

IMXMLUG

i.MX Machine Learning User's Guide

Rev. LF5.15.71_2.2.0 —
16 December 2022

User guide

Document information



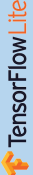








Information	Content
Keywords	i.MX, Linux, LF5.15.71_2.2.0
Abstract	The NXP eIQ Machine Learning Software Development Environment (hereinafter referred to as "NXP eIQ") provides a set of libraries and development tools for machine learning applications targeting NXP microcontrollers and application processors.



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* (IMXLXYOCTOUG) for more information.

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

NXP eIQ Inference Engines & Libraries	eIQ Inference Engine Deployment										
	 PyTorch	 ONNX RUNTIME	 TensorFlow Lite	 OpenCV	 DeepView RT	 ONNX RUNTIME	 TensorFlow Lite	 DeepView RT	 ONNX RUNTIME	 TensorFlow Lite	 DeepView RT
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 93	NA	NA	✓	NA	✓	NA	NA	NA	NA	✓	NA

✓ Supported

NA (Not Applicable)

Figure 1. NXP eIQ supported compute vs. inference engines

The NXP eIQ inference engines support multi-threaded execution on Cortex-A cores. Additionally, ONNX Runtime, TensorFlow Lite, and DeepViewRT also support acceleration on the GPU or NPU through Neural Network Runtime (NNRT). See also [Section 2](#). 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

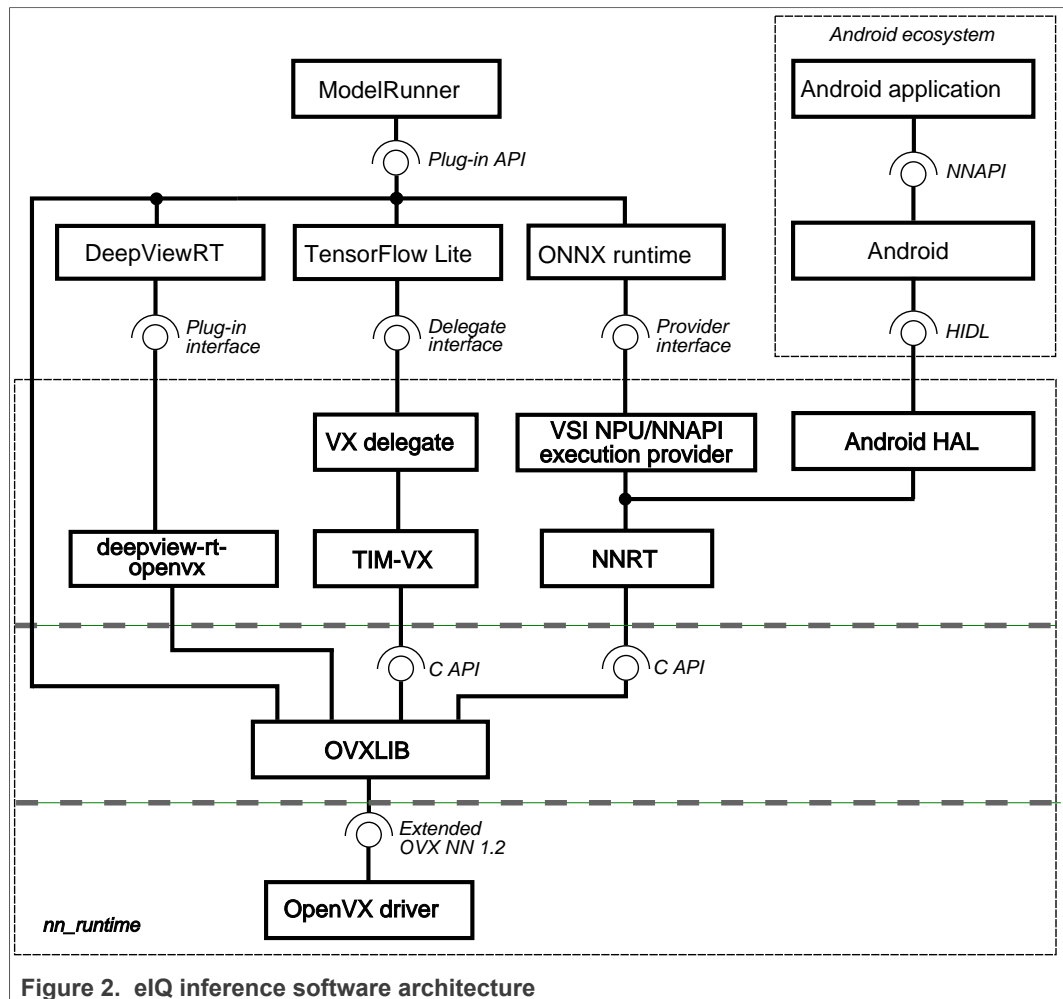
2 eIQ Inference Runtime Overview on i.MX 8 Series

The chapter describes an overview of the NXP eIQ software stack for use with the NXP Neural Network Accelerator IPs (GPU or NPU). The following figure shows the data flow between each element. The below diagram has two key parts:

- Neural Network Runtime (NNRT), which is a middleware bridging various inference frameworks and the NN accelerator driver.
- TIM-VX, which is a software integration module to facilitate deployment of Neural Networks on OpenVX enabled ML accelerators.

ModelRunner for DeepViewRT is a server application being able to receive requests using HTTP REST API, Python API, or UNIX RPC service, and delegate those to different inference engines, or the NN accelerator driver directly. See also [Section 7.4](#) for more details.

The NNRT supplies different backends for Android NN HAL, ONNX, and TensorFlow Lite allowing quick application deployment. The NNRT also empowers an application-oriented framework for use with i.MX8 processors. Application frameworks such as Android NN, and TensorFlow Lite can be speed-up by NNRT directly benefiting from its built-in backend plug-ins. Additional backend can be also implemented to expand support for other frameworks.



NNRT supports different Machine Learning frameworks by registering itself as a compute backend. Because each framework defines a different backend API, a lightweight backend layer is designed for each:

- For Android NN, the NNRT follows the Android HIDL definition. It is compatible with v1.2 HAL interface
- For ONNX Runtime, the NNRT registers itself as an execution provider

In doing so, NNRT unifies application framework differences and provides an universal runtime interface into the driver stack. At the same time, NNRT also acts as the heterogeneous compute platform for further distributing workloads efficiently across i.MX8 compute devices, such as NPU, GPU and CPU.

Note: Both the OpenCV and PyTorch inference engines are currently not supported for running on the NXP NN accelerators. Therefore, both frameworks are not included in the above NXP-NN architecture diagram.

3 TensorFlow Lite

TensorFlow Lite 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 Custom Operator on i.MX 93.

The TensorFlow Lite source code for this Yocto Linux release is available at this [repository](#), branch lf-5.15.71_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 93 platforms.

Features:

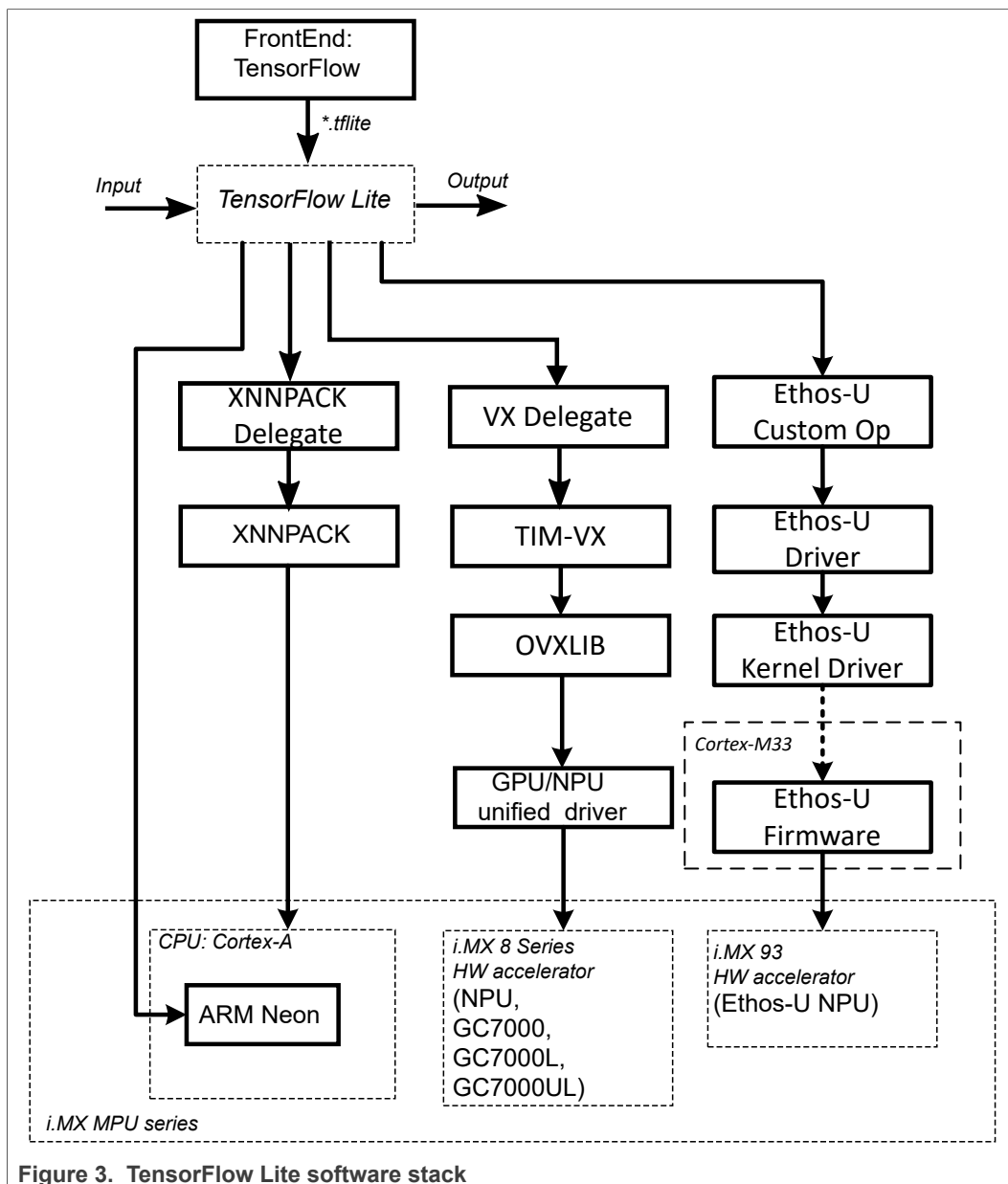
- TensorFlow Lite v2.9.1
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores
- Parallel computation using GPU/NPU hardware acceleration (on shader or convolution units)
- C++ and Python API (supported Python version 3)
- Per-tensor and Per-channel quantized models support

3.1 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
- NPU hardware acceleration on i.MX 93 NPU

See [Section 1](#) for some details about supporting of computation on GPU/NPU hardware accelerator on different hardware platforms.

**Note:**

On i.MX 8 family, the first execution of model inference using the VX Delegate will take longer, because of the time required for computational graph initialization by the GPU/NPU driver. The iterations following the graph initialization will perform much quicker. Note 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.

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 (IMXGRAPHICUG).

Refer to i.MX Graphics Users Guide for list GPUs with OpenVX support. Note the GC7000 Lite and GC7000 Ultra Lite GPUs does not support full OpenVX however still capable to run ML workload.

The GPU/NPU hardware accelerator driver support both per-tensor and per-channel quantized models. The GPU/NPU hardware accelerator on the i.MX 8 family is optimized for per-tensor quantized models. In case of per-channel quantized models, the performance might be lower. The actual impact depends on the model used.

3.2 Compute backends and delegates

TensorFlow Lite comes with options to execute compute operations of various compute units. We will refer to them as inference backends.

3.2.1 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.

3.2.2 XNNPACK delegate

[XNNPACK library](#) is a highly optimized library of floating-point 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 for floating point operators. In general, it provides better performance than the built-in kernels for floating point operators.

Note: Since TensorFlow Lite 2.6.0, the floating point models are executed via the XNNPACK Delegate by default.

3.2.3 VX Delegate

VX Delegate is a successor of the NNAPI Delegate on i.MX 8 Linux platforms. It enables accelerating the inference on on-chip hardware accelerator. The VX Delegate directly uses the hardware accelerator driver (OpenVX with extension) to fully utilize the accelerator capabilities. Over the NNAPI delegate it offers better alignment with the on-chip HW 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](#).

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](#)

3.2.4 Ethos-U custom operator

Ethos-U custom operator enables accelerating the inference on Ethos-U accelerator. The OP directly uses the Hardware accelerators driver to fully utilize the accelerators capabilities.

3.3 Delivery package

The TensorFlow Lite is available using Yocto Project recipes.

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

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 3.5](#) for more details.

There are following delegates available in the TensorFlow Lite 2.9.1 delivery package:

- XNNPACK Delegate
- VX Delegate

3.4 Build details

TensorFlow Lite uses CMake build system for compilation. Notable remarks to package building 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`)
- 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 3.5.1](#).
- On the i.MX 93 platform, the Ethos-U NPU support is enabled (`TFLITE_ENABLE_ETHOSU=On`).
- The package is compiled with the default `-O2` optimization level. Some CPU kernels, e.g. `RESIZE_BILINEAR`, are known to performs better with `-O3` optimization level,

however some performs better with -O2, e.g. 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.

3.5 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* (IMXLXOCTOUG) for detailed information how to generate Yocto SDK environment for cross-compiling. To activate this Yocto SDK environment on your host machine, use this command:

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

To build an application which uses the TensorFlow Lite, following options are available:

- Create CMake project which uses TensorFlow Lite (CMake superbuild pattern)
- Using Yocto SDK precompiled libraries

The TensorFlow Lite's CMake configuration file is in tensorflow/lite/CMakeLists.txt from the root repository (for NXP i.MX8 platforms).

3.5.1 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. Set up the Yocto SDK as described above
2. Configure the project using CMake:

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

3. Build the project:

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

4. 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
```

5. 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

3.5.2 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/6f9d96a1f41439ac172ee2ef7ccd8edf0e5d068c.tar.gz -O  
abseil-cpp.tar.gz  
$ tar -xzf abseil-cpp.tar.gz  
$ mv abseil-cpp-6f9d96a1f41439ac172ee2ef7ccd8edf0e5d068c  
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
```

3.6 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-2.9.1/examples
```



Figure 4. TensorFlow image classification input

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
```

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

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

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
```

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

```
/usr/bin/tensorflow-lite-2.9.1/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.

