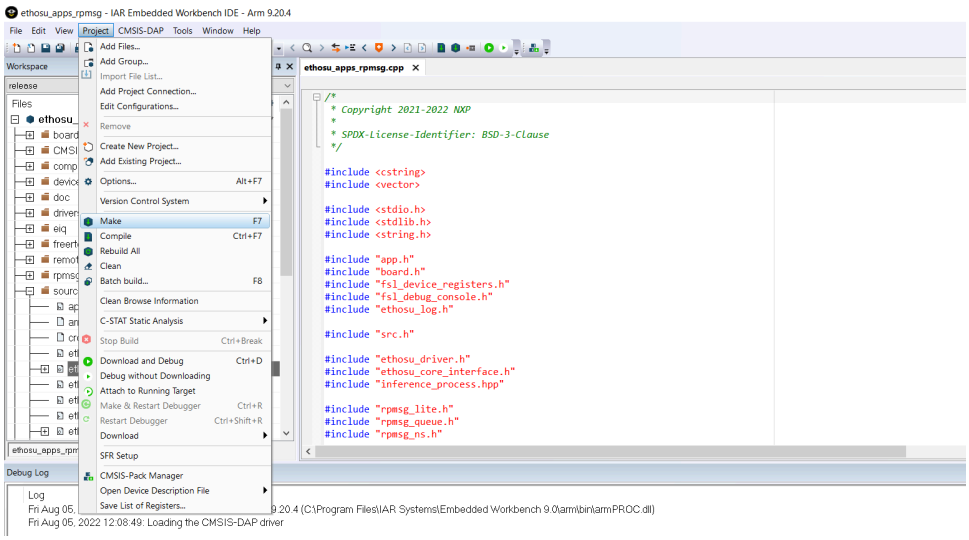

Figure 15. IAR Embedded Workbench for Arm

- ArmGCC - GNU Tools Arm Embedded
  Run the following command to build the project.

```
$ cd mcu-sdk-2.16/boards/mcimx93evk/demo_apps/ethosu_apps_rpmsg/armgcc
$ export ARMGCC_DIR=${YOUR_TOOLCHAIN_LOC}/gcc-arm-none-eabi-10-2020-q4-major
$ export PATH=$PATH:${YOUR_TOOLCHAIN_LOC}/gcc-arm-none-eabi-10-2020-q4-major/
bin
$ ./build_release.sh
```

7.2.7.3 Deploy procedure

1. Deploy the `ethosu_apps_rpmsg` firmware.
   Example `ethosu_apps_rpmsg` is built as `.out` or `.elf` and installed in rootfs as the name of “`ethosu_firmware`”. The pre-built binary is integrated in the rootfs and loaded by Linux Ethos-U driver upon an inference request.
   If the user rebuilds the firmware, the rebuilt `ethosu_apps_rpmsg.out` or `ethosu_apps_rpmsg.elf` should be copied to `/lib/firmware/` in rootfs and renamed as the name of “`ethosu_firmware`” as follows:

```
$ cp ethosu_apps_rpmsg.elf ./lib/firmware/ethosu_firmware
```

2. Deploy the `ethosu_apps` with U-Boot.
   The `ethosu_apps` is built as `.bin`. In U-Boot terminal, users can run the following command to do inference for the `conv2d op` model.

```
=> tftp 0x80000000 ethosu_apps.bin
=> cp.b 0x80000000 0x201e0000 0x20000;
=> bootaux 0x1ffe0000 0
```

When the example runs, the log and inference result would be displayed on the Cortex-M terminal as follows:

```
Initialize Arm Ethos-U
Inference status: success
```

Note:

The default `firmware ethosu_apps_rpmsg` contains the following operators support with TFLite-micro on Cortex-M33: Ethos-U, TFLite_Detection_PostProcess, and Dequantize. If an operator is supposed to fall back on Cortex-M33 but not included, rebuild the source code and deploy the firmware.

The `ethosu_apps` is a standalone Cortex-M application running without Cortex-A interacted, so it is deployed at the U-Boot stage.

7.2.7.4 Using the Ethos-U on Cortex-M

The Ethos-U NPU on i.MX 93 is accessible by the TFLite-Micro library. The TFLite-Micro interprets the optimized Vela model and delegates the kernels to different execution providers.

Currently, there are 3 types of execution provider supported:

- **NN Kernel**: default kernel implementation provided by TFLite-Micro for Cortex-M CPU.
- **CMSIS-NN kernel**: optimized kernel implementation by Arm using the CMSIS-NN library. The CMSIS-NN library executes the kernel on the Cortex-M CPU.
- **Ethos-U Kernel**: kernel implementation for the custom Ethos-U operator. This operator registered in TFLite-Micro framework and executes the computation on Ethos-U using the NPU driver.

7.2.7.4.1 Running Vela model with TFLite-Micro

The following provides the steps to run the Vela model on Cortex-M directly:

1. Get the flatbuffer Vela model.

```
const tflite::Model* model = tflite::GetModel(vela_model);
```

2. Configure/Allocate the inputs, outputs tensors statically.

```
constexpr int kTensorArenaSize = 1024 * 1024;
static uint8_t tensorArena[kTensorArenaSize];
```

3. Build the TFLite-Micro interpreter for the inference.

```
static tflite::MicroInterpreter interpreter(
model,  //the flatbuffer model
microOpResolver,  //resolve to kernel implementers
tensorArena,      // tensor memory address
kTensorArenaSize,  //tensor memory length
microErrorReporter); //error reporter
```

4. Set the input tensors.
   // Get access to the input tensor data

```
TfLiteTensor* inputTensor = interpreter->input(0);
```

   // Copy the input tensor data from an application buffer

```
for (int i = 0; i < inputTensor->bytes; i++)
inputTensor->data.int8[i] = input_data[i];
```

5. Run the inference and get the output.

```
// Invoke the inference
interpreter->Invoke();
// Get access to the output tensor data
TfLiteTensor* outputTensor = interpreter->output(0);
// Copy the output tensor data to an application buffer
for (int i = 0; i < outputTensor->bytes / sizeof(float32); i++)
    output_data[i] = outputTensor->data.f[i];
```

TFLite-Micro does not depend on dynamic memory allocation, so it requires users (application developers) to supply a memory arena when an interpreter is created. In practice, the user usually allocates this memory arena as a static buffer when the program starts, for example:

```
#define TENSOR_ARENA_SIZE (1024 * 1024 * 16)
uint8_t tensorArena[TENSOR_ARENA_SIZE];
```

TFLite-Micro framework uses this memory arena as inputs/outputs/intermediate tensors store. This memory size “TENSOR_ARENA_SIZE” must be adjusted according to the practical usage to consider the following points:

- Model used for the application
- Size of the input/output data
- Memory needed for intermediate result
- Memory arena mapping to SRAM or TCM, considering the effective usage of memory hierarchy

7.2.8 Memory hierarchy for Cortex-M

For Cortex-M, there are several types of memory media with different capacity, speed, and cost, which can be accessed by CPU. On i.MX 93, the memory hierarchy looks like below with speed decreasing order.

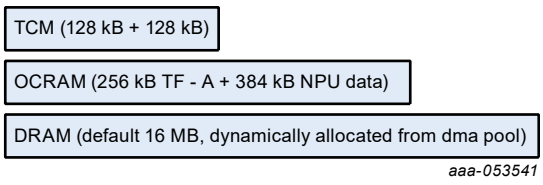

Figure 16. Memory hierarchy on i.MX 93

The TCM size is 256 KB, usually used for Cortex-M runtime data. By design, this memory space is not allocated for system purpose after booting. How to use it effectively is left for user decision.

The OCRAM size is 640 KB. By design, the first 256 KB is allocated for Arm Trusted Firmware (ATF), which is used to bootstrap the Cortex-A before the DRAM is available. The rear 384 KB is reserved for the NPU data: the weight/bias of an ML model.

The DRAM size is 2 GB on the i.MX 93 EVK board. However, only the shared DMA region between Cortex-A and Cortex-M can be used. The Ethos-U Linux driver requests DMA buffers for TensorArena dynamically from the DMA pool and passes the buffer address to the Ethos-U firmware on Cortex-M. If not explicitly specified, 16 MB DMA buffer is requested by default.

Ethos-U can only access the DRAM and OCRAM memory by design. The current memory mapping for the Ethos-U firmware is as shown in the following figure.

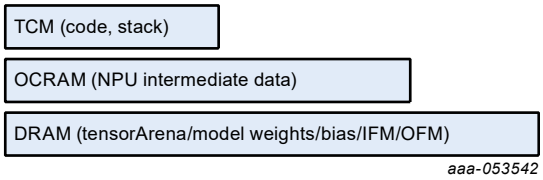

Figure 17. Memory mapping for Ethos-U firmware

With this configuration, the model data and TensorArena is allocated in DRAM and the OCRAM is used as NPU cache. The `Dedicated_Sram` memory mode should be used for model compilation with Vela:

```
vela --accelerator-config ethos-u65-256 --system-config Ethos_U65_High_End
--memory-mode Dedicated_Sram --config vela.ini {tflite-model}
```

For a standalone Cortex-M application, the memory mapping is as follows.

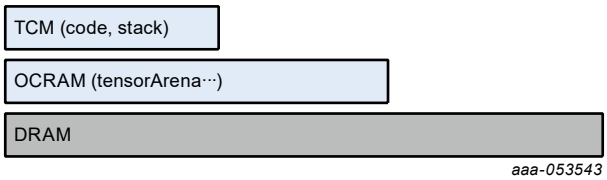

Figure 18. Memory mapping for a standalone Cortex-M application

With this configuration, No DRAM is used. All the model data and TensorArena memory for the NPU is allocated in OCRAM. The `Sram_Only` memory mode should be used for model compilation with Vela:

```
vela --accelerator-config ethos-u65-256 --system-config Ethos_U65_High_End
--memory-mode Sram_Only --config vela.ini {tflite-model}
```

7.2.9 Supported ML operators and constraints

See the supported operator list on the Ethos-U NPU. When the operator is not supported by the NPU, the Vela compiler displays the constraints information on the console. They are left untouched and scheduled on the CPU. Use the eIQ toolkit to view what operators are merged into Ethos-U operator for a model.

7.2.10 Profiling on hardware accelerators

This section describes how to enable profiler on the NPU, and how to capture logs.

There is model profiling tool (`vela-prof`) available on the i.MX 93 platform, which uses the NPU's Performance Monitoring Unit (PMU) to analyze NPU utilization, DRAM access, and SRAM usage for TensorFlow Lite (TFLite) models. The following are examples of usage:

```
root@imx93evk:~# vela-prof -i mobilenet_v1_1.0_224_int8.tflite
Info: Profiling model mobilenet_v1_1.0_224_int8_vela.tflite with PMU
=====================================
NPU Utilization: 62.0% = 572406226 / 256 / 3607842
DRAM Utilization: 47.5% = 401481 * 16 / 3607842 / 3.75
read : 40.1% = 338698 * 16 / 3607842 / 3.75
write: 7.4% = 62783 * 16 / 3607842 / 3.75
SRAM Utilization: 49.6% = 1789955 * 16 / 3607842 / 16.0
read : 37.4% = 1349578 * 16 / 3607842 / 16.0
write: 12.2% = 440377 * 16 / 3607842 / 16.0
root@imx93evk:~#
```

The tool first compiles the TFLite model using the Vela compiler to generate a Vela-optimized model, and then runs it on the board to collect real-time profiling data.

