Document identifier: IMXMLUG
Rev. LF5.10.72_2.2.0, 17 December 2021

i.MX Machine Learning User's Guide

Contents

Chapter 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 six inference engines are currently supported in the NXP eIQ software stack: Arm NN, TensorFlow Lite, ONNX Runtime, PyTorch, OpenCV, and DeepView™RT. The following figure shows the supported eIQ inference engines accross the computing units.

eIQ Inference Engine Deployment														
NXP eIQ Inference Engines & Libraries	PyTorch	arm NN	ONNX RUNTIME	TensorFlow Lite	OpenCV	DeepViewRT	arm NN	ONNX RUNTIME	TensorFlow Lite	DeepViewRT	arm NN	ONNX RUNTIME	TensorFlow Lite	DeepViewRT
Compute Engines	Cortex-A						GPU				NPU			
i.MX 8M Plus	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
i.MX 8QuadMax	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	NA	NA	NA	NA
i.MX 8QuadXPlus	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	NA	NA	NA	NA
i.MX 8M Quad, Nano	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	NA	NA	NA	NA
i.MX 8M Mini, 8ULP	✓	✓	✓	✓	✓	✓	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, Arm NN, ONNX Runtime, TensorFlow Lite, and DeepViewRT also support acceleration on the GPU or NPU through Neural Network Runtime (NNRT). See also eIQ Inference Runtime Overview.

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

Chapter 2
eIQ Inference Runtime Overview

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 ModelRunner for more details.

The NNRT supplies different backends for Android NN HAL, Arm NN, 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, TensorFlow Lite, and Arm NN 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.

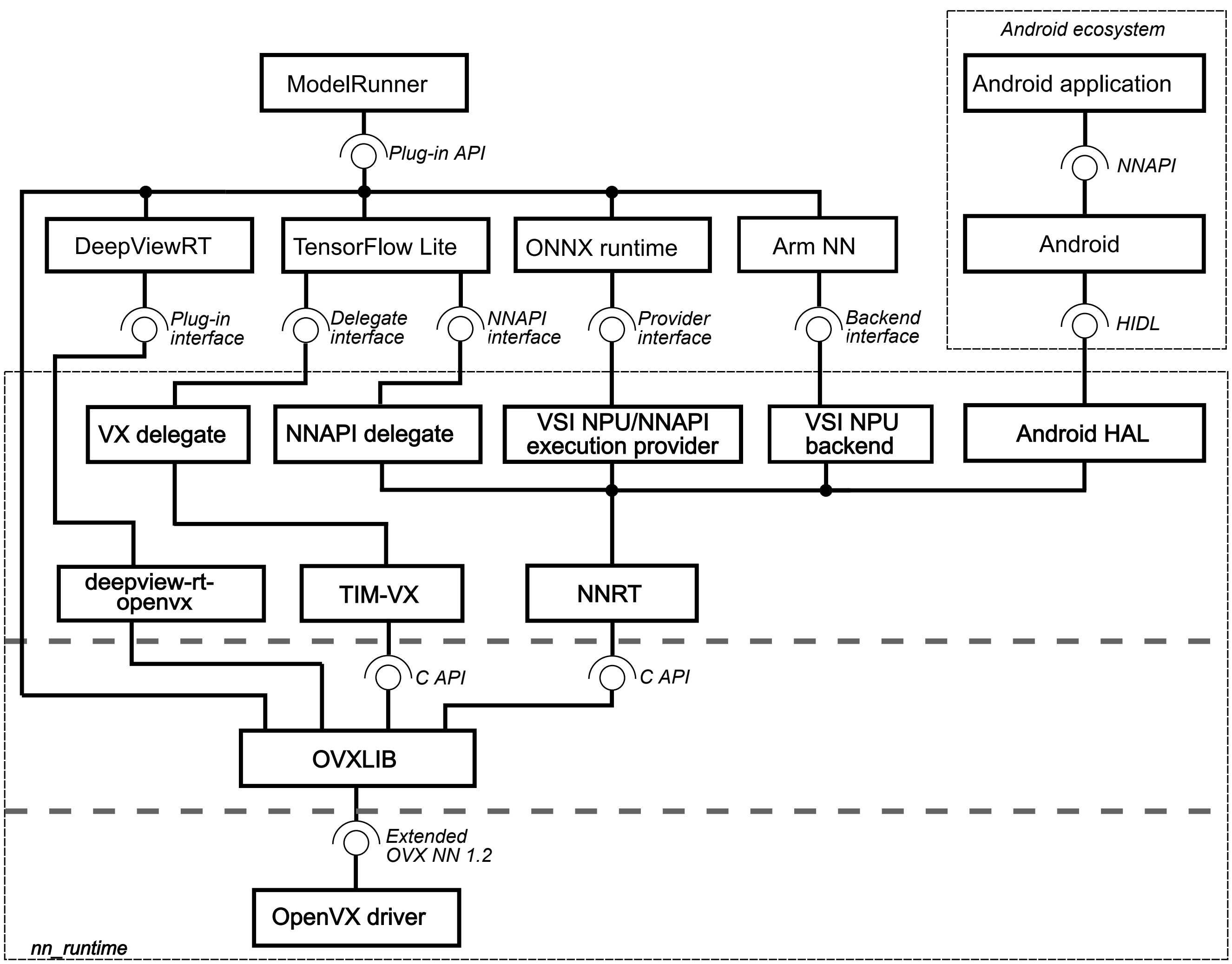

Figure 2. eIQ inference software architecture

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 TensorFlow Lite, the NNRT supports NNAPI Delegate. It supports most operations in Android NNAPI v1.2
- For Arm NN, the NNRT registers itself as a compute backend
- 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.

Chapter 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 or Android OS Neural Networks API (NNAPI) on various i.MX 8 platforms (in the NXP eIQ).

The TensorFlow Lite source code for this Yocto Linux release is available at this repository, branch lf-5.10.72_2.2.0. This repository is a fork of the mainline https://github.com/tensorflow/tensorflow, and it is optimized for NXP i.MX8 platforms.

Features:

- TensorFlow Lite v2.6.0
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores
- Parallel computation using GPU/NPU hardware acceleration (on shader or convolution units)
- C++ and Python API (supported Python version 3)
- Per-tensor and Per-channel quantized models support

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 Android NNAPI driver or VX Delegate

See Software Stack Introduction for some details about supporting of computation on GPU/NPU hardware accelerator on different hardware platforms.

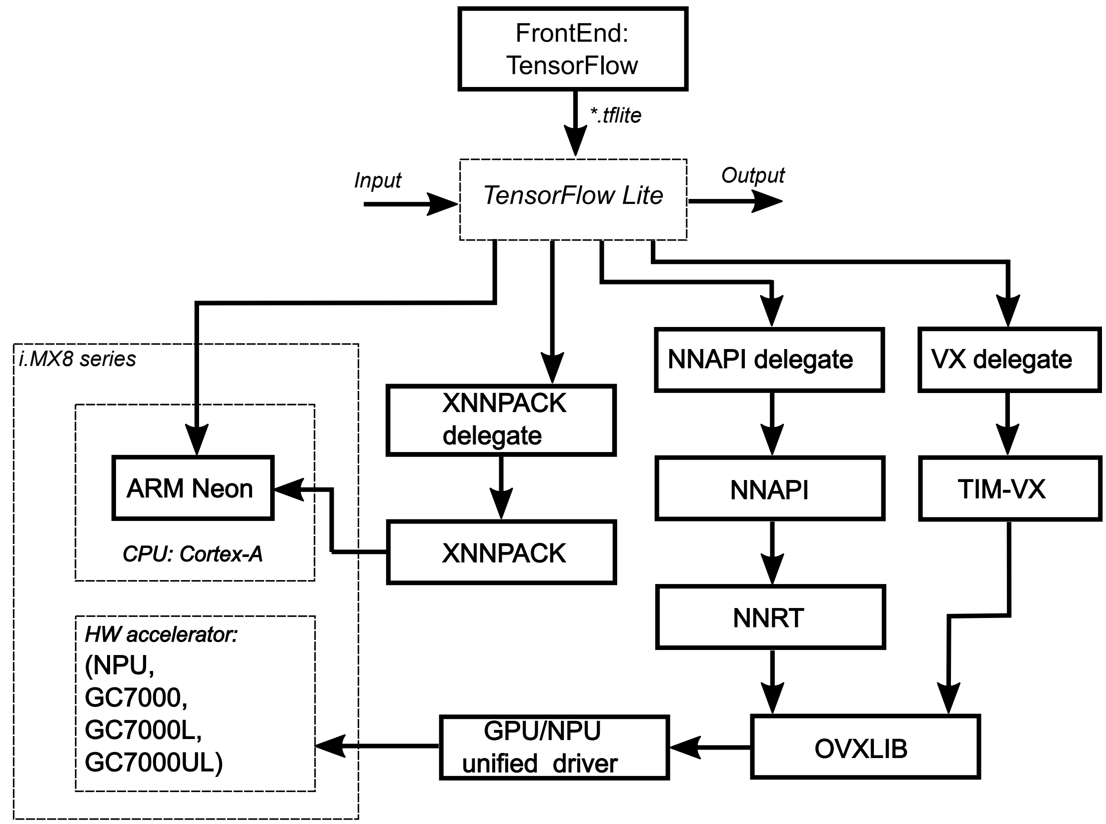

Figure 3. TensorFlow Lite software stack

NOTE

The first execution of model inference using the NNAPI or 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 theirs dependencies to perform computation specified by the model. The computation graph is built during the model parsing phase.

The NNAPI and VX Delagate 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).

The GPU/NPU hardware accelerator driver support both per-tensor and per-channel quantized models. The GPU/NPU hardware accelerator 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 NNAPI delegate

NNAPI delegate enables accelerating the inference on on-chip hardware accelerator. The delegate is based on Android's Neural Network API (NNAPI) specification. The full specification is available here: https://developer.android.com/ndk/reference/group/neural-networks.

The TensorFlow Lite library uses the Android NNAPI implementation from the GPU/NPU driver for running inference using the GPU/NPU hardware accelerator. The implemented NNAPI version is 1.2 which has some limitations in supported tensor data types and operations, compared to the feature set of TensorFlow Lite. Therefore, some models may work without acceleration enabled, but may fail when using the NNAPI. For the full list of supported features, see the NN HAL versions section of the NNAPI documentation.

NNAPI specification comes with its own operator set, which includes most but not all operator from TensorFlow Lite operator set. Moreover, not all variants of TensorFlow Lite operators are supported by NNAPI. This is valid for hardware accelerators operator support, where some operators are supported by the accelerator but are not part of NNAPI specification. Therefore, some layers execution can unnecessarily fall back on CPU, even if the HW accelerator supports the particular layer.

For all operators in the model, which was refused by the NNAPI delegate the TensorFlow Lite runtime print a warning message with reason why the operator was refused by the delegate:

```
WARNING: Operator ARG_MAX (v1) refused by NNAPI delegate: NNAPI only supports int32 output.
```

This information can be used to optimize the model for better performance.

NOTE

The NNAPI Delegate for Linux platform will be deprecated in the future. The NNAPI Delegate is not supported in the Python API.

3.2.4 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.

3.3 Delivery package

The TensorFlow Lite is available using Yocto Project recipes.

The TensorFlow Lite delivery package contains:

[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

- 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 Application development for more details.

There are following delegates available in the TensorFlow Lite 2.6.0 delivery package:

- XNNPACK Delegate
- NNAPI 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`)
- NNAPI Delegate support[2] (`TFLITE_ENABLE_NNAPI=On`), including warning messages for refused operation (`TFLITE_ENABLE_NNAPI_VERBOSE_VALIDATION=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 Create CMake project which uses TensorFlow Lite.
- 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* (IMXLXYOCTOUG) 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)

[2] Only for platforms with OpenVX support

- 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=${OE_CMAKE_TOOLCHAIN_FILE} -DTFLITE_ENABLE_XNNPACK=on \
-DTFLITE_ENABLE_RUY=on -DTFLITE_ENABLE_NNAPI=on -DTFLITE_ENABLE_VX=on \
-DTFLITE_ENABLE_NNAPI_VERBOSE_VALIDATION=on \
-DTIM_VX_INSTALL=${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 -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-2.6.0/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 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 with using the XNNPACK delegate, use the --use_xnnpack=true switch:

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

To run the example with the same model on the GPU/NPU hardware accelerator, add the --use_nnapi=true (for NNAPI Delegate) or --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 NNAPI 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 with NPU acceleration enabled should be 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-2.6.0/examples
```

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

```
$ python3 label_image.py
```

The output should be as follows:

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

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

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-2.6.0/examples
```

Benchmarking instructions are as follows:

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

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

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

The output of the benchmarking application should be similar to:

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

Running benchmark for at least 1 iterations and at least 0.5 seconds but terminate if exceeding
150 seconds.
count=4 first=160408 curr=155384 min=155384 max=160408 avg=156869 std=2076
Running benchmark for at least 50 iterations and at least 1 seconds but terminate if exceeding
150 seconds.
count=50 first=155586 curr=155424 min=155274 max=155622 avg=155443 std=81

Inference timings in us: Init: 3051, First inference: 160408, Warmup (avg): 156869, Inference
(avg): 155443
Note: as the benchmark tool itself affects memory footprint, the following is only APPROXIMATE to the
actual memory footprint of the model at runtime. Take the information at your discretion.
Peak memory footprint (MB): init=4.49219 overall=10.6133
```

To run the inference 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 NNAPI Delegate, add the `--use_nnapi=true` switch:

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

In addition to the output presented above, the NNAPI Delegate reports details on why a particular layer was refused by the delegate:

```
INFO: Created TensorFlow Lite delegate for NNAPI.
WARNING: Operator RESIZE_BILINEAR (v1) refused by NNAPI delegate: Operator refused due
performance reasons.
WARNING: Operator RESIZE_BILINEAR (v1) refused by NNAPI delegate: Operator refused due
performance reasons.
WARNING: Operator RESIZE_BILINEAR (v1) refused by NNAPI delegate: Operator refused due
performance reasons.
WARNING: Operator ARG_MAX (v1) refused by NNAPI delegate: NNAPI only supports int32 output.
Explicitly applied NNAPI delegate, and the model graph will be partially executed by the delegate w/
2 delegate kernels.
```

And detailed profiling information is available:

```
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/r2.6/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 2.6.0. It is recommended to use the TensorFlow Lite converter from corresponding TensorFlow version. The TensorFlow Lite runtime should be compatible with models generated by previous version of TensorFlow Lite Converter, however this backward compatibility is not guaranteed. Usage of successive version of TensorFlow Lite converter shall be avoided.

The 2.6.0 version of the converter has the following properties:

- In the post training quantization regime, the per-channel quantization is the only option. The per-tensor quantization is available only in connection with quantization aware training.
- Input and output tensors quantization is supported by setting the required data type in `inference_input_type` and `inference_output_type`.
- TOCO or MLIR based conversions are available. This is controlled by the `experimental_new_converter` attribute. As TOCO is becoming obsolete, MLIR-based conversion is already set by default in the 2.6.0 version of the converter.

  MLIR converter uses dynamic tensor shapes, what means the batch size of the input tensor is unspecified. Dynamic tensor shapes are not supported, by the GPU and NPU hardware accelerators and this shall be turned off. Standard installation of TensorFlow does not provide API to control the dynamic tensor shape feature, but can be deactivated in the tensorflow 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.

Chapter 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.

Arm Compute Library is designed as a compute engine for the Arm NN framework, so it is suggested to use Arm NN unless there is a need for a more optimized runtime.

Source codes are available at https://source.codeaurora.org/external/imx/arm-computelibrary-imx.

