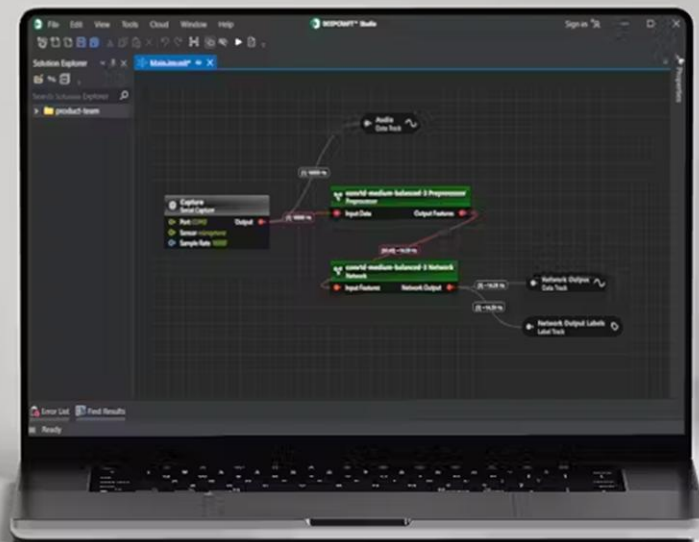




DEEPCRAFT™ Model Converter Hands-On Training Manual

Version 1.0 – August 2026

Hands-On Guide to DEEPCRAFT™ Model Converter, Infineon's tool for converting models to embedded code.

This tutorial will guide users through deploying and profiling models on Infineon microcontrollers.

Lukas.wirkestrand@imagimob.com

Table of Contents

1. Introduction	2
1.1 – Description of the DEEPCRAFT™ Model Converter	3
1.2 – Supported Models and Layers	4
1.3 - Goal of the Training	4
2. Hands On – Converting an open-source Model	5
2.1 - Exercise 1: Downloading the ResNet Model	5
2.2 - Exercise 2: General Settings of the DEEPCRAFT™ Model Converter	5
2.3 - Exercise 3: Inspecting the Model in the Model Viewer	7
2.4 - Exercise 4: Quantizing the Model	8
2.5 - Exercise 5: Advanced Options (Solved)	8
2.6 - Exercise 6: Preparing on-device Validation Data and Generating the Code	10
2.7 - Exercise 7: Deploying & Profiling the Model on PSOC™ Edge Kit	11
2.8 - Exercise 8: Validating the Model in Model Converter (Optional)	18
2.8.1 – Target Validation	18
2.8.2 – Desktop Validation	20
2.9 - Exercise 9: Using the Command-Line Interface (CLI) (Optional)	23
3. Final Considerations and Additional Documentation	23

DEEPCRAFT™ Model Converter Hands-On Training Manual

Version 1.0 – August 2026

1. Introduction

DEEPCRAFT™ Model Converter is a comprehensive tool designed to facilitate the conversion of pre-trained models for deployment on an Infineon target platform. By leveraging the Model Converter, users can generate code for existing models, optimize pre-trained models for specific target devices through configurable optimization parameters, and validate model performance on the desktop prior to actual device deployment. This allows for the deployment of a wide host of open-sourced models, alongside existing models, to be deployed on Infineon microcontrollers. Naturally these models must still be small enough to fit onto the targets. However, with the release of larger and more powerful microcontrollers there are many doors opened to take existing models and run them on a more lightweight edge device as opposed to the cloud.

The Model Converter is one of the tools in the DEEPCRAFT™ AI Suite alongside DEEPCRAFT™ Studio. Whereas Studio is an end-to-end platform for defining and building new models, Model Converter instead assumes a fully developed model already exists. As such, it is a much simpler tool which still allows for additional optimizations based on the selected hardware target. If changes need to be made to a model's architecture, then the developer must return to the source, make the changes, and then come back to the DEEPCRAFT™ Model Converter.

There are plenty of places to find models to deploy. Open-sourced repositories like HuggingFace and RoboFlow include models for many common and niche applications which could potentially be deployed on a microcontroller by using the Model Converter. Additionally, the DEEPCRAFT™ AI Hub is a centralized place for Infineon customers to find DEEPCRAFT™ software for any Edge AI needs: platform, tools, solutions, and models all in one place. These models can be used with the model converter to deploy on a target microcontroller or otherwise serve as a repository of example projects for Edge AI.