The following describes the advanced profiling on using Ethos-U PMU events.

The PMU profiling is supported with Ethos-U delegate options. The option is a string to specify what PMU events will be captured, which includes:

- `enable_cycle_counter:true`
  Shows the cycle counter values for an inference.
- `pmu_event config register mapping`
  Up to 4 PMU event IDs can be recorded, which is specified by `pmu_event register mapping`.
  For example, `pmu_event0:1` means the `pmu_event0` register will record the values of event 1.
  `pmu_event0:1;pmu_event1:3` means the `pmu_event0` register will record the value of event 1, and the `pmu_event1` register will record the value of event 3.

The following table shows all the event IDs supported by Ethos-U.

Table 5. Event IDs supported by Ethos-U

Event type	Event ID
CYCLE	1
NPU_IDLE	2
CC_STALLED_ON_BLOCKDEP	3
CC_STALLED_ON_SHRAM_RECONFIG	4
NPU_ACTIVE	5
MAC_ACTIVE	6
MAC_ACTIVE_8BIT	7
MAC_ACTIVE_16BIT	8
MAC_DPU_ACTIVE	9
MAC_STALLED_BY_WD_ACC	10
MAC_STALLED_BY_WD	11
MAC_STALLED_BY_ACC	12
MAC_STALLED_BY_IB	13
MAC_ACTIVE_32BIT	14
MAC_STALLED_BY_INT_W	15
MAC_STALLED_BY_INT_ACC	16
AO_ACTIVE	17
AO_ACTIVE_8BIT	18
AO_ACTIVE_16BIT	19
AO_STALLED_BY_OFMP_OB	20
AO_STALLED_BY_OFMP	21
AO_STALLED_BY_OB	22
AO_STALLED_BY_ACC_IB	23
AO_STALLED_BY_ACC	24
AO_STALLED_BY_IB	25

Table 5. Event IDs supported by Ethos-U*...continued*

Event type	Event ID
WD_ACTIVE	26
WD_STALLED	27
WD_STALLED_BY_WS	28
WD_STALLED_BY_WD_BUF	29
WD_PARSE_ACTIVE	30
WD_PARSE_STALLED	31
WD_PARSE_STALLED_IN	32
WD_PARSE_STALLED_OUT	33
WD_TRANS_WS	34
WD_TRANS_WB	35
WD_TRANS_DW0	36
WD_TRANS_DW1	37
AXI0_RD_TRANS_ACCEPTED	38
AXI0_RD_TRANS_COMPLETED	39
AXI0_RD_DATA_BEAT_RECEIVED	40
AXI0_RD_TRAN_REQ_STALLED	41
AXI0_WR_TRANS_ACCEPTED	42
AXI0_WR_TRANS_COMPLETED_M	43
AXI0_WR_TRANS_COMPLETED_S	44
AXI0_WR_DATA_BEAT_WRITTEN	45
AXI0_WR_TRAN_REQ_STALLED	46
AXI0_WR_DATA_BEAT_STALLED	47
AXI0_ENABLED_CYCLES	48
AXI0_RD_STALL_LIMIT	49
AXI0_WR_STALL_LIMIT	50
AXI_LATENCY_ANY	51
AXI_LATENCY_32	52
AXI_LATENCY_64	53
AXI_LATENCY_128	54
AXI_LATENCY_256	55
AXI_LATENCY_512	56
AXI_LATENCY_1024	57
ECC_DMA	58
ECC_SB0	59
AXI1_RD_TRANS_ACCEPTED	60
AXI1_RD_TRANS_COMPLETED	61

Table 5. Event IDs supported by Ethos-U*...continued*

Event type	Event ID
AXI1_RD_DATA_BEAT_RECEIVED	62
AXI1_RD_TRAN_REQ_STALLED	63
AXI1_WR_TRANS_ACCEPTED	64
AXI1_WR_TRANS_COMPLETED_M	65
AXI1_WR_TRANS_COMPLETED_S	66
AXI1_WR_DATA_BEAT_WRITTEN	67
AXI1_WR_TRAN_REQ_STALLED	68
AXI1_WR_DATA_BEAT_STALLED	69
AXI1_ENABLED_CYCLES	70
AXI1_RD_STALL_LIMIT	71
AXI1_WR_STALL_LIMIT	72
ECC_SB1	73

After setting the `external_delegate_options` with PMU event capture, you can run the TensorFlow Lite application. The PMU counter result is displayed on the console.

```
./label_image -m ~/mobilenet_v1_1.0_224_int8_vela.tflite --
external_delegate_path=/usr/lib/libethosu_delegate.so -l labels.txt --
external_delegate_options="enable_cycle_counter:true;pmu_event0:1;pmu_event1:3;pmu_event2:4;pmu_event3:5"

Ethos_u PMUs : [ 3674981 25689 54788 3667061 ]
Ethos-u cycle counter: 3676058
Ethos_u PMUs : [ 3673491 25287 54785 3665621 ]
Ethos-u cycle counter: 3674570
Ethos_u PMUs : [ 3680291 25502 54784 3672421 ]
Ethos-u cycle counter: 3681370
```

7.3 NPU transition guide from i.MX 8M Plus to i.MX 93

This section describes how to port Machine Learning application from i.MX 8M Plus to i.MX 93 with NPU acceleration.

7.3.1 Tensorflow Lite difference between i.MX 8M Plus and i.MX 93 NPU acceleration

See Figure 3 for Tensorflow Lite software stack. Both i.MX 8M Plus and i.MX 93 support Tensorflow Lite with NPU acceleration. i.MX 93 also supports TensorFlow Lite external delegate mechanism.

From the development perspective of the Machine Learning application, users can use the same Tensorflow API to develop the Machine Learning application. The only difference is that users need to use the Ethos-U Delegate instead of VX Delegate.

7.3.2 NPU supported operator list

While porting the Machine Learning application from i.MX 8M Plus to i.MX 93, check whether the NPU supported operators in your model are supported on the i.MX 93 NPU. This ensures that you leverage i.MX 93 NPU acceleration.

See supported operator list for i.MX 93 NPU operator support status and Table 13 for i.MX 8M Plus NPU operator support status.

7.4 Hardware acceleration with eIQ Neutron NPU on i.MX 9 series platform

eIQ Neutron NPU is a Neural Processing Unit (NPU) developed by NXP. It is designed to accelerate the Machine Learning inference. The Neutron-S version of the eIQ Neutron NPU comprises of 3 main blocks:

- Neutron computation core (Neutron) doing MACs, can be pipelined with multiple instances.
- Neutron controller (RISC-V core) to program Neutron registers and control the Neutron block.
- DMA like memory controller (Data Mover) to exchange data between host DDR and Neutron dedicated TCM.

Neutron-S NPU main features:

- Targets quantized Convolutional Neutral Networks (CNN) and supports 8 bit weights and 8/16 bit activations.
- Supports TensorFlow Lite (TFLite) inference with fallback to Cortex-A for unsupported operations.
- Supports TFLite API to offload a custom TFLite node - neutronGraph, to Neutron-S NPU.
- Provides model converter tool (through eIQ toolkit) to optimize the model performance and memory usage for Neutron-S NPU target.

7.4.1 Neutron-S NPU overview

The Neutron-S NPU involves several hardware blocks working together to support the acceleration of the tensor computation defined by the Machine Learning model:

- SoC main CPU (Cortex-A55)
- RISC-V Controller
- Data Mover
- Neutron compute block

The SoC main CPU runs the software under Linux OS, like the TFLite inference engine. It is responsible for loading the Machine Learning model, capturing and pre-processing the inputs and handing over the tensor computation to the NPU. The RISC-V controller in the Neutron-S NPU orchestrates the Neutron compute blocks and Data Mover. The Data Mover is a DMA like engine used for moving data between the SoC DDR and the NPU TCM.

7.4.2 Neutron-S software architecture

The software for Neutron-S NPU includes three main components, as shown in the following figure.

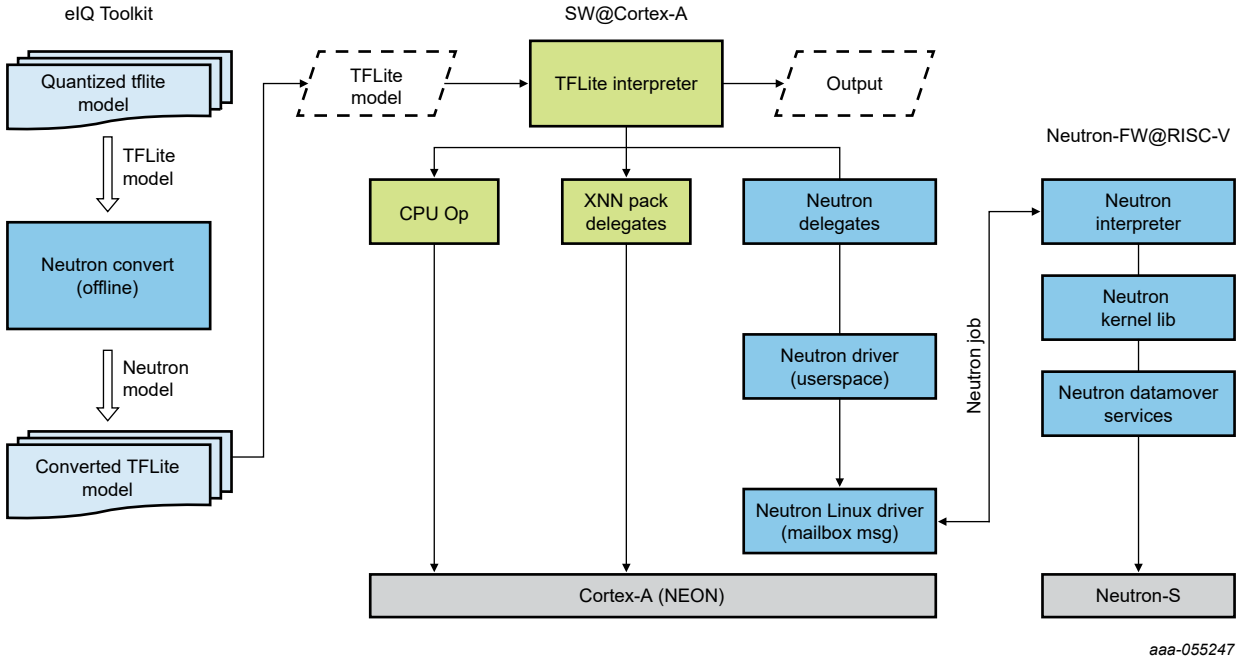

Figure 19. Neutron-S software architecture

- The Neutron model converter is an offline tool to compile the TFLite model for Neutron-S. The converter replaces supported operators in the model with a custom `neutronOp` node containing a model-specific firmware binary, static data, like weights, and inputs/outputs memory areas for Neutron-S. The output of the converter is a modified TFLite model graph for TFLite inference engine. For inference, it needs to use the corresponding TFLite Neutron Delegate.
- The Cortex-A software stack for Linux contains the TFLite inference engine, Neutron delegate library, user space Neutron driver library, and Neutron device driver for the Linux kernel.

