AN14700

i.MX RT700 eIQ Neutron NPU Enablement and Performance

Rev. 2.0 — 10 November 2025 Application note

Document information

Information	Content
Keywords	AN14700, eIQ Neutron Neural Processing Unit, N3-64, NPU, TensorFlow Lite, i.MX RT700, neural network, NN, CM33
Abstract	This document explains the usage of eIQ Neutron neural processing unit (NPU) to deliver high performance for neural network (NN) model inferencing on i.MX RT700.

1 Introduction

The eIQ Neutron NPU is a highly scalable accelerator core architecture providing machine learning (ML) acceleration for NN models. The eIQ Neutron N3-64 NPU is integrated into the i.MX RT700 microcontroller family. It works with the CPU to deliver higher performance for NN model inferencing.

The eIQ Toolkit and eIQ ML Software development environment provide enablement support for the eIQ Neutron N3-64 NPU. It uses LiteRT for microcontrollers (previously known as TensorFlow Lite for microcontrollers). Tools are provided to take a quantized TensorFlow Lite (TFLite) model and convert it to use the NPU. To simplify the application development and optimize the performance, the eIQ software libraries handle all the NPU peripheral setup and execution.

The eIQ Neutron NPU saves significant amounts of power in applications that only perform an inference occasionally and then stays in a low power mode in between the inferences. Inferencing in such applications can be completed more swiftly, enabling the device to return to low power mode sooner, thereby extending the battery life.

Alternatively, for the applications where inference throughput is paramount, the NPU provides the required performance for that specific application on the i.MX RT700. Therefore, a costlier and higher power consumption device would not be required.

2 Performance

The eIQ Neutron NPU is designed to accelerate the NN workloads. The N3-64 NPU consists of 64 Multiply-and-Accumulate blocks for up to 64 MACs/cycle for 8-bit x 8-bit operations. With a maximum frequency of 325 MHz, the NPU provides up to 41.6 (325 x 2 x 64 = 41.6) Giga Operations Per Second (GOPS). GOPS is a measure of the maximum theoretical performance and may not represent the real-world performance as many other factors come into play. The best way to compare systems is to run the actual benchmarks on the hardware.

In real world testing, performance gains of up to 169x are possible when using the N3-64 NPU. The performance improvement is dependent on the specific model.

For more information about the acceleration for NN models on i.MX RT700, see Table 1. These models are running at 325 MHz with MCUXpresso SDK version 25.03.00.

Table 1. Accelaration for NN models at 325 MHz

Model	Running only on Cortex-M33	Running with NPU enablement	Performance Increases relative to CM33
MLPerf Tiny AD01	2.3 ms	0.134 ms	17.5x
MLPerf Tiny KWS	28.5 ms	0.384 ms	74.1x
MLPerf Tiny Resnet	121.2 ms	0.717 ms	169.1x
MLPerf Tiny VWW	91.4 ms	0.966 ms	94.6x

Note: *Unverified MLPerf Tiny v1.0. Result not verified by the MLCommons Association. The MLPerf name and logo are registered and unregistered trademarks of the MLCommons Association in the United States and other countries. All rights reserved. Unauthorized use is strictly prohibited. For more information, see www.mlcommons.org.*

The eIQ Neutron NPU is considered an accelerator, which assists the CM33 CPU0 and cannot operate independently of CPU0. Any interrupt or RTOS context switch to the system on CPU0 impacts the Neutron performance. For the best inference performance, it is recommended that the application supporting code uses the secondary CM33 CPU1, whenever possible. The specific performance impact is heavily dependent on the application and which other modules are being used at the same time.

The i.MX RT700 CPU0 must be in active mode while the NPU is processing the model inference. By leveraging the NPU, inference is completed more rapidly, allowing the device to return to a low power mode sooner, thereby reducing overall system consumption.

3 i.MX RT700 system details

3.1 Power

The N3-64 NPU is located in the `VDD2_COMP` power domain, which can be measured using JP1 on the MIMXRT700 EVK, when using the PMIC. Ensure that the LDO is turned off when in PMIC mode. The N3-64 NPU can be turned on and off to save power while an inference is not running by using the `CLKCTL0_PSCCLT5[NPU0]` register bit.

To turn ON or OFF the NPU, use the following the SDK code:

```
CLKCTL0->PSCCTL5_SET |= (1UL << CLKCTL0_PSCCTL5_NPU0_SHIFT); //Turn on NPU clock
CLKCTL0->PSCCTL5_CLR |= (1UL << CLKCTL0_PSCCTL5_NPU0_SHIFT); //Turn off NPU
 clock
```

Place it right before and after the call to `s_interpreter->Invoke()` respectively, in the eIQ examples.

NPU usage draws more current while it is active. However, since the NPU can perform inference up to 169x faster, it can reduce overall energy consumption per inference by up to 83x compared to execution on M33 running at 325 MHz. This is important for the applications which only need to perform inferencing occasionally, since i.MX RT700 can go back to sleep much faster.

Table 2 contains power details captured on the VDD2 compute domain where the NPU resides on and only looks at the active inference time. It was captured by combining the `Power Comp Only` SDK project with the eIQ label image project to reduce the power consumption. The project code is also executed out of RAM which saves about 8 mA of current, though that slightly increases inference times.

Table 2. Energy per inference measured at 325 MHz

Model	Average current (mA)		Inference time (ms)		Power (mW)		Energy per inference (µJ)		NPU power savings factor
	No NPU	NPU	No NPU	NPU	No NPU	NPU	No NPU	NPU	
MLPerf Tiny AD01	37.7	54.1	2.434	0.134	41.4	59.3	100.7	7.9	12.7x
MLPerf Tiny KWS	32.1	54.0	28.906	0.384	35.2	59.2	1017.8	22.7	44.8x
MLPerf Tiny Resnet	31.5	64.7	122.951	0.720	34.6	70.9	4249.2	51.0	83.3x
MLPerf Tiny VWW	31.8	54.4	92.933	0.951	34.9	59.7	3240.6	56.7	57.1x

Running at a lower frequency can improve efficiency as well. The inference time scales linearly with frequency, but the power drops at a faster rate, resulting in less energy per inference, if power savings are a premium.

Table 3. NPU inference time across frequency

	Inference time (ms)				
	45 MHz	110 MHz	192 MHz	250 MHz	325 MHz
MLPerf Tiny AD01	0.955	0.390	0.225	0.178	0.134
MLPerf Tiny KWS	2.772	1.135	0.663	0.507	0.384
MLPerf Tiny Resnet	5.19	2.123	1.214	0.939	0.720
MLPerf Tiny VWW	6.82	2.791	1.622	1.243	0.951

Table 4. Power across frequency while inferencing using the NPU

	Power (mW)				
	45 MHz	**110 MHz**	**192 MHz**	**250 MHz**	**325 MHz**
MLPerf Tiny AD01	3.8	11.1	23.4	37.3	59.3
MLPerf Tiny KWS	3.7	10.9	22.8	37.0	59.2
MLPerf Tiny Resnet	4.3	13.1	27.5	44.5	70.9
MLPerf Tiny VWW	3.7	11.1	23.0	37.3	59.7

Table 5. NPU energy per inference across frequency

	Energy per inference (µJ)				
	45 MHz	**110 MHz**	**192 MHz**	**250 MHz**	**325 MHz**
MLPerf Tiny AD01	3.7	4.3	5.3	6.6	7.9
MLPerf Tiny KWS	10.3	12.3	15.1	18.8	22.7
MLPerf Tiny Resnet	22.6	27.8	33.3	41.8	51.0
MLPerf Tiny VWW	25.4	30.9	37.3	46.4	56.7

3.2 NPU clock source

The N3-64 NPU runs at the same clock frequency as CPU0 as they share the same clock source. The maximum clock frequency for i.MX RT700 CPU0 is 325 MHz, so the maximum NPU frequency is also 325 MHz. For a clock frequency of 325 MHz, a voltage of 1.1 V is required on the VDD2 compute domain. This voltage must be supplied by an external PMIC, in this case the PCA9422UK found on the MIMXRT700 EVK, as the internal LDOs can only supply up to 1.05 V for VDD2. The eIQ examples in MCUXpresso SDK configure the PMIC and clock configuration to run at 325 MHz by default.