This manual does not go into detail about machine learning theory or basics, instead serving to present how to practically apply such knowledge to port existing machine learning models optimized to run on microcontrollers. Here are some recommended references to start, but there is also plenty available online:

- [Machine Learning Crash Course](#)
Online class developed by Google
- [Deep Learning with Python 2nd Edition by François Chollet](#)
E-book with lots of examples that cover deep learning using Python, TensorFlow and Keras
- [Machine Learning Mastery](#)
Website of guides and tutorials that cover and define many ML concepts

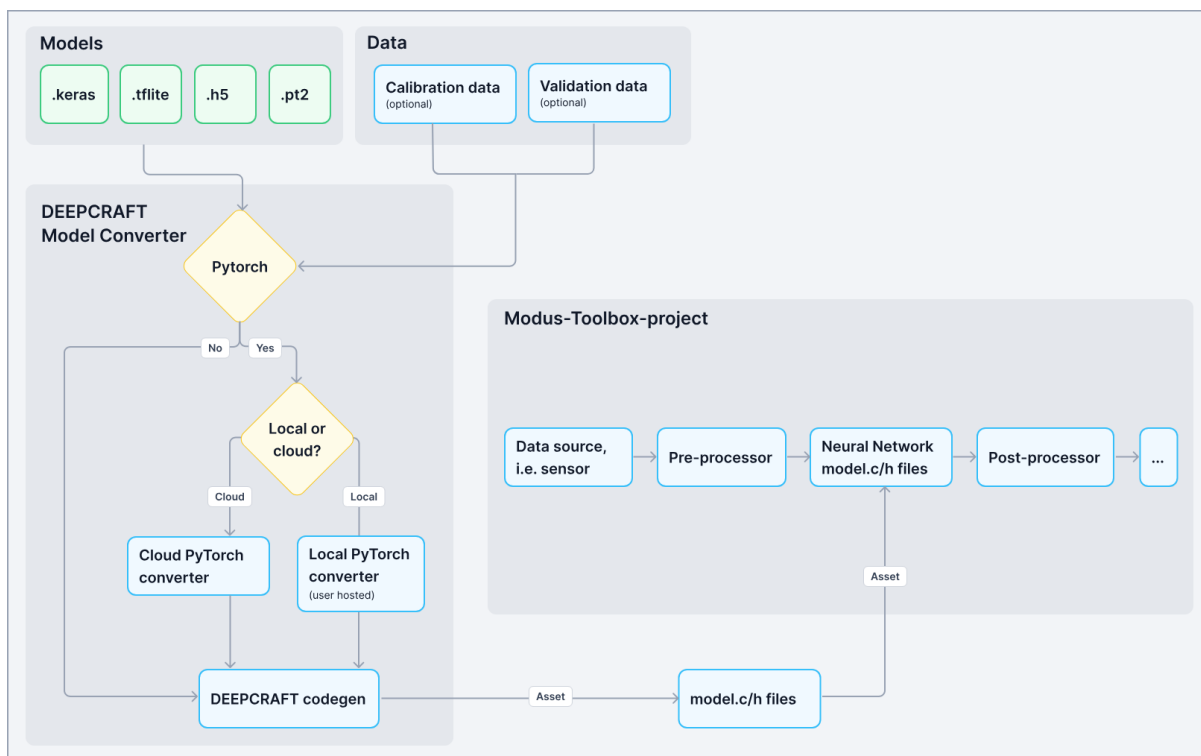
You can read the technical documentation of the DEEPCRAFT™ Model Converter [here](#).

1.1 Description of the DEEPCRAFT™ Model Converter

DEEPCRAFT™ Model Converter generates code files that enable the execution of Neural Networks on an Infineon PSOC™ 6 or PSOC™ Edge target. The DEEPCRAFT™ Model Converter accepts the network file, along with optional calibration or validation data, as input and generates model.c and model.h files that can be imported into a project in an embedded IDE like ModusToolbox™. The Model Converter runs completely locally on your PC, with the exception of an optional cloud-based PyTorch conversion.