The Neutron-FW stack contains code for the RISC-V controller, interpreting the microcode in the neutronGraph node.

7.4.3 NPU performance tuning

NXP Yocto Linux enables power-saving technology by default. To boost Machine Learning Performance on the i.MX 95 EVK board, perform the following tuning operations:

1. Disable DDR clock gating.
   DDR clock gating is the hardware feature applied to the i.MX 9 family. It is enabled by default in Yocto Linux for maximum power saving with performance trade-off.
   On i.MX 95, send the following command on the System Manager console (the serial port right after the Linux console) to disable DDR clock gating.

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

2. Set the CPU to performance mode.
   In Yocto Linux, the CPU is in `ondemand` mode by default, which makes the CPU work at low frequency when there is low workload. Make the CPU in performance mode before eIQ benchmarking.

```
# cpufreq-set -g performance
# cat /sys/devices/system/cpu/*/cpufreq/cpuinfo_cur_freq
1800000
```

3. Disable CPU idle.

For the Linux system, when disabling CPU Idle management, it can reduce context switch overhead.

```
# echo 1 | tee  /sys/devices/system/cpu/cpu*/cpuidle/state*/disable
```

7.4.4 Neutron NPU power management

Neutron device implements power management techniques to optimize energy consumption: supporting clock gating and power gating.

It provide different power mode strategies as follows:

- Automatic mode (default): Balance performance and power consumption, automatically clock gate for NPU compute and suspend (power NPU off) after a period of idleness (default 1 second).
- Performance mode: The Neutron NPU remains power ON and clock ON for a long time.
- Low power mode: Further power savings on top of the automatic mode. Neutron NPU performs clock gating immediately once inference is completed. Cuts off the clocks for NPU Compute, TCM, and ZenV core.

The default power mode strategy is automatic and the default suspend delay time is 1000 milliseconds. Users can change them using the following commands:

1. Change the power mode through the Linux model parameters on the U-Boot command line interface:

```
=> setenv bootargs "neutron.power_mode=0|1|2 $bootargs"
```

2. Change power gating (suspend) delay time on the Linux command line interface:

```
# cat /sys/devices/platform/soc/4ab00004.neutron/power/autosuspend_delay_ms
1000
# echo {time_millisecond} > /sys/devices/platform/soc/4ab00004.neutron/power/
autosuspend_delay_ms
```

8 Vision Pipeline with NNStreamer

NNStreamer is an efficient and flexible stream pipeline framework for complex neural network applications. It was initially developed by Samsung and then transferred to LF AI Foundation as an incubation project.

It is a set of GStreamer plugins that allows both GStreamer developers to adopt neural network models and neural network developers to manage neural network pipelines and their filters easily and efficiently.

The project is well documented through its dedicated github documentation site, but the main takeaways are described below for convenience.

In addition to the standard GStreamer data types, NNStreamer adds new data types “other/tensor” and “other/tensors” using a dedicated converter element. This data type represents a stream of multidimensional array and a stream of a container of multiple instances of such arrays, respectively.

NNStreamer provides a set of stream filters applying multiple operations on tensors:

- `tensor_converter` converts audio, video, text, or arbitrary binary streams to `others/tensor` streams.
- `tensor_decoder` converts `other/tensor(s)` to video or text stream with assigned sub-plugins.
- `tensor_filter` invokes a neural network model with the given model path and neural network framework name.
- `tensor_transform` applies various operators to tensors including typecast, add, mul, transpose, and normalize. For faster processing, it supports SIMD instructions and multiple operators in a single filter.
- `tensor_crop` crops the regions of incoming tensor.
- `tensor_rate` controls a frame rate of tensor streams.

- `tensor_mux`, `tensor_demux`, `tensor_merge`, `tensor_split`, `tensor_if`, and `tensor_aggregator` support tensor stream path controls.
- `tensor_sink` is a sink plug-in for making an application to get a buffer of `other/tensor(s)`.
- `tensor_source` allow non GStreamer standard input sources, such as sensors, to supply `other/tensor(s)` stream.
- `tensor_reposink` and `tensor_reposrc` implement recurrent path helpers, cutting GStreamer pipeline cycle through a dedicated shared repository. The `tensor_reposink` pushes data to the repository, this latter reinjecting data upstream through a `tensor_reposrc` element.

The following figure shows the general architecture of a NNStreamer pipeline.

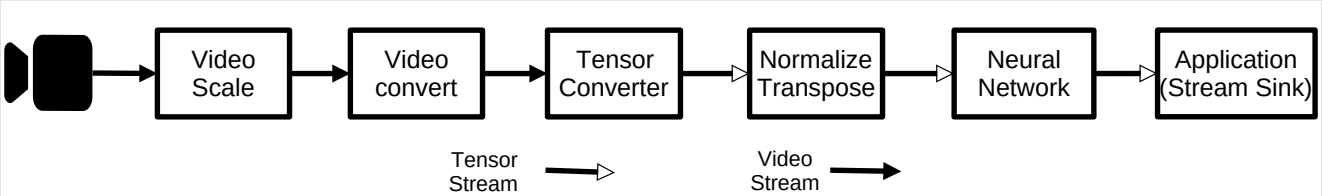

Figure 20. NNStreamer pipeline

There are two elements allowing adding user created features in run-time: ***tensor_filter*** and ***tensor_decoder***:

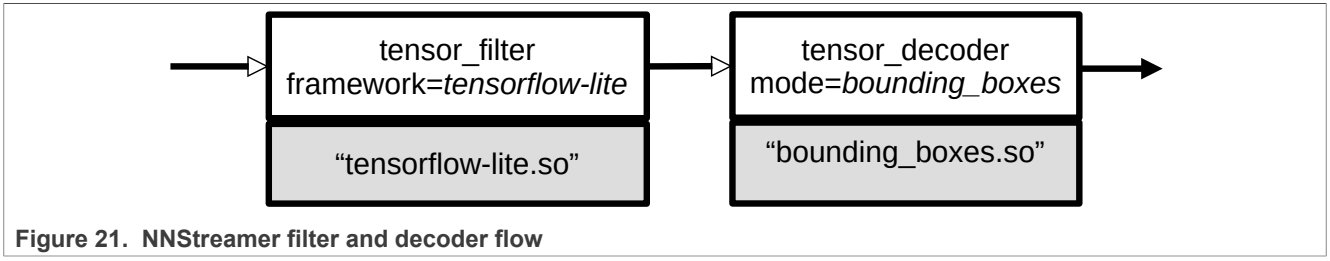

Figure 21. NNStreamer filter and decoder flow

While instantiating the *tensor_filter* and *tensor_decoder*, the framework and mode options respectively specify the target implementation through a dedicated shared library loaded at runtime. NNStreamer supplies a set of filters and decoders which are described briefly below, and APIs to implement customized user sub-plugins. Hence, it is possible to use a proprietary inference engine sub-plugin as tensor filter, or a specialized NN decoder.

NNStreamer supports the most popular inference engines (open source or not). On this release, TensorFlow Lite and TVM engines are supported.

Table 6. NNStreamer supported features

Framework/Tool	i.MX 95	i.MX 93	i.MX 8M Plus	i.MX 8M Quad/8M Nano/8QuadMax/8 QuadXPlus	i.MX 8M Mini/8 ULP
TensorFlow Lite	CPU/GPU/NPU	CPU/NPU	CPU/NPU/GPU	CPU/GPU	CPU
TVM	-	-	CPU/NPU/GPU	-	-
Custom C++	CPU	CPU	CPU	CPU	CPU
Custom Python	CPU	CPU	CPU	CPU	CPU
NNShark	-	-	CPU	-	-

In case an inference engine might be supported on multiple hardware backend, one can specify the device mapping the neural network.

Even though Tensor decoder element might not be appropriate for building an application which usually does not consume the neural network outputs for display purpose only, it is especially useful for implementing a

prototype during the development phase which might focus on the neural network model or optimizing the data path. Indeed, most neural networks topologies are supported for classical computer vision use cases: classification, object detection, pose estimation or segmentation.

NNStreamer tensor filter element has to be configured to use specific engine and hardware accelerator. Available options are listed in the following tables.

Table 7. TensorFlow Lite engine

Delegate	Tensor filter properties	USE_GPU_INFERENCE env variable
No delegate	`framework=tensorflow-lite model=<path to .tflite model file> custom=NumThreads:<cpu cores>` ***Note:*** *<cpu core> values:* • ***2** for i.MX 93 and i.MX 8ULP* • ***4** for others* • ***6** for i.MX 95*	-
XNNPACK Delegate	`framework=tensorflow-lite model=<path to .tflite model file> custom=Delegate:XNNPACK,NumThreads:<cpu cores>` ***Note:*** *<cpu core> values:* • ***2** for i.MX 93 and i.MX 8ULP* • ***4** for others* • ***6** for i.MX 95*	-
Neutron Delegate (i.MX 95 only)	`framework=tensorflow-lite model=<path to .tflite model file> custom=Delegate:External,ExtDelegate Lib:libneutron_delegate.so`	-
VX Delegate (applicable for supported i.MX 8)	`framework=tensorflow-lite model=<path to .tflite model file> custom=Delegate:External,ExtDelegate Lib:libvx_delegate.so`	0: NPU 1: GPU
Ethos-U Delegate (i.MX 93 only)	`framework=tensorflow-lite model=<path to .tflite model file> custom=Delegate:External,ExtDelegate Lib:libethosu_delegate.so`	-

Table 8. TVM engine

Tensor filter properties	USE_GPU_INFERENCE env variable
`framework=tvm model=<path to .so model library> custom=num_input_tensors:<number of input tensors>` where `<number of input tensors>` is typically **1**.	**0**: NPU **1**: GPU Relevant for models compiled to use OpenVX

8.1 Object detection pipeline example

This section provides implementation details for an object detection pipeline running on i.MX 8M Plus. Additional pipeline examples targeting more use-cases and i.MX platforms can be found in Section 8.2.

In this example, the following pipeline will be implemented leveraging all the compute backend available on i.MX 8M Plus to build an object detection scenario.