3.3 Memory requirements

There are two important memory locations used during eIQ enablement:

- `modeldata`: Location of the model in memory. Can be located in RAM or Flash. Inference time will be faster, if located in RAM.
- `s_tensorArena`: Memory area that is used for input, output, and intermediate arrays. Must be located in RAM.

For i.MX RT700 eIQ projects, by default the model is placed at location 0x2040_0000 in RAM using the `modeldata` location name.

The MCUXpresso SDK eIQ projects also create a `s_tensorArena` buffer that is placed in RAM and used by the model. In the MCUXpresso SDK projects, the default size for this buffer is 256 KB long and the size is set by the `kTensorArenaSize` variable, which is declared in the model data header file:

```
constexpr int kTensorArenaSize = 256 * 1024;
```

This buffer size can be optimized for the particular model being used by using the `s_interpreter->arena_used_bytes();` API call after model initialization. An example of using this API can be found in the eIQ SDK examples as it prints out the memory used. To get this value, the project must be run at least once and then the `kTensorArenaSize` variable can be modified to reduce RAM consumption. Buffer size requirements are model-specific and will depend on the structure and complexity of each model.

For larger models, the RAM allocated by default in the eIQ SDK projects for the `s_tensorArena` buffer or `modeldata` area may not be sufficient. In such case, the **NCACHE_REGION** size and modeldata size can be increased in the SDK project settings by modifying the RAM table allocation.

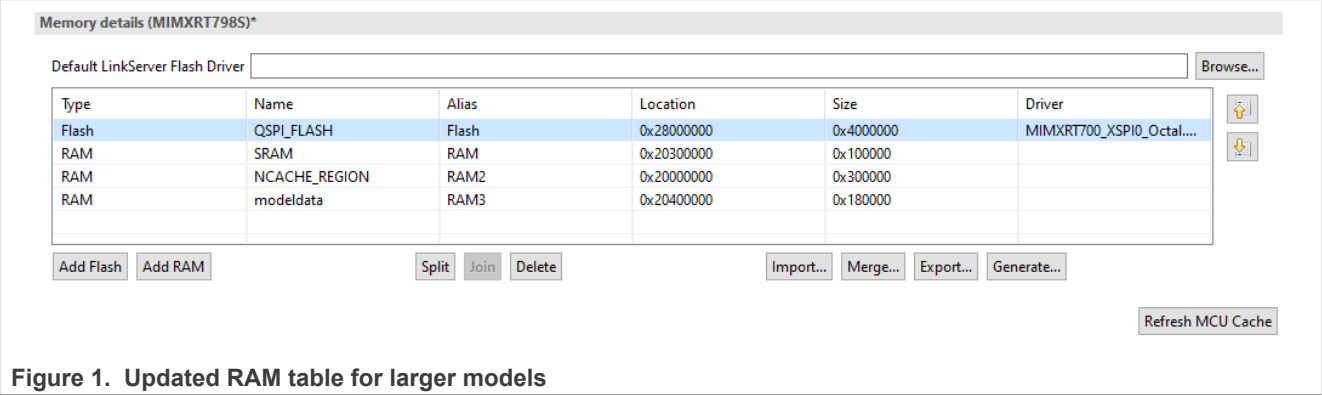

Figure 1. Updated RAM table for larger models

3.3.1 NPU memory access

The NPU has access to 5.5 MB of internal RAM from address **0x2000_0000** to **0x2058_0000** which consists of partitions SRAM P0 to SRAM P17. The model should be located inside this RAM area if it is placed in internal RAM. If using external memory, then the eIQ libraries will automatically copy the necessary model data to that internal memory location. The **s_tensorArena** buffer must also be within this memory area.

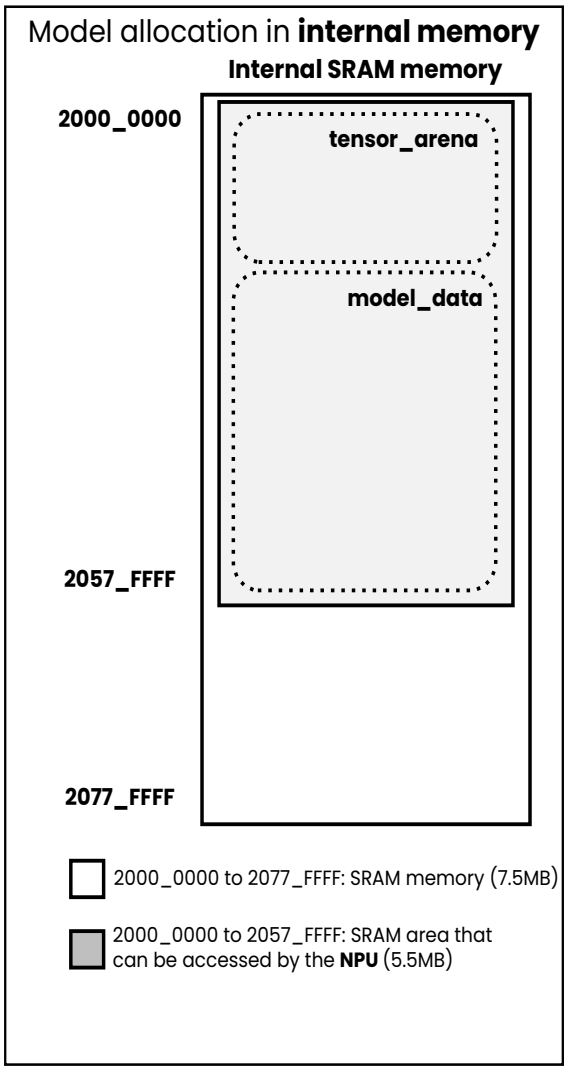

Figure 2. Model allocation in internal memory

3.4 Camera interface

The i.MX RT700 does not contain a CSI camera interface module. To support vision models, there will be examples using both a parallel camera with the OV7670 sensor and a USB camera. There is also an option of using an external CSI to parallel converter chip.

4 External memory

For large models, or in cases where RAM is limited, the model can be stored in external flash memory via XSPI0. Placing the model in the external memory has an impact on performance, however. An example of this can be found in the `tflm_label_image_ext_mem` example in the MCUXpresso SDK where the model is at address **0x2820_0000**.