Features:

- Arm Compute Library 21.08
- 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-21.08/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-21.08/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
```

Chapter 5
Arm NN

Arm NN is an open-source inference engine framework developed by Linaro Artificial Intelligence Initiative, which NXP is a part of. It does not perform computations on its own, but rather delegates the input from multiple model formats such as TensorFlow Lite, or ONNX, to specialized compute engines.

Source codes are available at https://source.codeaurora.org/external/imx/armnn-imx.

Features:

- Arm NN 21.08
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores provided by the ACL Neon backend
- Parallel computation using GPU/NPU hardware acceleration (on shader or convolution units) provided by the VSI NPU backend
- C++ and Python API (supported Python version 3)
- Supports multiple input formats (TensorFlow Lite, ONNX)
- Off-line tools for serialization, deserialization, and quantization (must be built from source)

5.1 Arm NN software stack

The Arm NN software stack is shown in the picture below. Arm NN supports computation on the following HW units:

- 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

See Software Stack Introduction for details about the support of GPU/NPU accelerators for each hardware platform.

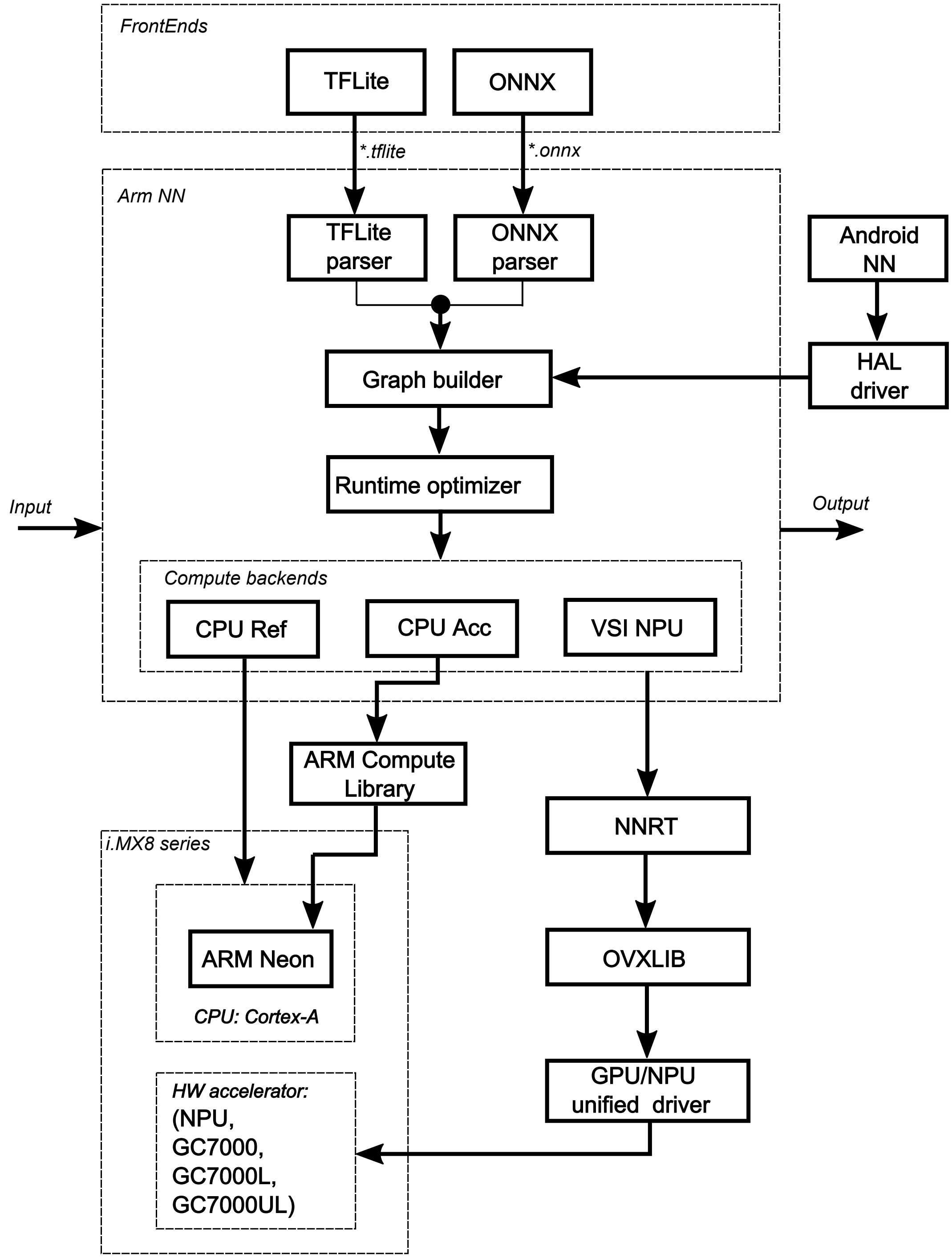

Figure 5. Arm NN SW stack

5.2 Compute backends

Arm NN on its own does not specialize in implementing compute operations. There is only the C++ reference backend running on the CPU, which is not optimized for performance and should be used for testing, checking results, prototyping, or as the final fallback, if none of the other backends supports a specific layer. The other backends delegate compute operations to other more specialized libraries such as Arm Compute Library (ACL).

- **For the CPU:** there is the NEON backend, which uses Arm Compute Library with the Arm NEON SIMD extension.

- **For the GPUs and NPUs:** NXP provides the VSI NPU backend, which leverages the full capabilities of i.MX 8's GPUs/NPUs using OpenVX and provides a great performance boost. ACL OpenCL backend, which you might notice in the source codes, is not supported due to Arm NN OpenCL requirements not being fulfilled by the i.MX 8 GPUs.

To activate the chosen backend while running the examples described in the following sections, add the following argument. The user can give multiple backends for the example applications. A layer in the model will be executed by the first backend, which supports the layer:

```
<example_binary> --compute=arg
```

Where `arg` can be:

- **CpuRef** = Arm NN C++ backend (no SIMD instructions); a set of reference implementations with NO acceleration on the CPU, which is used for testing, prototyping, or as the final fallback. It is very slow.
- **CpuAcc** = ACL NEON backend (runs on CPU with NEON instructions = SIMD)
- **VsiNpu** = For the GPUs and NPUs, NXP provides the VSI NPU backend, which leverages the full capabilities of i.MX 8's GPUs.

To develop your own application, make sure that you pass the chosen backend (CpuAcc, VsiNpu, or CpuRef) to the Optimize function for inference.

NOTE

VsiNpu backend delegates execution to the OpenVX driver. It depends on the driver if the workload is executed on the NPU or the GPU.

5.3 Running Arm NN tests

Arm NN SDK provides a set of tests, which can also be considered as demos showing what Arm NN does and how to use it. They load neural network models of various formats (TensorFlow Lite, ONNX), run the inference on a specified input data, and output the inference result. Arm NN tests are built by default when building the Yocto image and are installed in `/usr/bin/armnn-21.08`. Note that input data, model configurations, and model weights are not distributed with Arm NN. The user must download them separately and make sure they are available on the device before running the tests. However, Arm NN tests do not come with a documentation. Input file names are hardcoded, so investigate the code to find out what input file names are expected.

To help get started with Arm NN, the following sections provide details about how to prepare the input data and how to run Arm NN tests. All of them use well-known neural network models. Therefore, with only a few exceptions, such pre-trained networks are available freely on the Internet. Input images, models, formats, and their content was deduced using code analysis. However, this was not possible for all the tests, because either the models are not publicly available or it is not possible to deduce clearly what input files are required by the application. General workflow is first to prepare data on a host machine and then to deploy it on the board, where the actual Arm NN tests will be run.

The following sections assume that neural network model files are stored in a folder called models and input image files are stored in a folder called data. Create this folder structure on the larger partition using the following commands:

```
$ cd /usr/bin/armnn-21.08
$ mkdir data
$ mkdir models
```

5.3.1 TensorFlow Lite tests

Arm NN SDK provides the following test for TensorFlow Lite models:

```
/usr/bin/armnn-21.08/TfLiteInceptionV3Quantized-Armnn
/usr/bin/armnn-21.08/TfLiteInceptionV4Quantized-Armnn
/usr/bin/armnn-21.08/TfLiteMnasNet-Armnn
/usr/bin/armnn-21.08/TfLiteMobileNetSsd-Armnn
/usr/bin/armnn-21.08/TfLiteMobilenetQuantized-Armnn
/usr/bin/armnn-21.08/TfLiteMobilenetV2Quantized-Armnn
```

```
/usr/bin/armnn-21.08/TfLiteResNetV2-Armnn
/usr/bin/armnn-21.08/TfLiteVGG16Quantized-Armnn
/usr/bin/armnn-21.08/TfLiteResNetV2-50-Quantized-Armnn
/usr/bin/armnn-21.08/TfLiteMobileNetQuantizedSoftmax-Armnn
/usr/bin/armnn-21.08/TfLiteYoloV3Big-Armnn
```

NOTE

For the full list of the supported operators, see TensorFlow Lite support.

The following table provides the list of all dependencies for each Arm NN TensorFlow Lite binary example.

Table 1. Arm NN TensorFlow Lite example dependencies

Arm NN binary	Model file name	Renamed input files and data
TfLiteInceptionV3Quantized-Armnn	inception_v3_quant.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteMnasNet-Armnn	mnasnet_1.3_224.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteMobilenetQuantized-Armnn	mobilenet_v1_1.0_224_quant.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteMobilenetV2Quantized-Armnn	mobilenet_v2_1.0_224_quant.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteResNetV2-50-Quantized-Armnn	Model not available	N/A
TfLiteInceptionV4Quantized-Armnn	Model not available	N/A
TfLiteMobileNetSsd-Armnn	Model not available	N/A
TfLiteResNetV2-Armnn	Model not available	N/A
TfLiteVGG16Quantized-Armnn	Model not available	N/A
TfLiteMobileNetQuantizedSoftmax-Armnn	Model not available	N/A
TfLiteYoloV3Big-Armnn	Model not available	N/A

NOTE

Some models or input files are not publicly available.

Perform the following steps to run each of the examples above:

1. Download the model (column 2 of the table) and copy it to the *models* folder on the device.
2. Download the input data (column 3 of the table) and copy it to the *data* folder on the device. Rename all JPG images according to the expected input (shark.jpg, Dog.jpg, Cat.jpg). All these names are case sensitive.
3. Run the test:

```
$ cd /usr/bin/armnn-21.08
$ ./<armnn_binary> --data-dir=data --model-dir=models
```

5.3.2 ONNX tests

The Arm NN provides the following set of tests for ONNX models:

```
/usr/bin/armnn-21.08/OnnxMnist-Armnn
/usr/bin/armnn-21.08/OnnxMobileNet-Armnn
```

The following table provides the list of all dependencies for each Arm NN ONNX binary example.

Table 2. Arm NN ONNX example dependencies

Arm NN binary	Model file name	Renamed input files and data	Renamed model file name
OnnxMnist-Armnn	model.onnx	t10k-images.idx3-ubyte, t10k-labels.idx1-ubyte	mnist_onnx.onnx
OnnxMobileNet-Armnn	mobilenetv2-1.0.onnx	shark.jpg, Dog.jpg, Cat.jpg	mobilenetv2-1.0.onnx

Perform the following steps to run each of the examples above:

1. Download the model (column 2 of the table).
2. Rename the original model name to the new model name (column 4 of the table) and copy it to the *models* folder on the device.
3. Download the input data (column 3 of the table) and copy it to the *data* folder on the device.
4. Rename all the JPG images according to the expected input (shark.jpg, Dog.jpg, Cat.jpg). All these names are case sensitive.
5. Run the test:

```
$ cd /usr/bin/armnn-21.08
$ ./<armnn_binary> --data-dir=data --model-dir=models
```

5.4 Using Arm NN in a custom C/C++ application

You can create your own C/C++ applications for the i.MX 8 family of devices using Arm NN capabilities. This requires writing the code using the Arm NNAPI, setting up the build dependencies, cross-compiling the code for an aarch64 architecture, and deploying your application. Below is a detailed description for each of these steps:

1. Write the code.

   A good starting point to understand how to use Arm NNAPI in your own application is to go through "How-to guides" provided by Arm. These include application which shows how to load and run inference for an MNIST TensorFlow model.

2. Prepare and install the SDK.

   From a software developer's perspective, Arm NN is a library. Therefore, to create and build an application, which uses Arm NN, you need header files and matching libraries. For how to build the Yocto SDK, see the *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG). By default, header files and libraries are not added. To make sure that the SDK contains both the header files and the libraries, add the following to your `local.conf`.

   ```
   TOOLCHAIN_TARGET_TASK_append += " armnn-dev"
   ```

3. Build the code.

   To build the "armnn-mnist" example provided by Arm, you need to make a few modifications to make it work with a Yocto cross-compile environment:

   - Remove the definition of *ARMNN_INC* and all its uses from Makefile. The Arm NN headers are already available in the default include directories.
   - Remove the definition of *ARMNN_LIB* and all its uses from Makefile. The Arm NN libraries are already available in the default linker search path.
   - Replace "g++" with "${CXX}" in Makefile.

   Build the example:

- Setup the SDK environment:

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

- Run make:

```
$ make
```

4. Copy the built application to the board.

Input data are described in the "How-to guides". If the image you are using on your board is the same as the one for which you built the SDK, all the runtime dynamic libraries needed to run the application should be available on the board.

5.5 Python interface to Arm NN (PyArmNN)

PyArmNN is a Python extension for Arm NN SDK. PyArmNN provides interface similar to Arm NN C++ API. It is supported only for Python 3.x and not Python 2.x.

For full API documentation please refer to NXPmicro GitHub: https://github.com/NXPmicro/pyarmnn-release

5.5.1 Getting started

The easiest way to begin using PyArmNN is by using the Parsers. We will demonstrate how to use them below:

Install dependency.

```
pip3 install imageio
```

Create a parser object and load your model file.

```
import pyarmnn as ann
import imageio
# ONNX parser also exist.
parser = ann.ITfLiteParser()
network = parser.CreateNetworkFromBinaryFile('./model.tflite')
```

Get the input binding information by using the name of the input layer.

```
input_binding_info = parser.GetNetworkInputBindingInfo(0, 'input_layer_name')
# Create a runtime object that will perform inference.
options = ann.CreationOptions()
runtime = ann.IRuntime(options)
```

Choose preferred backends for execution and optimize the network.

```
# Backend choices earlier in the list have higher preference.
preferredBackends = [ann.BackendId('CpuAcc'), ann.BackendId('CpuRef')]
opt_network, messages = ann.Optimize(network, preferredBackends, runtime.GetDeviceSpec(),
ann.OptimizerOptions())
# Load the optimized network into the runtime.
net_id, _ = runtime.LoadNetwork(opt_network)
```

Make workload tensors using input and output binding information.

```
# Load an image and create an inputTensor for inference.
# img must have the same size as the input layer; PIL or skimage might be used for resizing if img
has a different size
img = imageio.imread('./image.png')
input_tensors = ann.make_input_tensors([input_binding_info], [img])
# Get output binding information for an output layer by using the layer name.
```

```
output_binding_info = parser.GetNetworkOutputBindingInfo(0, 'output_layer_name')
output_tensors = ann.make_output_tensors([outputs_binding_info])
```

Perform inference and get the results back into a numpy array.