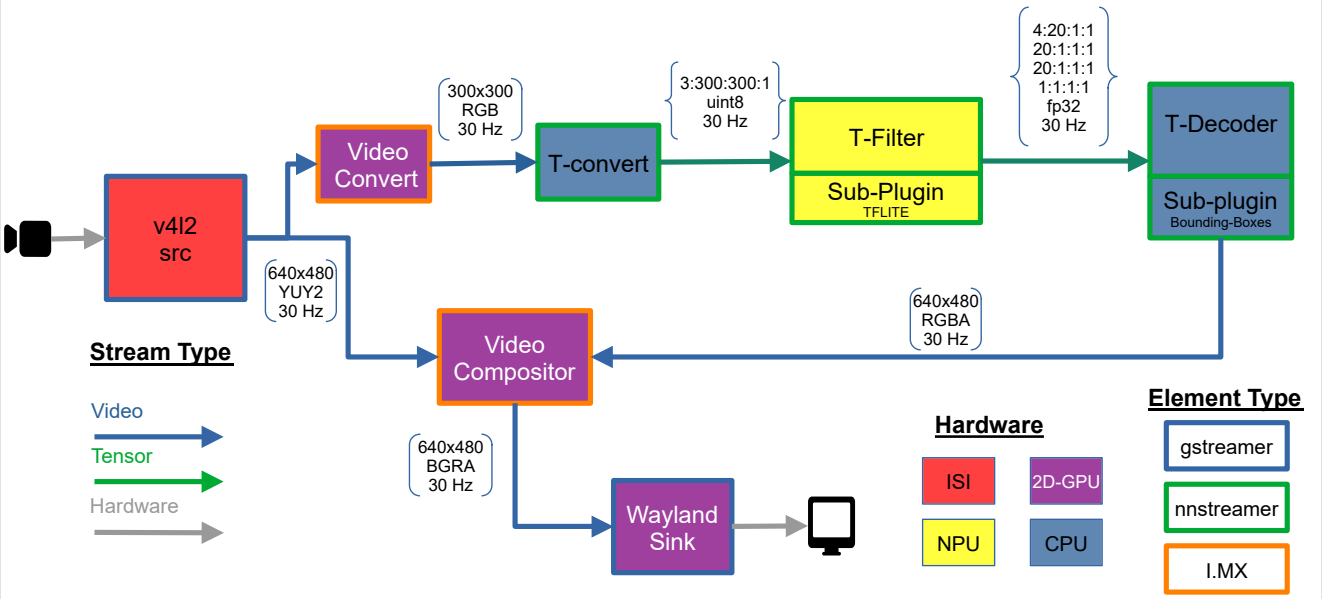

Figure 22. NNStreamer object detection example pipeline

On the target, download the trained neural network from google coral github site, and export the filenames to bash environment variables:

```
root:~# wget https://github.com/google-coral/test_data/raw/master/
ssd_mobilenet_v2_coco_quant_postprocess.tflite
root:~# wget https://github.com/google-coral/test_data/raw/master/coco_labels.txt
root:~# export MODEL=$(pwd)/ssd_mobilenet_v2_coco_quant_postprocess.tflite
root:~# export LABELS=$(pwd)/coco_labels.txt
```

Then builds and executes the GStreamer pipeline:

```
root:~# gst-launch-1.0 --no-position v4l2src device=/dev/video3 ! \
video/x-raw,width=640,height=480,framerate=30/1 ! \
tee name=t t. ! queue max-size-buffers=2 leaky=2 ! \
imxvideoconvert_g2d ! \
video/x-raw,width=300,height=300,format=RGBA ! \
videoconvert ! video/x-raw,format=RGB ! \
tensor_converter ! \
tensor_filter framework=tensorflow-lite model=${MODEL} \
custom=Delegate:External,ExtDelegateLib:libvx_delegate.so ! \
tensor_decoder mode=bounding_boxes option1=mobilenet-ssd-postprocess option2=${LABELS} \
option3=0:1:2:3,50 option4=640:480 option5=300:300 ! \
mix. t. ! queue max-size-buffers=2 ! \
imxcompositor_g2d name=mix latency=30000000 min-upstream-latency=30000000
 sink_0::zorder=2 sink_1::zorder=1 ! waylandsink
```

Note: *Hit CTRL+C keystroke to halt the execution if necessary.*

8.2 NXP NNStreamer pipeline examples

Pipelines targeting i.MX platforms are published to provide working examples for different use cases and implementation options.

Those examples are hosted on the GitHub server in a dedicated tree:

https://github.com/nxp-imx/nxp-nnstreamer-examples

Refer to the included README documentation for pipelines descriptions and instructions for dependencies download (models, metadata) and execution.

The following table lists the features covered by pipeline examples.

Table 9. Features of NXP NNStreamer examples

Category	Engine	Platform	Implementation
Object detection: MobileNet SSD V2, Yolov4-tiny Image classification: MobileNet V1 Image segmentation: DeepLab V3 Pose detection: MoveNet Face detection: UltraFace Face recognition: FaceNet512 Emotion detection: DeepFace Depth estimation: Midas V2	TensorFlow Lite	i.MX 8M Plus i.MX 93 i.MX 95	Shell script (gst-launch) Python, C++ Custom Python tensor_filter

8.3 Pipeline profiling

NNStreamer team developed NNShark, a profiling tool based on GstShark, to monitor several pipeline metrics useful to assess the SoC hardware usage.

NNShark can be used on the i.MX 8M Plus only, where specific metrics were added:

- 2D GPU (GC520L) utilization load
- 3D GPU (GC7000UL) utilization load
- NPU (GC8000) utilization load
- SoC masters bandwidth, as reported by Linux kernel perf tool
- Additionally, power domain consumption, as reported by power measurement tool (PMT) if the power measurement evaluation kit is available to the user.

Considering the complex GPU/NPU architecture involving concurrent stages, their reported utilization loads shall be considered as an order of magnitude and might not precisely reflect each individual stage's status.

Note:

For the source code demo location see the nnshark repository.

8.3.1 Enable profiling with NNShark

It is recommended to connect to the target through SSH as the NNShark UI refresh rate might not render well on the serial console.

Enable NNShark profiling through environment variables:

```
root:~# export GST_DEBUG="GST_TRACER:7"
root:~# export GST_TRACERS="live"
```

To get GPU usage measurements, disable power saving in the GPU driver (galcore) using command line Linux kernel parameters. You can manually edit the bootargs U-Boot variable before executing the boot command. Add the following parameters:

```
galcore.gpuProfiler=1 galcore.powerManagement=0
```

Then run the previous gst-launch command line, and the following screen should now be displayed on your terminal screen. You can scroll through all the pipeline elements with up/bottom direction key to select the desired element and display its connections with other pipeline elements.

You can select the element pads with left/right direction keys to highlight its connection to other elements' pads.

On this example, the tensor filter has an average processing time of 21.64 ms and its sink orange highlighted pad is connected to source pad of tensorconverter0 element (green highlighted).

Press 'q' or 'Q' to exit the profiling tool and return to the shell terminal. You can quit the application as previously explained through CTRL+C.

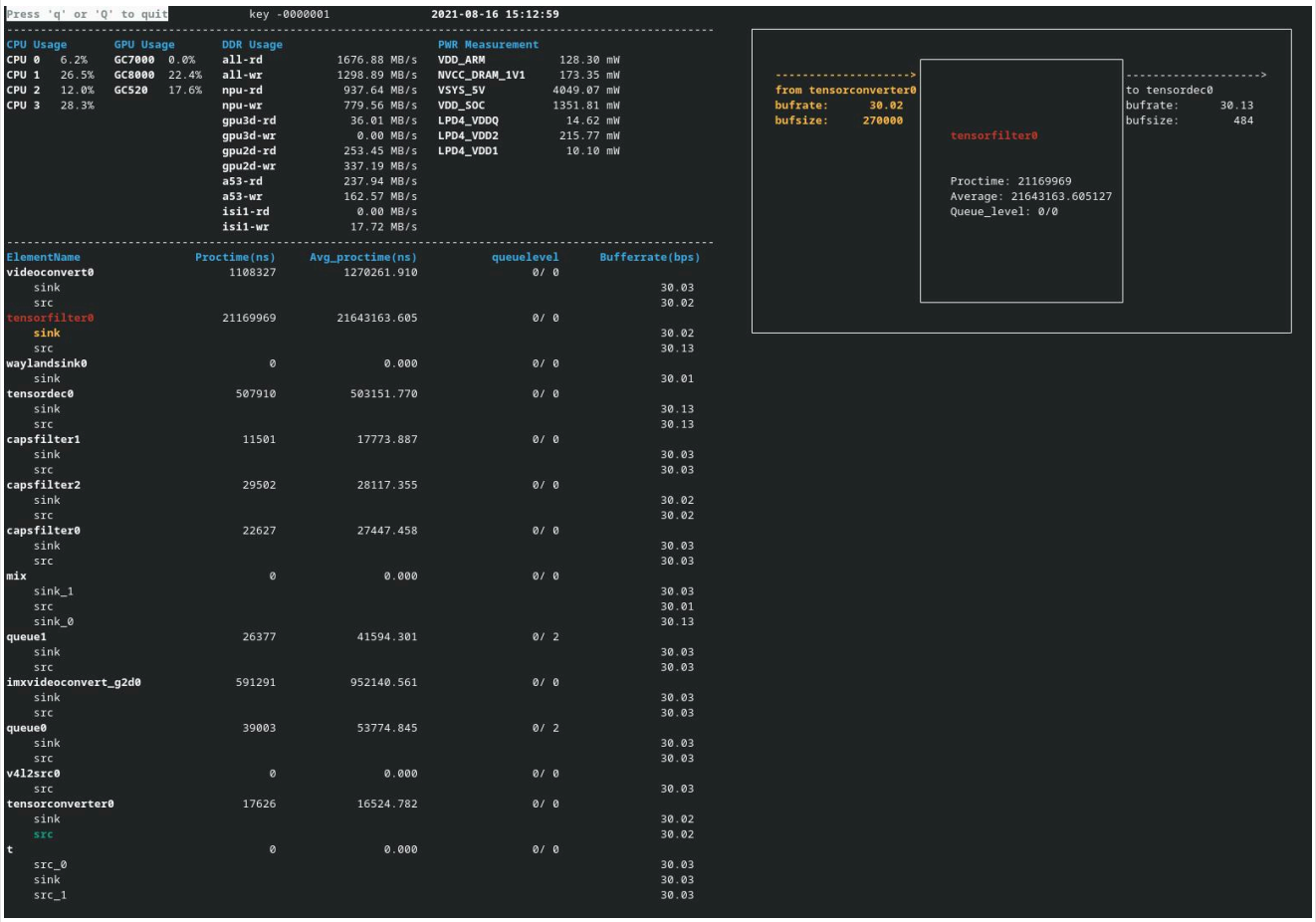

Figure 23. NNShark i.MX 8M Plus example screenshot

8.3.2 Adding power measurement to NNShark

On the desktop PC connected to the power measurement evaluation kit, execute the power measurement tool (PMT) in server mode such as the power measurements are collected and available on 65432 TCP/IP port.

```
user@localhost:pmt# python3 main.py server -b imx8mpevkpwra0 -p 65432
```

On the target, export the desktop PC ip address (192.168.1.99 for this example):

```
root:~# export GST_TRACERS_PWR_SERVER_IP=192.168.1.99
```

Note: *The user can run the NNShark without the power measurement kit.*

8.3.3 Known issues and limitations

In case perf reports inconsistent high numbers, this means that a perf process is still running in background of the previous run. If so, you must terminate manually their execution.

For your convenience, the below command can be used:

```
root:~# kill -9 $(ps -ef | grep nnshark-perf-ddr.sh | grep -v grep | tr -s ' ' |
 cut -d ' ' -f 2)
```

9 eIQ Demos

9.1 TensorFlow Lite Demos for i.MX 93

This section provides implementation details for several TensorFlow Lite demos running on i.MX 93.