3.7 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-2.9.1/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 using the XNNPACK delegate, add the `--use_xnnpack=true` switch:

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

To run the inference using the GPU/NPU hardware accelerator for VX Delegate, add the `--external_delegate_path=/usr/lib/libvx_delegate.so` switch:

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite
--external_delegate_path=/usr/lib/libvx_delegate.so
```

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 a 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%]
TfLiteNnapiDelegate  0.000      14.890      14.894      11.349%
11.349%
    RESIZE_BILINEAR    14.896      1.331      1.331      1.014%
12.363%
TfLiteNnapiDelegate  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%
TfLiteNnapiDelegate  0.000      14.890      14.894      11.349%
96.559%
TfLiteNnapiDelegate  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
TfLiteNnapiDelegate  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 TfLiteNnapiDelegate falls back to CPU. In the example above, the ARG_MAX and RESIZE_BILINEAR nodes fall back to CPU.

3.8 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 model into TensorFlow Lite model format. 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/r2.8/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 2.9.1. 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 2.9.1 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 2.9.1 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 instalation, 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.

3.9 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 TF 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/nlib-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.

4 Arm Compute Library

Arm Compute Library (ACL) is a collection of low-level functions optimized for Arm CPU and GPU architectures targeted at image processing, computer vision, and machine learning.

Source codes are available at <https://github.com/nxp-imx/arm-computelibrary-imx>.

Features:

- Arm Compute Library 22.05

- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A CPU cores
- C++ API only
- Low-level control over computation

Note:

The GPU OpenCL backend is not supported on i.MX 8 devices.

4.1 Running a DNN with random weights and inputs

Arm Compute Library comes with examples for most common DNN architectures like: AlexNet, MobileNet, ResNet, Inception v3, Inception v4, SqueezeNet, etc.

All available examples can be found in this example build location:

```
/usr/bin/arm-compute-library-22.05/examples
```

Each model architecture can be tested using `graph_[dnn_model]` application.

For example, to run the MobileNet v2 DNN model, use the following command:

```
$ ./graph_mobilenet_v2 --data=<path_cnn_data> --  
image=<input_image> --labels=<labels> --target=neon --  
type=<data_type> --threads=<num_of_threads>
```

The parameters are not mandatory. When not provided, the application runs the model with random weights and inputs. If inference finishes successfully, the "Test passed" message is printed.

4.1.1 Running AlexNet using graph API

In 2012, AlexNet shot to fame when it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual challenge that aims to evaluate algorithms for object detection and image classification. AlexNet is made up of eight trainable layers: five convolution layers and three fully connected layers. All the trainable layers are followed by a ReLu activation function, except for the last fully connected layer, where the Softmax function is used.

Location of the C++ AlexNet example implementation using the graph API is in this folder:

```
/usr/bin/arm-compute-library-22.05/examples
```

Demo instructions:

- Download the archive file ([compute_library_alexnet.zip](#)) to the example location folder.
- Create a new sub-folder and unzip the file:

```
$ mkdir assets_alexnet  
$ unzip compute_library_alexnet.zip -d assets_alexnet
```

- Set environment variables for execution:

```
$ export PATH_ASSETS=/usr/bin/arm-compute-library-21.08/  
examples/assets_alexnet/
```

- Run the example with following command line arguments:

```
$ ./graph_alexnet --data=$PATH_ASSETS --image=$PATH_ASSETS/  
go_kart.ppm --labels=$PATH_ASSETS/labels.txt --target=neon --  
type=f32 --threads=4
```

The output of a successful classification should be similar as the one below:

```
----- Top 5 predictions -----  
0.9736 - [id = 573], n03444034 go-kart  
0.0108 - [id = 751], n04037443 racer, race car, racing car  
0.0118 - [id = 518], n03127747 crash helmet  
0.0022 - [id = 817], n04285008 sports car, sport car  
0.0006 - [id = 670], n03791053 motor scooter, scooter  
Test passed
```

5 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](#).

Features:

- ONNX Runtime 1.10.0
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores provided by the CPU execution provider
- Parallel computation using GPU/NPU hardware acceleration (on shader or convolution units) provided by the VSI NPU and NNAPI execution providers
- C++ and Python API (supported Python version 3)
- ONNX Runtime 1.10.0 supports [ONNX 1.10](#) and Opset version 15.

Note:

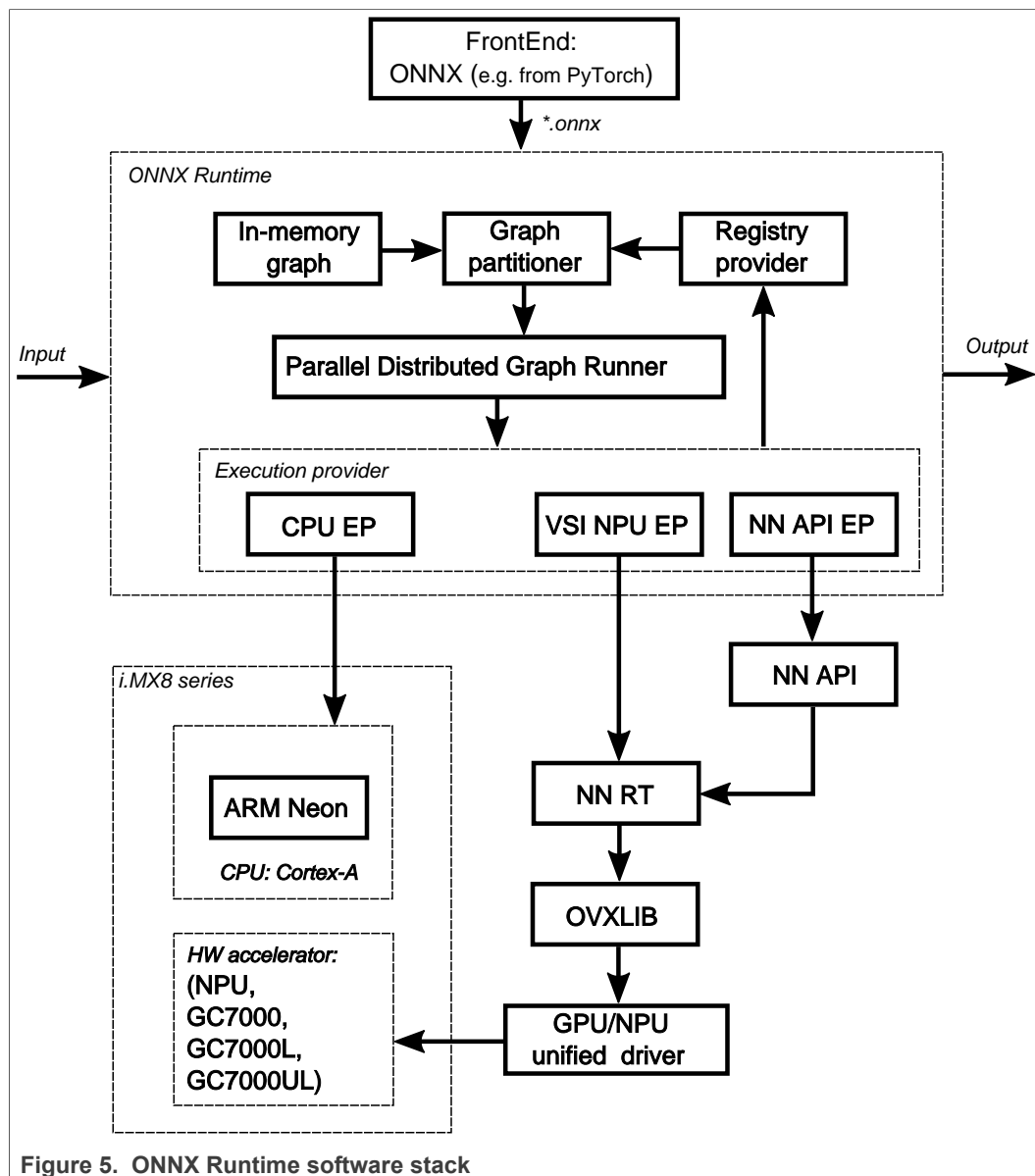
The opset only defines all the operators which are available. It does not necessarily mean they are implemented in the execution provider in use. See section [Section 5.2](#) for more details.

5.1 ONNX Runtime software stack

The ONNX Runtime software stack is shown in the following figure. The ONNX Runtime supports computation on the following HW units:

- CPU Arm Cortex-A cores using CPU execution provider
- GPU/NPU hardware accelerator using VSI NPU (deprecated) or NNAPI execution providers (experimental)

See [Section 1](#) for some details about supporting of computation on GPU/NPU hardware accelerator on different HW platforms.



5.2 Execution providers

Execution providers (EP) are a mechanism to delegate inference execution to an underlying framework or hardware. By default, the ONNX Runtime uses the CPU EP, which executes inference on the CPU.

Officially supported Execution Providers which provide means of acceleration compared to the default CPU EP are the following:

- `vsi_npu` - runs either on the GPU or the NPU depending on what HW is available. Leverages OpenVX implementation directly.
- `nnapi` - runs either on the GPU or the NPU depending on what HW is available. Leverages the NNAPI implementation which uses OpenVX.

Note:

The `VSI_NPU` Execution Provider is deprecated and will be removed in the future. The `NNAPI` Execution Provider is experimental.

5.2.1 ONNX model test

ONNX Runtime provides a tool that can run the collection of standard tests provided in the ONNX Model Zoo. The tool named `onnx_test_runner` is installed in `/usr/bin/onnxruntime-1.10.0`.

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 advised to store the ONNX test files on a larger partition, as described in the SD card image flashing section.

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

- Download and unpack the [mobilenet version 2](#) test archive to some folder, for example to `/home/root`:

```
$ cd /home/root
$ wget https://github.com/onnx/models/raw/main/vision/classification/mobilenet/model/mobilenetv2-7.tar.gz
$ tar -xzf mobilenetv2-7.tar.gz
$ ls ./mobilenetv2-7
mobilenetv2-7.onnx  test_data_set_0  test_data_set_1
test_data_set_2
```

- Run the `onnx_test_runner` tool providing `mobilenetv2-7` folder path and setting the execution provider:

```
$ /usr/bin/onnxruntime-1.10.0/onnx_test_runner -j 1 -c 1 -r 1
-e [cpu/vsi_npu/nnapi] ./mobilenetv2-7/
result:
Models: 1
Total test cases: 3
Succeeded: 3
Not implemented: 0
Failed: 0
Stats by Operator type:
Not implemented(0):
Failed:
Failed Test Cases:
$
```

Note:

Use `onnx_test_runner -h` for the full list of supported options.

5.2.2 C API

ONNX Runtime also provides a C API sample code described here: https://github.com/microsoft/onnxruntime/blob/v1.10.0/docs/C_API_Guidelines.md.

To build the sample from the [repository](#), run the following build command under the generated Yocto SDK environment (make sure that the `onnxruntime-dev` Yocto package is installed in the SDK, it should be installed by default):

```
$CXX -std=c++0x -I$SDKTARGETSYSROOT/usr/include/
onnxruntime/core/session -lonnxruntime C_Api_Sample.cpp -o
onnxruntime_sample
```

Note:

SqueezeNet model included in the BSP can be used with the executables.

5.2.2.1 Enabling execution provider

5.2.2.1.1

To enable a specific execution provider, you need to do the following in your code:

- Set the execution provider in code (see the previous C API sample how that is done for the CUDA EP). If not set, the default CPU EP would be used:

```
OrtSessionOptionsAppendExecutionProvider_<execution_provider>(<parameters>);
```
- Include headers based on the EP used in the code: `#include "<execution_provider>_provider_factory.h"`.
- Add includes to the build command: `-I/usr/include/onnxruntime/core/providers/<execution_provider>/`

5.2.3 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-1.10.0`. In order to run it, the user must provide an `.onnx` model file together with test data. To benchmark the SqueezeNet model running a single iteration using the VSI NPU execution provider, run to the following command:

```
$/usr/bin/onnxruntime-1.10.0/onnxruntime_perf_test /usr/bin/onnxruntime-1.10.0/squeezenet/model.onnx -r 1 -e vsi_npu
```

Note:

Use `onnxruntime_perf_test -h` for the full list of supported options.

6 PyTorch

PyTorch is a scientific computing package based on Python that facilitates building deep learning projects using power of graphics processing units.

Features:

- PyTorch 1.9.1
- Python version 3 supported
- Deep neural networks built on a tape-based autograd system

Note:

This release of PyTorch does not yet support the tensor computation on the NXP GPU/NPU. 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'
```

6.1 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)
```

6.2 Building and installing wheel packages

This release includes building script for PyTorch and TorchVision on aarch64 platform. Currently, it supports the native building on the NXP aarch64 platform with BSP SDK.

Note: Generally, in the yocto rootfs of the BSP SDK, the PyTorch and TorchVision wheel packages are already integrated. There is no need to build and install from scratch. If you would like to build them by your own, perform the steps below.

6.2.1 How to build

Perform the following steps:

1. Get the latest i.MX BSP from <https://github.com/nxp-imx/imx-manifest>.
2. Set up the build environment for one of the NXP aarch64 platforms and edit the *local.conf* to add the following dependency for PyTorch native build:

```
IMAGE_INSTALL_append = " python3-dev python3-pip python3-
wheel python3-pillow python3-setuptools python3-numpy
python3-pyyaml
python3-cffi python3-future cmake ninja packagegroup-core-
builddessential git git-perltools libxcrypt libxcrypt-dev
```


3. Build the BSP images using the following command:

```
$ bitbake imx-image-full
```

4. Get into the pytorch folder and execute the build script on NXP aarch64 platform to generate wheel packages. You can get the source from <https://github.com/NXPmicro/pytorch-release> as well:

```
$ cd /path/to/pytorch/src  
$ ./build.sh
```

6.2.2 How to install

If the building is successful, the wheel packages should be found under `/path/to/pytorch/src/dist`:

```
$ pip3 install /path/to/torch-1.9.1.post2-cp310-cp310-  
linux_aarch64.whl  
$ pip3 install /path/to/torchvision-0.10.0-cp310-cp310-  
linux_aarch64.whl
```

7 DeepViewRT

DeepViewRT is a proprietary neural network inference engine optimized for NXP microprocessors and microcontrollers, which not only implements its own compute engine, but it is also able to leverage popular 3rd party ones.

Features:

- DeepViewRT 2.4.46
- Plug-in API allowing for various compute engines:
 - DeepViewRT (CPU/Neon)
 - DeepViewRT (OpenVX)
 - TensorFlow Lite
 - ONNX Runtime
- C and Python API
- Per-tensor and per-channel quantization model support
- Defines custom operations or custom behavior for existing operations
- Models to be deployed to all targets without explicitly programming the computation graph

7.1 DeepViewRT software stack

The DeepViewRT Software stack includes DeepViewRT library, modelrunner library and modelrunner server - see the following picture:

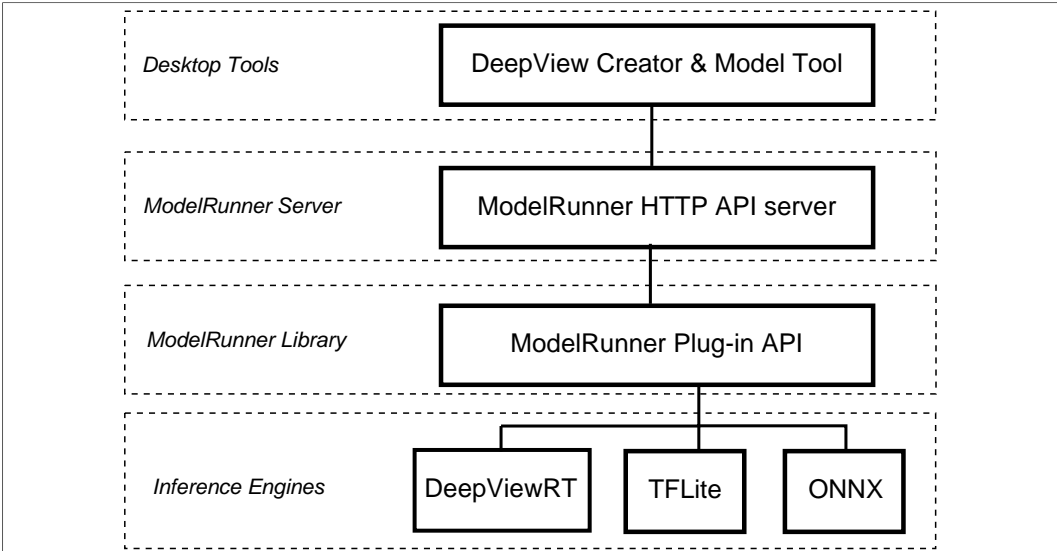


Figure 6. DeepViewRT SW stack

Note:

eIQ Portal and Model Tool are parts of the eIQ Toolkit.

DeepViewRT supports the following hardware:

- CPU Arm Cortex-A cores
- GPU/NPU hardware accelerator using the VSI NPU backend, which runs on both the GPU and the NPU depending on which is available

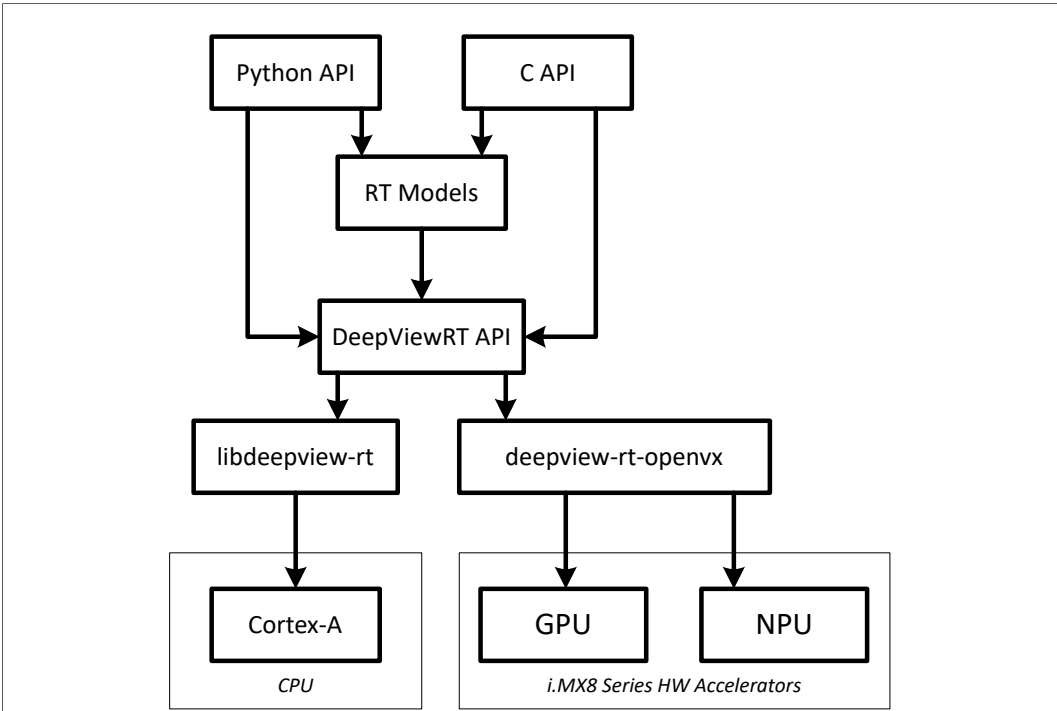


Figure 7. DeepViewRT computing engines

Note:

Refer to DeepViewRT User Manual, included in the eIQ Toolkit docs folder, for more information about the DeepViewRT API.

7.2 Delivery packages

The DeepViewRT is available in Yocto recipe and able to get DeepViewRT package through the DeepViewRT recipe.

The DeepViewRT packages include followings components for Yocto BSP release:

- DeepViewRT shared library (dynamic library)
- DeepViewRT header file
- DeepViewRT Python module
- ModelRunner binary and library
- ModelRunner plug-in libraries (OpenVX, TensorFlow Lite, ONNX Runtime)
- DeepViewRT examples (labelimg, detect_ssd, python examples for image classification and object detection)

7.3 Example applications

All example application were integrated into the Yocto BSP image. You can use this Yocto command to extract source code and build all examples:

```
bitbake -c patch deepview-rt-examples
```

The deepview-rt-examples source code were put under `tmp/work/cortexa53-crypto-mx8mp-poky-linux/deepview-rt-examples/1.5-r0/deepview-rt-examples-1.3`.

The folder structure looks like as follows.

```

├── CMakeLists.txt
├── COPYING
├── detecting
│   ├── CMakeLists.txt
│   ├── detectv4.c
│   ├── detectv4_remote.c
│   ├── Makefile
│   └── README.md
├── labelcam-gst
│   ├── cmake
│   │   ├── FindGStreamer.cmake
│   │   └── MacroFindGStreamerLibrary.cmake
│   ├── CMakeLists.txt
│   ├── demo.c
│   ├── Makefile
│   ├── README.md
│   ├── README.pdf
│   └── VERSION
├── labeling
│   ├── CMakeLists.txt
│   ├── labeling.c
│   ├── labeling_remote.c
│   ├── Makefile
│   └── README.md
├── LICENSE.txt
├── Makefile
├── SCR-deepview-rt-examples.txt
└── ssdcam-gst
    ├── cmake
    │   ├── FindGStreamer.cmake
    │   └── MacroFindGStreamerLibrary.cmake
    ├── CMakeLists.txt
    ├── demo.c
    ├── Makefile
    ├── README.md
    ├── README.pdf
    └── VERSION

```

Figure 8. DeepViewRT Yocto folder structure

For cross-compile of those examples, use Makefile under example source folder.

7.3.1 Image labelling applications

There are two example applications which demonstrate how to implement an image labelling application, targeting either the direct DeepViewRT C API or Python API.

The "labelimg" application directly calls DeepViewRT C API on CPU or NPU/GPU:

```

$ ./labelimg -e deepview-rt-openvx.so
mobilenet_v1_0.25_224_quant.rtm eagle.png
# running on CPU
$ cd /usr/bin/deepview-rt-examples
# Running example with DeepViewRT C API on NPU
$ ./labelimg -e deepview-rt-openvx.so python-examples/models/
mobilenet_v1_0.25_224.rtm python-examples/images/panda.jpg
# Running example with DeepViewRT C API on CPU

```

```
$ ./labelimg python-examples/models/mobilenet_v1_0.25_224.rtm
python-examples/images/panda.jpg
# Running example with DeepViewRT Python API on NPU
$ python3 python-examples/rt_classify_image.py --engine
deepview-rt-openvx.so --model python-examples/models/
mobilenet_v1_0.25_224.rtm --image python-examples/images/
panda.jpg
# Running example with DeepViewRT Python API on CPU
$ python3 python-examples/rt_classify_image.py --model python-
examples/models/mobilenet_v1_0.25_224.rtm --image python-
examples/images/panda.jpg
```

7.3.2 Object detection applications

There are two example applications which demonstrate how to implement an object detection application, targeting either the direct DeepViewRT C API or the ModelRunner REST API using the libCurl library.

The "detecting" application directly calls DeepViewRT C API:

```
$ cd /usr/bin/deepview-rt-examples
$ ./detect_ssd DATA_PATH//
mobilenet_ssd_v1_1.00_trimmed_new.rtm DATA_PATH/
ssd_resized.jpg -T 0.5 -I 0.5 -i 50 -e /usr/lib/deepview-rt-
openvx.so
```

Note:

All examples use DeepViewRT RTM model format. The .rtm can be converted from .tflite. For a model conversion, refer to the eIQ Toolkit User's Guide (EIQTUG).

7.4 ModelRunner

The ModelRunner application provides an HTTP service for hosting DeepViewRT models, TensorFlow Lite models, ONNX Runtime models and remote evaluation. The service also provides a low-level UNIX socket service for low-latency video processing. It was integrated into BSP through the DeepViewRT Yocto recipe.

For ModelRunner HTTP REST API, please refer to *DeepViewRT User Manual*, which is included in the eIQ Toolkit *docs* folder.

To use Modelrunner for benchmark evaluation, refer to below commands (chapters) to measure the performance.

7.4.1 DeepViewRT

To run modelrunner with DeepViewRT backend and measure its performance:

```
$ modelrunner -e rt -c 0 -m mobilenet_v1_1.0_224_quant.rtm -b
50 -t 4
Plugin: libmodelrunner-rt.so;
Average model run time: 129.0078 ms (layer sum: 0.0000 ms)
```

Note:

Number of threads (-t parameter) should correlate with the number of device computing cores to get the best performance. For example, for i.MX 8QM device use -t 6, etc.

7.4.2 OpenVX

To run modelrunner with OpenVX by accelerating with NPU and measure its performance:

```
$ modelrunner -e rt -c 1 -m mobilenet_v1_1.0_224_quant.rtm -b 50
Plugin: libmodelrunner-ovx.so;
RTMx Output indices = [87 ]
Created empty VX graph, inputs = 1, outputs = 1
RTMx Layer count = 88
...
Average model run time: 2.2397 ms
```

7.4.3 TensorFlow Lite

To run modelrunner with TensorFlow Lite and Vx Delegate and measure its performance:

```
$ modelrunner -e tflite -c 3 -m
mobilenet_v1_1.0_224_quant.tflite -b 50
Plugin: libmodelrunner-tflite.so;
Loaded model
resolved reporter
INFO: Created TensorFlow Lite delegate for NNAPI.
Applied NPU delegate.
interpreter invoked
average time: 2.51356 ms
Average layer sum: 2.5105 ms
```

Note:

It can be changed to use CPU by replacing “-c 1” with “-c 0”. Use “-c 2” for XNNPACK and “-c 3” for VX Delegate.

7.4.4 ONNX Runtime

To run modelrunner with ONNX Runtime and Vsi_Npu execution provider and measure its performance:

```
$ modelrunner -e onnx -c 3 -m mobilenet_v1_1.0_224_quant.onnx -b 50
Plugin: libmodelrunner-onnx.so;
WARNING: Since openmp is enabled in this build, this API cannot be used to configure intra op num threads. Please use the openmp environment variables to control the number of threads.
Prefer Vsi_Npu execution provider
Input name=input, type=1, num_dims=4, shape=[ 1 3 224 224 ]
Number of outputs = 1
Output 0 : name=TFLITE2ONNX_Quant_MobilenetV1/Predictions/Reshape_1_dequantized
Loaded ONNX model.
Average model run time: 434.220155 ms
```

8 TVM

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

Features:

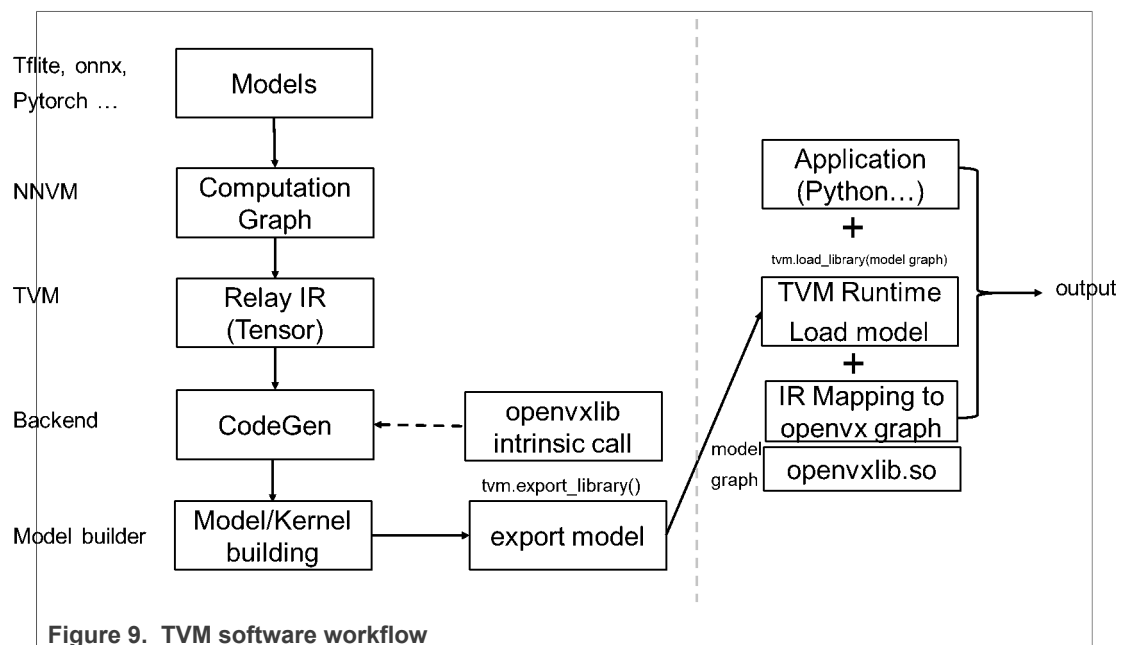
- TVM 0.7.0
- Compilation of deep learning models into minimum deployable modules
- Infrastructure to automatic generate and optimize models on more backend with better performance
- GPU/NPU support for i.MX 8 (except for i.MX8MM) platforms with OpenVX library
- TVM builder supported for Ubuntu 18.04, x86_64 platform

Note:

For more detailed information, see [TVM Documentation](#).

8.1 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.



8.2 Getting started

8.2.1 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 tvml.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_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_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
```

8.2.2 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.