Note: *If placed in external memory then the* `fetch_constants_to_sram` *option must be used during NPU conversion of the model.*

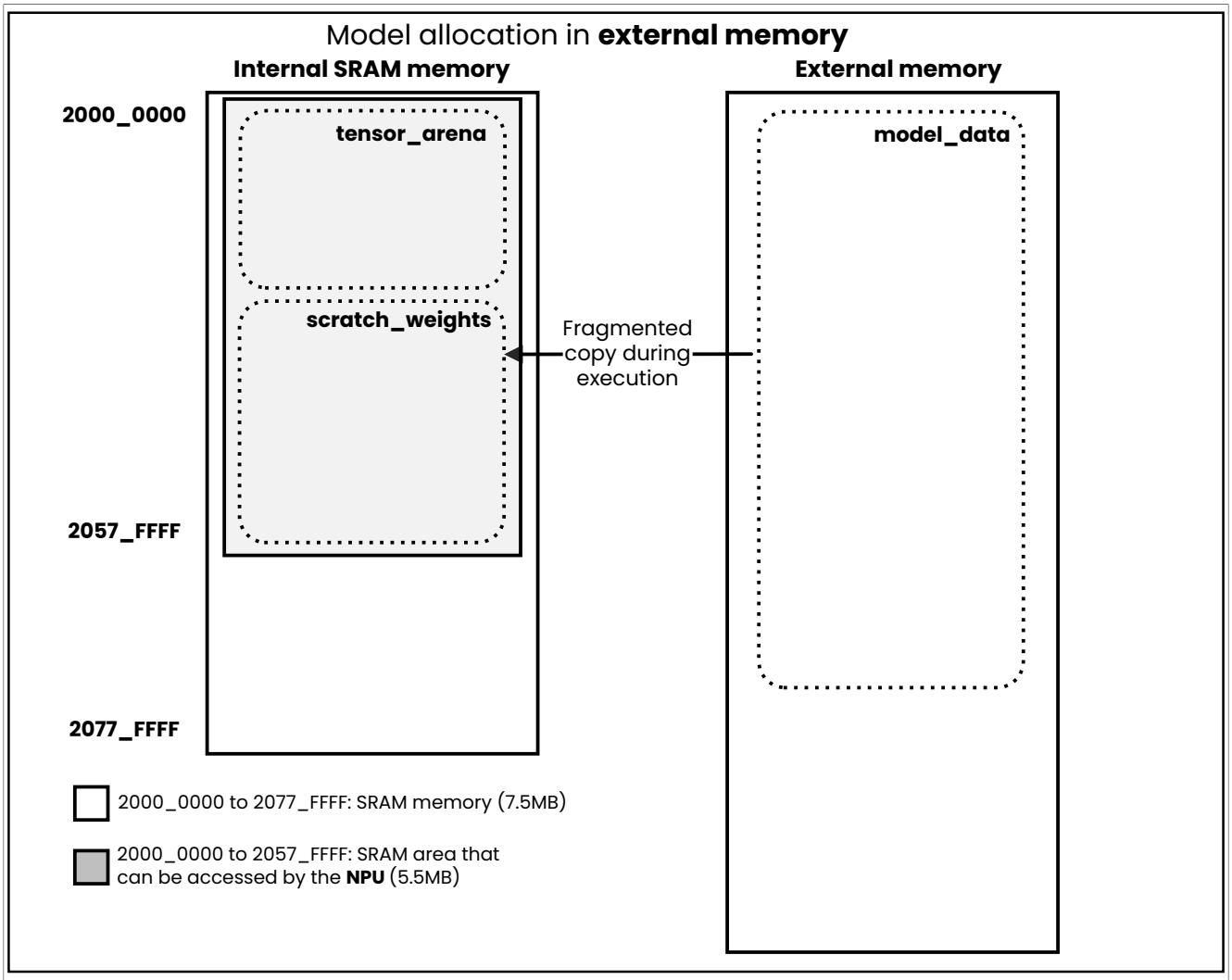

Figure 3. Model allocation in external memory

For proving the performance impact of the model being stored in internal memory (SRAM) compared to the external memory (Flash), a set of benchmarks were executed for 8 Mobilenet models with incremental sizes from 0.48 MB to 14.79 MB. These models were generated by using the standard Mobilenet structure and adjusting the alpha parameter value to get different network width and model sizes. Since the purpose of the benchmark is to evaluate model runtime performance on the embedded target and not the model prediction performance, the models were trained to classify only between two types of images. For example, lions and tigers.

The following code snippet was used to create and train the models:

```
# Define supported alpha parameter values (network width)
MODEL_WIDTH_SUPPORTED = [0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00]

# Select a particular alpha value
MODEL_WIDTH = MODEL_WIDTH_SUPPORTED[0]

# Create the MobileNet model
model =
 tf.keras.applications.mobilenet.MobileNet(input_shape=(IMG_SIZE_W,IMG_SIZE_H,3),alpha=MODEL_WIDTH,
 depth_multiplier=1, include_top=True,weights=None)
preds=Dense(2,activation='softmax')(model.output)
model = tf.keras.Model(inputs=model.input,outputs=preds)
model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
# Train the model
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
train_generator = train_datagen.flow_from_directory(DIR_IMG, target_size=(IMG_SIZE_H,
 IMG_SIZE_W),color_mode='rgb', batch_size=BATCH_SIZE,class_mode='categorical', shuffle=True)
hist = model.fit(train_generator, epochs=EPOCHS, validation_data=train_generator,
 callbacks=callbacks)
```

Table 6 captures the sizes of the generated models according to the alpha parameter value.

Table 6. Models and alpha values

Model version	Model name	Alpha value	Model parameters count (x10^6)	TFLite float model size [MB]	TFLite int8 NPU model size [MB]
1	Mobilenet 0.48_r128_a0.25	0.25	0.48	1.8	0.48
2	Mobilenet 1.34_r128_a0.5	0.5	1.34	5.08	1.32
3	Mobilenet 2.60_r128_a0.75	0.75	2.60	9.85	2.54
4	Mobilenet 4.25_r128_a1.0	1	4.26	16.1	4.13
5	Mobilenet 6.30_r128_a1.25	1.25	6.30	23.8	6.10
6	Mobilenet 8.73_r128_a1.5	1.5	8.74	33.1	8.44
7	Mobilenet 11.56_r128_a1.75	1.75	11.57	43.9	11.1
8	Mobilenet 14.79_r128_a2.0	2	14.79	56.1	14.2

Table 7 shows the performance impact of the model being in RAM compared to external Flash for different Mobilenet models from 0.48 MB to 14.79 MB in size:

Table 7. External memory impact on NPU performance (inference in milliseconds)

Model version	Mobilenet name	Model in RAM	Model in external Flash via XSPI0	Difference
1	Mobilenet 0.48_r128_a0.25	3.451 ms	5.26 ms	1.52x
2	Mobilenet 1.34_r128_a0.5	6.015 ms	10.447 ms	1.74x
3	Mobilenet 2.60_r128_a0.75	9.598 ms	18.093 ms	1.89x
4	Mobilenet 4.25_r128_a1.0	14.287 ms	27.855 ms	1.95x
5	Mobilenet 6.30_r128_a1.25	Does not fit	38.933 ms	Not available
6	Mobilenet 8.73_r128_a1.5	Does not fit	54.272 ms	Not available
7	Mobilenet 11.56_r128_a1.75	Does not fit	69.862 ms	Not available
8	Mobilenet 14.79_r128_a2.0	Does not fit	88.886 ms	Not available

5 eIQ Neutron NPU documentation

The N3-64 NPU module chapter in the *i.MX RT700 Reference Manual* contains an overview of the module and the supported operators. All enablement is handled using the eIQ neutron libraries, included in the MCUXpresso SDK. Therefore, the register details for the NPU module are not described in the reference manual.

Note: *The list of operators in the i.MX RT700 Reference Manual is the theoretical capabilities of the NPU but not all operators have been enabled with eIQ software. To view the current list of operators supported by eIQ enablement, browse the latest version of eIQ Toolkit software for the document eIQ Toolkit User Guide.*

6 Model conversion for NPU

The Neutron converter tool (NCT) is a utility included in the eIQ Toolkit. The NCT analyzes a quantized TFLite model, determines layers that can be accelerated by the NPU and then groups them in a custom node. The custom node is called as a **NeutronGraph** node.