TensorFlow Lite demos (binaries) are located at: `/usr/bin/eiq-examples-git`.

Binary models are not located in the image because of the size. Before running the demos, these files should be downloaded to the device:

```
$ cd /usr/bin/eiq-examples-git
$ python3 download_models.py
```

Note: *This script is downloaded from GitHub and Google drive. Make sure the device network is correctly configured and can access the Internet.*

9.1.1 Image classification demo

Note: *All the demos require X11 to display, so use the XWayland distro images.*

This demo performs image classification using a pretrained mobilenet-v1 network. Demo dependencies are from:

`/usr/bin/eiq-examples-git/image_classification`.

- `grace_hopper.bmp`
- `label_image.py`
- `labels.txt`

The demo network model dependencies:

- `mobilenet_v1_1.0_224_quant.tflite`

Run the Python example with the image input from the default location:

```
$ cd /usr/bin/eiq-examples-git/image_classification
$ python3 label_image.py -i grace_hopper.bmp -l labels.txt
0.874510: military uniform
0.031373: Windsor tie
0.015686: mortarboard
0.011765: bulletproof vest
0.007843: bow tie
```

```
time: 4.126ms
```

9.1.2 SSD object detection demo

The SSD object detection demo performs object detection using the Single-Shot multibox Detection (SSD) detector. It detects objects on camera, video, or image. Demo dependencies are from: `/usr/bin/eiq-examples-git/object_detection`.

- `cars0.bmp`
- `labels.py`
- `main.py`

The demo network model dependencies:

- `ssd_mobilenet_v1_quant.tflite`

Run the Python example with the image input from the default location:

```
$ cd /usr/bin/eiq-examples-git/object_detection
$ python3 main.py -i cars0.bmp
rectangle:(640,493),(1756,881) label:car
rectangle:(1470,466),(1947,694) label:car
rectangle:(803,462),(846,502) label:car
rectangle:(733,451),(788,493) label:car
rectangle:(573,473),(705,565) label:car
rectangle:(608,465),(679,519) label:car
rectangle:(203,455),(271,596) label:person
rectangle:(910,461),(956,500) label:car
rectangle:(1020,453),(1076,497) label:person
```

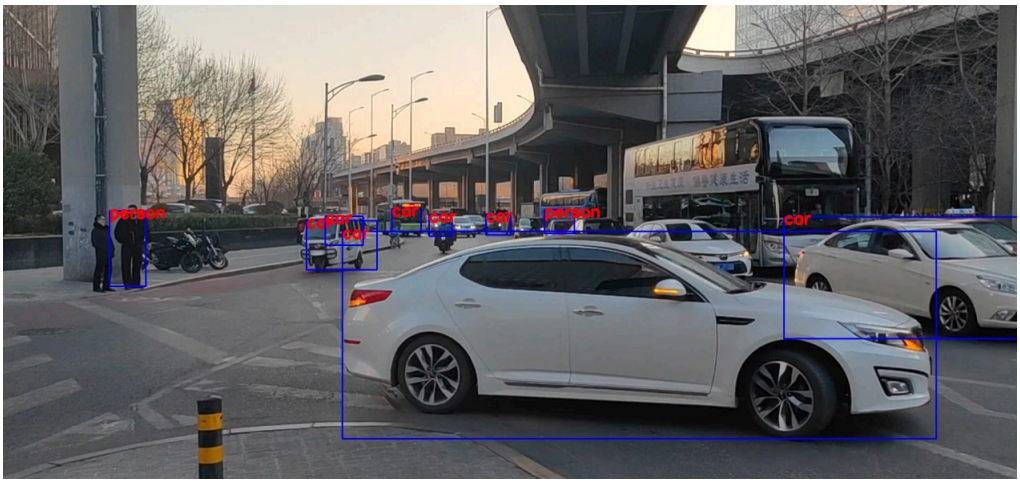

Figure 24. SSD object detection demo

Run the Python example with the live camera connected to port 0.

```
$ python3 main.py -i /dev/video0
```

Note: *Choose the right port where the camera is currently connected. Use the `v4l2-ctl --list-devices` command to check it.*

9.1.3 Hand gesture detection demo

This application demonstrates hand detection and gesture detection. It detects objects on camera, video, or image. Demo dependencies are from: `/usr/bin/eiq-examples-git/gesture_detection`.

- `anchors.csv`
- `hand0.bmp`
- `hand_tracker.py`
- `main.py`

The demo network model dependencies:

- `palm_detection_builtin_256_integer_quant.tflite`
- `hand_landmark_3d_256_integer_quant.tflite`

Run the Python example with the image input from the default location:

```
$ cd /usr/bin/eiq-examples-git/gesture_detection
$ python3 main.py -i hand0.bmp
```

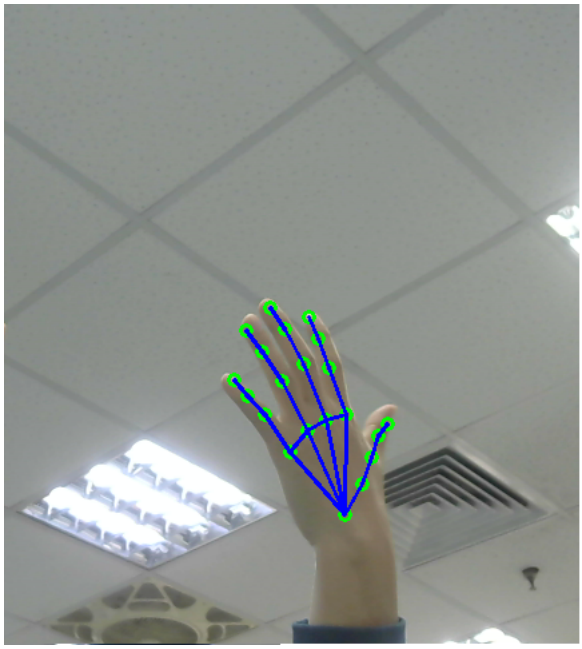

Figure 25. Hand gesture detection demo

Run the Python example with the live camera connected to port 0.

```
$ python3 main.py -i /dev/video0
```

Note: *Choose the right port where the camera is currently connected. Use the* `v4l2-ctl --list-devices` *command to check it.*

9.1.4 Face recognition demo

This application is a demonstration for real-time face recognition. It uses pretrained yoloface model for face detection, and facenet model to calculate face landmark. The demo supports the live camera input only.

Demo dependencies are from: `/usr/bin/eiq-examples-git/face_recognition`.

- face_database.py
- face_detection.py
- face_recognition.py
- main.py

The demo network model dependencies:

- yoloface_int8.tflite
- facenet_512_int_quantized.tflite

Before running the demo, connect a keyboard to the board.

1. Run the Python example with the live camera connected to port 0.

```
$ cd /usr/bin/eiq-examples-git/face_recognition
$ python3 main.py -i /dev/video0
```

Note: *Choose the right port where the camera is currently connected. Use the* `v4l2-ctl --list-devices` *command to check it.*

2. Add a name to the face database.
   Face the camera and press ‘a’ on the keyboard, which is connected to the board, and then input a new name.
3. Delete the name from the face database.
   Press ‘d’ on the keyboard, which is connected to the board, and then input the name.

10 Release Notes

10.1 Known issues and limitations

- Inline model compilation on i.MX 95 is not available.
- Hardware Accelerators on i.MX 8 does not support layers with dynamic shapes.
- The NPU on i.MX 8M Plus is not optimized for models with dynamic weights. The layers with dynamic weights (for example, in the FullyConnected layer) are computed significantly slower.
- Some of the links for the models in the `download_models.py` script from Section 9.1 are no longer available.

10.2 Release notes for LF6.12.49_2.2.0

i.MX 95/943:

- Upgraded Neutron Software Stack to v2.2.2.
- Integrated the Neutron driver dynamic library `libNeutronDriver.so`.
- Fixed potential Neutron SMMU address access errors when Neutron encounters an error.

10.3 Release notes for LF6.12.34_2.1.0

General:

- Neutron Converter upgraded to v2.1.3, will be released in eiq-toolkit v1.17.
- Neutron runtime upgraded to v2.1.3 as Neutron software stack.

TensorFlow Lite:

- Upgraded to 2.19.0.
- Support int8 data type for `detection_postprocess`.

ONNX Runtime:

- Experimental support of CNN model on Neutron NPU.
- Bug fix for VSINPU EP on i.MX 8M Plus.

i.MX 93:

- Arm Vela Compiler updated to version 4.3.0.
- Added Vela model profiling tool (`vela-prof`).
- Supports Elu OP by Vela.

i.MX 95, i.MX 943:

- Upgraded Neutron Software Stack to v2.1.3.
- The Neutron NPU runs in interrupt (IRQ) mode by default, generating an IRQ to signal the completion of an inference task. To switch to polling mode, add `neutron.use_irq=0` to the U-Boot boot arguments.

10.4 Release notes for LF6.12.20_2.0.0

General:

- For Neutron model conversion, eIQ Toolkit 1.16 includes newer Neutron converter tool, which optimizes the model conversion speed.
- LLM model support with ONNX Runtime (CPU EP and Neutron EP) and LiteRT (CPU).

LiteRT:

- v1.2.0 is supported as an experimental feature for all devices, the same feature coverage as TFLite runtime.
- Only Python API is available for example development.

ONNX Runtime:

- Upgraded to 1.22.0-pre-release (April 1st).
- Supports Neutron EP (MatMul operators) for the LLM model running on i.MX 95 as an experimental feature.
- Supports VsiNPU EP for CNN models running on i.MX 8 series.
- Supports MatMulNBits for CPU Execution provider and Neutron Execution provider, enabling matrix multiplication with weights quantized to 8 bits and 4 bits.
- Integrated KleidiAI library into ONNX Runtime MLAS for enhanced performance on Arm architectures.

i.MX 93:

- Arm Vela Compiler updated to version 4.2.0.
- Ethos-U firmware supports build from source. It is available since MCU SDK25.06.00-pvw2.

i.MX 95:

- Neutron Software Stack upgraded.
- NPU Power Management feature supports clock gating and power gating.

i.MX 943:

- Neutron Software Stack supported.

10.5 Release notes for LF6.6.52_2.2.0

i.MX 95:

- Neutron Software Stack upgraded.

- Not support inline compilation, only the converted TensorFlow Lite model is interpreted by Neutron Runtime.
- Pre-compiled model specific binary in NeutronOp with significant performance improvement for inference.

10.6 Release notes for LF6.6.36_2.1.0

General:

- Arm Compute Library is removed since this release.

TensorFlow Lite:

- Upgraded to 2.16.2.
- Fixed missing PIL module when running the `label_image` Python example (`label_image.py`)
- Modified Flex delegate build onto a 2-stage process.

ONNX Runtime

- Fixed compilation with Yocto SDK.

i.MX 8M Plus:

- VX Delegate update and bug fixes
- TIM-VX update, and internal OVXLIB updated to 1.2.14.

i.MX 93:

- Arm Vela Compiler updated to version 3.12.
- Ethos-U software updated to 24.05.

i.MX 95:

- Neutron Software Stack upgraded.