8.3 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 tvml, 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 tvml -j4 # make tvml build stack
```

8.4 Supported models

The following models are verified with TVM.

Table 1. 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
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

9 NN Execution on Hardware Accelerators

9.1 Hardware acceleration on i.MX 8 Series

9.1.1 Hardware accelerator description

The i.MX8 class devices are deployed with two kind of NN accelerators (see also Figure 1):

- Neural Processing Unit (NPU)

- Graphical 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.

Graphical 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.

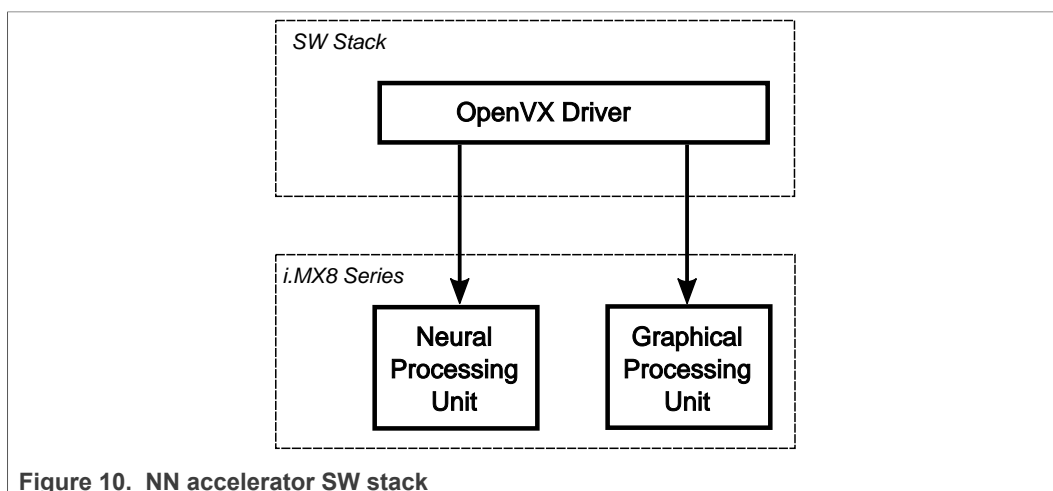


Figure 10. NN accelerator SW stack

Interface to NPU/GPU HW accelerator is provided via the OpenVX v1.2 with NN Extensions. OpenVX is an open, royalty-free standard for cross platform acceleration of computer vision applications. It provides²

- 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](https://www.khronos.org/registry/OpenVX/specs/1.2/html/index.html).

Note:

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

9.1.2 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
```

² OpenVX 1.2 specification; <https://www.khronos.org/registry/OpenVX/specs/1.2/html/index.html>:

```
edit: setenv bootargs ${jh_clk} console=${console} root=
${mmcrout} 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. 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-2.8.0/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.

9.1.3 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.

9.1.4 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
```

9.2 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.

9.2.1 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.

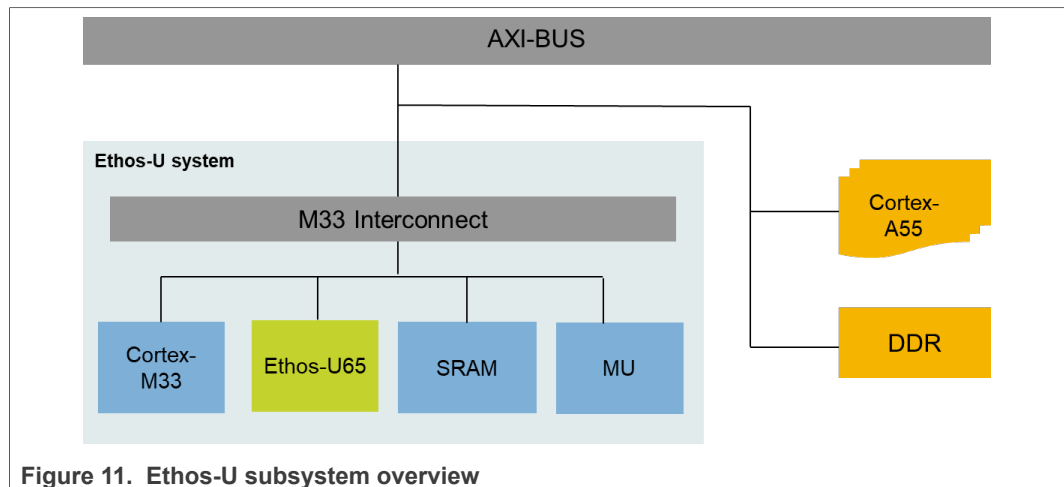


Figure 11. Ethos-U subsystem overview

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 trigger 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.

9.2.2 Ethos-U software architecture

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

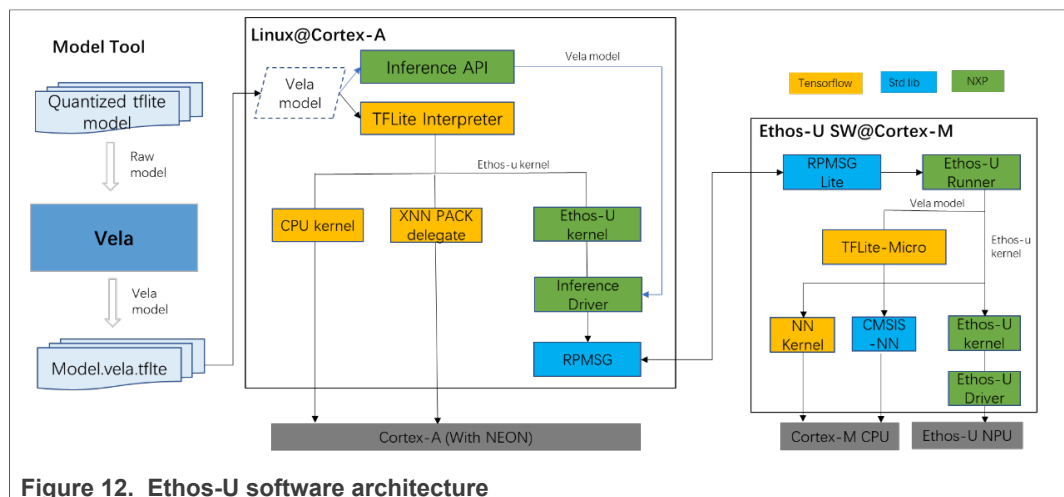


Figure 12. Ethos-U software architecture

- Vela model compiler: offline tool to compile the TFLite 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 TFLite model graph for TFLite/TFLite-Micro inference engines.
- Cortex-A SW stack for Linux: containing MPU inference engine (TensorFlow Lite), driver library, and kernel-side device driver for Linux Kernel.
- Cortex-M SW stack: containing MCU inference engine SW (TFLite-Micro, CMSIS-NN), and NPU driver.

The typical inference workflow is as follows:

1. Converts the TFLite model into Vela model using the Vela model compiler and generates the optimized version for Ethos-U NPU.
2. The optimized model is either:
 - a. TFLite inference engine, which recognizes the custom “ethos-u” operator, allocates the buffer for input/output feature map (IFM/OFM) and executes the operator via Ethos-U Linux driver.
 - b. 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 TFLite-Micro or Ethos-U driver directly according to the task type.
 - a. If the task type is accelerating the “ethos-u” operator (using the TFLite), 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 TFLite-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 TFLite or Inference API by opening the Ethos-U device.

9.2.3 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-2.9.1/examples/labels.txt ./
$ cp ../../tensorflow-lite-2.9.1/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-2.9.1/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-2.9.1/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 model with TFLite inference engine (offload the “ethos-u” operator to Cortex-M).

```
$ cd /usr/bin/tensorflow-lite-2.9.1/examples
$ ./label_image -m
../../../../ethosu/examples/output/
mobilenet_v1_1.0_224_quant_vela.tflite
```

The following will be printed if no error occurs:

```
INFO: Loaded mode[ 2712.710545] imx-rproc imx93-cm33: can't
change firmware while running
../../../../ethosu/examples/output/
mobilenet_v1_1.0_224_quant_vela.tflite
INFO: resolved reporter
INFO: invoked
INFO: average time: 4.433 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
```

9.2.4 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.

9.2.4.1 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-5.15.71_2.2.0
$ pip3 install .
```

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