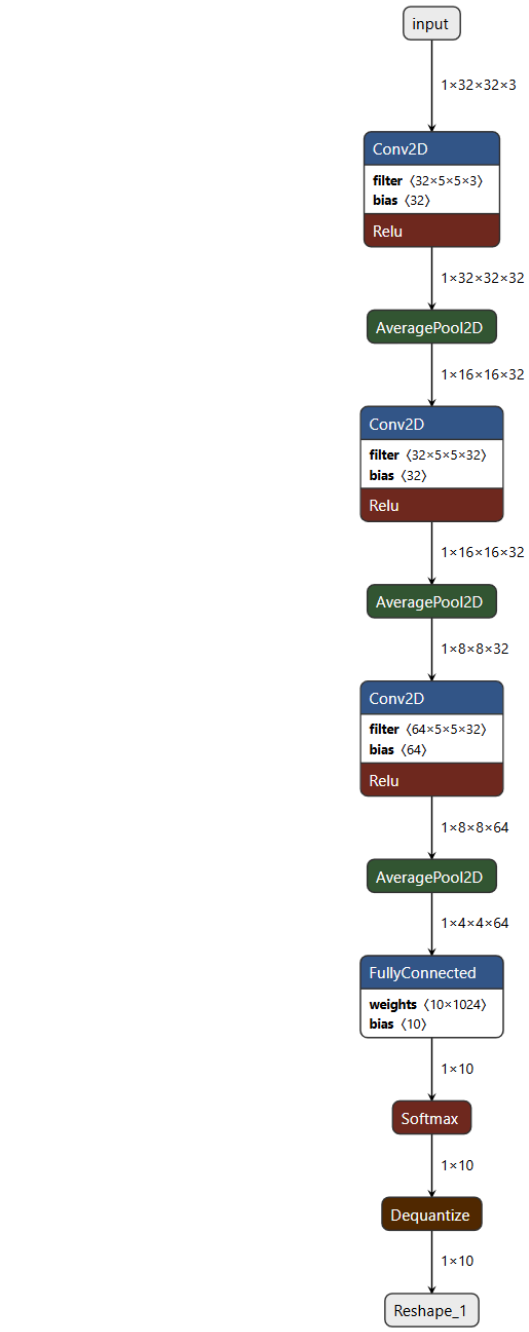

Figure 4. Original model

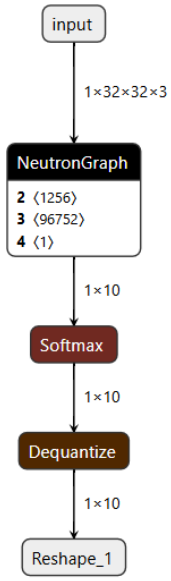

Figure 5. Neutron NPU converted model

The output of the NCT is a TFLite file that contains NeutronGraph nodes. The more layers of a model that can be optimized to run on the NPU, the faster the model will execute. For optimal performance, design the model using supported TFLite operators to maximize the number of operations that can be offloaded to the NPU.

It is required that the correct NCT version is used to convert a quantized TFLite model for the associated MCUXpresso SDK version. If the wrong version is used, then the following error is generated:

```
Incompatible Neutron NPU microcode and driver versions! Please, convert the model with Neutron converter tool
 intended for this SDK release.
```

The eIQ Toolkit comes with current and previous NCT version for different versions of MCUXpresso SDK and Linux BSP. Ensure that the correct NCT version is selected for the firmware that is used in the eIQ Toolkit settings.

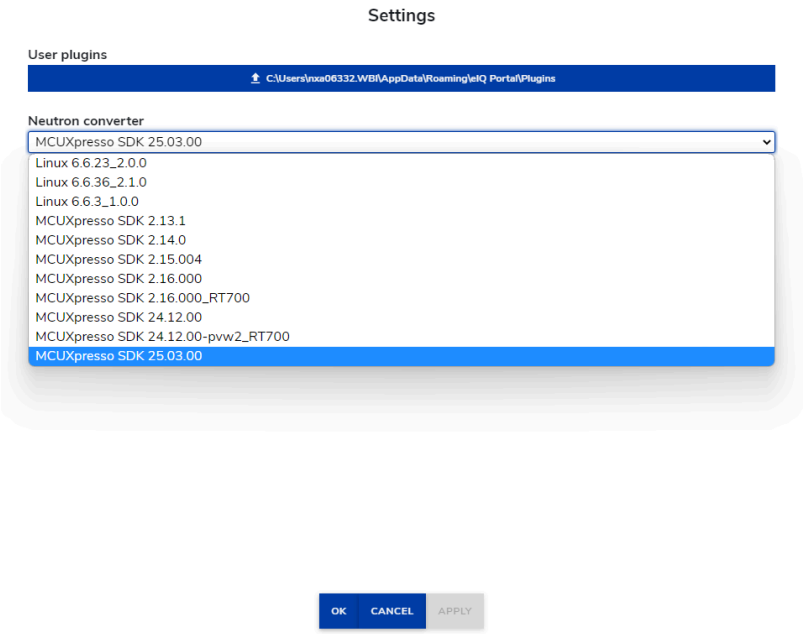

Figure 6. Selecting Neutron converter version in eIQ toolkit

During the conversion process, there are several options available:

- `--input`: Specify the quantized **.tflite* file to use as input.
- `--output`: Specify the output filename and location of the NPU converted file.
- `--target`: Specify the device that the model runs on. For i.MX RT700, it is `imxrt700`.
- `--dump-header-file-output`: Create a **.h* file containing the converted model.
- `--dump-header-file-input`: Create a **.h* file containing an array with the original nonconverted input model. Useful for comparing the performance increase that NPU provides.
- `--use-sequencer`: Use a sequencer mode during conversion which prioritizes model performance over model size. For details, see Section 6.1.
- `--fetch-constants-to-sram`: Allow storing of the model weights in an external memory. Required if the model is read from an external flash during inferencing.
- `--help`: Display the other commands and details on each command.

6.1 Model size reduction

When using the NCT for model conversion, it reduces the size of the model due to how the eIQ Neutron NPU converted models are able to more efficiently store quantization parameters in the NeutronGraph nodes compared to the original TFLite files. In a standard TFLite model, each quantization parameter requires 20 bytes but in an eIQ Neutron converted model, each quantization parameter requires only 2 bytes. There are also savings due to reduction in layer names in the eIQ Neutron converted models. The extent of file size reduction is model dependent; in certain cases, there might be a marginal increase in size due to structural modifications introduced during model restructuring.

See Table 8 for the impact of NPU conversion using the NCT version compatible with SDK version 25.03, on these particular models found in MCUXpresso SDK (using non-sequencer mode):

Table 8. Conversion file size differences

Model	Original TFLite size	After NPU conversion	Size reduction
MLPerf Tiny AD01	277 KB	273 KB	2%
MLPerf Tiny KWS	54 KB	32 KB	41%
MLPerf Tiny Resnet	98 KB	85 KB	14%
MLPerf Tiny VWW	333 KB	242 KB	27%

6.2 Sequencer mode

This is a feature available for i.MX RT700 that trades a larger model size for increased performance. The specific increase in size and performance are model-dependent. Performance improvement can vary from a 1% up to 18% on some models, while size increase can be anywhere from 2.1% to up to 82%. Also note that models that utilize sequencer mode cannot be executed from external memory. They must be located in internal SRAM.