10.7 Release notes for LF6.6.23_2.0.0

General:

- Added support for i.MX 91.

TensorFlow Lite:

- Upgraded to 2.15.0.
- Added GPU Delegate for i.MX 95. The GPU delegate is available in the C++ API.

ONNX Runtime

- Upgraded to 1.17.1.

PyTorch

- PyTorch framework not available by default in the BSP. Can be deployed using pip from the PyPI registry.

i.MX 8M Plus:

- VX Delegate update and bug fixes
- TIM-VX update and bug fixes.

i.MX 93:

- Arm Vela Compiler updated to version 3.11.
- Ethos-U software updated to 24.02.

i.MX 95:

- Added support to offload ML workload on the on-chip Arm Mali G310 GPU with TensorFlow Lite, using the GPU Delegate.
- Added support for eIQ Neutron Neural Processing Unit using the inline compilation.
- Full NPU acceleration for mobilenetv1 and mobilenetv2 models or models with similar operators. Expect other popular CNN models in the upcoming releases.

10.8 Release notes for LF6.6.3_1.0.0

General:

- Initial support for the i.MX 95 platform.

TensorFlow Lite:

- Upgraded to 2.14.0.
- Added helper script to generate reduced-size Flex Delegate Bazel artifacts.

i.MX 8M Plus:

- VX Delegate update and bug fixes
- TIM-VX update and bug fixes.

i.MX 93:

- Arm Vela Compiler updated to version 3.10.
- Ethos-U software updated to 23.11.

i.MX 95:

- Added eIQ for i.MX 95.
- Added support for eIQ Neutron Neural Processing Unit using offline compilation. The compiler is available in the eIQ Toolkit.

10.9 Release notes for LF6.1.55_2.2.0

General:

- Model Runner was removed from Linux BSP.
  The eIQ Toolkit deploys the compatible Model Runner instance automatically.

TensorFlow Lite:

- Upgraded to 2.12.1.

ONNX Runtime:

- Upgraded to 1.16.1.
- NNAPI execution provider support was removed.

i.MX 8M Plus:

- VX Delegate update and bug fixes.
- TIM-VX update and bug fixes.

i.MX 93:

- Arm Vela Compiler updated to version 3.9.
- Ethos-U software updated to 23.08.

eIQ Demos:

- Removed the support for AWS end-to-end SageMaker demo.

10.10 Release notes for LF6.1.36_2.1.0

TensorFlow Lite

- Upgraded to 2.11.1.
- Bug fixes.
- Added Flex Delegate support, including the binary size reduction described here: www.tensorflow.org/lite/guide/reduce_binary_size

VX Delegate

- Synchronized with TensorFlow 2.11.1.
- Bug fixes.

DeepViewRT

- DeepViewRT inference engine was removed.

10.11 Release notes for LF6.1.22_2.0.0

VX Delegate

- Bug fixes.
- Added support for EmbeddingLookup, Cast, and BroadcastTo.
- Fixed performance on MobilenetV1, MobileNetV2, VGG16, VGG19, and NasNet Mobile.

ONNX Runtime

- Upgraded to 1.13.1.
- VSI-NPU Execution provider is obsolete and was removed from ONNX Runtime.
- Added support to run dynamic-shape models using NNAPI Execution Provider.

PyTorch

- Upgraded to 2.0.0.

DeepViewRT

- DeepViewRT inference engine is deprecated and will be removed in the future.

i.MX 93

- Arm Vela Compiler: Updated to version 3.7.

10.12 Release notes for LF6.1.1_1.0.0

TensorFlow Lite

- Upgraded to 2.10.0.
- Deprecated Ethos-U Custom operator on i.MX 93. The preferred way for models with Ethos-U Operator is using the Ethos-U Delegate.

VX Delegate

- Bug fixes.
- Added support for UnidirectionalSequenceLSTM, BidirectionalSequenceLSTM, Shape, HashtableLookup operators.

- Updated C++ Standard to C++17.
- Fixed TransposeConv2d operator.
- Known issue: Decreased performance on MobilenetV1, MobileNetV2, VGG16, VGG19, and NasNet Mobile.

i.MX 93

- Arm Vela Compiler: Updated to version 3.6.
- Introduced Ethos-U Delegate for i.MX 93.

eIQ Demos

- Added TensorFlow Lite demo application for i.MX 93.

10.13 Release notes for LF5.15.71_2.2.0

TensorFlow Lite

- Added option to inference diff tool to compare the inference to reference model. This enables validation of the model on i.MX 93 accelerated by Ethos-U NPU.
- Ethos-U: Enables getting PMU counters from the NPU.
- Ethos-U: Uses one flash and Arena buffer for multiple Ethos-U Operators.

VX Delegate

- Bug fixes.
- Fixed failures with TensorFlow Lite kernel tests: `expand_dims`, `LRN`, `strided-slice`, `resize`, `maximum`, `minimum`, and `conv3d`.

i.MX 93

- NPU profiling support.
- Ethos-u-driver-stack: Updated to version 22.08.
- Arm Vela Compiler: Updated to version 3.5.

10.14 Release notes for LF5.15.52_2.1.0

- General
  - Added support for i.MX 93 platform, including NN acceleration on Ethos-U NPU.
- TensorFlow Lite
  - TensorFlow Lite updated from version from 2.8.0. to 2.9.1. For details, see `RELEASE.md` in the source code repository.
  - Added support for Ethos-U HW acceleration for i.MX 93 platform.
- VX Delegate
  - Added support for ReverseV2, UnidirectionalSequenceLSTM and Unpack operators.
  - Fixed bug in reshape for `inception_v1_224_quant` model.
  - Fixed Yolo-V4-tiny.

  - Other minor bug fixes.
- TIM-VX
  - TIM-VX updated from 1.1.42 to 1.1.50.
- Arm Compute Library
  - Arm Compute Library updated from 21.08 to 22.05.
- DeepViewRT
  - DeepViewRT updated from 2.4.42. to 2.4.46.
- eIQ Examples
  - Resolved dependency issue due to Yocto BSP upgrade: AWS end-to-end SageMaker demo can be built with latest Yocto BSP (LF5.15.52_2.1.0).

10.15 Release notes for LF5.15.32_2.0.0

- ArmNN inference engine was removed from eIQ.
- TensorFlow Lite
  - TensorFlow Lite was updated from version 2.6.0 to 2.8.0. For details, see `RELEASE.md` in the source code repository.
  - Features and improvements:
    - Fixed evaluation tools build with Yocto SDK. Prior to build of the evaluation tools with CMake, it is necessary to build and install the required tooling (protobuf compiler - `protoc`). Use the `CMakeLists.txt` from `tensorflow/lite/tools/cmake/native_tools/`.
- ONNX Runtime
  - Features and improvements:
    - ArmNN and ACL Execution providers were removed from eIQ.
    - VSI_NPU backend is deprecated and will be removed in the future.
    - NNAPI execution provider is experimental feature.
- TIM-VX
  - TIM-VX was updated from 1.1.37 to 1.1.42.
- DeepViewRT
  - DeepViewRT was updated from 2.4.37 to 2.4.42.

10.16 Release notes for LF5.15.5-1.0.0

- Arm NN inference engine is deprecated in this release and will be removed in the future.
- NNAPI Delegate of TensorFlow Lite and NNAPI Execution Provider of ONNX Runtime is deprecated and will be removed in the future. For leveraging ML model acceleration use VX Delegate instead.
- TensorFlow Lite:
  - Features and improvements:
    - Fixed unit test build with TensorFlow Lite static library.
    - Support FullyConnected layer with implicit bias in VX Delegate.
    - Fix bug in `stride_slice` if `end_dim` set as -1 in VX Delegate.
    - Other minor fixes.
- ONNX Runtime:
  - Features and improvements:
    - Version update from 1.8.2 to 1.10.0.
    - Updated to GCC11 toolchain.
    - NNAPI Execution Provider is ported from 1.5.3 (does not contain latest 1.10.0 updates) and it is considered experimental. We do not suggest using it in production.
    - Arm NN and ACL Execution providers are deprecated and will be removed in the future

- PyTorch upgraded to version 1.9.1.
- TIM-VX:
  - Features and improvements:
    - Version update from 1.1.34 to 1.1.37.
    - DMA Buffer support.
    - Support for additional operators (SVDF, GlobalPool2D, AdaptivePool2D, Erf, grouped Conv1D, Signal Frame, RNN Cell, One Hot).
    - Support Layout inference for additional operators (Batch Norm, Transpose, Fully Connected with no explicit bias).
- DeepViewRT:
  - Features and improvements:
    - Version update from 2.4.36 to 2.4.37
    - C and Python API for NPU support are available.
    - Align modelrunner plugin with TFLite/Arm NN/ONNX Runtime inference engine.
  - Issues and limitations:
    - Bug fix for deepview-rt library and example codes.

10.17 Release notes for LF5.10.72-2.2.0

- TensorFlow Lite:
  - Upgraded to version 2.6.0.
  - VX Delegate changed to external delegate.
  - Optimization of the PCQ Transpose Convolution operator on the NPU hardware accelerator.
  - Python API support external Delegates:
    - With this change, the `label_image.py` Python example support the use of external delegates with arguments. See the help for more information.
    - Python API supports using external delegate via the `tflite.load_delegate()` call.
    - NNAPI delegate not available in Python API. For the model acceleration on the HW accelerator, the VX delegate can be used:

```
ext_delegate = [ tflite.load_delegate("/usr/lib/libvx_delegate.so") ]
                                          interpreter
 = tflite.Interpreter(model_path=args.model_file,
 experimental_delegates=ext_delegate, num_threads=args.num_threads)
```

- Arm Compute Library:
  - Features and improvements:
    - Major version update from 21.02 to 21.08.
  - Issues and limitations:
    - Only the CPU-accelerated NEON backend is being built. Use Arm NN with the VSI NPU backend to leverage acceleration on the GPU or the NPU.
- Arm NN:
  - Features and improvements:
    - Major version update from 21.02 to 21.08.
    - TensorFlow Parser, Caffe Parser and Quantizer were removed and are no longer available. Only ONNX Parser, TensorFlow Lite Parser and Arm NN Delegate for TF Lite are now available to load *.tflite* and *.onnx* models.

  - See full list of changes added by the community.
- Issues and limitations:
  - Only ACL NEON backend is being built. Use the VSI NPU Backend instead of ACL OpenCL to leverage acceleration on the GPU or the NPU.
  - There are significant performance optimizations for the NPU to TransposeConv2D which are not supported in the VSI NPU backend. If your model uses TransposeConv2D heavily try to use TF Lite with VXDelegate instead.