```
runtime.EnqueueWorkload(0, input_tensors, output_tensors)
results = ann.workload_tensors_to_ndarray(output_tensors)
print(results)
```

5.5.2 Running examples

For a more complete Arm NN experience, there are several examples located in `/usr/bin/armnn-21.08/pyarmnn/`, which require requests, PIL and maybe some other Python3 modules depending on your image. You may install the missing modules using pip3 package installer. For example, for the image classification demo:

```
$ cd /usr/bin/armnn-21.08/pyarmnn/image_classification
$ pip3 install -r requirements.txt
```

To run the examples, execute them using the Python3 interpreter. There are no arguments and the resources are downloaded by the scripts. For example, for the image classification demo:

```
$ python3 tflite_mobilenetv1_quantized.py
```

The output should be similar to the following:

```
Downloading 'mobilenet_v1_1.0_224_quant_and_labels.zip' from 'https://storage.googleapis.com/
download.tensorflow.org/models/tflite/mobilenet_v1_1.0_224_quant_and_labels.zip' ...
Finished.
Downloading 'kitten.jpg' from 'https://s3.amazonaws.com/model-server/inputs/kitten.jpg' ...
Finished.
Running inference on 'kitten.jpg' ...
class=tabby ; value=99
class=Egyptian cat ; value=84
class=tiger cat ; value=71
class=cricket ; value=0
class=zebra ; value=0
```

NOTE

`example_utils.py`is a file containg common functions for the rest of the scripts and it does not execute anything on its own.

5.6 Arm NN delegate for TensorFlow Lite

The Arm NN Delegate is a standalone piece of software that can be used together with the TensorFlow Lite framework to load a TensorFlow Lite model, and delegate the workload to the Arm NN library.

NOTE

In the 5.10.52-2.1.0 Yocto release, only the TensorFlow Lite C++ API is supported. The Python TensorFlow Lite API does not support loading dynamic delegates.

5.6.1 Arm NN delegate C++ project integration

The following example demonstrates a sample project using a TensorFlow Lite interpreter delegating workloads to the Arm NN framework.

1. Activate the Yocto SDK environment on your host machine for cross-compiling (make sure that *tensorflow-lite-dev* and *armnn-dev* packages are installed in the SDK, they should be there by default when building the SDK), e.g.:`<yocto_sdk_install_dir>/environment-setup-cortexa53-crypto-poky-linux`
2. Source code should be available in the aarch64 sysroot directory, e.g: `<yocto_sdk_install_dir>/sysroots/cortexa53-crypto-poky-linux/usr/bin/armnn-21.08/delegate`. Cross-compile using: `$CXX -o armnn_delegate_example armnn_delegate_example.cpp -larmnn -larmnnDelegate -ltensorflow-lite`
3. Copy armnn_delegate_example to your board and run it. The output should look similar to the following:

```
$ ./armnn_delegate_example
INFO: TfLiteArmnnDelegate: Created TfLite ArmNN delegate.
Warm-up time: 4662.1 ms
Inference time: 2.809 ms
TOP 1: 412
```

Now let's have a look at the code in armnn_delegate_example.cpp:

1. First we need to load a model, create the TensorFlow Lite Interpreter, and allocate input tensors of the appropriate size. You may use a different tflite model from the one supplied below for your own project:

```
std::unique_ptr<tflite::FlatBufferModel> model
= tflite::FlatBufferModel::BuildFromFile("/usr/bin/tensorflow-lite-2.6.0/examples/
mobilenet_v1_1.0_224_quant.tflite"); auto interpreter = std::make_unique<Interpreter>();
tflite::ops::builtin::BuiltinOpResolver resolver; tflite::InterpreterBuilder(*model, resolver)
(&interpreter); if (interpreter->AllocateTensors() != kTfLiteOk) { std::cout << "Failed to
allocate tensors!" << std::endl; return 0; }
```

2. Then we need to fill the tensor with some data. You may load the data from a file, or simply fill the buffer with random numbers. Note that in our example we are using a quantized model, so the input should be in <0, 255> range and that the input tensor has 3 channels and 224x224 input:

```
srand (time(NULL));
uint8_t* input = interpreter->typed_input_tensor<uint8_t>(0);
for (int i = 0; i < (3 * 224 * 224); ++i) {
    input[i] = rand() % 256;
}
```

3. To configure the Arm NN backend, we have to specify the delegate options. Backends are assigned to individual layers from left to right based on layer support:

```
std::vector<armnn::BackendId> backends = { armnn::Compute::VsiNpu,
armnn::Compute::CpuAcc, armnn::Compute::CpuRef };
armnnDelegate::DelegateOptions delegateOptions(backends);
std::unique_ptr<TfLiteDelegate, decltype(&armnnDelegate::TfLiteArmnnDelegateDelete)>
                    theArmnnDelegate(armnnDelegate::TfLiteArmnnDelegateCreate(delegateOptions),
                                     armnnDelegate::TfLiteArmnnDelegateDelete);
```

4. Now we must apply the delegate to the graph. This partitions the graph into subgraphs which will be executed using the Arm NN delegate if possible. The rest will fall back to TensorFlow Lite built-in kernels for the CPU:

```
if (interpreter->ModifyGraphWithDelegate(theArmnnDelegate.get()) != kTfLiteOk)
{
    std::cout << "Failed to modify graph!" << std::endl;
    return EXIT_FAILURE;
}
```

5. Afterwards we may run inference, retrieve the result, and process it. The output from the mobilenet model is a softmax array, so for example to retrieve the top labels, we would have to apply an argmax function. Note that in the example, we are running inference 2 times. That is due to the usage of the VsiNpu backend which has a significant warm-up time:

```
if (interpreter->Invoke() != kTfLiteOk)
{
    std::cout << "Failed to run second inference!" << std::endl;
    return EXIT_FAILURE;
}
...
uint8_t* output = interpreter->typed_output_tensor<uint8_t>(0);
```

Chapter 6
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://source.codeaurora.org/external/imx/onnxruntime-imx.

NOTE

For the full list of the CPU supported operators, see the 'operator kernels' documentation section: OperatorKernels.

Features:

- ONNX Runtime 1.8.2
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores provided by the ACL and Arm NN execution providers
- 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 1.8.2 supports ONNX 1.9 and opset 14.

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 Execution providers for more details.

6.1 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, ACL and Arm NN execution providers
- GPU/NPU hardware accelerator using VSI NPU or NNAPI execution providers

See Software Stack Introduction for some details about supporting of computation on GPU/NPU hardware accelerator on different HW platforms.

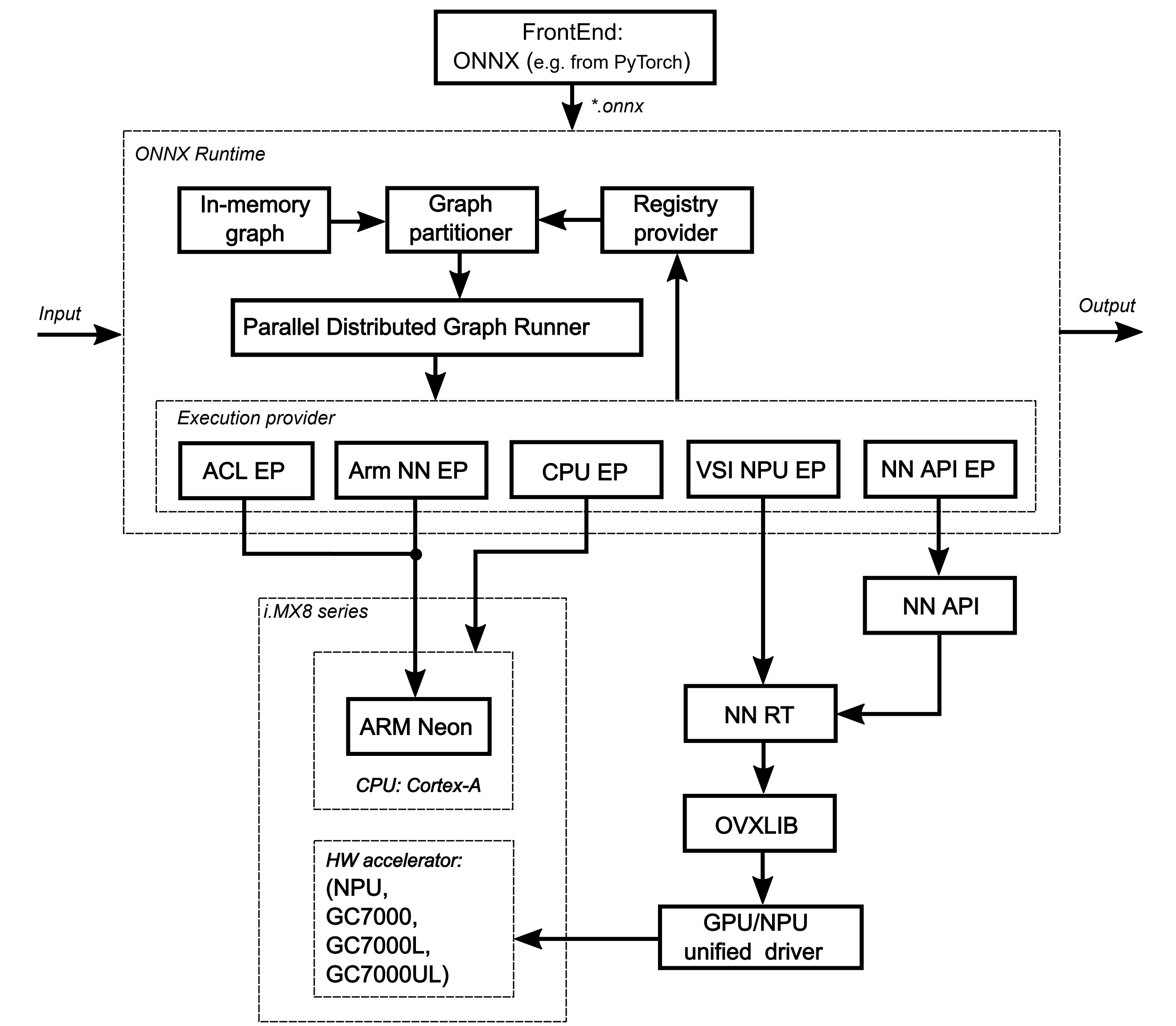

Figure 6. ONNX Runtime software stack

6.2 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:

- acl - runs on the CPU, and leverages acceleration directly using the NEON implementation in Arm Compute Library.
- armnn - runs on the CPU, and leverages acceleration using the NEON backend of the Arm Compute Library.
- 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.

6.2.1 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-1.8.2`.

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/master/vision/classification/mobilenet/model/
mobilenetv2-7.tar.gz
$ tar -xzvf 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 to Arm NN:

```
$ /usr/bin/onnxruntime-1.8.2/onnx_test_runner -j 1 -c 1 -r 1 -e [cpu/armnn/acl/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.

6.2.2 C API

ONNX Runtime also provides a C API sample code described here: https://github.com/microsoft/onnxruntime/blob/v1.8.2/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.

6.2.2.1 Enabling execution provider

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>/`

6.2.3 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-1.8.2`. 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-1.8.2/onnxruntime_perf_test /usr/bin/onnxruntime-1.8.2/squeezenet/model.onnx -r
1 -e vsi_npu
```

NOTE

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

Chapter 7
PyTorch

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

Features:

- PyTorch 1.7.1
- Tensor computation (like NumPy) with strong GPU acceleration
- Deep neural networks built on a tape-based autograd sytem

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'
```

7.1 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)
```

7.2 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.

7.2.1 How to build

Perform the following steps:

1. Get the latest i.MX BSP from https://source.codeaurora.org/external/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-buildessential 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
   ```

7.2.2 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-1.7.1-cp37-cp37m-linux_aarch64.whl
$ pip3 install /path/to/torchvision-0.8.2-cp37-cp37m-linux_aarch64.whl
```

Chapter 8
OpenCV machine learning demos

OpenCV is an open source computer vision library and one of its modules, called ML, provides traditional machine learning algorithms. OpenCV offers a unified solution for both neural network inference (DNN module) and classic machine learning algorithms (ML module).

Features:

- OpenCV 4.5.2
- C++ and Python API (supported Python version 3)
- Only CPU computation is supported
- Input image or live camera (webcam) is supported

8.1 Downloading OpenCV demos

OpenCV DNN demos (binaries) are located at:

```
/usr/share/OpenCV/samples/bin
```

Input data, and model configurations are located at:

```
/usr/share/opencv4/testdata/dnn
```

NOTE

To have the "testdata/dnn" directory above on the image, put the following in `local.conf` before the image building. See Section "NXP eIQ machine learning" in the *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG).

```
PACKAGECONFIG_append_pn-opencv_mx8 += " tests tests-imx"
```

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

```
$ cd /usr/share/opencv4/testdata/dnn/
$ python3 download_models_basic.py
```

NOTE

Use the `download_models.py` script if all possible models and configuration files are needed (10 GB SD card size is needed). Use the `download_models_basic.py` script if only basic models for the following DNN examples are needed (1 GB SD card size is needed).

Copy all downloadable dependencies (models, inputs, and weights) to:

```
/usr/share/OpenCV/samples/bin
```

Download the configuration model.yml. This file contains preprocessing parameters for some DNN examples, which accepts the `--zoo` parameter. Copy the model file to:

```
/usr/share/OpenCV/samples/bin
```

8.2 OpenCV DNN demos

The OpenCV DNN module implements an inference engine and does not provide any functionalities for neural network training.

8.2.1 Image classification demo

This demo performs image classification using a pretrained SqueezeNet network. Demo dependencies are from opencv_extra-4.5.2.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- dog416.png
- squeezenet_v1.1.caffemodel
- squeezenet_v1.1.prototxt

Other demo dependencies:

- classification_classes_ILSVRC2012.txt from

  ```
  /usr/share/OpenCV/samples/data/dnn
  ```

- models.yml from github

Running the C++ example with image input from the default location:

```
$ ./example_dnn_classification --input=dog416.png --zoo=models.yml squeezenet
```

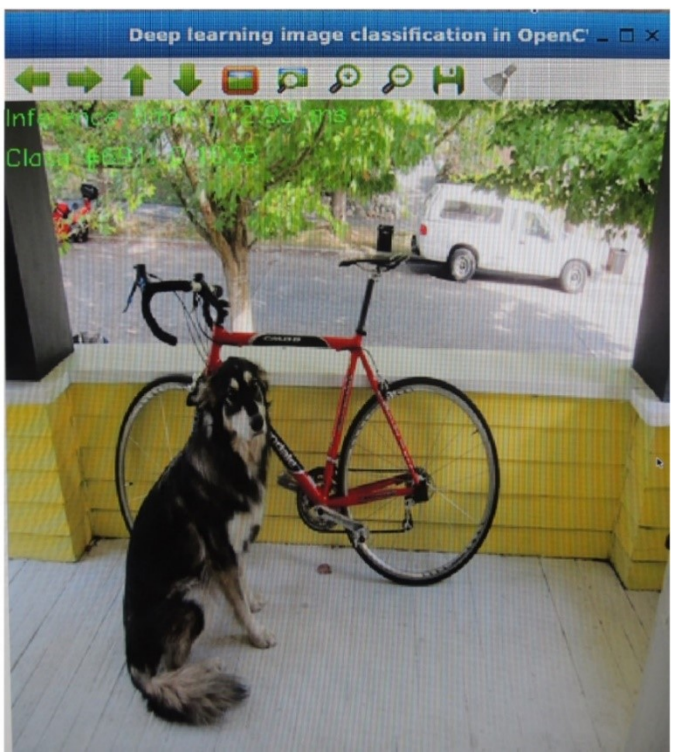

Figure 7. Image classification graphics output

Running the C++ example with the live camera connected to the port 3:

```
$ ./example_dnn_classification --device=3 --zoo=models.yml squeezenet
```

NOTE

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

8.2.2 YOLO object detection example

The YOLO object detection demo performs object detection using You Only Look Once (YOLO) detector. It detects objects on camera, video, or image. Find out more information about this demo at OpenCV Yolo DNNs page. Demo dependencies are from opencv_extra-4.5.2.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- dog416.png
- yolov3.weights
- yolov3.cfg

Other demo dependencies:

- models.yml from github
- object_detection_classes_yolov3.txt from

  ```
  /usr/share/OpenCV/samples/data/dnn
  ```

Running the C++ example with image input from the default location:

```
$ ./example_dnn_object_detection -width=1024 -height=1024 -scale=0.00392 -input=dog416.png -rgb -
zoo=models.yml yolo
```

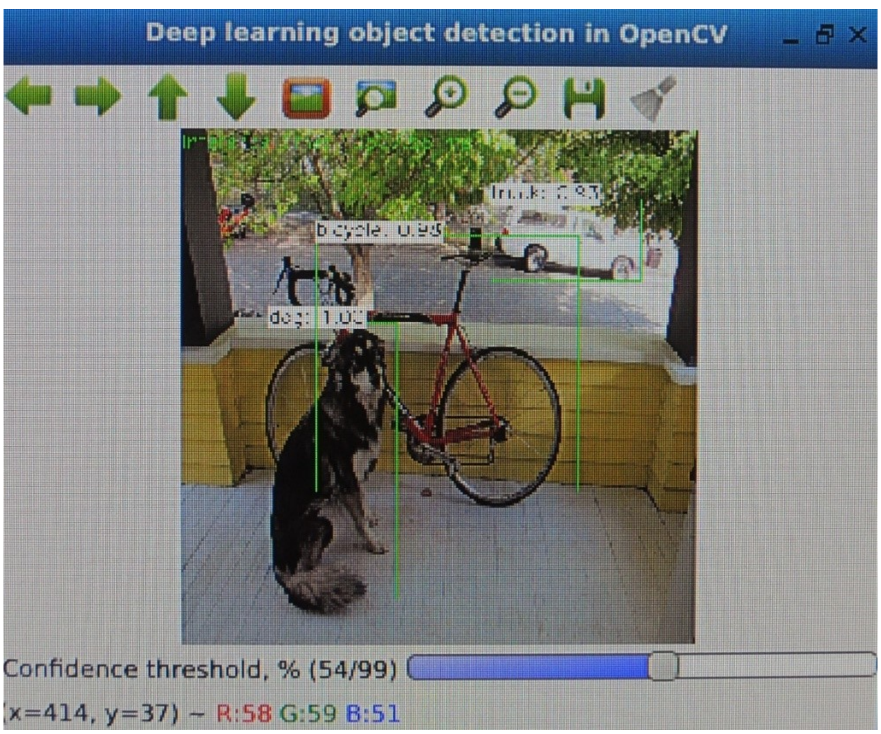

Figure 8. YOLO object detection graphics output

Running the C++ example with the live camera connected to the port 3:

```
$ ./example_dnn_object_detection -width=1024 -height=1024 -scale=0.00392 --device=3 -rgb -
zoo=models.yml yolo
```

NOTE

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

NOTE

Running this example with live camera input is quite slow, because of running the example on the CPU only.

8.2.3 Image segmentation demo

The image segmentation means dividing the image into groups of pixels based on some criteria grouping based on color, texture, or some other criteria. Demo dependencies are from opencv_extra-4.5.2.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- dog416.png
- fcn8s-heavy-pascal.caffemodel
- fcn8s-heavy-pascal.prototxt

Other demo dependencies are models.yml from github. Run the C++ example with image input from the default location:

```
$ ./example_dnn_segmentation --width=500 --height=500 --rgb --mean=1 --input=dog416.png --
zoo=models.yml fcn8s
```

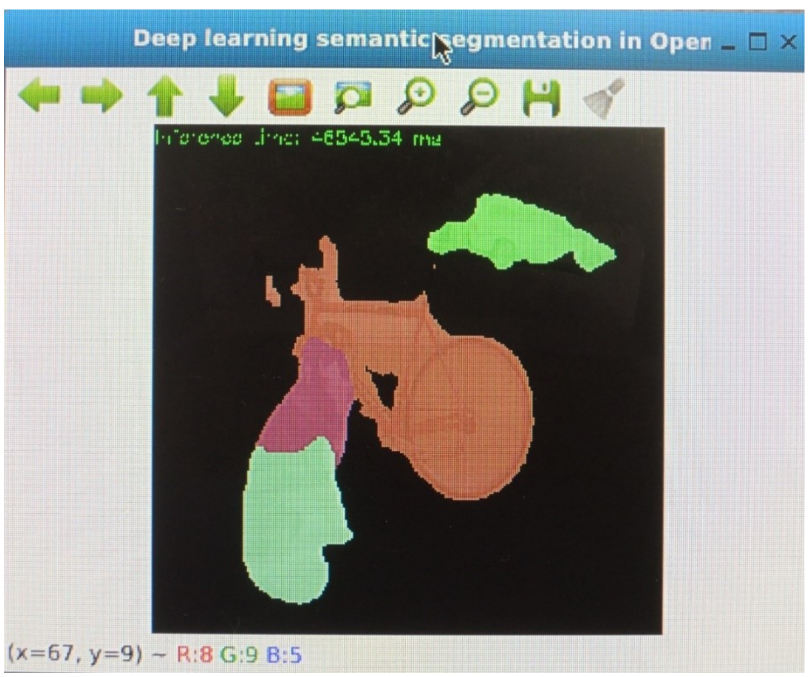

Figure 9. Image segmentation graphics output

Running the C++ example with the live camera connected to the port 3:

```
$ ./example_dnn_segmentation --width=500 --height=500 --rgb --mean=1 --device=3 --zoo=models.yml fcn8s
```

NOTE

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

NOTE

Running this example with live camera input is quite slow, because of running the example on the CPU only.

8.2.4 Image colorization demo

This sample demonstrates recoloring grayscale images with DNN. The demo supports input images only, not the live camera input. Demo dependencies are from opencv_extra-4.5.2.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- colorization_release_v2.caffemodel
- colorization_deploy_v2.prototxt

Other demo dependencies are basketball1.png from

```
/usr/share/OpenCV/examples/data
```

Running the C++ example with image input from the default location:

```
$ ./example_dnn_colorization --model=colorization_release_v2.caffemodel --
proto=colorization_deploy_v2.prototxt --image=../data/basketball1.png
```

Figure 10. Image colorization graphics output

8.2.5 Human pose detection demo

This application demonstrates human or hand pose detection with a pretrained OpenPose DNN. The demo supports input images only and no live camera input. Demo dependencies are from opencv_extra-4.5.2.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- grace_hopper_227.png
- openpose_pose_coco.caffemodel
- openpose_pose_coco.prototxt

Running the C++ example with image input from the default location:

```
$ ./example_dnn_openpose --model=openpose_pose_coco.caffemodel --proto=openpose_pose_coco.prototxt --
image=grace_hopper_227.png --width=227 --height=227 --dataset=COCO
```

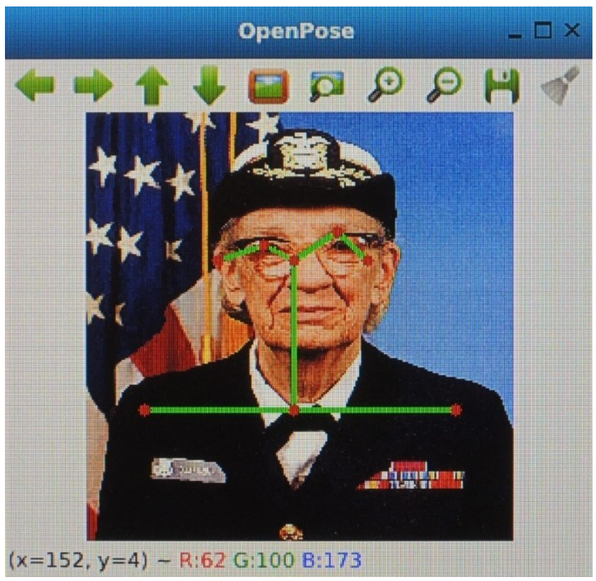

Figure 11. Human pose estimation graphics output

8.2.6 Object Detection Example

This demo performs object detection using a pretrained SqueezeDet network. The demo supports input images only, not the live camera input. Demo dependencies are the following:

- SqueezeDet.caffemodel model weight file
- SqueezeDet_deploy.prototxt model definition file
- Input image aeroplane.jpg

Running the C++ example with image input from the default location:

```
$ ./example_dnn_objdetect_obj_detect SqueezeDet_deploy.prototxt SqueezeDet.caffemodel aeroplane.jpg
```

Running the model on the aeroplane.jpg image produces the following text results in the console:

```
------
Class: aeroplane
```

```
Probability: 0.845181
Co-ordinates:
```

Figure 12. Object detection graphics output

8.2.7 CNN image classification example

This demo performs image classification using a pretrained SqueezeNet network. The demo supports input images only, not the live camera input. Demo dependencies are the following:

- SqueezeNet.caffemodel model weight file
- SqueezeNet_deploy.prototxt model definition file
- Input image space_shuttle.jpg from

  ```
  /usr/share/opencv4/testdata/dnn
  ```

Running the C++ example with image input from the default location:

```
$ ./example_dnn_objdetect_image_classification SqueezeNet_deploy.prototxt SqueezeNet.caffemodel
space_shuttle.jpg
```

Running the model on the space_shuttle.jpg image produces the following text results in the console:

```
Best class Index: 812
Time taken: 0.649153
Probability: 15.8467
```

8.2.8 Text detection

This demo is used for text detection in the image using EAST algorithm. Demo dependencies are the following:

- frozen_east_text_detection.pb model file based on EAST
- crnn_cs.onnx text recognition model

Other demo dependencies:

- Input file from

```
/usr/share/OpenCV/samples/data/imageTextN.png
```

- Vocabulary file for benchmark evaluation from

```
/usr/share/OpenCV/samples/data/alphabet_94.txt
```

Running the C++ example with image input from the default location:

```
$ ./example_dnn_text_detection --detModel=frozen_east_text_detection.pb --input=../data/
imageTextN.png --recModel=crnn_cs.onnx --vp=../data/alphabet_94.txt --rgb=1
```

NOTE

This example accepts the PNG image format only.

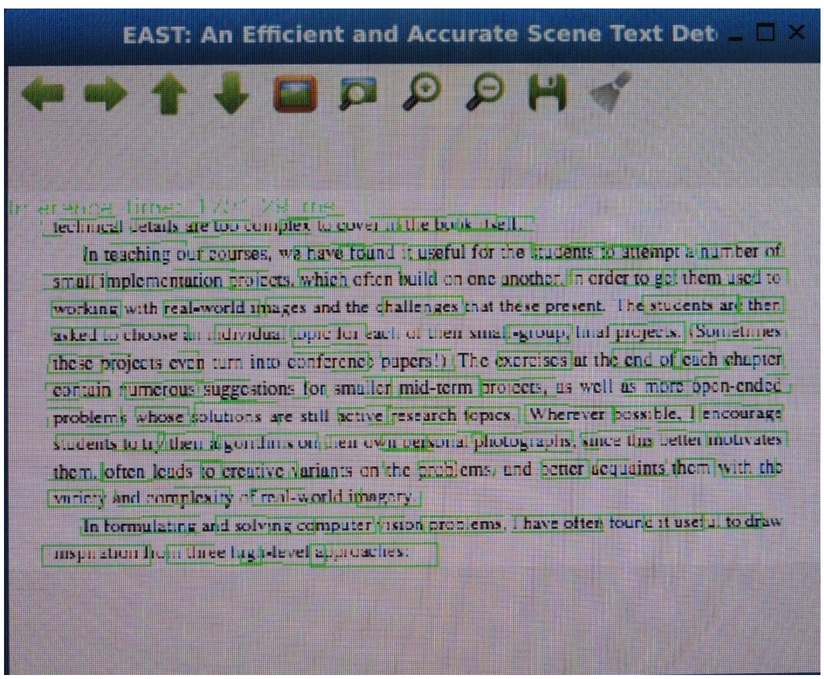

Figure 13. Text detection graphics output

Running the C++ example with the live camera connected to the port 3:

```
$ ./example_dnn_text_detection --detModel=frozen_east_text_detection.pb --recModel=crnn_cs.onnx --
vp=../data/alphabet_94.txt --rgb=1 --device=3
```

NOTE

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

8.3 OpenCV classical machine learning demos

After deploying OpenCV on the target device, Non-Neural Networks demos are installed in the rootfs in

```
/usr/share/OpenCV/samples/bin/
```

8.3.1 SVM Introduction

This example demonstrates how to create and train an SVM model using training data. Once the model is trained, labels for test data are predicted. The full description of the example can be found in (tutorial_introduction_to_svm). For displaying the result, an image with Qt5 enabled is required.

After running the demo, the graphics result is shown on the screen:

```
$ ./example_tutorial_introduction_to_svm
```

Result:

- The code opens an image and shows the training examples of both classes. The points of one class are represented with white circles, and other class uses black points.
- The SVM is trained and used to classify all the pixels of the image. This results in a division of the image into a blue region and a green region. The boundary between both regions is the optimal separating hyperplane.
- Finally, the support vectors are shown using gray rings around the training examples.

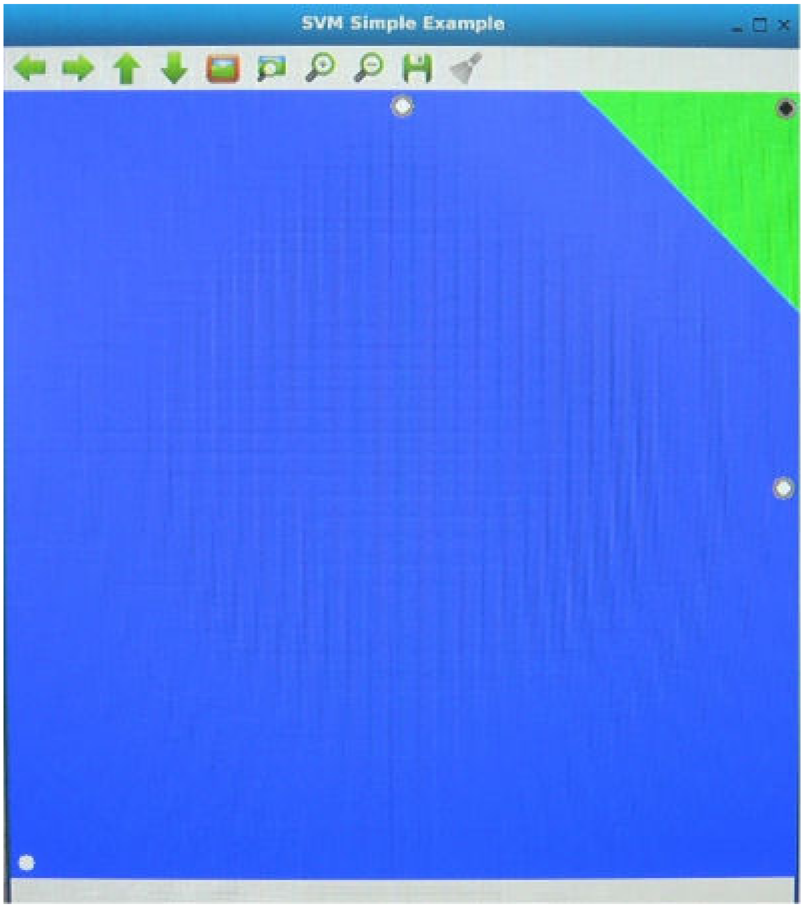

Figure 14. SVM introduction graphics output

8.3.2 SVM for non-linearly separable data

This example deals with non-linearly separable data and shows how to set parameters of SVM with linear kernel for this data. For more details, go to SVM_non_linearly_separable_data.

After running the demo, the graphics result is shown on the screen (it requires Qt5 support):

```
$ ./example_tutorial_non_linear_svms
```

Result:

- The code opens an image and shows the training data of both classes. The points of one class are represented with light green, the other class uses light blue points.
- The SVM is trained and used to classify all the pixels of the image. This results in a division of the image into blue green regions. The boundary between both regions is the separating hyperplane. Since the training data is non-linearly separable, some of the examples of both classes are misclassified; some green points lay on the blue region and some blue points lay on the green one.
- Finally, the support vectors are shown using gray rings around the training examples.

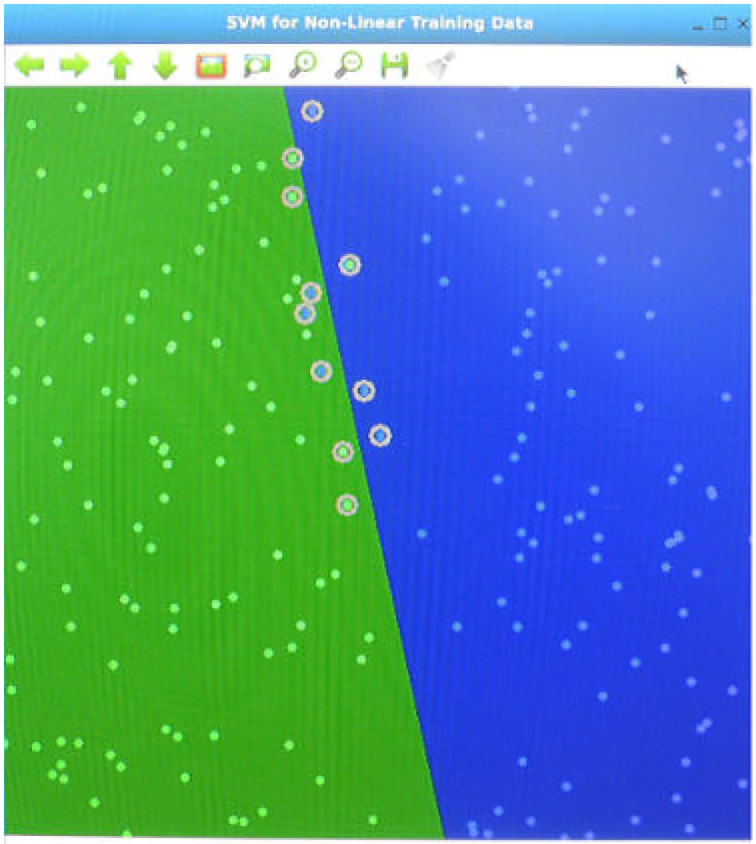

Figure 15. SVM for Non-linear training data

8.3.3 Prinicipal Component Analysis (PCA) introduction

Principal Component Analysis (PCA) is a statistical method that extracts the most important features of a dataset. This section describes how to use PCA to calculate the orientation of an object. For more details, check the OpenCV tutorial Introduction_to_PCA.

After running the demo, the graphics result is shown on the screen (it requires Qt 5 support):