```
$ vela --version
3.x.x
```

9.2.4.2 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.

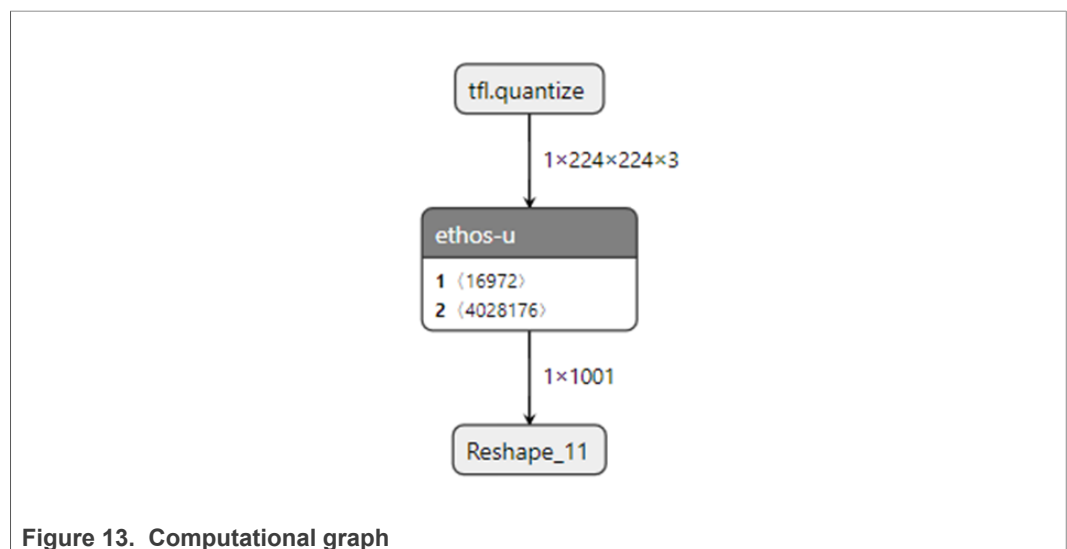


Figure 13. Computational graph

9.2.5 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.

9.2.5.1 Ethos-U driver library

The Ethos-U Driver provides a C++ APIs for dispatching the inference to the Ethos-U 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.

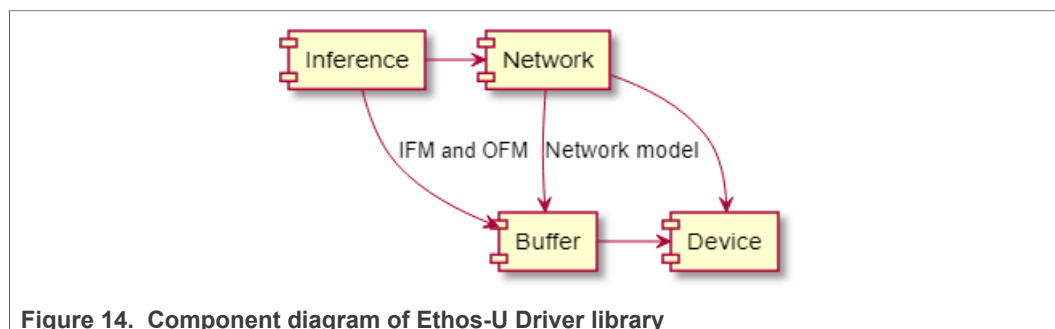


Figure 14. Component diagram of Ethos-U Driver library

The inference runner demonstrates how to dispatch inferences to the Ethos-U 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 Kernel 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.

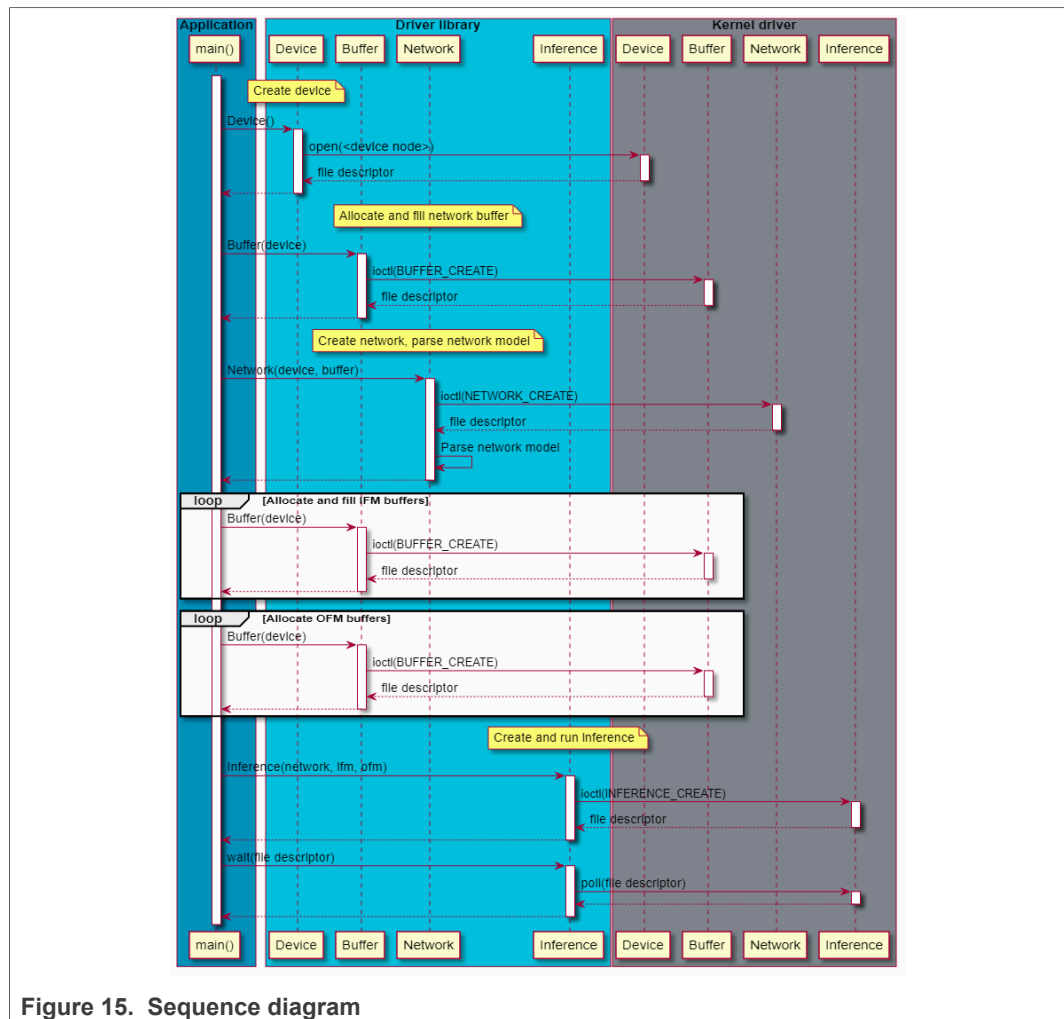


Figure 15. Sequence diagram

9.2.5.2 Ethos-U kernel driver interface

The Ethos-U 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 kernel driver.

When the 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.

9.2.5.3 Device and Buffer class

The Kernel 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.

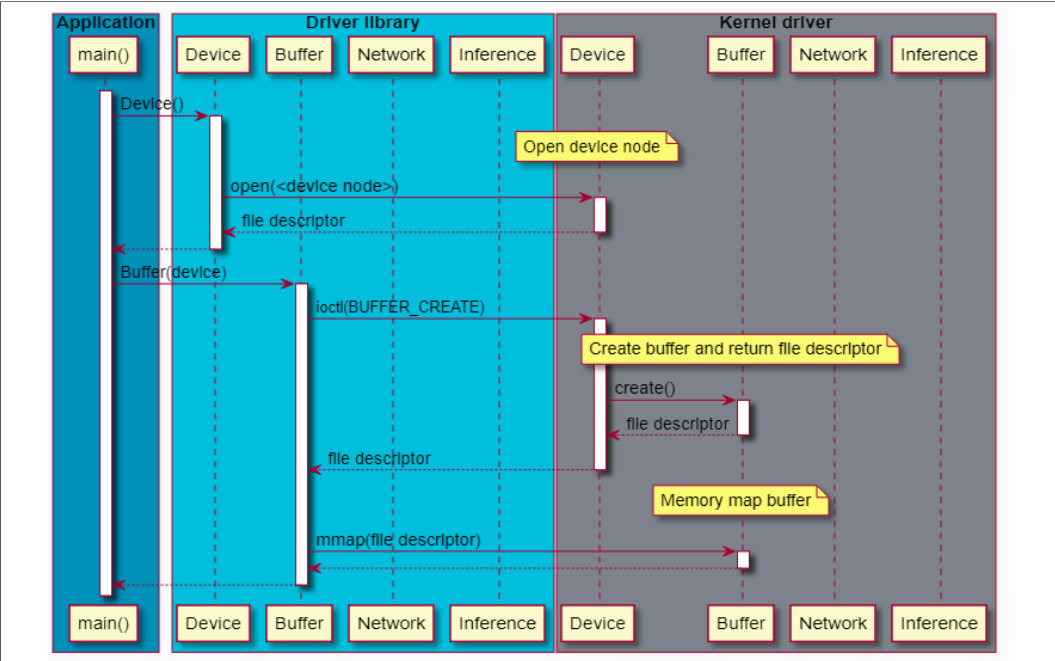


Figure 16. Device and Buffer class

9.2.5.4 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.

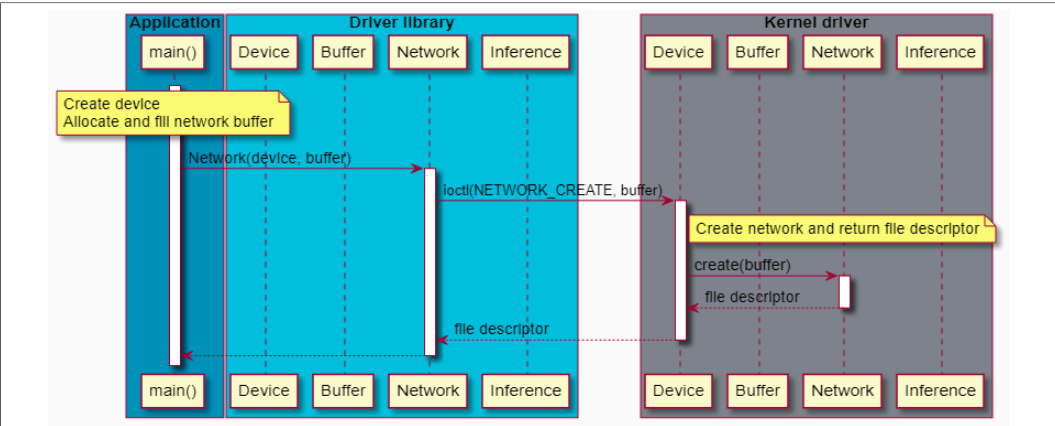


Figure 17. Network class

9.2.5.5 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 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.

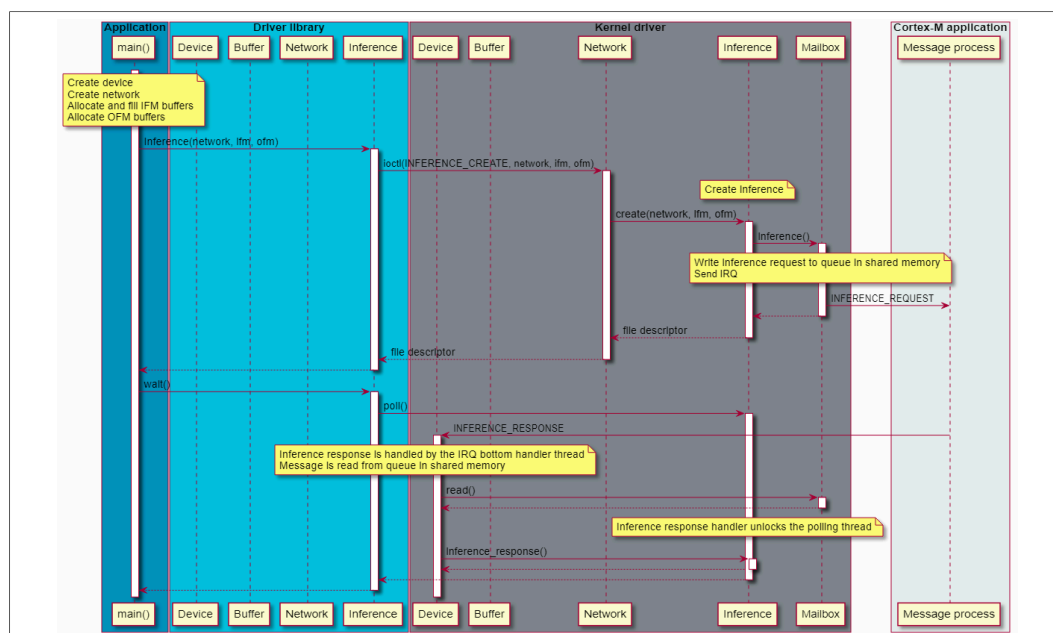


Figure 18. Inference class

9.2.5.6 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()
```

9.2.5.7 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...).

9.2.5.8 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)

# associate 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)
```

9.2.6 Inference with TFLite

9.2.6.1 Ethos-U custom operation

Ethos-U custom operator enables accelerating the inference on Ethos-U accelerator. The OP directly uses the Hardware accelerators driver to fully utilize the accelerators capabilities.

9.2.6.2 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.

9.2.6.3 Running image classification example

See [Section 9.2.3](#) to try the example.

See [Section 7.4.3](#) for how to build and use the Tensorflow Lite API with an application.

9.2.7 Building and deploying the Ethos-U firmware

9.2.7.1 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.

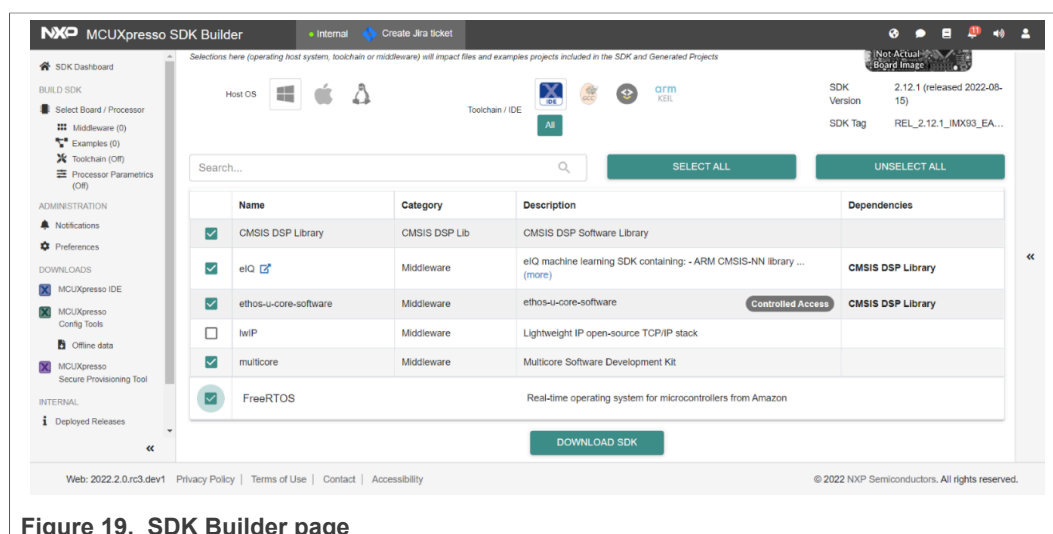


Figure 19. SDK Builder page

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
```

9.2.7.2 Ethos-U example applications

9.2.7.2.1 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 app 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.

9.2.7.2.2 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.

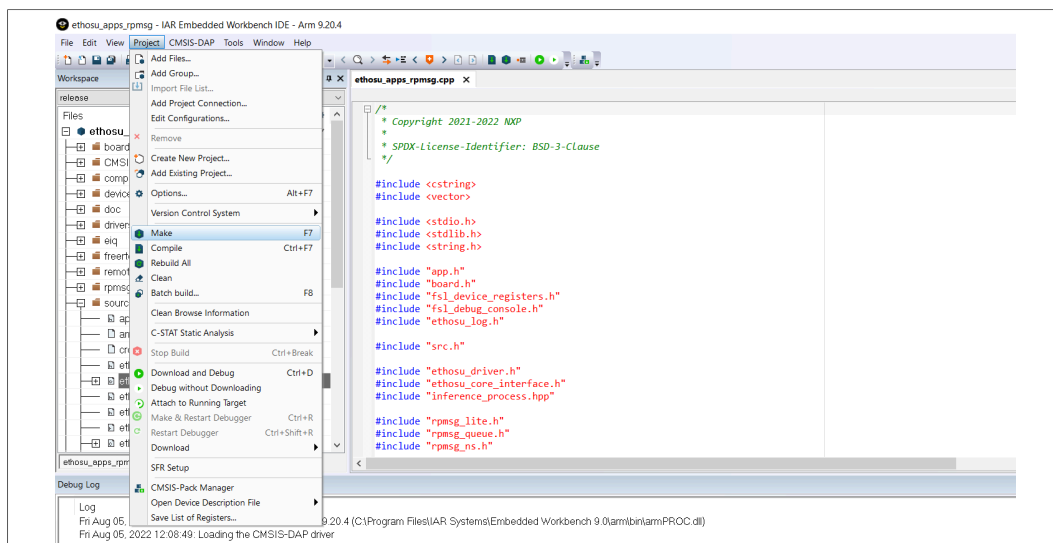


Figure 20. IAR Embedded Workbench for Arm

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

```
$ cd mcu-sdk-2.0/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
```

9.2.7.3 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 0x201e0000 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.

9.2.7.4 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 Cortex-M CPU or <TBD>.
- **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.

9.2.7.4.1 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:

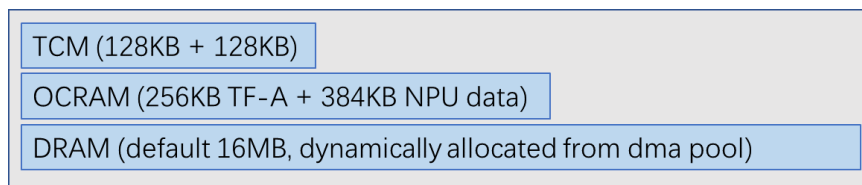
```
#define TENSOR_ARENA_SIZE (1024 * 1024 * 16)
uint8_t tensorArena[TENSOR_ARENA_SIZE];
```

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

9.2.8 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:

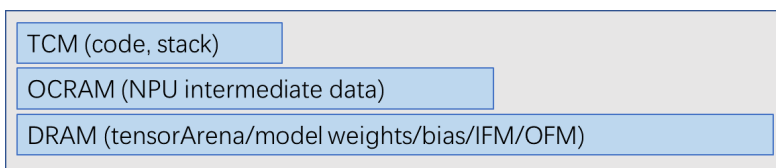


TCM size is 256 KB, usually used for Cortex-M runtime data. By design, this memory space is not allocated for system purpose after booting. How to use it effectively is left for user decision.