- ONNX Runtime:
  - Features and improvements:
    - Minor version update from 1.8.1 to 1.8.2.
    - Experimental Python API enablement including support for all available Execution Providers (CPU, ACL, Arm NN, NNAPI, VSI NPU).
    - Added `/usr/bin/onnxruntime-1.8.2/onnxruntime_peft_test`. Use this instead of `onnx_test_runner` to measure performance of your model.
    - Fixed verbose logging during inference on NPU.
    - Updated ACL and Arm NN Backends to leverage ACL and Arm NN 21.08.
    - All ONNX Runtime artifacts are being installer to `/usr/bin/onnxruntime-1.8.2` instead of `/usr/bin`.
    - See full list of changes added by the community.
  - Issues and limitations:
    - There are significant performance optimizations for the NPU to TransposeConv2D which are not supported in the VSI NPU Execution Provider. If your model uses TransposeConv2D heavily try to use TF Lite with VXDelegate instead.
    - Running SqueezeNet with the NNAPI execution provider produces incorrect results.
- DeepViewRT:
  - Features and improvements:
    - Minor version update from 2.4.30 to 2.4.36.
    - C API for NPU support is available.
    - Performance optimization for DeepViewRT CPU.
    - Bug fix for shuffle layer.
  - Issues and limitations:
    - `nn_tensor_load_file_ex` is one convenience function and not well optimized.

11 List of Used Variables

The following table provides the summary of used variables described in this document for the particular inference engine. Use the `export` command to apply these variables.

Table 10. System variables summary

Variable name	Description
CNN_PERF	0: Disable (default) 1: Prints the execution time for each operation (requires VIV_VX_DEBUG_LEVEL=1). If VIV_VX_PROFILE=1 is set, the default value is 1.
NN_EXT_SHOW_PERF	0: Disable (default) 1: Shows more profiling details (requires VIV_VX_DEBUG_LEVEL=1)
PATH_ASSETS	Sets the export path for user assets.

Table 10. System variables summary*...continued*

Variable name	Description
USE_GPU_INFERENCE	Selection between the 3D GPU (1) and the NPU (otherwise).
VIV_VX_CACHE_BINARY_GRAPH_DIR	Specifies the path of the cached NBG. Default is the current work directory.
VIV_VX_DEBUG_LEVEL	0: Disable (default) 1: Prints the debug information of driver on the console. Generally, this environment variable is used together with other environment variables to print logs.
VIV_VX_ENABLE_CACHE_GRAPH_BINARY	0: Disable (default) 1: Enables graph cache mode. The network loads the NBG file to run if the cached NBG file exists. Otherwise, it generates an NBG file. It can save the time for the verification stage.
VIV_MEMORY_PROFILE	0: Disable (default) 1: Prints the memory footprint of the system (CPU) and GPU (VIP) (requires VIV_VX_DEBUG_LEVEL=1)
VIV_VX_PROFILE	0: Disable (default) 1: Prints the DDR read and write bandwidth, AXI_SRAM read and write bandwidth, and the cycle count of VIP execution. The counter is per-node-process (requires VIV_VX_DEBUG_LEVEL=1). 2: Prints the DDR read and write bandwidth, AXI_SRAM read and write bandwidth, and the cycle count of VIP execution. The counter is per-graph-process (requires VIV_VX_DEBUG_LEVEL=1).

12 Neural Network API Reference

The neural-network operations and corresponding supported API functions are listed in the following table. See also Section 2.2.3 for details about supported operators.

Table 11. Neural-network operations and supported API functions

Op Category/Name	TensorFlow Lite 2.8.0	ONNX 1.22.0
Activation		
elu	ELU	Elu
gelu	-	Gelu
floor	Floor	Floor
leakyrelu	-	LeakyReL
prelu	PRELU	PreLu
relu	RELU	ReLu
relu1	RELU1	-
relu6	RELU6	-
Hard_swish	HARD_SWISH	-
rsqrt	RSQRT	-
selu	-	Selu
sigmoid	LOGISTIC	Sigmoid
softplus	-	Softplus

Table 11. Neural-network operations and supported API functions...*continued*

Op Category/Name	TensorFlow Lite 2.8.0	ONNX 1.22.0
softmax	SOFTMAX	Softmax
softrelu	-	-
sqrt	SQRT	Sqrt
tanh	TANH	TanH
bounded	-	-
linear	-	-
Dense Layers		
dense	-	-
Element Wise		
abs	ABS	Abs
add	ADD	Add
clip_by_value	-	Clip
div	DIV	Div
equal	EQUAL	Equal
exp	EXP	Exp
log	LOG	Log
greater	GREATER	Greater
greater_equal	GREATER_EQUAL	-
less	LESS	Less
less_equal	LESS_EQUAL	-
logical_and	LOGICAL_AND	And
logical_or	LOGICAL_OR	Or
minimum	MINIMUM	Min
maximum	MAXIMUM	Max
multiply	MUL	Mul
negative	NEG	Neg
not_equal	NOT_EQUAL	-
pow	POW	POW
select	SELECT	-
square	-	-
sub	SUB	Sub
where	-	Where
Image Processing		
resize_bilinear	RESIZE_BILINEAR	Unsample
resize_nearest_neighbor	RESIZE_NEAREST_NEIGHBOR	Resize
Matrix Multiplication		

Table 11. Neural-network operations and supported API functions*...continued*

Op Category/Name	TensorFlow Lite 2.8.0	ONNX 1.22.0
fullconnect	FULLY_CONNECTED	-
matrix_mul	-	-
MatMulInteger	-	MatMulInteger
MatMulIntegerToFloat	-	MatMulIntegerToFloat
MatMulNBits	-	MatMulNBits
QLinearMatMul	-	QLinearMatMul
Normalization		-
batch_normalize	-	BatchNormalization
instance _normalize	-	InstanceNormalization
l2normalize	L2_NORMALIZATION	-
localresponsenormalization	LOCAL_RESPONSE_ NORMALIZATION	LRN
Reshape		
batch2space	BATH_TO_SPACE_ND	-
concat	CONCATENATION	Concat
depth_to_space	DEPTH_TO_SPACE	DepthToSpace
expanddims	EXPAND_DIMS	-
flatten	-	-
gather	GATHER	Gather
pad	PAD	Pad
permute	TRANSPOSE	Transpose
reducemean	MEAN	ReduceMean
reducesum	REDUCE_SUM	ReduseSum
gathernd	-	GatherND
reducemax	REDUCE_MAX	ReduceMax
reducemin	REDUCE_MIN	ReduceMin
reduceproduct	-	-
reshape	RESHAPE	Reshape
reverse	-	ReverseSequence
slice	SLICE	Slice
space2batch	SPACE_TO_BATCH_ND	-
split	SPLIT	Split
squeeze	SQUEEZE	Squeeze
transpose	-	Transpose
strided_slice	STRIDED_SLICE	-
unstack	-	-
RNN		

Table 11. Neural-network operations and supported API functions*...continued*

Op Category/Name	TensorFlow Lite 2.8.0	ONNX 1.22.0
gru	-	GRU
lstm	UNIDIRECTIONAL_ SEQUEENCE_LSTM	-
lstmunit	LSTM	LSTM
rnn	RNN	-
Sliding Window		
avg_pool	AVERAGE_POOL_2D	AveragePool
convolution	CONV_2D	Conv
deconvolution	TRANSPOSE_CONV	ConvTranspose
depthhwise_convolution	DEPTHWISE_CONV_2D	-
Log_softmax	LOG_SOFTMAX	Logsoftmax
l2pooling	L2_POOL_2D	-
max_pool	MAX_POOL_2D	MaxPool
Others		
argmax	ARGMAX	ArgMax
argmin	ARGMIN	ArgMin
dequantize	DEQUANTIZE	DequantizeLinear
quantize	QUANTIZE	QuantizeLinear
roi_pool	-	-
shuffle_channel	-	-
tile	TILE	Tile
svdf	SVDF	-
embedding_lookup	EMBEDDING_LOOKUP	-
cast	CAST	Cast
ssd	-	-

13 OVXLIB Operation Support with GPU

This section provides a summary of the neural network OVXLIB operations supported by the NXP Graphics Processing Unit (GPU) IP with hardware support for OpenVX and OpenCL and a compatible Software stacks. OVXLIB operations are listed in the following table.

The following abbreviations are used for format types:

- **asym-u8**: asymmetric_affine-uint8
- **asym-i8**: asymmetric_affine-int8
- **fp32**: float32
- **pc-sym-i8**: perchannel_symmetric_int8
- **fp16**: float16
- **bool8**: bool8
- **int16**: int16

- **int32**: int32

Table 12. OVXLIB operation support with GPU

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
Basic Operations					
VSI_NN_OP_CONV2D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_CONV1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_DEPTHWISE_ CONV1D	asym-u8	asym-u8	asym-u8	✓	
	asym-i8	asym-i8	asym-i8	✓	
VSI_NN_OP_DECONVOLUTION1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_DECONVOLUTION	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_FCL	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_GROUPED_CONV1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_GROUPED_CONV2D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
Activation Operations					
VSI_NN_OP_ELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_HARD_SIGMOID	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SWISH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LEAKY_RELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_PRELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RELUN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RSQRT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SIGMOID	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SOFTRELU	asym-u8		asym-u8	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SQRT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_TANH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ABS	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CLIP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_EXP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LOG	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_NEG	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MISH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
VSI_NN_OP_LINEAR	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ERF	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SOFTMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LOG_SOFTMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SQUARE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SIN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
Elementwise Operations					
VSI_NN_OP_ADD	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SUBTRACT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MULTIPLY	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_DIVIDE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MAXIMUN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MINIMUM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_POW	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_FLOORDIV	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MATRIXMUL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RELATIONAL_OPS	asym-u8		bool8	✓	✓
	asym-i8		bool8	✓	✓
	fp32		bool8	✓	✓
	fp16		bool8	✓	✓
	bool8		bool8	✓	✓
VSI_NN_OP_LOGICAL_OPS	bool8		bool8	✓	✓
VSI_NN_OP_LOGICAL_NOT	bool8		bool8	✓	✓
VSI_NN_OP_SELECT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
	bool8		bool8	✓	✓
VSI_NN_OP_ADDN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
Normalization Operations					
VSI_NN_OP_BATCH_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LRN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LRN2	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_L2_NORMALIZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_L2NORMALZESCALE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LAYER_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp16		fp16	✓	✓
VSI_NN_OP_GROUP_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_BATCHNORM_SINGLE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MOMENTS	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
Reshape Operations					
VSI_NN_OP_EXPAND_ BROADCAST	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SLICE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SPLIT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CONCAT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_STACK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_UNSTACK	asym-u8		asym-u8	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RESHAPE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SQUEEZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_PERMUTE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_REORG	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SPACE2DEPTH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_DEPTH2SPACE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_BATCH2SPACE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SPACE2BATCH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
VSI_NN_OP_PAD	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_REVERSE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_STRIDED_SLICE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CROP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_REDUCE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ARGMX	asym-u8		asym-u8/int16/ int32	✓	✓
	asym-i8		asym-u8/int16/ int32	✓	✓
	fp32		int32	✓	✓
	fp16		asym-u8/int16/ int32	✓	✓
VSI_NN_OP_ARGMIN	asym-u8		asym-u8/int16/ int32	✓	✓
	asym-i8		asym-u8/int16/ int32	✓	✓
	fp32		int32	✓	✓
	fp16		asym-u8/int16/ int32	✓	✓
VSI_NN_OP_SHUFFLECHANNEL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp16		fp16	✓	✓
RNN Operations					
VSI_NN_OP_LSTMUNIT_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_LSTM_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_GRUCELL_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_GRU_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_SVDF	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
Pooling Operations					
VSI_NN_OP_ROI_POOL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_POOLWITHARGMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_UPSAMPLE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
Miscellaneous Operations					
VSI_NN_OP_PROPOSAL	asym-u8		asym-u8	✓	
	asym-i8		asym-i8	✓	
	fp32		fp32	✓	
	fp16		fp16	✓	
VSI_NN_OP_VARIABLE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_DROPOUT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RESIZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_INTERP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_DATACONVERT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_A_TIMES_B_PLUS_C	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_FLOOR	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_EMBEDDING_LOOKUP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_GATHER	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_GATHER_ND	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SCATTER_ND	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_TILE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RELU_KERAS	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ELTWISEMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_FCL2	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_POOL	asym-u8		asym-u8	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SIGNAL_FRAME	asym-u8		asym-u8	✓	
	asym-i8		asym-i8	✓	
	fp32		fp32	✓	
	fp16		fp16	✓	
VSI_NN_OP_CONCATSHIFT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_UPSAMPLESCALE	asym-u8		asym-u8	✓	
	asym-i8		asym-i8	✓	
	fp16		fp16	✓	
VSI_NN_OP_ROUND	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CEIL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SEQUENCE_MASK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_REPEAT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ONE_HOT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CAST	asym-u8		asym-u8	✓	✓

Table 12. OVXLIB operation support with GPU*...continued*

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓

14 OVXLIB Operation Support with NPU

This section provides a summary of the neural network OVXLIB operations supported by the NXP Neural Processor Unit (NPU) IP and a compatible Software stacks. OVXLIB operations are listed in the following table.