See the chart below for the impact that sequencer mode has on these particular models in both performance and model size in MCUXpresso SDK version 25.03:

Table 9. Sequencer mode performance impact

Model	NPU without sequencer	NPU with sequencer	Performance increase
MLPerf Tiny AD01	0.130 ms	0.134 ms	-3.1%
MLPerf Tiny KWS	0.475 ms	0.384 ms	19.2%
MLPerf Tiny Resnet	0.725 ms	0.717 ms	1.1%
MLPerf Tiny VWW	1.087 ms	0.966 ms	11.1%

Table 10. Sequencer mode model size impact

Model	NPU without sequencer	NPU with sequencer	Size increase
MLPerf Tiny AD01	273 KB	278 KB	2.1%
MLPerf Tiny KWS	32 KB	57 KB	81.4%
MLPerf Tiny Resnet	85 KB	89 KB	6.3%
MLPerf Tiny VWW	242 KB	282 KB	16.5%

7 eIQ enablement

This section provides a high level overview of the steps required to convert a model and integrate it into a MCUXpresso SDK eIQ example. For more specific details updated for each new SDK release, check the NXP Community for lab guides that walk through this process step by step.

7.1 Software requirements

Download the following software:

1. MCUXpresso SDK for i.MX RT700.
2. eIQ Toolkit: Available for both Windows and Linux.

In addition, install the GCC, Python, and Git dependencies either by using the MCUXpresso Installer tool or following the instructions on this webpage.

Note: *The GitHub version of the SDK does not support the MCUXpresso IDE. Therefore, use GCC and/or VSCode.*

7.2 Model conversion

After installing the eIQ Toolkit, use the NCT to convert the quantized TFLite model. In this example, Mobilenet v1 0.25 model is used.

1. To convert the TFLite model from the command line, add the file *neutron-converter.exe* located at *C:\NXP\eIQ_Toolkit_v1.15.1\bin\neutron-converter\MCU_SDK_25.03.00* to your executable path.
   Ensure that the path matches with the SDK version being used.
2. Use the following command for conversion:

```
neutron-converter --input mobilenet_quant.tflite --output
 mobilenet_npu.tflite --dump-header-file-input --dump-header-file-output --
target imxrt700 --use-sequencer
```

3. After conversion, compare the original model with the new converted model.
   For example:
   - The original TFLite file: **mobilenet_quant.tflite**

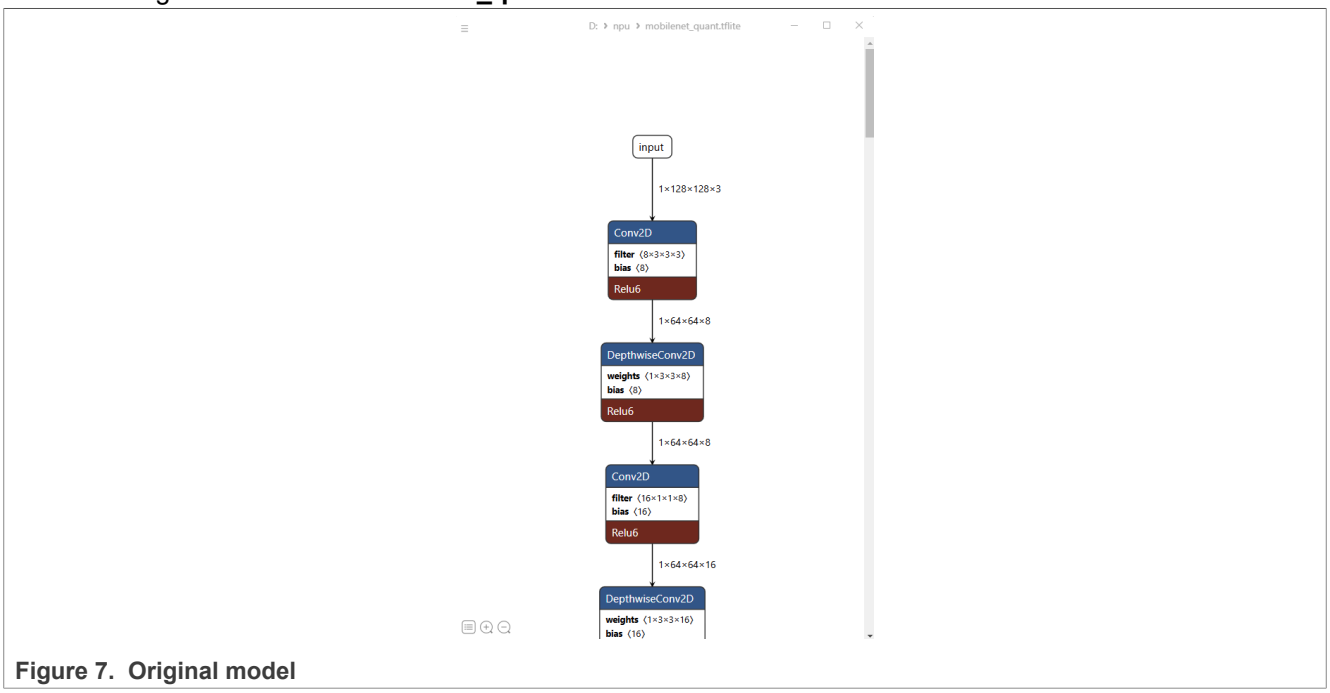

Figure 7. Original model

   - The Neutron converted file: **mobilenet_npu.tflite**

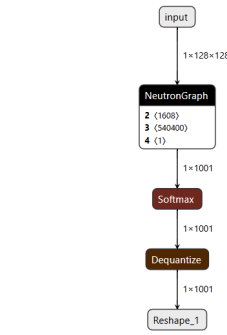

Figure 8. Neutron NPU converted model

You can observe the following:

- All the operators in the original model are replaced with a **NeutronGraph** operator. The **NeutronGraph** operators are executed on the eIQ Neutron NPU when this model runs on the i.MX RT700.
  Any layers that were not converted to a NeutronGraph operator runs on the Cortex-M33 core.
- In general, the NPU converted **.tflite* file will take up less memory space. Note that this might be counteracted by the slightly increased size required for using the eIQ Neutron libraries.
- During the conversion process, the `--dump-header-file-output` argument generated the **.h* header file for the NPU optimized model that can be used in the eIQ MCUXpresso SDK projects.
- The `--dump-header-file-input` argument generated the **.h* header file for the original non-converted model. This will be used so the inference time of the original model that only runs on the Cortex-M33 core can be compared to the NPU converted model that makes use of the eIQ Neutron NPU.

7.3 Run the converted model

To use the converted model inside a MCUXpresso SDK eIQ project, perform the following steps:

1. Download the SDK from NXP's GitHub repository using West.
   a. Run the following commands in an empty directory:
   Note: *The `west update` command takes several minutes to download the SDK repository*