DEEPCRAFT™ Model Converter is available in both Graphical User Interface (GUI) and Command-Line Interface (CLI) versions, providing flexibility and convenience for developers working in a variety of workflows. Model Converter is fully compatible with **Windows**, **macOS**, and **Linux** operating systems allowing users to integrate the models into the preferred development environments. This training will primarily show the GUI view for explainability, but everything can be done identically in the CLI which is briefly described at the end.

The flowchart below shows the workflow of converting a model. As you can see there is a slightly different path taken when a PyTorch model is inputted due to the need for an additional conversion step. This step can be run either via the Infineon cloud by having an account or in a local container with no required connection. The core part of the tool is the DEEPCRAFT™ Codegen, which takes a TensorFlow model and converts it into C code. The optional Calibration and Validation data are used for quantization and validation, which will be explained later. Setup for the local container can be found [here](#).



1.2 Supported Models and Layers

As stated in the description, the supported file formats of models are:

- TensorFlow Keras (.h5 or .keras)
- LiteRT (formerly TFlite) (.tflite)
- PyTorch (.pt2)

Also note that other common model formats like .pt can easily be converted to .pt2 using the PyTorch package with torch v2.6.0+. More details on that can be found in [torch.export](#).

Currently the only targets which are supported for output are the PSOC™ 6 and PSOC™ Edge.

Because Model Converter utilizes TensorFlow in the backend for the conversion to C code, it is limited in which models it supports based on the layers supported by that framework. A full list of the exact layers supported can be found [here](#). However, this does not outline which layers get accelerated by the Neural Processing Units of the U55 or NNLite, yet.

The Table below outlines which quantization levels are supported and which model format:

Supported Quantization	TensorFlow Keras	TensorFlow Lite	PyTorch
Float32 activations + Float32 weights	✓	✓	✓
Int8 activations + Int8 weights	✓	✓	✓
Int16 activations + Int8 weights	✓	✓	x

For models to be quantized you must provide representative input samples.

1.3 Goal of the Training

The goal of this training is to take an existing model downloaded from an open-sourced repository and learn the process of converting using the Model Converter. We will focus on the GUI and explore the different options that can be used like: Model Viewer, Quantization, Ethos NPU options, Interpreterless and various validations.

In total this Hands-on is expected to take 90min to complete.

2. Hands On – Converting an open-source Model

Pre-requisites:

- DEEPCRAFT™ Model Converter >= 5.10
- ModusToolbox™ version >= 3.8
- PSOC™ Edge E84 AI Evaluation Kit or PSOC™ 6 AI Evaluation Kit
- USB-C Cable

ModusToolbox™ and DEEPCRAFT™ Model Converter can both be downloaded from the [ModusToolbox™ Setup Tool](#).

2.1 Exercise 1: Downloading the ResNet Model

For this exercise, we will utilize the well-known open-sourced model ResNet. ResNet was released in 2015 for image recognition and utilizes residual connections to overcome common problems like the vanishing gradient problem and the degradation problem. For our purposes today we will not dive into the details of this model, it is simply chosen as an arbitrary example to run through this entire exercise with. You are encouraged to try another model, so long as it aligns with the limitations outlined in Chapter 1.2.