The following abbreviations are used for format types:

- **asym-u8**: asymmetric_affine-uint8
- **asym-i8**: asymmetric_affine-int8
- **fp32**: float32
- **pc-sym-i8**: perchannel_symmetric-int8
- **fp16**: float16
- **bool8**: bool8
- **int16**: int16
- **int32**: int32

The following abbreviations are used to reference key Execution Engines (NPU) in the hardware:

- **NN**: **N**eural-**N**etwork Engine
- **PPU**: **P**arallel **P**rocessing **U**nit
- **TP**: **T**ensor **P**rocessor

Table 13. OVXLIB operation support with NPU

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
Basic Operations						
VSI_NN_OP_CONV2D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_CONV1D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_CONV3D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
VSI_NN_OP_DEPTHWISE_CONV1D	asym-u8	asym-u8	asym-u8			✓
	asym-i8	asym-i8	asym-i8			✓
VSI_NN_OP_DECONVOLUTION	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_DECONVOLUTION1D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_FCL	asym-u8	asym-u8	asym-u8		✓	
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	
VSI_NN_OP_GROUPED_CONV1D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_GROUPED_CONV2D	asym-u8	asym-u8	asym-u8			
	asym-i8	pc-sym-i8	asym-i8			✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
Activation Operations						
VSI_NN_OP_ELU	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_HARD_SIGMOID	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SWISH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
VSI_NN_OP_LEAKY_RELU	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_PRELU	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_RELU	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_RELUN	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_RSQRT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SIGMOID	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SOFTRELU	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SQRT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_TANH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp16		fp16		✓	
VSI_NN_OP_ABS	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_CLIP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_EXP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_LOG	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_NEG	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MISH	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SOFTMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_LOG_SOFTMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SQUARE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SIN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_LINEAR	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ERF	asym-u8		asym-u8		✓	✓
	asym-i8		asym-i8		✓	✓
	fp32		fp32			✓
	fp16		fp16		✓	✓
Elementwise Operations						
VSI_NN_OP_ADD	asym-u8		asym-u8	✓		
	asym-i8		asym-i8	✓		
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SUBTRACT	asym-u8		asym-u8	✓		
	asym-i8		asym-i8	✓		
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MULTIPLY	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_DIVIDE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MAXIMUN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
VSI_NN_OP_MINIMUM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_POW	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_FLOORDIV	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MATRIXMUL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_RELATIONAL_OPS	asym-u8		bool8			✓
	asym-i8		bool8			✓
	fp32		bool8			✓
	fp16		bool8			✓
	bool8		bool8			✓
VSI_NN_OP_LOGICAL_OPS	bool8		bool8			✓
VSI_NN_OP_LOGICAL_NOT	bool8		bool8			✓
VSI_NN_OP_SELECT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
	bool8		bool8			✓
VSI_NN_OP_ADDN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
Normalization Operations						
VSI_NN_OP_BATCH_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_LRN	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_LRN2	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_L2_NORMALIZE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_L2NORMALZESCALE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_LAYER_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_BATCHNORM_SINGLE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MOMENTS	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_GROUP_NORM	asym-u8		asym-u8			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
Reshape Operations						
VSI_NN_OP_EXPAND_ BROADCAST	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SLICE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SPLIT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_CONCAT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_STACK	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_UNSTACK	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_RESHAPE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SQUEEZE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp16		fp16		✓	
VSI_NN_OP_PERMUTE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_REORG	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SPACE2DEPTH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_DEPTH2SPACE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
	bool8		bool8			
VSI_NN_OP_BATCH2SPACE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SPACE2BATCH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_PAD	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_REVERSE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_STRIDED_SLICE	asym-u8		asym-u8		✓	

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_CROP	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_REDUCE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ARGMAX	asym-u8		asym-u8/int16/ int32			✓
	asym-i8		asym-u8/int16/ int32			✓
	fp32		int32			✓
	fp16		asym-u8/int16/ int32			✓
VSI_NN_OP_ARGMIN	asym-u8		asym-u8/int16/ int32			✓
	asym-i8		asym-u8/int16/ int32			✓
	fp32		int32			✓
	fp16		asym-u8/int16/ int32			✓
VSI_NN_OP_SHUFFLECHANNEL	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
RNN Operations						
VSI_NN_OP_LSTMUNIT_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	✓
VSI_NN_OP_LSTM_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp16	fp16	fp16		✓	✓
VSI_NN_OP_GRUCELL_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	✓
VSI_NN_OP_GRU_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	✓
VSI_NN_OP_SVDF	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	✓
Pooling Operations						
VSI_NN_OP_ROI_POOL	asym-u8		asym-u8		✓	✓
	asym-i8		asym-i8		✓	✓
	fp32		fp32			✓
	fp16		fp16		✓	✓
VSI_NN_OP_POOLWITHARGMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_UPSAMPLE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
Miscellaneous Operations						
VSI_NN_OP_PROPOSAL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_VARIABLE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
VSI_NN_OP_DROPOUT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_RESIZE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_INTERP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_DATACONVERT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_A_TIMES_B_PLUS_C	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_FLOOR	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_EMBEDDING_ LOOKUP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_GATHER	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_GATHER_ND	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp16		fp16			✓
VSI_NN_OP_SCATTER_ND	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_TILE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_RELU_KERAS	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_ELTWISEMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_FCL2	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_POOL	asym-u8		asym-u8	✓	✓	
	asym-i8		asym-i8	✓	✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SIGNAL_FRAME	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_CONCATSHIFT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓

Table 13. OVXLIB operation support with NPU*...continued*

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_UPSAMPLESCALE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp16		fp16			✓
VSI_NN_OP_ROUND	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_CEIL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SEQUENCE_MASK	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_REPEAT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ONE_HOT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_CAST	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓

15 Note About the Source Code in the Document

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

Copyright 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.

16 Revision History

This table provides the revision history.

Revision history

Document ID	Release date	Description
UG10166 v.LF6.12.49_2.2.0	23 January 2026	Updated the ONNX Runtime version information.
UG10166 v.LF6.12.49_2.2.0	12 December 2025	Upgraded to the 6.12.49 kernel, and added the i.MX 95 FRDM board support.
UG10166 v.LF6.12.34_2.1.0	25 September 2025	Upgraded to the 6.12.34 kernel, i.MX 943 A0 and i.MX 95 B0 support with Beta quality, and added the i.MX 8MP/91/93 FRDM boards support.
UG10166 v.LF6.12.20_2.0.0	26 June 2025	Upgraded to the 6.12.20 kernel.
UG10166 v.LF6.12.3_1.0.0	30 April 2025	Updated the command in Section 4.1.
UG10166 v.LF6.12.3_1.0.0	31 March 2025	Upgraded to the 6.12.3 kernel.
UG10166 v.LF6.6.52_2.2.0	16 December 2024	Upgraded to the 6.6.52 kernel.
UG10166 v.LF6.6.36_2.1.0	30 September 2024	Upgraded to the 6.6.36 kernel.
IMXMLUG_6.6.23_2.0.0	28 June 2024	Upgraded to the 6.6.23 kernel, U-Boot v2024.04, TF-A v2.10, OP-TEE 4.2.0, Yocto 5.0 Scarthgap, and added the i.MX 91 as Alpha quality, i.MX 95 as Beta quality.
IMXMLUG v.LF6.6.3_1.0.0	29 March 2024	Upgraded to the 6.6.3 kernel, removed the i.MX 91P, and added the i.MX 95 as Alpha Quality.
IMXMLUG v.LF6.1.55_2.2.0	12/2023	Upgraded to the 6.1.55 kernel.
IMXMLUG v.LF6.1.36_2.1.0	09/2023	Upgraded to the 6.1.36 kernel.
IMXMLUG v.LF6.1.22_2.0.0	06/2023	Upgraded to the 6.1.22 kernel.
IMXMLUG v.LF6.1.1_1.0.0	03/2023	Upgraded to the 6.1.1 kernel.
IMXMLUG v.LF5.15.71_2.2.0	12/2022	Upgraded to the 5.15.71 kernel.
IMXMLUG v.LF5.15.52_2.1.0	09/2022	Upgraded to the 5.15.52 kernel, and added the i.MX 93.

Revision history...*continued*

Document ID	Release date	Description
IMXMLUG v.LF5.15.32_2.0.0	06/2022	Upgraded to the 5.15.32 kernel, U-Boot 2022.04, and Kirkstone Yocto.
IMXMLUG v.LF5.15.5_1.0.0	03/2022	Upgraded to the 5.15.5 kernel, Honister Yocto, and Qt6.
IMXMLUG v.LF5.10.72_2.2.0	12/2021	Upgraded the kernel to 5.10.72 and updated the BSP.
IMXMLUG v.LF5.10.52_2.1.0	09/2021	Updated for i.MX 8ULP Alpha and the kernel upgraded to 5.10.52.
IMXMLUG v.LF5.10.35_2.0.0	06/2021	Upgraded to Yocto Project Hardknott and the kernel upgraded to 5.10.35.
IMXMLUG v.L5.4.70_2.3.2	04/2021	Patch release.
IMXMLUG v.LF5.10.9_1.0.0	03/2021	Kernel upgrade to 5.10.9 and Machine Learning upgrades.
IMXMLUG v.L5.4.70_2.3.0	01/2021	i.MX 5.4 consolidated GA for release i.MX boards including i.MX 8M Plus and i.MX 8DXL.
IMXMLUG v.L5.4.47_2.2.0	09/2020	Initial release.

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.

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.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

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.

Contents