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

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

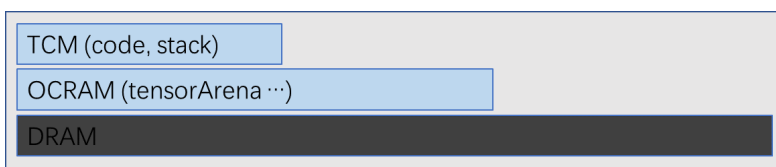
Ethos-U can only access the DRAM and OCRM memory by design. The current memory mapping for Ethos-U firmware is as follows:



With this configuration, the model data and tensor arena is allocated in DRAM and the OCRAM is used as NPU cache. “Dedicated_Sram” memory mode has to 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 standalone Cortex-M app, the memory mapping is as follows:



With this configuration, No DRAM is used. All the model data and tensorArena memory for NPU is allocated in OCRAM. “Sram_Only” memory mode has to 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}
```

9.2.9 Supported ML operators and constraints

The following table lists the TFLite operators that can be placed on the Ethos-U NPU. If the constraints are not met, then that operator is scheduled on the CPU instead. For any other TFLite operators not listed, they will be left untouched and scheduled on the CPU. Use the eIQ toolkit to view what operators are merged into Ethos-U operator for a model.

Table 2. Supported ML operators and constraints

Operator	Constraints
ABS	Generic , Specific
ADD	Generic , Specific
AVERAGE_POOL_2D	Generic , Specific
CONCATENATION	Generic , Specific
CONV_2D	Generic , Specific
DEPTHWISE_CONV_2D	Generic , Specific
EXPAND_DIMS	Generic , Specific
FULLY_CONNECTED	Generic , Specific
HARD_SWISH	Generic , Specific
LEAKY_RELU	Generic , Specific

Table 2. Supported ML operators and constraints...continued

Operator	Constraints
LOGISTIC	Generic
MAXIMUM	Generic , Specific
MAX_POOL_2D	Generic , Specific
MEAN	Generic , Specific (removed constraints #1)
MINIMUM	Generic , Specific
MUL	Generic , Specific
PACK	Generic
PAD	Generic , Specific (removed constraints #2)
QUANTIZE	Generic
RELU	Generic
RELU6	Generic
PRELU	Generic (newly added)
RELU_N1_TO_1	Generic
RESHAPE	Generic , Specific
RESIZE_BILINEAR	Generic , Specific
SHAPE	Generic
SLICE	Generic
SOFTMAX	Generic , Specific
SPLIT	Generic
SPLIT_V	Generic , Specific
SQUEEZE	Generic , Specific
STRIDED_SLICE	Generic , Specific
SUB	Generic , Specific
TANH	Generic
TRANSPOSE_CONV	Generic , Specific
UNPACK	Generic

Removed Constraints:

- Product of IFM height and width must be no greater than 256.
- The pad tensor can only pad width and height.

9.2.10 Profiling on hardware accelerators

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

Two environment variables are prepared for PMU profiling,
 ETHOSU_ENABLE_CYCLE_COUNTER and ETHOSU_PMU_CONFIG.

Set `ETHOSU_ENABLE_CYCLE_COUNTER` to **1** to enable cycle counter, **0** to disable cycle counter. The default value is **0**.

```
$ export ETHOSU_ENABLE_CYCLE_COUNTER="1"
```

Set `ETHOSU_PMU_CONFIG` to enable PMU counter. Up to 4 PMU event IDs can be added in this variable, e.g.,

```
$ export ETHOSU_PMU_CONFIG="3 4 5 6"
```

or

```
$ export ETHOSU_PMU_CONFIG="3 4"
```

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

Table 3. 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 3. 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 3. 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 one or two environment variables, you can run the TensorFlow Lite application. The PMU counter result is displayed on the console.

```
$ export ETHOSU_ENABLE_CYCLE_COUNTER="1"
$ export ETHOSU_PMU_CONFIG="1 3 4 5"
$ ./label_image -m mobilenet_v2_1.0_224_quant_vela.tflite -l
  labels.txt -i stopwatch.bmp
.....
Ethos_u PMUs : [ 201717971 34712 41893 3457751 ]
Ethos-u cycle counter: 237501410
.....
```

9.3 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.

9.3.1 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. The major differences are as follows:

- i.MX 93 NPU software stack depends on the offline tool to compile the Tensorflow Lite model to Ethos-U command stream for Ethos-U NPU execution, while i.MX 8M Plus uses online compilation to generate NPU commands stream for NPU execution. This means that i.MX 93 NPU users need to use the Vela tool to convert the Tensorflow Lite model to Vela model first. See [Section 9.2.4](#) for more details about the Vela tool.
- i.MX 8M Plus uses the Tensorflow Lite external delegate mechanism to support NPU acceleration, but i.MX 93 uses the Tensorflow Lite Custom OP mechanism to support NPU acceleration.

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 converted Vela model for i.MX 93 NPU acceleration. The following is the code snippet based on the Tensorflow Lite Python API. The major difference is the model that you use.



```

61 | '--ext_delegate_options',
62 | help= 'external delegate options, \
63 |     format: "option1: value1; option2: value2"'
64 |
65 | args = parser.parse_args()
66 |
67 | ext_delegate = None
68 | ext_delegate_options = {}
69 |
70 | # parse external delegate options
71 | if args.ext_delegate_options is not None:
72 |     options = args.ext_delegate_options.split(';')
73 |     for o in options:
74 |         kv = o.split(':')
75 |         if (len(kv) == 2):
76 |             ext_delegate_options[kv[0].strip()] = kv[1].strip()
77 |         else:
78 |             raise RuntimeError('Error parsing delegate option: ' + o)
79 |
80 | # Load external delegate
81 | if args.ext_delegate is not None:
82 |     print('Loading external delegate from {} with args: {}'.format(
83 |         args.ext_delegate, ext_delegate_options))
84 |     ext_delegate = [
85 |         tflite.load_delegate(args.ext_delegate, ext_delegate_options)
86 |     ]
87 |
88 | interpreter = tflite.Interpreter(
89 |     model_path=args.model_file,
90 |     experimental_delegates=ext_delegate,
91 |     num_threads=args.num_threads)
92 | interpreter.allocate_tensors()
93 |
94 | input_details = interpreter.get_input_details()
95 | output_details = interpreter.get_output_details()
96 |

```

Figure 21. Tensorflow Lite Python API

Another change is to use the converted Vela model for i.MX 93 NPU.

For TFLite C++ API, it is similar with the Python example above, and the only change is the model.

9.3.2 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 [Table 2](#) for i.MX 93 NPU operator support status and [Table 12](#) for i.MX 8M Plus NPU operator support status.

10 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 GStreamer developers to adopt neural network models easily and efficiently 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” thanks to 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 recurrence path helpers, cutting GStreamer pipeline cycle thanks to 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.

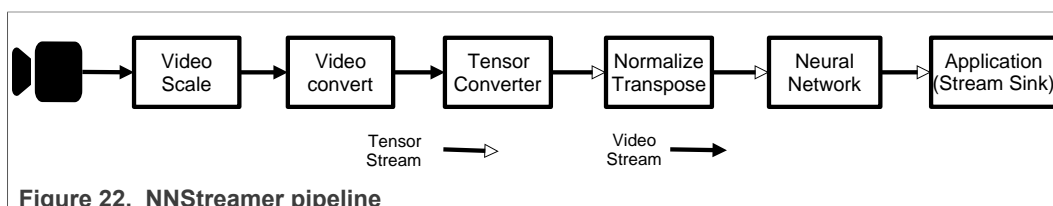


Figure 22. NNStreamer pipeline

There are two elements allowing adding user created features in run-time: [tensor_filter](#) and [tensor_decoder](#).

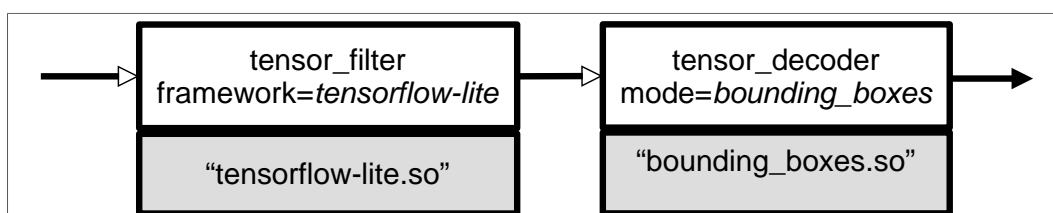


Figure 23. NNStreamer filter and decoder flow

While instantiating the `tensor_filter` and `tensor_decoder`, the framework and mode options respectively specify the target implementation thanks to 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, TVM, and DeepViewRT engines are supported.

Table 4. NNStreamer supported features

Framework/Tool	i.MX 93	i.MX 8M Plus	i.MX 8M Quad/8M Nano/8Quad Max/8QuadXPlus	i.MX 8M Mini/8 ULP
TensorFlow Lite	CPU/NPU	CPU/NPU/GPU	CPU/GPU	CPU
TVM	-	CPU/NPU/GPU	CPU/GPU	CPU (i.MX 8ULP only)
DeepViewRT	CPU	CPU/NPU/GPU	CPU/GPU	CPU
Custom C++	CPU	CPU	CPU	CPU
Custom Python	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 5. 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	-
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	-

Table 5. TensorFlow Lite engine...continued

Delegate	Tensor filter properties	USE_GPU_INFERENCE env variable
VX Delegate	framework=tensorflow-lite model=<path to .tflite model file> custom=Delegate:External ,ExtDelegateLib:libvx_delegate.so	0: NPU 1: GPU

Table 6. 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

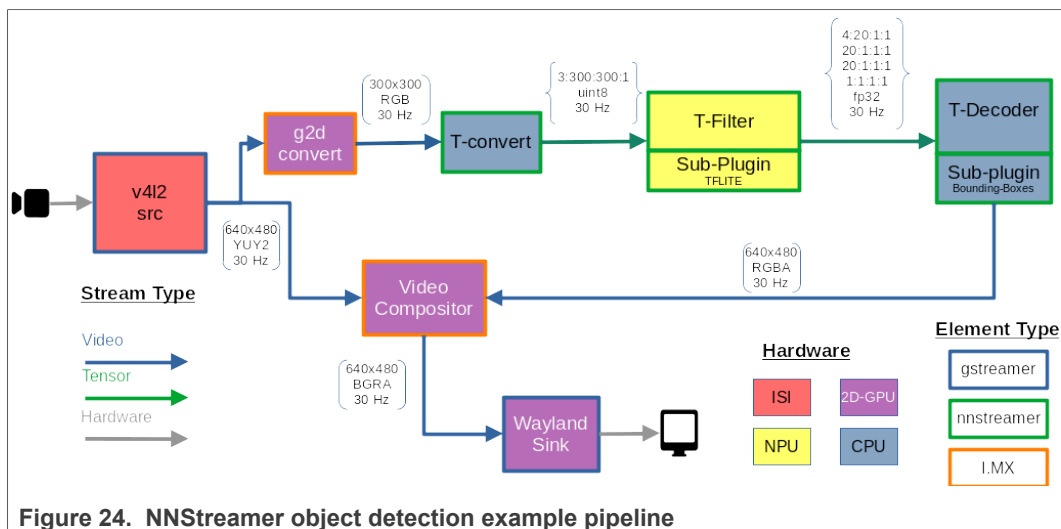
Table 7. DeepViewRT

Inference engine	Tensor filter properties	USE_GPU_INFERENCE env variable
None	framework=deepview-rt model=<path to .rtm model file>	-
deepview-rt-openvx	framework=deepview-rt model=<path to .rtm model file> custom=Engine:/usr/lib/deepview-rt-openvx.so	0: NPU 1: GPU

10.1 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 10.2](#).

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



10.2 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/NXPmicro/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 8. Features of NXP NNStreamer examples

Category	Engine	Platform	Implementation
Object detection: MobileNet SSD V2 Image Classification: MobileNet V1 Image Segmentation: DeepLab V3 Pose Detection: MoveNet	TensorFlow Lite DeepViewRT	i.MX 8M Plus i.MX 93	Shell script (gst-launch) Python

10.3 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.MX8M 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.

10.3.1 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"
```

In order to get GPU usage measurements, you must disable power saving in the GPU driver (galcore) thanks to command line kernel parameters. You can manually edit the bootargs uboot variable prior to execute the boot command, adding 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.

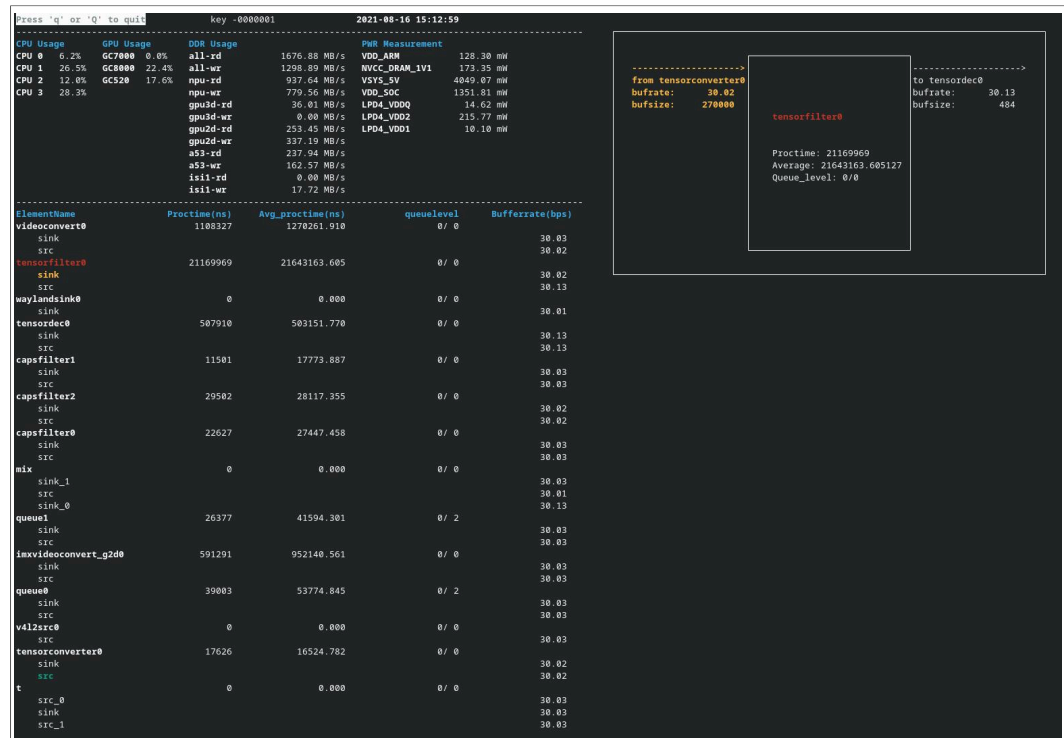


Figure 25. NNShark i.MX8M Plus example screenshot

10.3.2 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.