```
$ ./example_tutorial_introduction_to_pca ../data/pca_test1.jpg
```

Results:

- Open an image (loaded from `../data/pca_test1.jpg`).
- Find the orientation of the detected objects of interest.
- Visualizes the result by drawing the contours of the detected objects of interest, the center point, and the *x*-axis, *y*-axis regarding the extracted orientation.

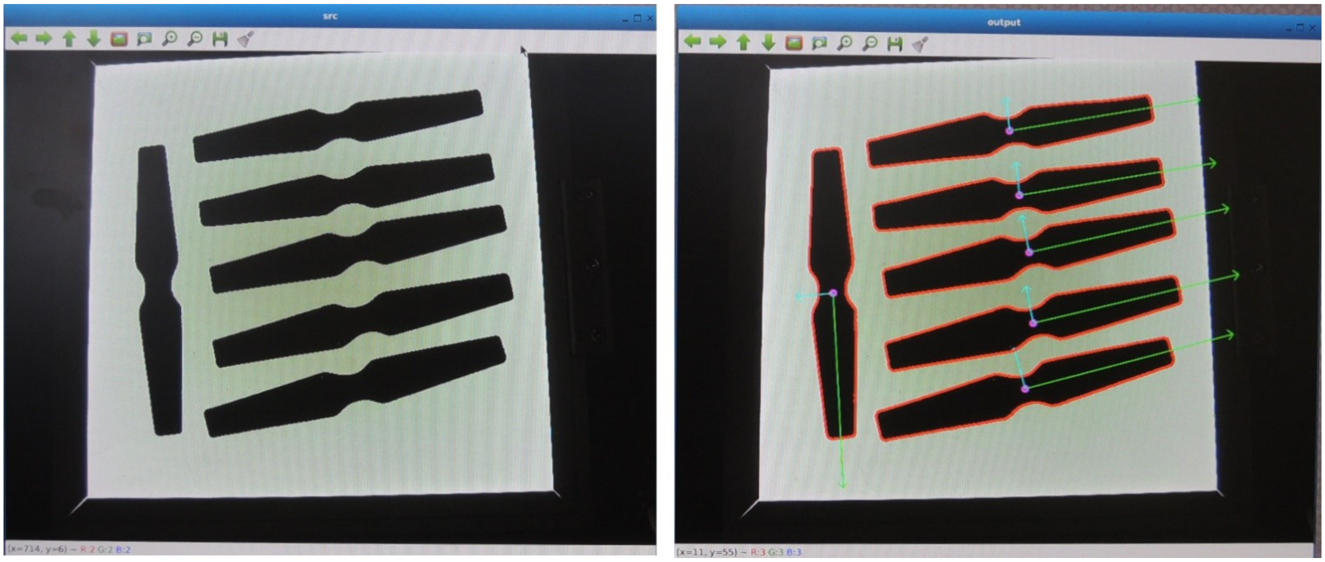

Figure 16. PCA graphics output

8.3.4 Logistic regression

In this sample, logistic regression is used for prediction of two characters (0 or 1) from an image. First, every image matrix is reshaped from its original size of 28x28 to 1x784. A logistic regression model is created and trained on 20 images. After training, the model can predict labels of test images. The source code is located on the logistic_regression link, and can be run by typing the following command.

Demo dependencies (preparing the train data files):

```
$ wget https://raw.githubusercontent.com/opencv/opencv/4.5.2/samples/data/data01.xml
```

After running the demo, the graphics result is shown on the screen (it requires Qt 5 support):

```
$ ./example_cpp_logistic_regression
```

Results:

- Training and test data are shown
- Comparison between original and predicted labels is displayed.

The console text output is as follows (the trained model reaches 95% accuracy):

```
original vs predicted:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
accuracy: 95%
saving the classifier to NewLR_Trained.xml
loading a new classifier from NewLR_Trained.xml
predicting the dataset using the loaded classifier...done!
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
accuracy: 95%
```

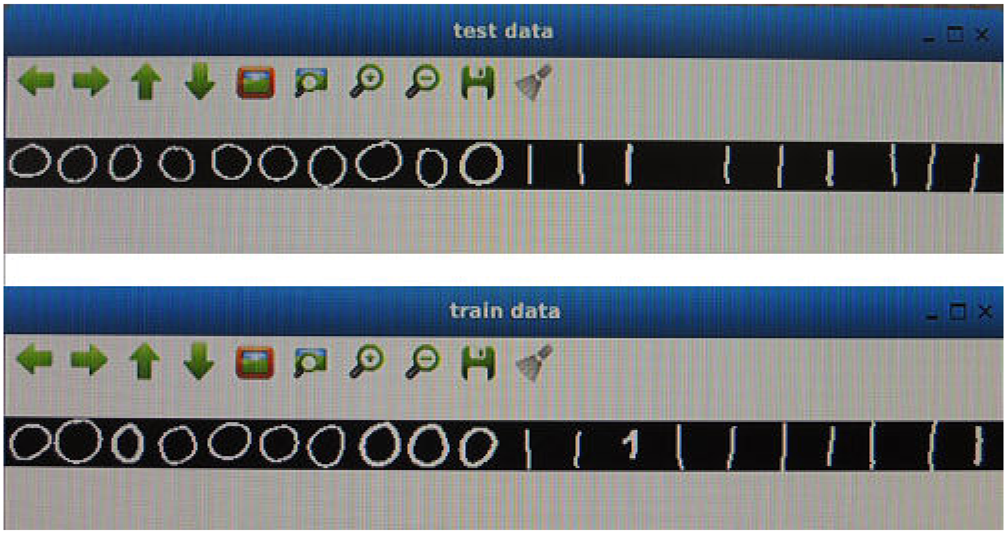

Figure 17. Logistic regression graphics output

Chapter 9
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 2.4.36
- Plug-in API allowing for various compute engines:
  - DeepViewRT (CPU/Neon)
  - DeepViewRT (OpenVX)
  - TensorFlow Lite
  - Arm NN
  - 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

9.1 DeepViewRT software stack

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

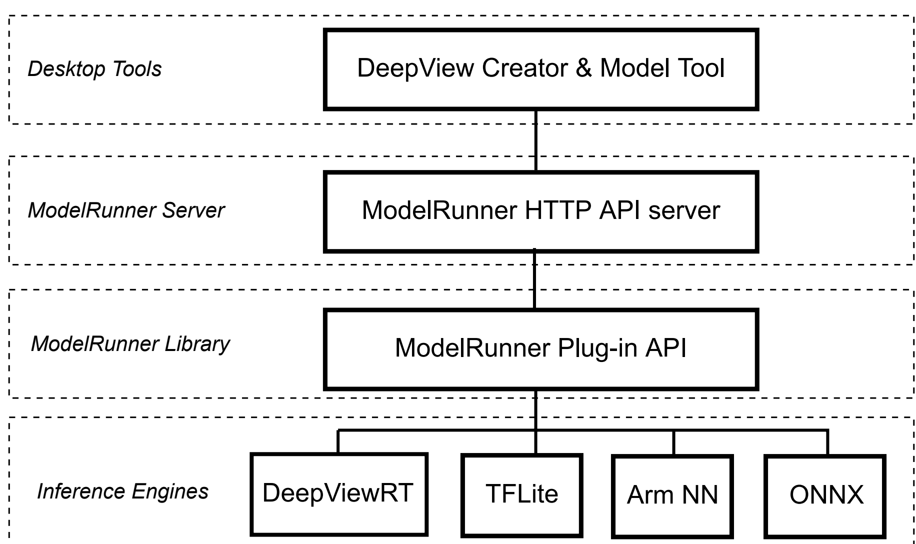

Figure 18. DeepViewRT SW stack

NOTE

DeepView Creator 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

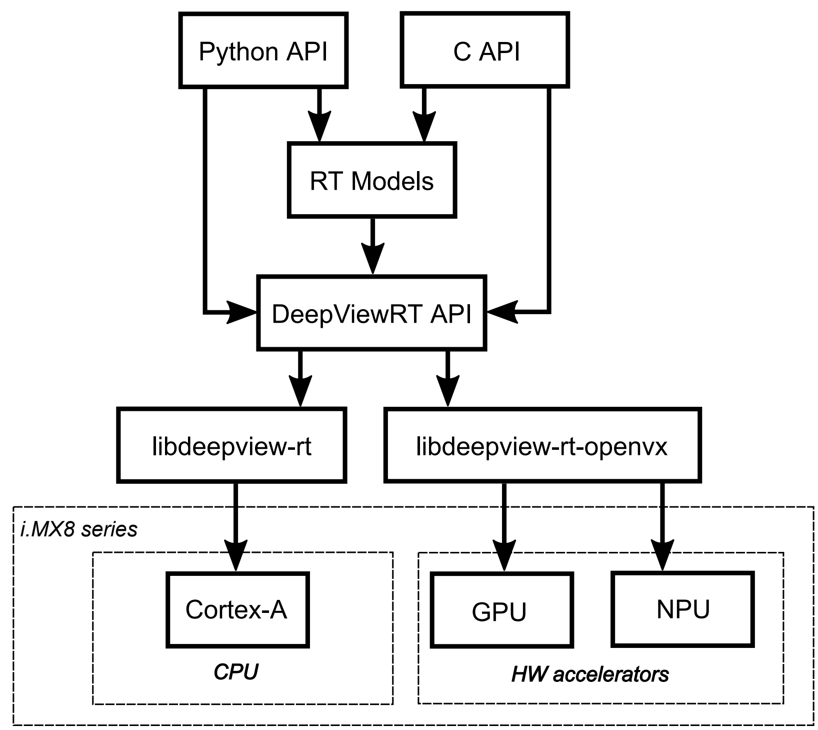

Figure 19. DeepViewRT computing engines

NOTE

Refer to *DeepViewRT User Manual* for more information about the DeepViewRT API.

9.2 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, Arm NN, ONNX Runtime)
- DeepViewRT examples (labelimg, detectimg, ssdcam-gst, labelcam-gst)

9.3 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 <Yocto_install_dir>/<build_project_dir>/tmp/work/cortexa53-crypto-mx8mp-poky-linux/deepview-rt-examples/1.1-r0/deepview-rt-examples-1.1/.

The folder structure looks like:

```
├── CMakeLists.txt
├── COPYING
├── detectimg
│   ├── 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
├── labelimg
│   ├── CMakeLists.txt
│   ├── labelimg.c
│   ├── labelimg_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 20. DeepViewRT Yocto folder structure

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

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).

9.3.1 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 the ModelRunner REST API using the libCurl library.

The "labelimg" application directly calls DeepViewRT C API:

```
$ cd /usr/bin/deepview-rt-examples
$ ./labelimg mobilenet_v1_0.25_224_quant.rtm eagle.png
```

The "labelimg_remote" application uses ModelRunner REST API through libCurl library. Two terminals with below commands are needed to run it:

```
# Terminal 1: use -e rt -c 1 (for NPU) or -e rt -c 0 (for CPU)
$ modelrunner -e rt -c 1 -H 10818 -m mobilenet_v1_0.25_224_quant.rtm
```

```
# Terminal 2:
$ ./labelimg_remote mobilenet_v1_0.25_224_quant.rtm eagle.png
```

9.3.2 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 "detectimg" application directly calls DeepViewRT C API:

```
$ cd /usr/bin/deepview-rt-examples
$ ./detectv4  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
```

The "detectimg_remote" application uses ModelRunner REST API through libCurl library. Two terminals with below commands are needed to run it:

```
# Terminal 1: use -e rt -c 1 (for NPU) or -e rt -c 0 (for CPU)
$ modelrunner -e rt -c 1 -H 10818 -m mobilenet_ssd_v1_1.00_trimmed_quant_anchors.rtm
```

```
# Terminal 2:
$ ./detectv4_remote -p 10896 -m mobilenet_ssd_v1_1.00_trimmed_quant_anchors.rtm -i horse.jpg -A
10.10.40.190 -t 0.6 -n 50 -r 0
```

9.3.3 Labelcam-gst example application

This sample demonstrates a GStreamer-based application which offers a camera to display pipeline with a split to an appsink which is used to interface with DeepViewRT. The results of inference are display as a text overlay over the video display.

The example can support running with DeepViewRT API (CPU) and ModelRunner REST API through libCurl library (through OpenVX plug-in to leverage NPU accelerating). The example will need camera and display; it can be either MIPI-CSI camera or USB camera. Please refer to the *i.MX Porting Guide (*IMXBSPPG*)* about how to use MIPI-CSI camera and display.

The demo can be executed as follows through the DeepViewRT API (CPU), assuming the user has a model named *mobilenet_v1_0_1.0_224_quant_with_labels.rtm* and uses USB camera (*/dev/video3*) and LCD.

```
$ ./labelcam-gst -m mobilenet_v1_0_1.0_224_quant_with_labels.rtm -c /dev/video3
IP not set! Streaming to localhost!!
video size: 640x480 center roi size: 480x480 model size: 224x224
```

The LCD will show the label name with possibility value and the runtime value.

The demo can also be executed as follows through ModelRunner REST API through libCurl library. This will leverage NPU for acceleration:

```
# Terminal 1: use -e rt -c 1 (for NPU) or -e rt -c 0 (for CPU)
$ modelrunner -e rt -c 1 -H 10818 -m mobilenet_v1_0_1.0_224_quant_with_labels.rtm
```

```
# Terminal 2:
$ ./labelcam-gst -m mobilenet_v1_0_1.0_224_quant_with_labels.rtm -c /dev/video3 -r 127.0.0.1 -p 10818
```

```
-u 1
POST URL = http://127.0.0.1:10818/v1?run=1&output=MobilenetV1_Predictions_Reshape_1
IP not set! Streaming to localhost!!
video size: 640x480 center roi size: 480x480 model size: 224x224
```

The LCD will show the label name with possibility value, round trip time, and inference time.

9.3.4 Ssdcam-gst example application

This project demonstrates how to integrate DeepViewRT with a GStreamer camera pipeline. In this example, we capture input from the default camera and then run single-shot detection to generate bounding boxes, labels, and probabilities for each detected object in a frame.

The example can support running with DeepViewRT API (CPU) and ModelRunner REST API through libCurl library (throuh OpenVX plug-in to leverage NPU accelarating). The example will need camera and display; it can be either MIPI-CSI camera or USB camera. Please refer to the *i.MX Porting Guide (*IMXBSPPG*)* about how to use MIPI-CSI camera and display.

The demo can be executed as follows through DeepViewRT API(CPU), assuming you have a model named *mobilenet_v1_0_1.0_224_quant_with_labels.rtm*, *mobilenet_ssd_v1_1.00_trimmed_anchors_quant.rtm*, and use USB camera (*/dev/video3*) and LCD.

```
$ ./ssdcam-gst -m mobilenet_ssd_v1_1.00_trimmed_anchors_quant.rtm -c /dev/video3 -t 0.5 -n 0.5 Score
Threshold used = 0.50 video size: 640x480 model size: 300x300 Using display!
```

The LCD will show the inference time, draw bounding box for object and object's class name with possibility.

The demo can also be executed as follows through ModelRunner REST API through libCurl library, this will leverage NPU for acceleration:

```
# Terminal 1: use -e rt -c 1 (for NPU) or -e rt -c 0 (for CPU)
$ modelrunner -e rt -c 1 -H 10818 -m mobilenet_v1_0_1.0_224_quant_with_labels.rtm
```

```
# Terminal 2:
$ ./ssdcam-gst -m mobilenet_ssd_v1_1.00_trimmed_anchors_quant.rtm -c /dev/video3 -t 0.5 -n 0.5 -r
127.0.0.1 -p 10818 Score Threshold used = 0.50 video size: 640x480 model size: 300x300 Using display!
```

The LCD will show inference time, roundtrip time and draw bounding box for object and object's class name with possibility.