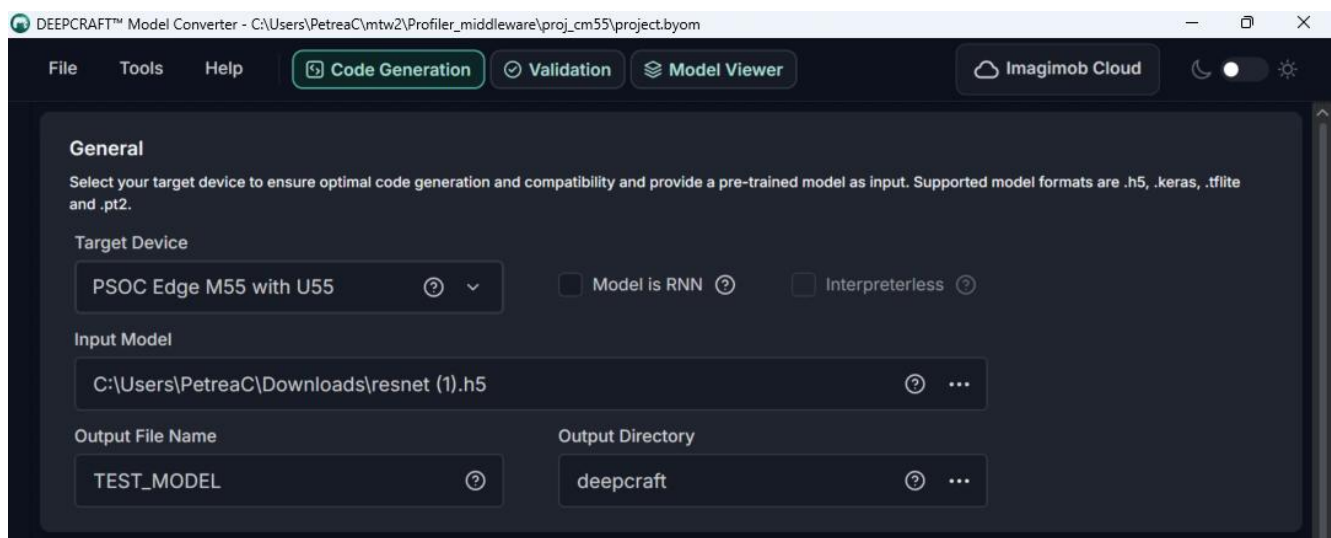
This model is available in various versions and formats, many of which are way too big to deploy on an embedded target. Therefore, the version we will utilize today takes only a 32x32x3 input, available in our the Lab_Source folder of the GitHub.

Alongside the model we will also require reference data to perform the quantization and validation steps. This data file is also found in the repo.

Download both the resnet.h5 and resnet_inference_output.npz files.

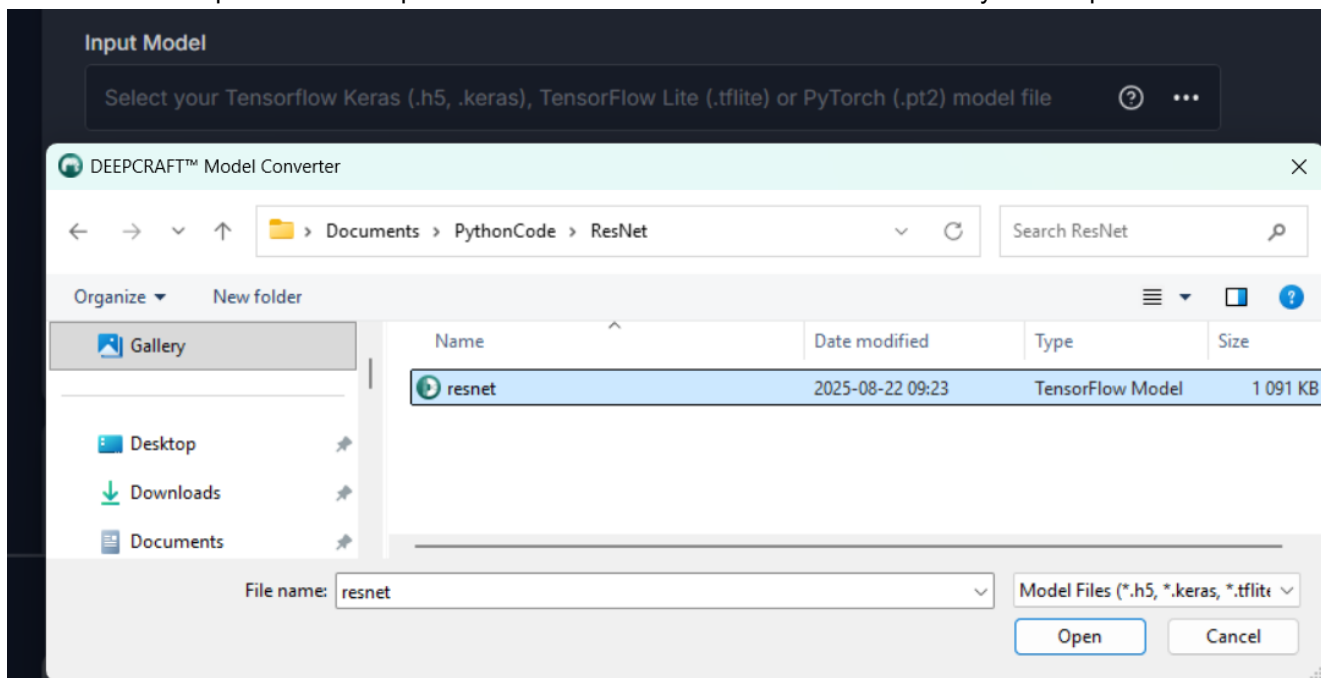
2.2 Exercise 2: General Settings of the DEEPCRAFT™ Model Converter