10.3.3 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)
```

11 eIQ Demos

11.1 AWS end-to-end SageMaker demo

AWS SageMaker demo shows how to use the pre-built AWS IoT Greengrass and SageMaker Edge Manager packages in i.MX BSP to build, deploy and manage machine learning model and device software with the cloud services.

AWS IoT Greengrass is software that extends cloud capabilities to local devices. It enables local device messaging via MQTT protocol, establishing a secure connection to the cloud. AWS SageMaker Edge Manager provides a software agent that runs on edge device for model inference and a separate SageMaker Neo cloud service for managing models on edge devices.

Features:

- AWS IoT Greengrass v2
- AWS Sagemaker Edge Manager agent
- AWS commad-line interface (AWS CLI) v1.21.12
- Auto script examples based on AWS CLI to provision, operate cloud service and devices
- Video inference demo, performing these tasks:
 - model deployment from cloud
 - USB camera capturing image frame
 - inference result return to cloud

11.1.1 AWS Greengrass/SageMaker demo workflow

This end-to-end flow (see also the following figure) uses a pre-trained mobilenetv2 image classification model to perform image classification at the edge with images captured from an USB camera. Inference is performed on the NPU of the i.MX 8M Plus, which allows for up to 50x performance increase when compared to running it on a CPU only. Results are uploaded to AWS IoT and input and output tensors are uploaded to Amazon S3.

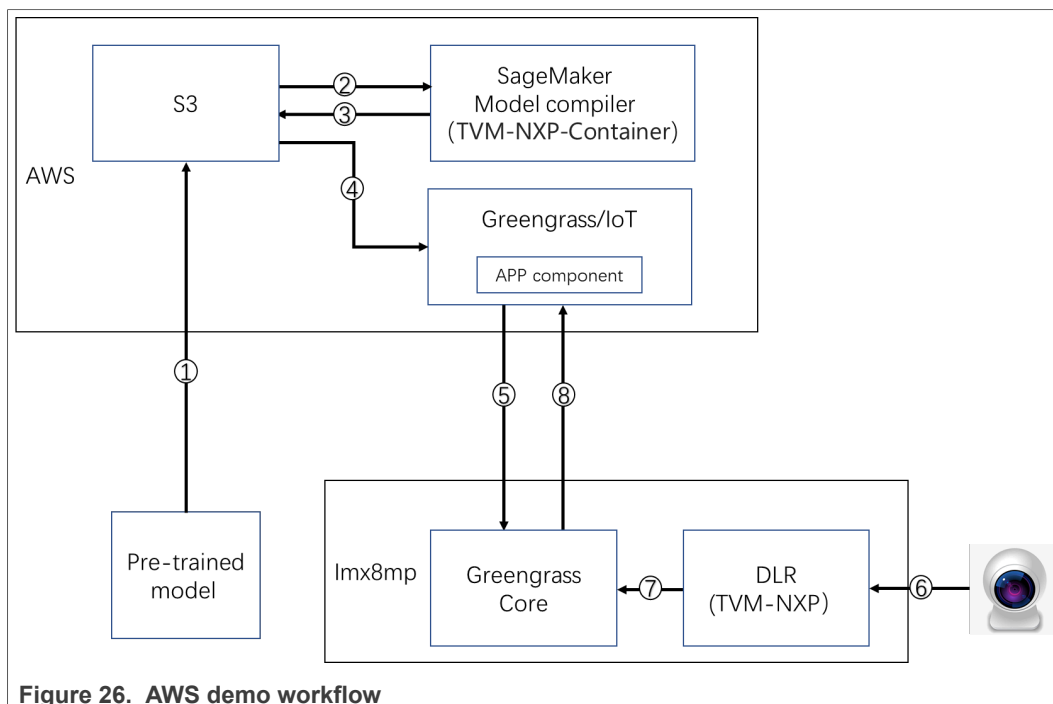


Figure 26. AWS demo workflow

The demo workflow is the following:

1. User uploads the pre-trained model to AWS S3.
2. SageMaker model compiler (a container that NXP offered to AWS) gets the model and compile the binary for the i.MX 8M Plus NPU.
3. The container uploads the binary back to S3.
4. Greengrass/IoT packages the model binary and users' codes to APP component.
5. Greengrass/IoT deploys the APP component to the edge device (i.MX 8M Plus).
6. The APP component gets the image from the camera.
7. The APP component runs the model on DLR (the TVM runtime offered by NXP).
8. Greengrass Core sends the inference result to AWS.

Requirements:

- NXP i.MX8MP-EVK BSP with pre-builtin AWS device packages
- An AWS account
- A certificate and private key for the AWS account
- An USB camera that connected to the NXP i.MX8MP-EVK

11.1.2 Getting started

11.1.2.1 Building BSP image

The building is based on using AWS packages and demo scripts:

- Follow the *i.MX Yocto Project User's Guide (IMXLXYOCTOUG)* to setup the project
- Repository initialization:

```
repo init -u https://github.com/nxp-imx/imx-manifest -b imx-
linux-hardknott -m imx-5.15.52-2.1.0_aws.xml
repo sync
```

- Build the image:

```
$ DISTRO=fsl-imx-wayland MACHINE=imx8mpevk source imx-aws-  
setup-release.sh -b build-imx8mp  
$ bitbake imx-image-full
```

- Flash the image to the SD card

```
$ sudo dd if=imx-image-full-imx8mpevk.wic of=/dev/xxxx
```

- Bootup the board with this SD card

11.1.2.2 Running demo scripts on device

The demo scripts can be found under `/usr/bin/dlr-demo-scripts` folder after booting-up the board. These scripts can operate with cloud resources and can setup the demo environment:

```
root@imx8mpevk:/usr/bin/dlr-demo-scripts# ls -l *.sh  
00_setup_cloud_services.sh  
01_create_greenrass_core.sh  
02_create_greenrass_role.sh  
03_upload_component_version.sh  
04_create_device_fleet_register_device.sh  
05_compile_and_package_neo_model.sh  
06_create_greenrass_deployment.sh  
07_setup_device_greenrass.sh  
10_clean_up.sh  
setup_cloud_service_and_device.sh
```

Before running these scripts, below environment variables needs to be specified:

- Set the AWS key environment:

```
$ export AWS_ACCESS_KEY_ID="YOUR AWS ACCESS KEY ID"  
$ export AWS_SECRET_ACCESS_KEY="YOUR AWS SECRET ACCESS KEY"  
$ export AWS_SESSION_TOKEN="YOUR AWS SESSION TOKEN"  
$ export AWS_REGION="us-west-2"    #replace with your aws  
region
```

- Optionally, set the ARN permission boundary if necessary. You can find it in *AWS management Console->IAM->Policies*:

```
$ export PERMISSIONS_BOUNDARY="YOUR PERMISSIONS BOUNDARY ARN"
```

- Optionally, set the camera device ID if necessary. The default value is 3:

```
$ export CAMERA_DEVICE=3
```

- Set the PROJECT_NAME to one unique string with lowercase letters only:

```
$ export PROJECT_NAME={project_name}
```

- Run the demo script:

```
$ cd /usr/bin/dlr-demo-scripts  
$ ./setup_cloud_service_and_device.sh
```

11.1.2.3 Check inference result

You can check inference results in two ways:

1. From the device Greengrass log file:

```
$ cd /greengrass/v2/logs
$ tail -f aws.sagemaker.
${project_name}_edgeManagerClientCamera Integration.log

stdout. {'index': '750', 'confidence': '0.4980392156862745',
'performance': '9.131669998168945', 'model_name':
'mobilenetv2-224-10-quant'}.
stdout. {'index': '831', 'confidence': '0.49411764705882355',
'performance': '15.126943588256836', 'model_name':
'mobilenetv2-224-10-quant'}.
```

2. From the cloud service console:

Navigate to the **AWS IoT Console** -> **Test** -> **MQTT test client** (see the below figure). Under "Subscribe" menu, select "em/inference". Every second, inference results should arrive on the "em/inference" topic with the result and confidence level.

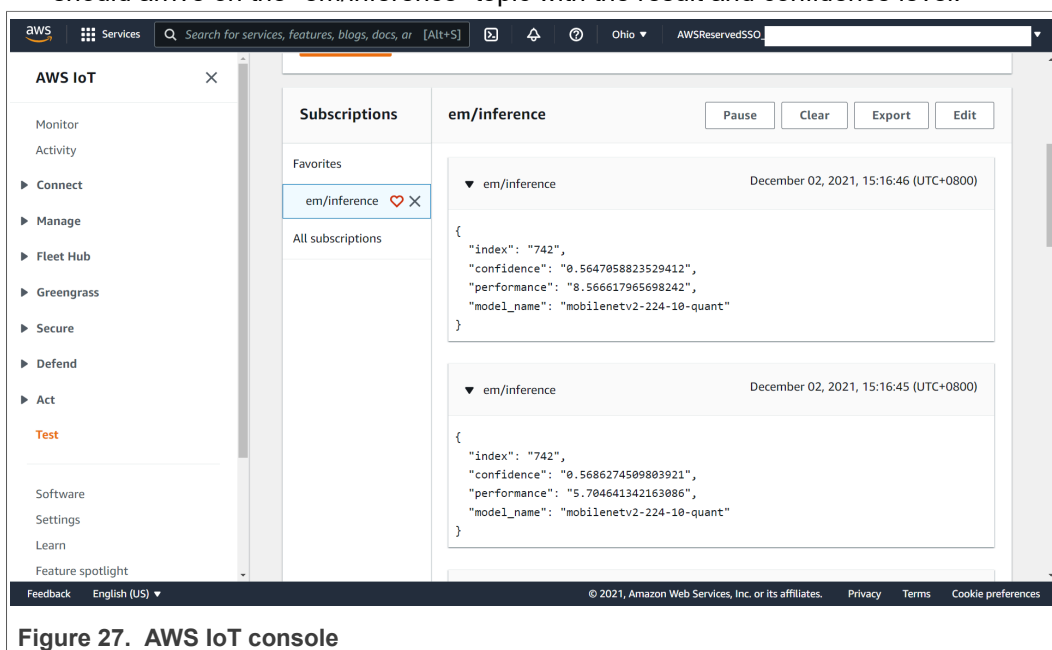


Figure 27. AWS IoT console

11.1.2.4 Clean up cloud environment

After testing, release cloud resources to save the cost:

```
$ /usr/bin/dlr-demo-scripts/10_clean_up.sh
```

11.1.3 Additional resources

Refer below links for more detailed information about AWS IoT Greengrass:

- AWS IoT Greengrass: [What is AWS IoT Greengrass? - AWS IoT Greengrass \(amazon.com\)](https://aws.amazon.com/greengrass/)
- SageMaker Edge Manager: [SageMaker Edge Manager - Amazon SageMaker](https://aws.amazon.com/sagemaker-edge/)
- Greengrass sagemaker example: [Greengrass-v2-sagemaker-edge-manager-python](#)
- IAM & Permission boundary: [Permission boundary](#)

12 Release Notes

12.1 Known issues and limitations

- HW Accelerators on i.MX8 does not support layers with dynamic shapes.
- The NPU on i.MX8 M Plus is not optimized for models with dynamic weights. The layers with dynamic weights (e.g. in FullyConnected layer) are computed significantly slower.

12.2 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.

12.3 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).

12.4 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.

12.5 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.

12.6 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_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 installed 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.

13 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 9. 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.
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.

Table 9. System variables summary...continued

Variable name	Description
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).

14 Neural Network API Reference

The neural-network operations and corresponding supported API functions are listed in the following table. See also [Section 3.2.3](#) for details about supported operators.

Table 10. Neural-network operations and supported API functions

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.46	TensorFlow Lite 2.8.0	ONNX 1.10.0
Activation				
elu	-	-	ELU	Elu
floor	ANEURALNETWORKS_FLOOR	-	Floor	Floor
leakyrelu	-	leaky_relu	-	LeakyReL
prelu	ANEURALNETWORKS_PRELU	prelu	PRELU	PreLu
relu	ANEURALNETWORKS_RELU	relu	RELU	ReLu
relu1	ANEURALNETWORKS_RELU1	-	RELU1	-
relu6	ANEURALNETWORKS_RELU6	relu6	RELU6	-
Hard_swish	ANEURALNETWORKS_HARD_SWISH	swish	HARD_SWISH	-
rsqrt	ANEURALNETWORKS_RSQRT	rsqrt	RSQRT	-
sigmoid	ANEURALNETWORKS_LOGISTIC	sigmoid/sigmoid_fast	LOGISTIC	Sigmoid
softmax	ANEURALNETWORKS_SOFTMAX	softmax	SOFTMAX	Softmax
softrelu	-	-	-	-

Table 10. Neural-network operations and supported API functions...continued

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.46	TensorFlow Lite 2.8.0	ONNX 1.10.0
sqrt	ANEURALNETWORKS_SQRT	sqrt	SQRT	Sqrt
tanh	ANEURALNETWORKS_TANH	tanh	TANH	TanH
bounded	-	-	-	-
linear	-	linear	-	-
Dense Layers				
dense	-	dense	-	-
Element Wise				
abs	ANEURALNETWORKS_ABS	abs	ABS	Abs
add	ANEURALNETWORKS_ADD	add	ADD	Add
clip_by_value	-	-	-	Clip
div	ANEURALNETWORKS_DivV	divide	DIV	Div
equal	ANEURALNETWORKS_EQUAL	-	EQUAL	Equal
exp	ANEURALNETWORKS_EXP	exp	EXP	Exp
log	ANEURALNETWORKS_LOG	log	LOG	Log
greater	ANEURALNETWORKS_GREATER	-	GREATER	Greater
greater_equal	ANEURALNETWORKS_GREATER_EQUAL	-	GREATER_EQUAL	-
less	ANEURALNETWORKS_LESS	-	LESS	Less
less_equal	ANEURALNETWORKS_LESS_EQUAL	-	LESS_EQUAL	-
logical_and	ANEURALNETWORKS_LOGICAL_AND	-	LOGICAL_AND	And
logical_or	ANEURALNETWORKS_LOGICAL_OR	-	LOGICAL_OR	Or
minimum	ANEURALNETWORKS_MINIMUM	-	MINIMUM	Min
maximum	ANEURALNETWORKS_MAXIMUM	-	MAXIMUM	Max
multiply	ANEURALNETWORKS_MUL	multiply	MUL	Mul
negative	ANEURALNETWORKS_NEG	-	NEG	Neg
not_equal	ANEURALNETWORKS_NOT_EQUAL	-	NOT_EQUAL	-
pow	ANEURALNETWORKS_POW	-	POW	POW
select	ANEURALNETWORKS_SELECT	-	SELECT	-
square	-	-	-	-
sub	ANEURALNETWORKS_SUB	subtract	SUB	Sub
where	-	-	-	Where
Image Processing				
resize_bilinear	ANEURALNETWORKS_RESIZE_BILINEAR	-	RESIZE_BILINEAR	Unsample

Table 10. Neural-network operations and supported API functions...continued

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.46	TensorFlow Lite 2.8.0	ONNX 1.10.0
resize_nearest_neighbor	ANEURALNETWORKS_RESIZE_NEAREST_NEIGHBOR	resize	RESIZE_NEAREST_NEIGHBOR	Resize
Matrix Multiplication				
fullconnect	ANEURALNETWORKS_FULLY_CONNECTED	-	FULLY_CONNECTED	-
matrix_mul	-	matmul/matmul_cache	-	-
Normalization				
batch_normalize	-	batchnorm	-	Batch Normalization
instance_normalize	-	-	-	Instance Normalization
l2normalize	ANEURALNETWORKS_L2_NORMALIZATION	-	L2_NORMALIZATION	-
localresponsenormalization	ANEURALNETWORKS_LOCAL_RESPONSE_NORMALIZATION	-	LOCAL_RESPONSE_NORMALIZATION	LRN
Reshape				
batch2space	ANEURALNETWORKS_BATCH_TO_SPACE_ND	-	BATCH_TO_SPACE_ND	-
concat	ANEURALNETWORKS_CONCATENATION	-	CONCATENATION	Concat
depth_to_space	ANEURALNETWORKS_DEPTH_TO_SPACE	-	DEPTH_TO_SPACE	DepthToSpace
expanddims	ANEURALNETWORKS_EXPAND_DIMS	-	EXPAND_DIMS	-
flatten	ANEURALNETWORKS_RESHAPE	-	-	-
gather	ANEURALNETWORKS_GATHER	-	GATHER	Gather
pad	ANEURALNETWORKS_PAD	-	PAD	Pad
permute	ANEURALNETWORKS_TRANSPOSE	-	TRANSPOSE	Transpose
reducemean	ANEURALNETWORKS_MEAN	reduce_mean	MEAN	ReduceMean
reducesum	ANEURALNETWORKS_SUM	reduce_sum	REDUCE_SUM	ReduceSum
gathernd	-	-	-	GatherND
reducemax	ANEURALNETWORKS_REDUCE_MAX	reduce_max	REDUCE_MAX	ReduceMax
reducemin	ANEURALNETWORKS_REDUCE_MIN	reduce_min	REDUCE_MIN	ReduceMin
reduceproduct	-	reduce_product	-	-
reshape	ANEURALNETWORKS_RESHAPE	-	RESHAPE	Reshape

Table 10. Neural-network operations and supported API functions...continued

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.46	TensorFlow Lite 2.8.0	ONNX 1.10.0
reverse	-	-	-	Reverse Sequence
slice	ANEURALNETWORKS_SLICE	-	SLICE	Slice
space2batch	ANEURALNETWORKS_SPACE_TO_BATCH_ND	-	SPACE_TO_BATCH_ND	-
split	ANEURALNETWORKS_SPLIT	-	SPLIT	Split
squeeze	ANEURALNETWORKS_SQUEEZE	-	SQUEEZE	Squeeze
strided_slice	ANEURALNETWORKS_STRIDED_SLICE	-	STRIDED_SLICE	-
unstack	-	-	-	-
RNN				
gru	-	-	-	GRU
lstm	-	-	UNIDIRECTIONAL_SEQUENCE_LSTM	-
lstmunit	ANEURALNETWORKS_LSTM	-	LSTM	LSTM
rnn	ANEURALNETWORKS_RNN	-	RNN	-
Sliding Window				
avg_pool	ANEURALNETWORKS_AVERAGE_POOL	avgpool/avgpool_ex	AVERAGE_POOL_2D	AveragePool
convolution	ANEURALNETWORKS_CONV_2D	conv/conv_ex	CONV_2D	Conv
deconvolution	ANEURALNETWORKS_TRANSPOSE_CONV_2D	transpose_conv2d_ex	TRANSPOSE_CONV	ConvTranspose
depthwise_convolution	ANEURALNETWORKS_DEPTHWISE_CONV_2D	-	DEPTHWISE_CONV_2D	-
Log_softmax	ANEURALNETWORKS_LOG_SOFTMAX	-	LOG_SOFTMAX	Logsoftmax
l2pooling	ANEURALNETWORKS_L2_POOL	-	L2_POOL_2D	-
max_pool	ANEURALNETWORKS_MAX_POOL	maxpool/maxpool_ex	MAX_POOL_2D	MaxPool
Others				
argmax	ANEURALNETWORKS_ARGMAX	argmax	ARGMAX	ArgMax
argmin	ANEURALNETWORKS_ARGMIN	-	ARGMIN	ArgMin
dequantize	ANEURALNETWORKS_DEQUANTIZE	-	DEQUANTIZE	DequantizeLinear
quantize	ANEURALNETWORKS_QUANTIZE	-	QUANTIZE	QuantizeLinear
roi_pool	ANEURALNETWORKS_ROI_ALIGN	-	-	-
shuffle_channel	ANEURALNETWORKS_CHANNEL_SHUFFLE	-	-	-

Table 10. Neural-network operations and supported API functions...continued

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.46	TensorFlow Lite 2.8.0	ONNX 1.10.0
tile	ANEURALNETWORKS_TILE	-	TILE	Tile
svdf	ANEURALNETWORKS_SVDF	-	SVDF	-
embedding_lookup	ANEURALNETWORKS_EMBEDDING_LOOKUP	-	EMBEDDING_LOOKUP	-
cast	ANEURALNETWORKS_CAST	-	CAST	Cast
ssd	-	ssd_decode_ nms_standard_ bbx/ssd_decode_ nms_variance_ bbx/ssd_nms_full	-	-

15 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 11. 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	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
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_DECONVOLUTION2D	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	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	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	✓	✓
	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	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	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	✓	✓
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	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
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	✓	✓
	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	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	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_FLOOR DIV	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MATRIX MUL	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	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	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_L2NORMALIZE_SCALE	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_GROUP_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
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	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RESHAPE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	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	✓	✓
VSI_NN_OP_PAD	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
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	✓	✓
	fp16		fp16	✓	✓
RNN Operations					

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
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_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_SVD	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					

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
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	✓	✓
	fp32		fp32	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	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_ELTSWEMAX	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	✓	✓

Table 11. OVXLIB operation support with GPU...continued

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	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_UPSAMPLE_SCALE	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	✓	✓

Table 11. OVXLIB operation support with GPU...continued

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

16 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**: Neural-Network Engine
- **PPU**: Parallel Processing Unit
- **TP**: Tensor Processor

Table 12. 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			✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp16	fp16	fp16			✓
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		✓	

Table 12. 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_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			✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
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			✓
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			✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	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			✓
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			✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
VSI_NN_OP_MAXIMUM	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_FLOOR DIV	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MATRIX MUL	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			✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	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			✓
	fp16		fp16			✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
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			✓
	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		✓	

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
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			✓
	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		✓	

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
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		✓	
	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			✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	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			✓
	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		✓	✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
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		✓	
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		✓	

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
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			✓
	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_ELWISEMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	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			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_UPSAMPLE_SCALE	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			✓

Table 12. OVXLIB operation support with NPU...continued

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	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			✓

17 Note About the Source Code in the Document

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

Copyright 2019 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.

18 Revision History

This table provides the revision history.

Revision history

Revision number	Date	Substantive changes
L5.4.47_2.2.0	09/2020	Initial release
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
LF5.10.9_1.0.0	03/2021	Kernel upgrade to 5.10.9 and Machine Learning upgrades
L5.4.70_2.3.2	04/2021	Patch release
LF5.10.35_2.0.0	06/2021	Upgraded to Yocto Project Hardknott and the kernel upgraded to 5.10.35
LF5.10.52_2.1.0	09/2021	Updated for i.MX 8ULP Alpha and the kernel upgraded to 5.10.52
LF5.10.72_2.2.0	12/2021	Upgraded the kernel to 5.10.72 and updated the BSP
LF5.15.5_1.0.0	03/2022	Upgraded to the 5.15.5 kernel, Honister Yocto, and Qt6
LF5.15.32_2.0.0	06/2022	Upgraded to the 5.15.32 kernel, U-Boot 2022.04, and Kirkstone Yocto
LF5.15.52_2.1.0	09/2022	Upgraded to the 5.15.52 kernel, and added the i.MX 93.
LF5.15.71_2.2.0	12/2022	Upgraded to the 5.15.71 kernel.

19 Legal information

19.1 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.

19.2 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 <http://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 data sheet 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.

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.

19.3 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

1	Software Stack Introduction	2	9	NN Execution on Hardware Accelerators	34
2	elQ Inference Runtime Overview on i.MX 8 Series	3	9.1	Hardware acceleration on i.MX 8 Series	34
3	TensorFlow Lite	4	9.1.1	Hardware accelerator description	34
3.1	TensorFlow Lite software stack	5	9.1.2	Profiling on hardware accelerators	35
3.2	Compute backends and delegates	7	9.1.3	Hardware accelerators warmup time	36
3.2.1	Built-in kernels	7	9.1.4	Switching between GPU and NPU	37
3.2.2	XNNPACK delegate	7	9.2	Hardware acceleration with Ethos-U on i.MX 93 platform	37
3.2.3	VX Delegate	7	9.2.1	Ethos-U subsystem overview	37
3.2.4	Ethos-U custom operator	8	9.2.2	Ethos-U software architecture	38
3.3	Delivery package	8	9.2.3	Getting started	39
3.4	Build details	8	9.2.4	Vela tool	40
3.5	Application development	9	9.2.4.1	Installing the Vela tool	40
3.5.1	Create CMake project which uses TensorFlow Lite	9	9.2.4.2	Compiling the TFLite model	41
3.5.2	Using Yocto SDK precompiled libraries	10	9.2.5	Inference with Ethos-U inference API	42
3.6	Running image classification example	11	9.2.5.1	Ethos-U driver library	42
3.7	Running benchmark applications	13	9.2.5.2	Ethos-U kernel driver interface	44
3.8	Post training quantization using TensorFlow Lite converter	16	9.2.5.3	Device and Buffer class	44
3.9	TensorFlow Lite for Microcontrollers on Xtensa HiFi4 core	17	9.2.5.4	Network class	45
4	Arm Compute Library	18	9.2.5.5	Inference class	46
4.1	Running a DNN with random weights and inputs	19	9.2.5.6	How to use the inference API	46
4.1.1	Running AlexNet using graph API	19	9.2.5.7	Interpreter class	47
5	ONNX Runtime	20	9.2.5.8	Interpreter Python wrapper	48
5.1	ONNX Runtime software stack	20	9.2.6	Inference with TFLite	49
5.2	Execution providers	21	9.2.6.1	Ethos-U custom operation	49
5.2.1	ONNX model test	22	9.2.6.2	Delivery package	49
5.2.2	C API	22	9.2.6.3	Running image classification example	49
5.2.2.1	Enabling execution provider	23	9.2.7	Building and deploying the Ethos-U firmware	49
5.2.3	ONNX performance test	23	9.2.7.1	Getting the source	49
6	PyTorch	23	9.2.7.2	Ethos-U example applications	50
6.1	Running image classification example	24	9.2.7.3	Deploy procedure	51
6.2	Building and installing wheel packages	24	9.2.7.4	Using the Ethos-U on Cortex-M	52
6.2.1	How to build	24	9.2.8	Memory hierarchy for Cortex-M	53
6.2.2	How to install	25	9.2.9	Supported ML operators and constraints	54
7	DeepViewRT	25	9.2.10	Profiling on hardware accelerators	55
7.1	DeepViewRT software stack	25	9.3	NPU transition guide from i.MX 8M Plus to i.MX 93	58
7.2	Delivery packages	27	9.3.1	Tensorflow Lite difference between i.MX 8M Plus and i.MX 93 NPU acceleration	58
7.3	Example applications	27	9.3.2	NPU supported operator list	59
7.3.1	Image labelling applications	28	10	Vision Pipeline with NNStreamer	59
7.3.2	Object detection applications	29	10.1	Object detection pipeline example	62
7.4	ModelRunner	29	10.2	NXP NNStreamer pipeline examples	63
7.4.1	DeepViewRT	29	10.3	Pipeline profiling	64
7.4.2	OpenVX	30	10.3.1	Enable profiling with NNShark	64
7.4.3	TensorFlow Lite	30	10.3.2	Adding power measurement to NNShark	65
7.4.4	ONNX Runtime	30	10.3.3	Known issues and limitations	66
8	TVM	31	11	elQ Demos	66
8.1	TVM software workflow	31	11.1	AWS end-to-end SageMaker demo	66
8.2	Getting started	31	11.1.1	AWS Greengrass/SageMaker demo workflow	66
8.2.1	Running example with RPC verification	31	11.1.2	Getting started	67
8.2.2	Running example individually on device	32	11.1.2.1	Building BSP image	67
8.3	How to build TVM stack on host	33	11.1.2.2	Running demo scripts on device	68
8.4	Supported models	34	11.1.2.3	Check inference result	68

11.1.2.4	Clean up cloud environment	69
11.1.3	Additional resources	69
12	Release Notes	70
12.1	Known issues and limitations	70
12.2	Release notes for LF5.15.71_2.2.0	70
12.3	Release notes for LF5.15.52_2.1.0	70
12.4	Release notes for LF5.15.32_2.0.0	71
12.5	Release notes for LF5.15.5-1.0.0	71
12.6	Release notes for LF5.10.72-2.2.0	72
13	List of Used Variables	73
14	Neural Network API Reference	74
15	OVXLIB Operation Support with GPU	78
16	OVXLIB Operation Support with NPU	92
17	Note About the Source Code in the Document	106
18	Revision History	107
19	Legal information	108

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