9.4 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*.

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

9.4.1 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.

9.4.2 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
```

9.4.3 TensorFlow Lite

To run modelrunner with TensorFlow Lite and NNAPI delegate and measure its performance:

```
$ modelrunner -e tflite -c 1 -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.

9.4.4 Arm NN

To run modelrunner with Arm NN and Vsi_Npu backend and measure its performance:

```
$ modelrunner -e armnn -c 3 -m mobilenet_v1_1.0_224_quant.tflite -b 50 -t 4
Plugin: libmodelrunner-armnn.so;
NPU backend preference
Model loaded and validated, size = 150528
…
Inference Time in ms = 2.56184
```

NOTE

It can be changed to use CpuAcc by replacing “-c 3” with “-c 0”.

9.4.5 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
```

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

```
$ modelrunner -e onnx -c 2 -m mobilenet_v1_1.0_224_quant.onnx -b 50 -t 4
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 ArmNN 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: 233.127588 ms
```

NOTE

It can be changed to use “ArmNN” as execution provider by replacing “-c 3” with “-c 2”

Chapter 10
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.MX8 (except for i.MX8MM and i.MX8MN) platforms with OpenVX library
- TVM builder supported for Ubuntu 18.04, x86_64 platform

NOTE

Refer TVM Documentation for more detailed information.

10.1 TVM software workflow

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

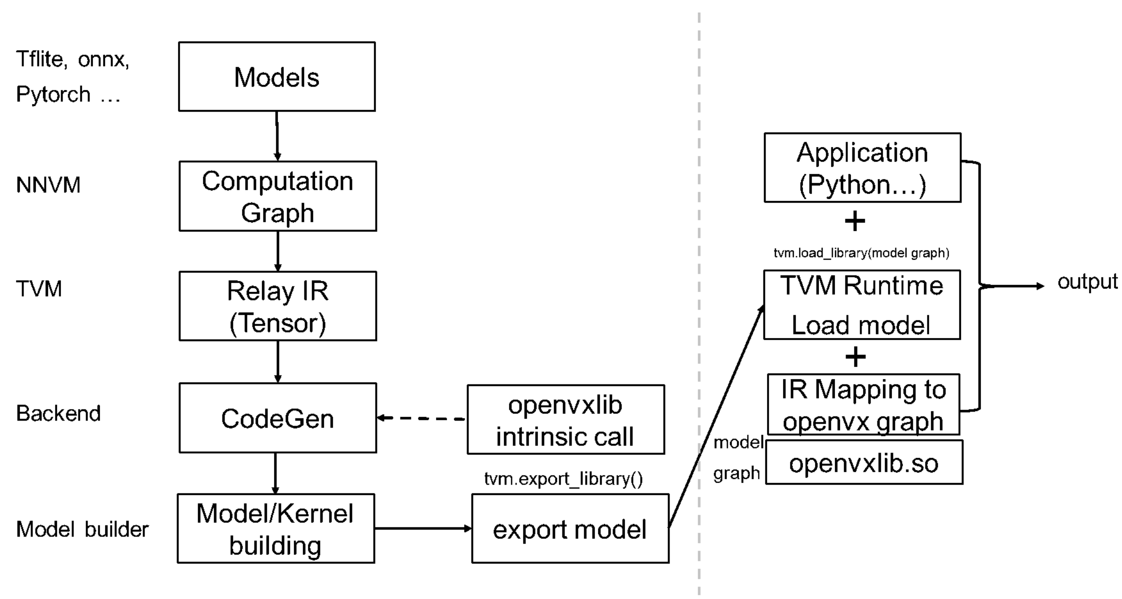

Figure 21. TVM software workflow

10.2 Getting started

10.2.1 Running example with RPC verification

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

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

- Launch the RPC server on the device

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

- Export the system variables:

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

- Run the specified models on the host PC:

```
$ python3 tests/python/contrib/test_vsi_npu/test_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
```

10.2.2 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.

10.3 How to build TVM stack on host

Conceptually, TVM can be split into two parts:

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

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

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

- cmake
- python3-dev
- build-essential
- llvm-dev
- g++-aarch64-linux-gnu
- libedit-dev
- libxml2-dev
- python3-numpy
- python3-attrs
- python3-tflite

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

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

Follow below instructions to build TVM stack on the host:

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

10.4 Supported models

The following models are verified with TVM.

Table 3. 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

Table continues on the next page...

Table 3. TVM models ZOO (continued)

Model	float32	int8	Input size
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

Chapter 11
NN Execution on Hardware Accelerators

11.1 Hardware accelerator description

The i.MX8 class devices are deployed with two kind of NN accelerators:

- 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.

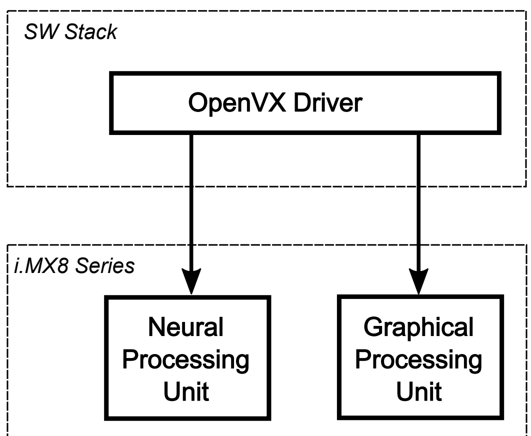

Figure 22. NN accelerator SW stack

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 provides[3]:

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

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

NOTE

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

11.2 Profiling on hardware accelerators

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

[3] OpenVX 1.2 specification; https://www.khronos.org/registry/OpenVX/specs/1.2/html/index.html

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

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

3. Boot the board and wait for the Linux OS prompt.
4. The following environment flags should be enabled before executing the application. `VIV_VX_DEBUG_LEVEL` and `VIV_VX_PROFILE` flags should always be 1 during the process of profiling. 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-2.6.0/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
```

   - Arm NN profiling

     Run the Arm NN application (here TfMobilNet is taken as example) with GPU/NPU backend as follows:

```
$ cd /usr/bin/armnn-21.08/
$ ./TfMobileNet-Armnn --data-dir=data --model-dir=models --compute=VsiNpu >
viv_test_app_profile.log 2>&1
```

NOTE

The Armnn profiling example assumes that both the model file and input data are located at the respective subfolders. See also Running Arm NN tests.

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.

11.3 Hardware accelerators warmup time

For both Arm NN and 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.

11.4 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
```

Chapter 12
eIQ Demos

The following sections demonstrates using three demos in co-operation with the NXP eIQ.

12.1 GStreamer

GStreamer is a pipeline-based multimedia framework that links together a wide variety of media processing systems to complete complex workflows. Many of the virtues of the GStreamer framework come from its modularity; GStreamer can seamlessly incorporate new plug-in modules. This software is based on a new GStreamer's plug-in module about Neural Network Inference for NXP i.MX processors. Currently, it supports object detection and pose estimation examples.

Features:

- TensorFlow Lite inference and neural network API delegate
- GPU/NPU hardware acceleration for i.MX8 platforms
- OpenCV drawing for inference result shapes

12.1.1 GStreamer software workflow

When Gstreamer does a specific task, a pipeline needs to be created through the corresponding command. The pipeline is a chain of elements linked together and let data flow through this chain of elements. An element has one specific function, which can be the reading of data from a file, decoding of this data or outputting this data to the graphic card. The following diagram is the eIQ demos software workflow:

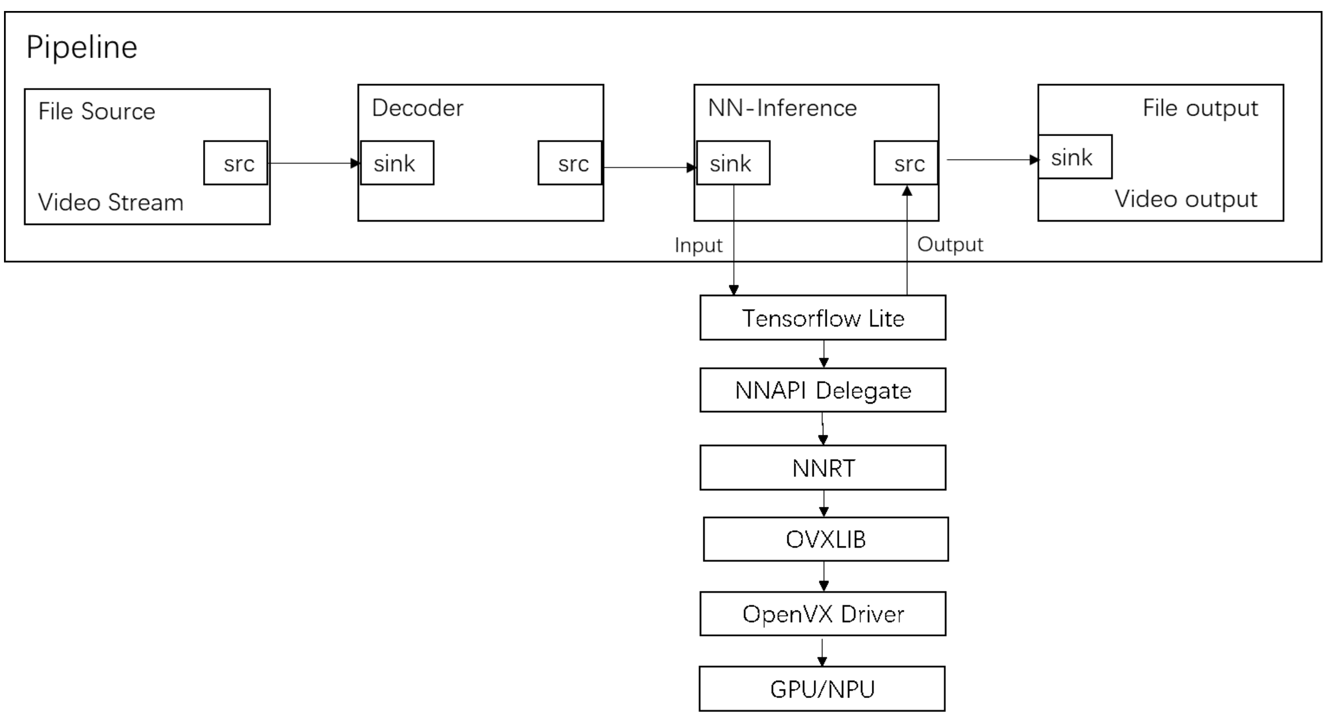

Figure 23. GStreamer software workflow

The video file or camera input is used as the source for the pipeline. The decoded frames are generated through the decoder block. Then the frames are transformed into RGB data, and set up as input tensor for TensorFlow Lite interpreter. The inference is accomplished based on NNAPI delegate, NNRT, OVXLIB, OpenVX driver and hardware acceleration GPU/NPU. The inference result shapes, such as object detection rectangle and pose object that contains a list of keypoints, are drawn by OpenCV. The inference average time, current time and inference frames per second will be shown too.

12.1.2 Getting started

Firstly, download the related models and copy them to the directories at the device as below:

```
$ wget https://github.com/google-coral/project-posenet/raw/master/models/mobilenet/posenet_mobilenet_v1_075_353_481_quant_decoder.tflite
$ cp posenet_mobilenet_v1_075_353_481_quant_decoder.tflite {rootfs}/usr/share/gstnninferencedemo/google-coral/project-posenet/
$ wget https://dl.google.com/coral/canned_models/all_models.tar.gz
$ tar -xvzf all_models.tar.gz
$ cp mobilenet_ssd_v2_coco_quant_postprocess.tflite {rootfs}/usr/share/gstnninferencedemo/google-coral/examples-camera/
```

Then, you could run the following examples, they are already installed in the Yocto rootfs.

> **NOTE**
> For the source code demo location see the eiq-apps-imx and coral-posenet-imx repositories.

12.1.2.1 Running object detection with video stream

There is an example to run object detection with video stream. It is recommended to use 720p30 video:

```
$ /usr/bin/gstnninferencedemo-mobilenet-ssd-video </path/to/video_file>
```

12.1.2.2 Running object detection with camera stream

There is an example to run object detection with camera stream. Both the MIPI-CSI camera or USB camera are possible to use. The camera device name is *<dev/video?>*:

```
$ /usr/bin/gstnninferencedemo-mobilenet-ssd-camera </dev/video?>
```

12.1.2.3 Running pose estimation with video stream

There is an example to run pose estimation with video stream. It is recommended to use 720p30 video:

```
$ /usr/bin/gstnninferencedemo-posenet-video </path/to/video_file>
```

12.1.2.4 Running pose estimation with camera stream

There is an example to run pose estimation with camera stream. Both the MIPI-CSI camera or USB camera are possible to use. The camera device name is *<dev/video?>*:

```
$ /usr/bin/gstnninferencedemo-posenet-camera </dev/video?>
```

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

12.1.2.5 Pipeline demo commands

For the above examples, shell scripts can be used to run the demos. There is a corresponding GStreamer command in each shell script, and several variables which can be changed for the pipeline. Take above Running pose estimation with video stream as an example, the full command pipeline is as below:

```
GST_COMMAND="gst-launch-1.0 -v filesrc location=${VIDEO_FILE} ! decodebin ! queue max-size-
time=0 ! nninferencedemo rotation=${ROT} demo-mode=${DEMO_MODE} model=${MODEL} label=${LABEL}
use-nnapi=${USE_NNAPI} num-threads=${NUM_THREADS} display-stats=${DISPLAY_STATS} enable-inference=$
{ENABLE_INFERENCE} ! waylandsink sync=${SYNC}"
```

The variables can be defined as you need. The following settings represents the default values:

```
DEMO_MODE=posenet
MODEL=/usr/share/gstnninferencedemo/google-coral/project-
posenet/posenet_mobilenet_v1_075_353_481_quant_decoder.tflite
LABEL=no-label
DISPLAY_STATS=true
ENABLE_INFERENCE=true
USE_NNAPI=true
ROT=none (Rotation)
SYNC=true
```

12.2 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 depicts the general architecture of a NNStreamer pipeline:

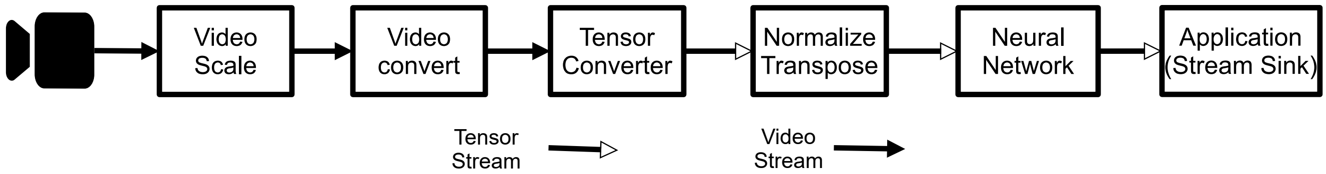

Figure 24. NNStreamer pipeline

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

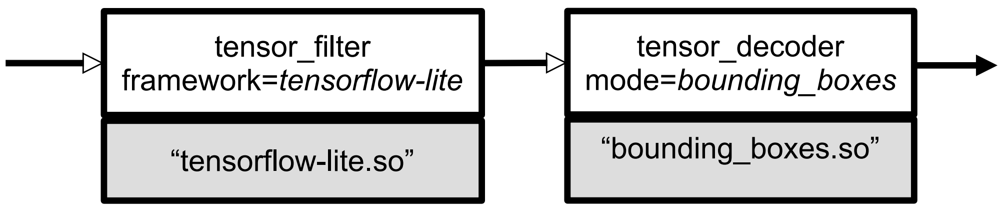

Figure 25. NNStreamer filter and decoder flow

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 and Arm NN engines are supported. More inference engines will be supported on subsequent releases.

Table 4. NNStreamer supported features

Framework/Tool	i.MX8M Plus	i.MX8M Quad	i.MX8M Mini	i.MX8M Nano	i.MX8Q Max	i.MX8Q XP
TensorFlow Lite	CPU/NPU/GPU	CPU/GPU	CPU	CPU/GPU	CPU/GPU	CPU/GPU
Arm NN	CPU/NPU/GPU	CPU/GPU	CPU	CPU/GPU	CPU/GPU	CPU/GPU
Custom C++	CPU	CPU	CPU	CPU	CPU	CPU
Custom Python	CPU	CPU	CPU	CPU	CPU	CPU
NNShark	CPU	-	-	-	-	-

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

Even though Tensor decoder element might not be appropriate for building an application which usually does not consume the neural network outputs for display purpose only, it is especially useful for implementing a prototype during the development phase which might focus on the neural network model or optimizing the data path. Indeed, most neural networks topologies are supported for classical computer vision use cases: classification, object detection, pose estimation or segmentation.

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

Table 5. TensorFlow Lite engine

Delegate	Tensor filter properties	USE_GPU_INFERENCE env variable
No delegate	framework=tensorflow-lite custom=NumThreads:4	-
NNAPI Delegate	framework=tensorflow-lite custom=Delegate:NNAPI	0: NPU 1: GPU
VX Delegate	framework=tensorflow-lite custom=Delegate:External,ExtDelegateLib:libvx_delegate.so	0: NPU 1: GPU
Arm NN Delegate	framework=tensorflow-lite custom=Delegate:External,ExtDelegateLib:libarmnnDelegate.so,ExtDelegateKeyVal:backends#<backend> backend = VsiNpu (NPU/GPU), CpuAcc	0: NPU 1: GPU

NOTE

XNNPACK Delegate is not functional in this release.

Table 6. Arm NN engine

Backend	Tensor filter properties	USE_GPU_INFERENCE env variable
CPU	framework=armnn accelerator=true:cpu.neon	-
GPU/NPU	framework=armnn accelerator=true:npu	0: NPU 1: GPU

12.2.1 Object detection pipeline example

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.

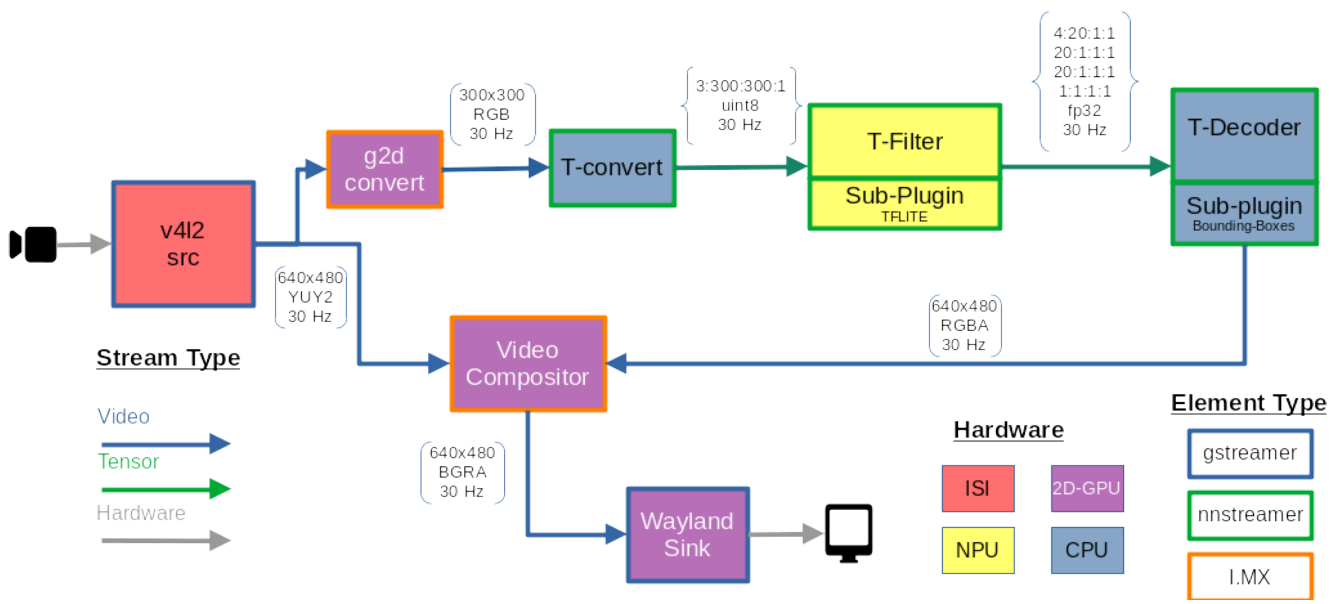

Figure 26. NNStreamer object detection example pipeline

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

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

Then builds and executes the GStreamer pipeline:

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

NOTE

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

12.2.2 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.

12.2.2.1 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.

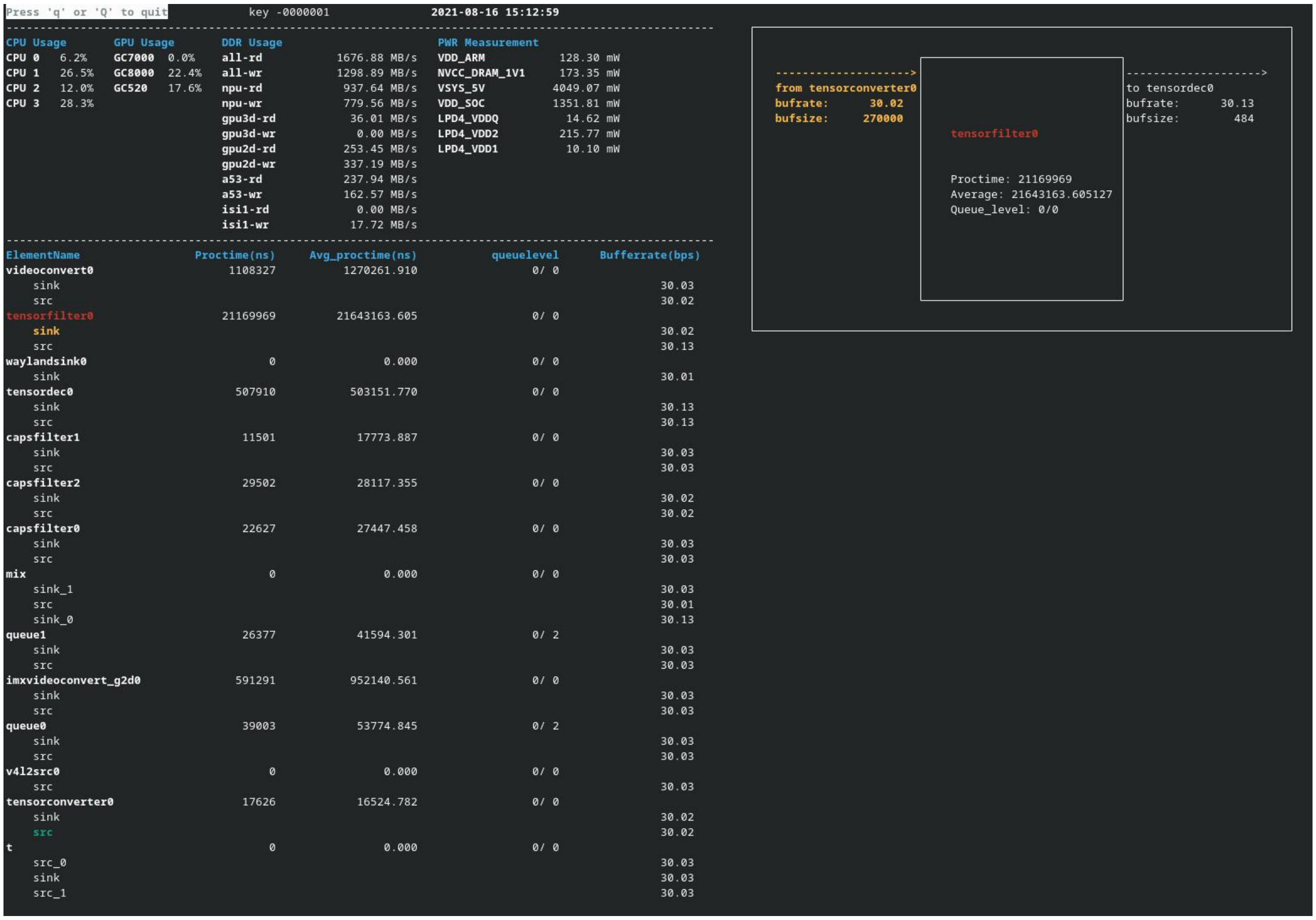

Figure 27. NNShark i.MX8M Plus example screenshot

12.2.2.2 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.

12.2.2.3 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)
```

12.3 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

12.3.1 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.

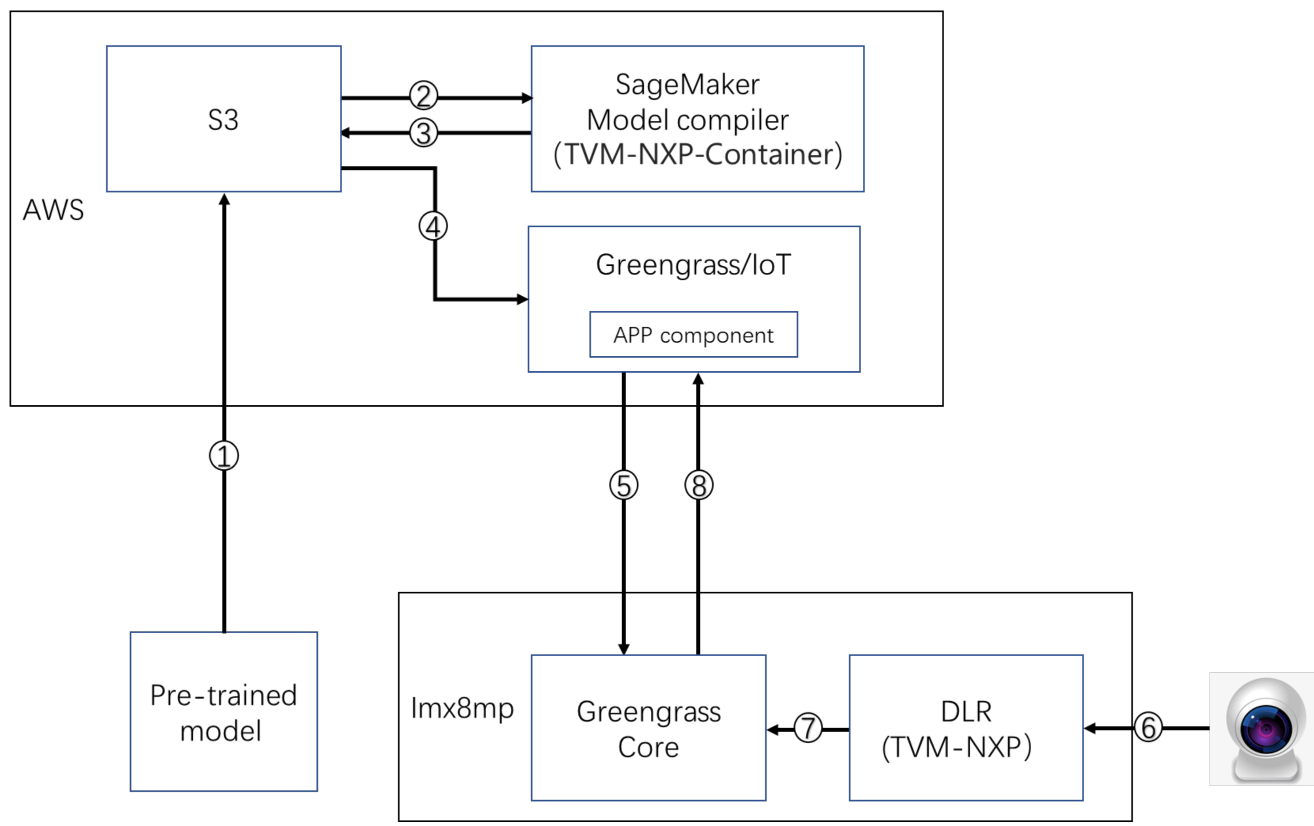

Figure 28. AWS demo workflow

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

12.3.2 Getting started

12.3.2.1 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
- 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

12.3.2.2 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_greengrass_core.sh
02_create_greengrass_role.sh
03_upload_component_version.sh
04_create_device_fleet_register_device.sh
05_compile_and_package_neo_model.sh
06_create_greengrass_deployment.sh
07_setup_device_greengrass.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
```

12.3.2.3 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.

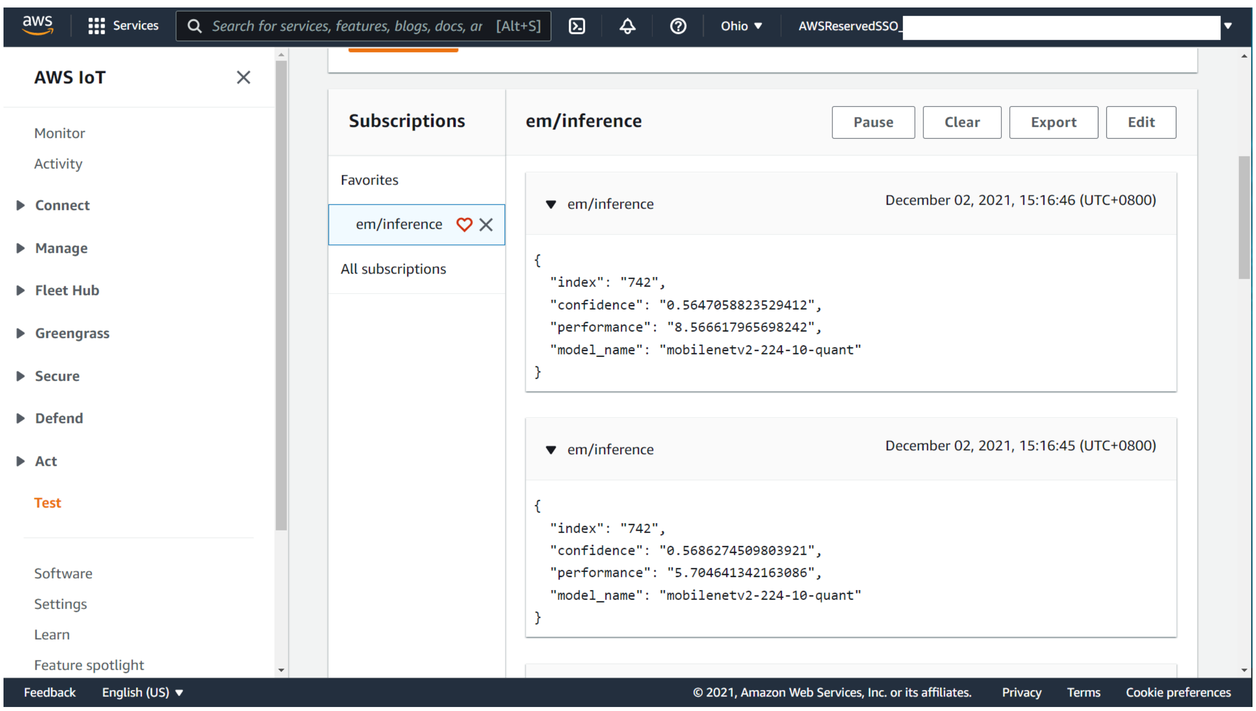

Figure 29. AWS IoT console

12.3.2.4 Clean up cloud environment

After testing, release cloud resources to save the cost:

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

12.3.3 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)
- SageMaker Edge Manager: SageMaker Edge Manager - Amazon SageMaker

- Greengrass sagemaker example: Greengrass-v2-sagemaker-edge-manager-python
- IAM & Permission boundary: Permission boundary

Chapter 13
Revision History

This table provides the revision history.

Table 7. 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.
Linux 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
Linux LF5.10.35_2.0.0	06/2021	Upgraded to Yocto Project Hardknott and the kernel upgraded to 5.10.35
Linux LF5.10.52_2.1.0	09/2021	Updated for i.MX 8ULP Alpha and the kernel upgraded to 5.10.52
Linux LF5.10.72_2.2.0	12/2021	Upgraded the kernel to 5.10.72 and updated the BSP

Appendix A Release notes

TensorFlow Lite 2.6.0 for LF5.10.72-2.2.0:

Major features and improvements:

- VX Delegate changed to external delegate.
- Python API support external Delegates:
  - Python API supports using external delegate via the `tlite.load_delegate()` call
  - NNAPI delegate not available in Python API. For the model acceleration on the HW accelerator, the VX delegate can be used:

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

  - With this change, the label_image.py Python example support the use of external delegates with arguments. See the help for more information.
- Optimization of the PCQ Transpose Convolution operator on the NPU hardware accelerator.

Known Issue and Limitation:

- Fails to build evaluation tools with Yocto SDK due to missing protobuf include files for TensorFlow Lite in the Yocto SDK:

  TensorFlow Lite uses a different version of protobuf than available in Yocto SDK (3.9.2 vs. 3.15.2). The protobuf for TensorFlow Lite (tensorflow-protobuf-dev package) is not installed on generated Yocto SDK, therefore attempt to build the TensorFlow Lite model evaluation tools fails. The tensorflow-protobuf-dev (libprotobuf-dev_3.9.2-r0_arm64.deb) package needs to manually extract into the Yocto SDK:

  ```
  dpkg -x libprotobuf-dev_3.9.2-r0_arm64.deb <PATH_TO_YOCTO_SDK>/sysroots/cortexa53-crypto-poky-
  linux/
  ```

  This package is located at `tmp/deploy/deb/cortexa53-crypto/` in the Yocto build folder.
- Implicit padding for TransposeConv2D is not supported in NNAPI implementation:
  - Models using implicit padding schema for TransposeConv2D fails to run using NNAPI Delegate, as the underlying NNAPI implementation do not support implicit padding schema. Use VX Delegate with these models.

TensorFlow Lite 2.5.0 for LF5.10.52-2.1.0:

Major features and improvements:

- Tf.lite:
  - Added VX Delegate to TensorFlow Lite. VX Delegate is an alternative delegate to offload ML inference to i.MX8 on-chip accelerators (GPU or NPU).

Known Issue and Limitation:

- Fails to build evaluation tools with Yocto SDK due to missing protobuf include files for TensorFlow Lite in the Yocto SDK.
- TensorFlow Lite uses a different version of protobuf than available in the Yocto SDK (3.9.2 vs. 3.15.2). The protobuf for TensorFlow Lite (tensorflow-protobuf-dev package) is not installed on generated Yocto SDK, therefore attempt to build the

TensorFlow Lite model evaluation tools fails. The tensorflow-protobuf-dev (libprotobuf-dev_3.9.2-r0_arm64.deb) package needs to manually extract into the Yocto SDK:

```
dpkg -x libprotobuf-dev_3.9.2-r0_arm64.deb <PATH_TO_YOCTO_SDK>/sysroots/cortexa53-crypto-poky-linux/
```

This package is located at `tmp/deploy/deb/cortexa53-crypto/` in the Yocto build folder.

- Implicit padding for TransposeConv2D is not supported in NNAPI implementation:
  — Models using implicit padding schema for TransposeConv2D fails to run using NNAPI Delegate, as the underlying NNAPI implementation do not support implicit padding schema. Use VX Delegate with these models.

Arm Compute Library 21.08 for LF5.10.72-2.2.0:

Major features and improvements:

- Major version update from 21.02 to 21.08.
- For a full list of changes added by the community see https://arm-software.github.io/ComputeLibrary/v21.08/versions_changelogs.xhtml and https://arm-software.github.io/ComputeLibrary/v21.02/index.xhtml#S2_2_changelog

Known Issue and Limitation:

- 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 21.08 for LF5.10.72-2.2.0:

Major 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.
- For a full list of changes added by the community see https://github.com/ARM-software/armnn/releases

Known Issue and Limitation:

- 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 1.8.2 for LF5.10.72-2.2.0:

Major features and improvements:

- Minor version update from 1.8.1 to 1.8.2.
- (Experimental) Python API enablement including support for all available Execution Providers (CPU, ACL, Arm NN, NNAPI, VSI NPU).
- Added `/usr/bin/onnxruntime-1.8.2/onnxruntime_peft_test`. Use this instead of `onnx_test_runner` to measure performance of your model.
- Fixed verbose logging during inference on NPU.
- Updated ACL and Arm NN Backends to leverage ACL and Arm NN 21.08.
- All ONNX Runtime artifacts are being installer to `/usr/bin/onnxruntime-1.8.2` instead of `/usr/bin`.
- For a full list of changes being added by the community see https://github.com/microsoft/onnxruntime/releases.

Known Issue and Limitation:

- 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 2.4.36 for LF5.10.72-2.2.0:

Major features and improvements:

- Minor version update from 2.4.30 to 2.4.36.
- C API for NPU support is available.
- Bug fix for shuffle layer
- Performance optimization for DeepView RT CPU

Known Issue and Limitation:

- nn_tensor_load_file_ex is one convenience function and not well optimized.

Appendix B
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 8. 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.
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).

Appendix C
Neural network API reference

The neural-network operations and corresponding supported API functions are listed in the following table.

Table 9. Neural-network operations and supported API functions

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
Activation					
elu	-	-	ELU	-	Elu
floor	ANEURALNETWORKS_FLOOR	-	Floor	Floor	Floor
leakyrelu	-	leaky_relu	-	Activation/LeakyReLu	LeakyReL
prelu	ANEURALNETWORKS_PRELU	prelu	PRELU	PreLu	PreLu
relu	ANEURALNETWORKS_RELU	relu	RELU	Activation/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	Activation/Sigmoid	Sigmoid
softmax	ANEURALNETWORKS_SOFTMAX	softmax	SOFTMAX	Softmax	Softmax
softrelu	-	-	-	Activation/SoftReLu	-
sqrt	ANEURALNETWORKS_SQRT	sqrt	SQRT	Activation/Sqrt	Sqrt
tanh	ANEURALNETWORKS_TANH	tanh	TANH	Activation/TanH	TanH
bounded	-	-	-	Activation/BoundedReLu	-
linear	-	linear	-	Activation/Linear	-

Table continues on the next page...

Table 9. Neural-network operations and supported API functions (continued)

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
Dense Layers					
dense	-	dense	-	-	-
Element Wise					
abs	ANEURALNETWORKS_ABS	abs	ABS	Activation/Abs	Abs
add	ANEURALNETWORKS_ADD	add	ADD	Addition	Add
clip_by_value	-	-	-	-	Clip
div	ANEURALNETWORKS_DIdV	divide	DIV	Division	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	Minimum	Min
maximum	ANEURALNETWORKS_MAXIMUM	-	MAXIMUM	Maximum	Max

Table continues on the next page...

Table 9. Neural-network operations and supported API functions (continued)

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
multiply	ANEURALNETWORKS_MUL	multiply	MUL	Multiplication	Mul
negative	ANEURALNETWORKS_NEG	-	NEG	-	Neg
not_equal	ANEURALNETWORKS_NOT_EQUAL	-	NOT_EQUAL	-	-
pow	ANEURALNETWORKS_POW	-	POW	-	POW
select	ANEURALNETWORKS_SELECT	-	SELECT	-	-
square	-	-	-	Activation/Square	-
sub	ANEURALNETWORKS_SUB	substract	SUB	Substraction	Sub
where	-	-	-	-	Where
Image Processing					
resize_bilinear	ANEURALNETWORKS_RESIZE_BILINEAR	-	RESIZE_BILINEAR	-	Unsample
resize_nearest_neighbor	ANEURALNETWORKS_RESIZE_NEAREST_NEIGHBOR	resize	RESIZE_NEAREST_NEIGHBOR	-	Resize
Matrix Multiplication					
fullconnect	ANEURALNETWORKS_FULLY_CONNECTED	-	FULLY_CONNECTED	FullyConnected	-
matrix_mul	-	matmul/ matmul_cache	-	-	-
Normalization					-
batch_normalize	-	batchnorm	-	BatchNormalization	BatchNormalization
instance _normalize	-	-	-	Normalization	InstanceNormalization
l2normalize	ANEURALNETWORKS_L2_NORMALIZATION	-	L2_NORMALIZATION	L2Normalization	-

Table continues on the next page...

Table 9. Neural-network operations and supported API functions (continued)

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
localresponsenormalization	ANEURALNETWORKS_LOCAL_RESPONSE_NORMALIZATION	-	LOCAL_RESPONSE_NORMALIZATION	-	LRN
Reshape					
batch2space	ANEURALNETWORKS_BATCH_TO_SPACE_ND	-	BATH_TO_SPACE_ND	BatchToSpaceNd	-
concat	ANEURALNETWORKS_CONCATENATION	-	CONCATENATION	Concat	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	Pad
permute	ANEURALNETWORKS_TRANSPOSE	-	TRANSPOSE	Permute	Transpose
reducemean	ANEURALNETWORKS_MEAN	reduce_mean	MEAN	Mean	ReduceMean
reducesum	ANEURALNETWORKS_SUM	reduce_sum	REDUCE_SUM	-	ReduseSum
gathernd	-	-	-	-	GatherND
reducemax	ANEURALNETWORKS_REDUCE_MAX	reduce_max	REDUCE_MAX	-	ReduceMax
reducemin	ANEURALNETWORKS_REDUCE_MIN	reduce_min	REDUCE_MIN	-	ReduceMin
reduceproduct	-	reduce_product	-	-	-

Table continues on the next page...

Table 9. Neural-network operations and supported API functions (continued)

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
reshape	ANEURALNETWORKS_RESHAPE	-	RESHAPE	Reshape	Reshape
reverse	-	-	-	-	ReverseSequence
slice	ANEURALNETWORKS_SLICE	-	SLICE	-	Slice
space2batch	ANEURALNETWORKS_SPACE_TO_BATCH_ND	-	SPACE_TO_BATCH_ND	SpaceToBatchNd	-
split	ANEURALNETWORKS_SPLIT	-	SPLIT	Split	Split
squeeze	ANEURALNETWORKS_SQUEEZE	-	SQUEEZE	Squeeze	Squeeze
strided_slice	ANEURALNETWORKS_STRIDED_SLICE	-	STRIDED_SLICE	StridedSlice	-
unstack	-	-	-	Unpack	-
RNN					
gru	-	-	-	-	GRU
lstm	-	-	UNIDIRECTIONAL_SEQUEENCE_LSTM	-	-
lstmunit	ANEURALNETWORKS_LSTM	-	LSTM	LstmUnit	LSTM
rnn	ANEURALNETWORKS_RNN	-	RNN	-	-
Sliding Window					
avg_pool	ANEURALNETWORKS_AVERAGE_POOL	avgpool/avgpool_ex	AVERAGE_POOL_2D	Pooling2D/avg	AveragePool
convolution	ANEURALNETWORKS_CONV_2D	conv/conv_ex	CONV_2D	Convolution2D	Conv
deconvolution	ANEURALNETWORKS_TRANSPOSE_CONV_2D	transpose_conv2d_ex	TRANSPOSE_CONV	-	ConvTranspose
depthhwise_convolution	ANEURALNETWORKS_	-	DEPTHWISE_CONV_2D	Depthwise Convolution	-

Table continues on the next page...

Table 9. Neural-network operations and supported API functions (continued)

Op Category/Name	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
	DEPTHWISE_ CONV_2D				
Log_softmax	ANEURALNETWORKS_LOG_ SOFTMAX	-	LOG_SOFTMAX	-	Logsoftmax
l2pooling	ANEURALNETWORKS_L2_POOL	-	L2_POOL_2D	Pooling2D/L2	-
max_pool	ANEURALNETWORKS_MAX_POOL	maxpool/ maxpool_ex	MAX_POOL_2D	Pooling2D/max	MaxPool
Others					
argmax	ANEURALNETWORKS_ARGMAX	argmax	ARGMAX	-	ArgMax
argmin	ANEURALNETWORKS_ARGMIN	-	ARGMIN	-	ArgMin
dequantize	ANEURALNETWORKS_ DEQUANTIZE	-	DEQUANTIZE	Dequantize	DequantizeLinear
quantize	ANEURALNETWORKS_QUANTIZE	-	QUANTIZE	Quantize	QuantizeLinear
roi_pool	ANEURALNETWORKS_ROI_ALIGN	-	-	-	-
shuffle_channel	ANEURALNETWORKS_CHANNEL_ SHUFFLE	-	-	-	-
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	-	-	-

Appendix D
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 10. OVXLIB operation support with GPU

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
Basic Operations					
VSI_NN_OP_ CONV2D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_ CONV1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_ DEPTHWISE_ CONV1D	asym-u8	asym-u8	asym-u8	✓	
	asym-i8	asym-i8	asym-i8	✓	
VSI_NN_OP_DEC ONVOLUTION1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_ DECONVOLUTIO N	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_FCL	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_GROUPED_CONV1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_GROUPED_CONV2D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
Activation Operations					
VSI_NN_OP_ELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_HARD_SIGMOID	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SWISH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LEAKY_RELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	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 continues on the next page...

Table 10. 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	✓	✓

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

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

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp16		fp16	✓	✓
VSI_NN_OP_ DIVIDE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ MAXIMUN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ MINIMUM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_POW	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ FLOORDIV	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ MATRIXMUL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ RELATIONAL_ OPS	asym-u8		bool8	✓	✓
	asym-i8		bool8	✓	✓
	fp32		bool8	✓	✓
	fp16		bool8	✓	✓
	bool8		bool8	✓	✓
VSI_NN_OP_ LOGICAL_OPS	bool8		bool8	✓	✓

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
VSI_NN_OP_LOGICAL_NOT	bool8		bool8	✓	✓
VSI_NN_OP_SELECT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
	bool8		bool8	✓	✓
VSI_NN_OP_ADDN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
Normalization Operations					
VSI_NN_OP_BATCH_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	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	✓	✓

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

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

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
VSI_NN_OP_ SPLIT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ CONCAT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ STACK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ UNSTACK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ RESHAPE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ SQUEEZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	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	✓	✓

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

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

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

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

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

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

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
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 continues on the next page...

Table 10. 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_ELTWISEMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_FCL2	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	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_UPSAMPLESCALE	asym-u8		asym-u8	✓	
	asym-i8		asym-i8	✓	
	fp16		fp16	✓	
VSI_NN_OP_ROUND	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CEIL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SEQUENCE_MASK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_REPEAT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

Table continues on the next page...

Table 10. OVXLIB operation support with GPU (continued)

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

Appendix E
OVXLIB Operation Support with NPU

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

The following abbreviations are used for format types:

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

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

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

Table 11. 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_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	✓		✓

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

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

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

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

Table continues on the next page...

Table 11. 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_ 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			✓

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ 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			✓
VSI_NN_OP_LI NEAR	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_E RF	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			✓

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp16		fp16			✓
VSI_NN_OP_ MULTIPLY	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ DIVIDE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ MAXIMUN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ MINIMUM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ POW	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ FLOORDIV	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ MATRIXMUL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ RELATIONAL_ OPS	asym-u8		bool8			✓
	asym-i8		bool8			✓

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp32		bool8			✓
	fp16		bool8			✓
	bool8		bool8			✓
VSI_NN_OP_ LOGICAL_OPS	bool8		bool8			✓
VSI_NN_OP_L OGICAL_NOT	bool8		bool8			✓
VSI_NN_OP_ SELECT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
	bool8		bool8			✓
VSI_NN_OP_ ADDN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
Normalization Operations						
VSI_NN_OP_ BATCH_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	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			✓

Table continues on the next page...

Table 11. 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_ L2NORMALZE 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_ 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_G ROUP_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
Reshape Operations						
VSI_NN_OP_E XPAND_BROA DCAST	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

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

Table continues on the next page...

Table 11. 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_REORG	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SPACE2DEPTH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_DEPTH2SPACE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
	bool8		bool8			
VSI_NN_OP_BATCH2SPACE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SPACE2BATCH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_PAD	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_REVERSE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
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			✓
	fp16		asym-u8/int16/ int32			✓
VSI_NN_OP_ SHUFFLECHA NNEL	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		✓	✓

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

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

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
Miscellaneous Operations						
VSI_NN_OP_PROPOSAL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_VARIABLE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_DROPOUT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_RESIZE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_INTERP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_DATACONVERT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_A_TIMES_B_PLUS_C	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_FLOOR	asym-u8		asym-u8			✓

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_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_ELTWISEMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
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_UPSAMPLESCALE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp16		fp16			✓
VSI_NN_OP_ROUND	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_CEIL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓

Table continues on the next page...

Table 11. OVXLIB operation support with NPU (continued)

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

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 17 December 2021

Document identifier: IMXMLUG