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git
 mcuxpresso-sdk
cd mcuxpresso-sdk/manifests
git branch --all
git checkout remotes/origin/release/25.03.00
cd ..
west update
```

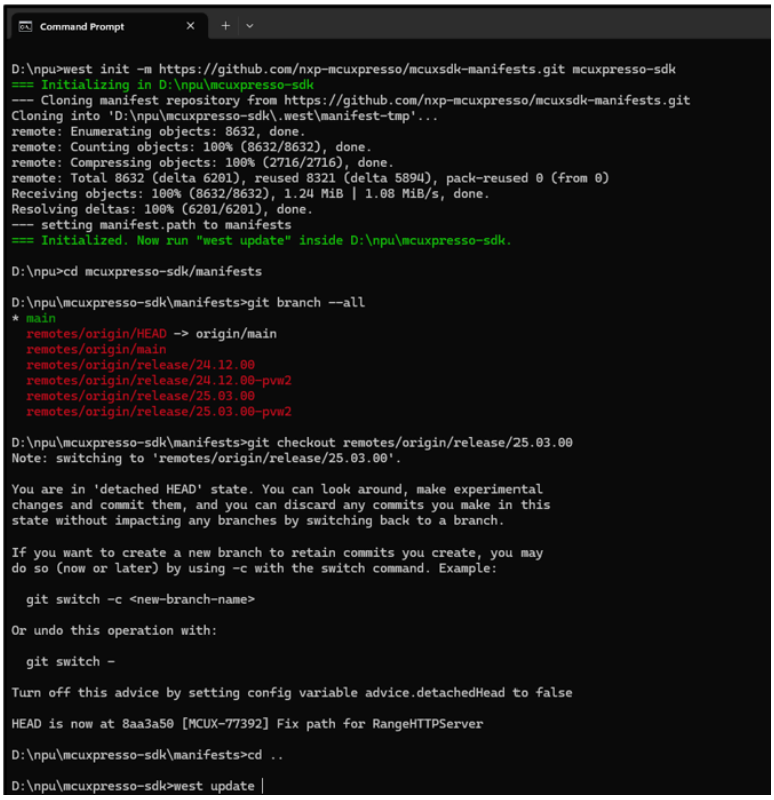

Figure 9. Download MCUXpresso SDK from GitHub

2. Import the models that were generated in the last section into this TFLM Label Images project.
   a. Go to the location:
   \mcuxsdk\examples_boards\mimxrt700evk\eiq_examples\tflm_label_image\pcq_npu
   b. Copy and paste the two **.h* header files that were generated in the previous section into this file location.

   It should look like the following when complete:

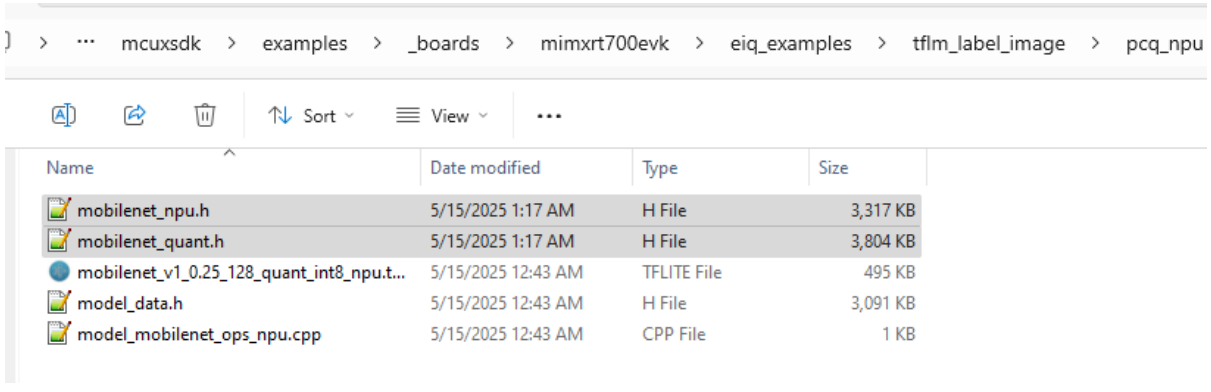

Figure 10. Copying in the generated header files

3. Modify the two header files to update some information for the eIQ MCUXpresso SDK project that describes :
   - How much memory this model requires and
   - Describe some of the normalization values that this model uses:

   a. Open *model_data.h* in that same folder which contains the default model for this example. Find the following section of code and copy it:

model_data.h

```
// Copyright 2022-2024 NXP.
// All rights reserved.
// SPDX-License-Identifier: BSD-3-Clause

// Neutron Converter Version: 1.2.0+0X1b86b19d
// Neutron Microcode Version: 0X1b86b19d

/*
// Register operators for TFLite Micro.
static tflite::MicroMutableOpResolver<3> s_microOpResolver;
s_microOpResolver.AddSoftmax();
s_microOpResolver.AddDequantize();
s_microOpResolver.AddCustom(tflite::GetString_NEUTRON_GRAPH(), tflite::Register_NEUTRON_GRAPH());
*/

#ifdef __arm__
#include <cmsis_compiler.h>
#else
#define __ALIGNED(x) __attribute__((aligned(x)))
#endif

#ifdef __MCUXPRESSO
#define __PLACEMENT __attribute__((section(".data.$modeldata")))
#else
#define __PLACEMENT __attribute__((section(".modeldata")))
#endif

#define MODEL_NAME "mobilenet_v1_0.25_128_quant_int8_npu"
#define MODEL_INPUT_MEAN 127.5f
#define MODEL_INPUT_STD 127.5f

constexpr int kTensorArenaSize = 256 * 1024;

static const uint8_t model_data[] __ALIGNED(16) __PLACEMENT = {
  0x08, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0xda, 0x4a, 0xf8, 0xff,
  0x03, 0x00, 0x00, 0x00, 0xe4, 0xb8, 0x07, 0x00, 0x10, 0xb5, 0x07, 0x00,
  0xa0, 0xb4, 0x07, 0x00, 0x04, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
```

Figure 11. model_data.h

b. Then in both the files *mobilenet_quant.h* and *mobilenet_ npu.h* copy that code above the array, overwriting the array declaration and the `#define` names originally.

model_data.h | mobilenet_npu.h

```
// Copyright 2022-2024 NXP.
// All rights reserved.
// SPDX-License-Identifier: BSD-3-Clause

// Neutron Converter Version: 1.2.0+0X1b86b19d
// Neutron Microcode Version: 0X1b86b19d

/*
// Register operators for TFLite Micro.
static tflite::MicroMutableOpResolver<3> s_microOpResolver;
s_microOpResolver.AddSoftmax();
s_microOpResolver.AddDequantize();
s_microOpResolver.AddCustom(tflite::GetString_NEUTRON_GRAPH(), tflite::Register_NEUTRON_GRAPH());
*/

#define MODEL_NAME "mobilenet_npu.tflite"
#define MODEL_SIZE 543312

static const unsigned char model_data[] __attribute__((aligned(16))) = {
  0x08, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0xea, 0xb9, 0xf7, 0xff,
  0x03, 0x00, 0x00, 0x00, 0xd4, 0x49, 0x08, 0x00, 0x00, 0x46, 0x08, 0x00,
```

Figure 12. Original mobilenet_quant.h and mobilenet_npu.h files

After copying the code, it should look like the following image. Ensure not to erase the commented lines at the top of the file as those comments will be used later. Note that the **MODEL_NAME** can be changed to “mobilenet”.

```
model_data.h   mobilenet_npu.h
 1  // Copyright 2022-2024 NXP.
 2  // All rights reserved.
 3  // SPDX-License-Identifier: BSD-3-Clause
 4
 5  // Neutron Converter Version: 1.2.0+0X1b86b19d
 6  // Neutron Microcode Version: 0X1b86b19d
 7
 8  /*
 9  // Register operators for TFLite Micro.
10  static tflite::MicroMutableOpResolver<3> s_microOpResolver;
11  s_microOpResolver.AddSoftmax();
12  s_microOpResolver.AddDequantize();
13  s_microOpResolver.AddCustom(tflite::GetString_NEUTRON_GRAPH(), tflite::Register_NEUTRON_GRAPH());
14  */
15
16  #ifdef __arm__
17  #include <cmsis_compiler.h>
18  #else
19  #define __ALIGNED(x) __attribute__((aligned(x)))
20  #endif
21
22  #ifdef __MCUXPRESSO
23  #define __PLACEMENT __attribute__((section(".data.$modeldata")))
24  #else
25  #define __PLACEMENT __attribute__((section(".modeldata")))
26  #endif
27
28  #define MODEL_NAME "mobilenet_v1_0.25_128_quant_int8_npu"
29  #define MODEL_INPUT_MEAN 127.5f
30  #define MODEL_INPUT_STD 127.5f
31
32  constexpr int kTensorArenaSize = 256 * 1024;
33
34  static const uint8_t model_data[]  __ALIGNED(16)  __PLACEMENT = {
35    0x08, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0xea, 0xb9, 0xf7, 0xff,
36    0x03, 0x00, 0x00, 0x00, 0xd4, 0x49, 0x08, 0x00, 0x00, 0x46, 0x08, 0x00,
```

Figure 13. Mobilenet_quant.h and mobilenet_npu.h files after modification

4. Next go to *\mcuxpresso-sdk\mcuxsdk\examples\eiq_examples\common\tflm* and open *model.cpp*
5. Go to line 27 and change it to point to the non-NPU accelerated model in *mobilenet_npu.h*. It should look like the following after changed:

```
19
20  #include "tensorflow/lite/micro/kernels/micro_ops.h"
21  #include "tensorflow/lite/micro/micro_interpreter.h"
22  #include "tensorflow/lite/micro/micro_op_resolver.h"
23  #include "tensorflow/lite/schema/schema_generated.h"
24
25  #include "fsl_debug_console.h"
26  #include "model.h"
27  #include "mobilenet_npu.h"
28
29  static const tflite::Model* s_model = nullptr;
30  static tflite::MicroInterpreter* s_interpreter = nullptr;
31
32  extern tflite::MicroOpResolver &MODEL_GetOpsResolver();
```

Figure 14. Using new model

6. Now, you must update the list of operators used by the model. Open the *model_mobilenet_ops_npu.cpp* file at *\mcuxsdk\examples_boards\mimxrt700evk\eiq_examples\tflm_label_image\pcq_npu*.
7. Inside the **MODEL_GetOpsResolver** function is a list of operators. We must replace this list of operators with the ones found in the *mobilenet_npu.h* header file.

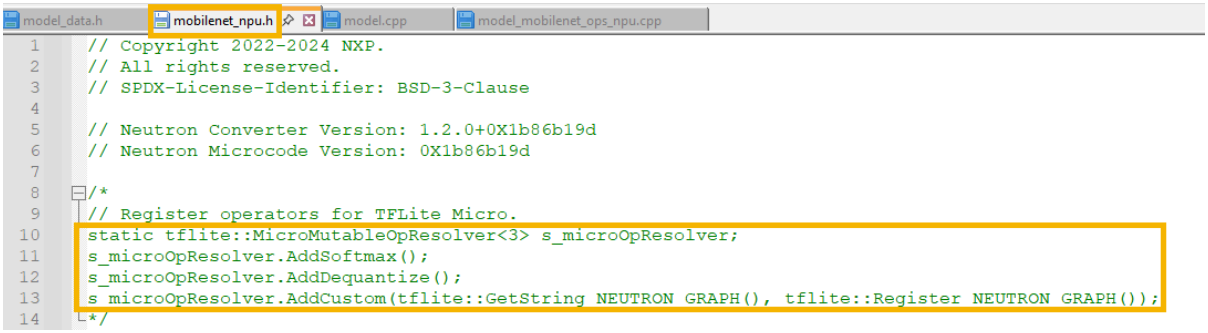

Figure 15. Ops used by model

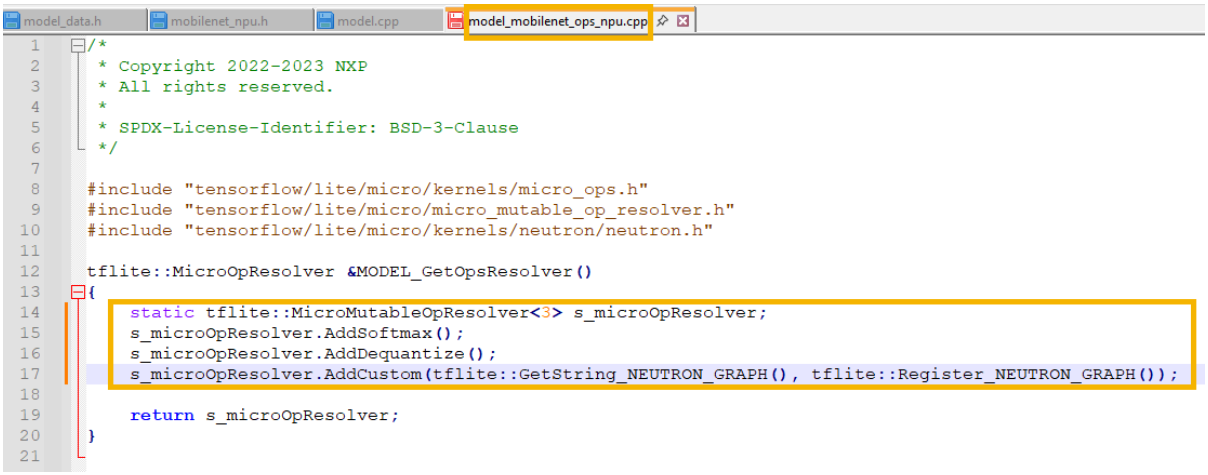

Figure 16. Updating the MODEL_GetOpsResolver function with Ops for the model

8. Compile the TFLM label image project:
   a. Go to the mcuxpresso-sdk\mcusdk directory and see the possible compile commands by using *west list_project -p examples/eiq_examples/tflm_label_image*:
   b. And then compile the project by running the following command:

```
west build -p always examples/eiq_examples/tflm_label_image --toolchain
 armgcc --config flash_debug -b mimxrt700evk -Dcore_id=cm33_core0
```

9. Connect a USB micro B cable from your computer to the USB port on the MIMXRT700 EVK at **J54**. Also ensure that **JP1** and **JP3** are shunted on the board and that **SW10** has pin 1 **OFF** and pin 2 **ON**

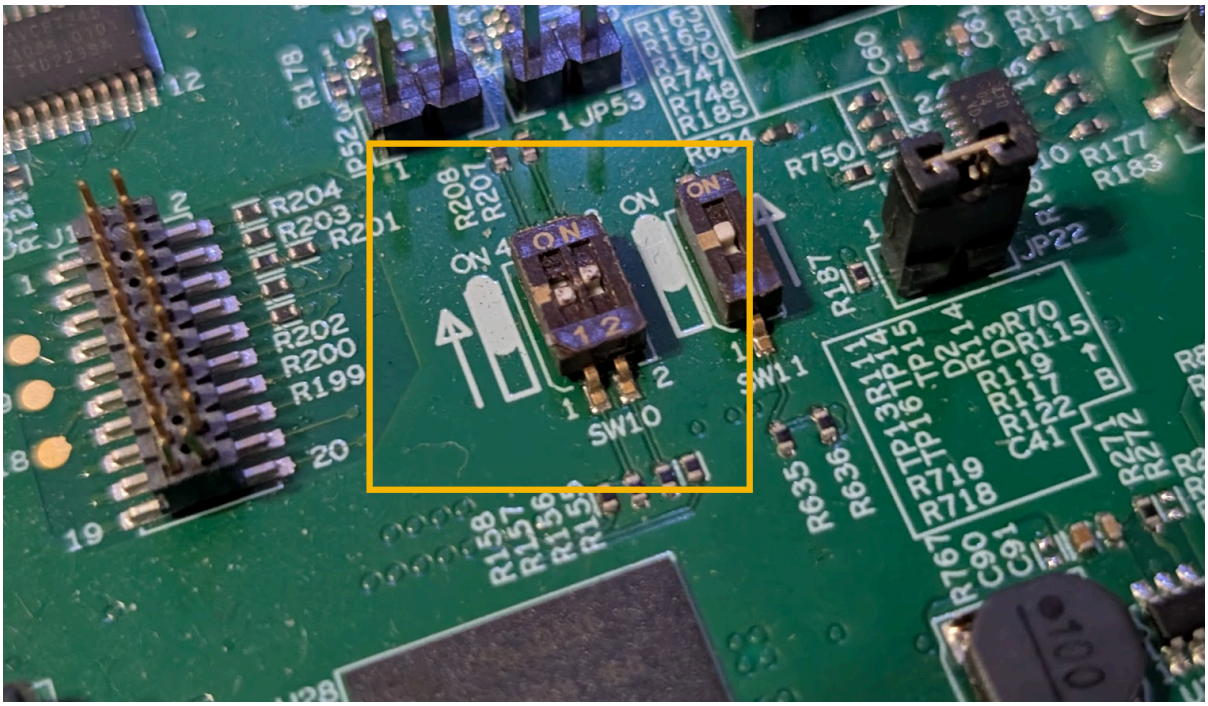

Figure 17. SW10 settings

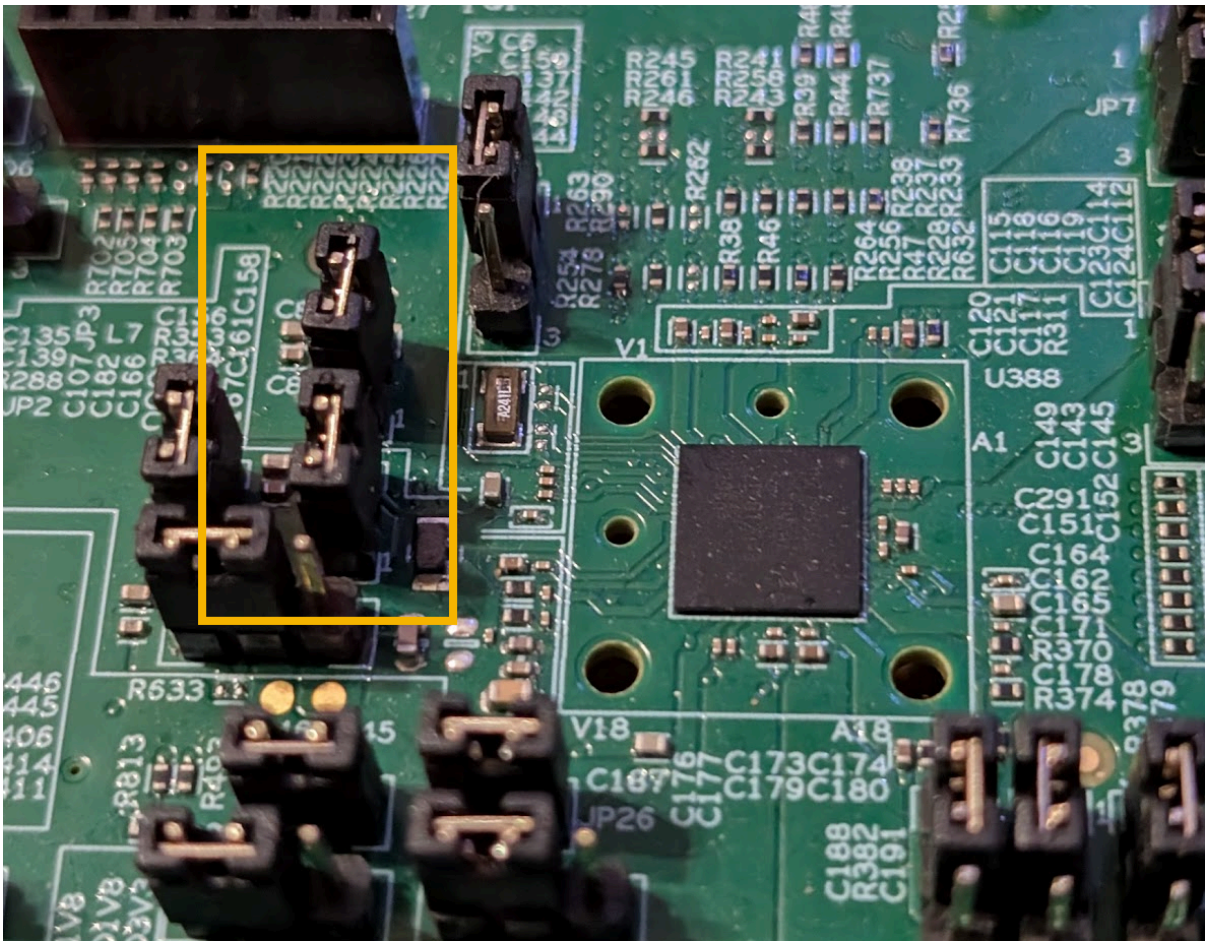

Figure 18. JP1 and JP3 settings

10. Open TeraTerm or other serial terminal program, and connect to the virtual COM port that board enumerated as when you plugged in the USB cable (your COM number will likely be different than the screenshot). Use 115200 baud, 1 stop bit, no parity.
11. Flash and run the TFLM label image project like done for the other MCUXpresso SDK examples as described in the MCUXpresso documentation.
12. You should now see the following on the serial terminal:

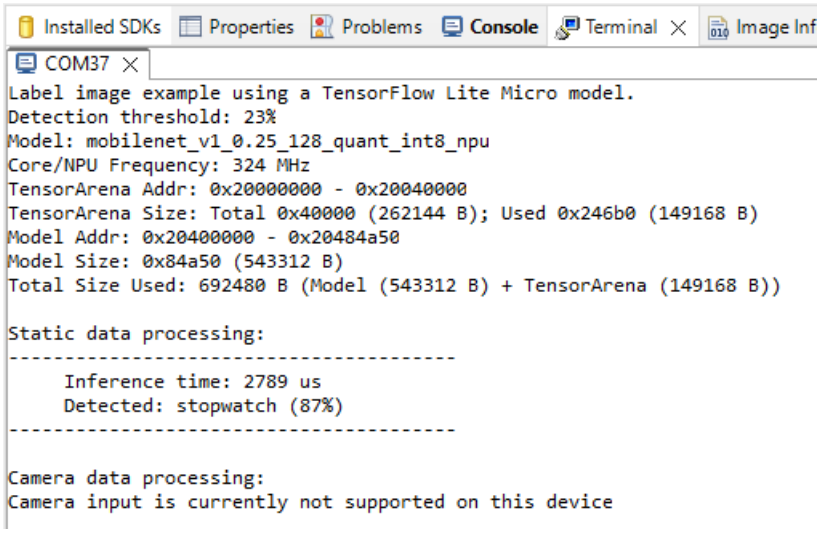

Figure 19. Terminal output for inference

13. If desired, the same process can be carried out to run the original version of the model to see the performance difference that the NPU provides. To get the best performance from running on the CM33 core, the declaration of the `s_tensorArena` buffer in the *model.cpp* file must be changed to the following. This should only be done when running the original non-NPU version of the model. If used for an NPU version of the model then the model generates incorrect results.

```
static uint8_t s_tensorArena[kTensorArenaSize] __ALIGNED(16)
```

8 Note about the source code in the document

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

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

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

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

9 Revision history

Table 11 summarizes the revisions done to this document.

Table 11. Revision history

Document ID	Release date	Description
AN14700 v.2.0	10 November 2025	Initial public release
AN14700 v.1.0	27 June 2025	Initial internal release

Legal information

Definitions

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

Disclaimers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Suitability for use in industrial applications (functional safety) — This NXP product has been qualified for use in industrial applications. It has been developed in accordance with IEC 61508, and has been SIL-classified accordingly. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

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

Trademarks

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

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

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

eIQ — is a trademark of NXP B.V.

TensorFlow, the TensorFlow logo and any related marks — are trademarks of Google Inc.

Contents