Open the DEEPCRAFT™ Model Converter and you will see this:



The process of generating a model is quite easy:

1. Select “Input Model” and point to the location where resnet.h5 is saved on your computer.



2. “Target Device” is the core we optimize the model for. Since we are using the PSOC™ Edge E84 we have three options: **M33 + NNLite**, **M55** or **M55 + U55**. In this training we will try various settings and deploy on both cores. Start by selecting **M55 + U55**, for code generated on the most powerful core. The generated code will consist of layers that are accelerated by the U55 and those that are not, and will then be automatically allocated to the corresponding core during runtime.

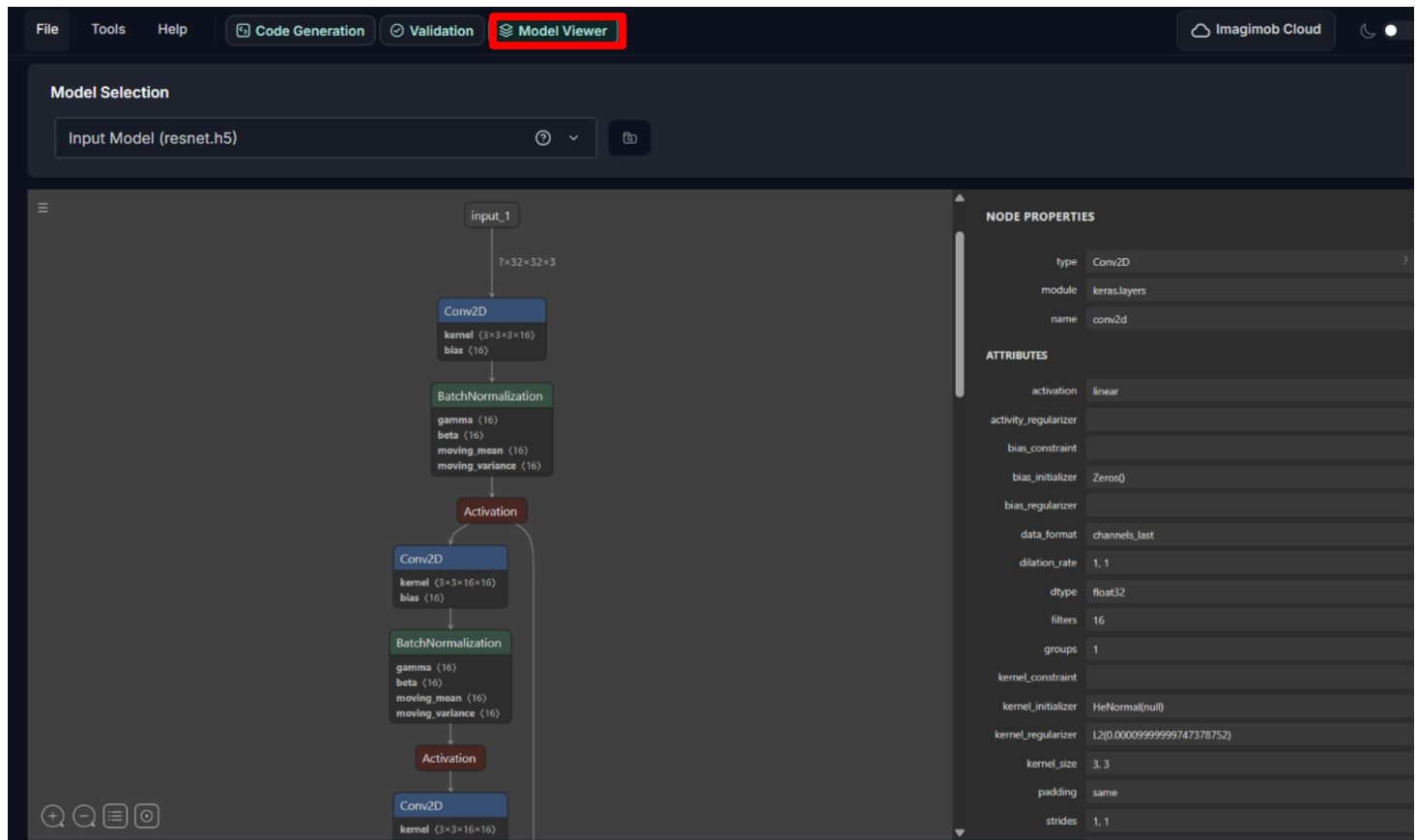
The rest of the options are optional and need not be changed for this workshop. However, here is an overview:

Option	Description
Model is RNN	This checkbox should be enabled when the input includes recurrent layers such as LSTM, GRU or similar. There is an automatic detection, but explicit setting is recommended. Doesn't affect the model conversion, instead controls how validation and calibration data are handled.
Interpreterless	Enabled: Generates static C++ code. Using Interpreterless pre-interprets the model, removing the need for LiteRT Micro interpreter during runtime. Consequently, runtime, memory planner and unused operator code are not compiled into final application. The exact memory savings varies but can be close to 200KB. Only supported on PSOC™ 6 and PSOC™ Edge M33 + NNLite.
Output File Name	Name of generated model files. Recommended to keep as “model” to align with Code Example.
Output Directory	Location of the output files.

Make sure the Output File Name is 'TEST_MODEL'.

2.3 Exercise 3: Inspecting the Model in the Model Viewer

It is important to understand what layers a model consists of in order to select settings for conversion and to undertake troubleshooting. The DEEPCRAFT™ Model Converter contains a 'Model Viewer', leveraging NETRON, which allows such visualizations of selected models. Navigate to the model viewer in the top right.



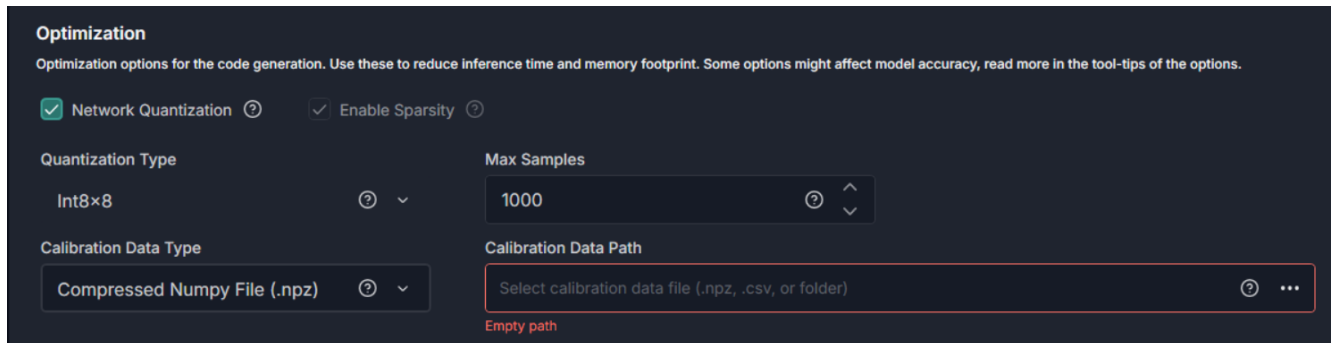
By clicking on a layer in the network there are plenty of properties which can be ascertained, including the quantization level of the weights and bias. As we can see in the Conv2D layer, the “dtype” is float32 meaning that the weights of the ResNet model are not quantized. Selecting “input_1” at the top shows us that the expected input shape of this model is 32x32x3 of float32 data. While troubleshooting, visualizing the layers and cross referencing to the supported layers outlined in Chapter 1.2 is a recommended first step if model conversion fails.

Take some time to look around at the ResNet model in the visualizer.

More details on the Model Viewer can be found [here](#).

2.4 Exercise 4: Quantizing the Model

The “Optimization” section is where you outline the desired level of quantization of the resulting model. Most commonly, models are quantized from float32 to int8x8, but int16x8 is also available. Int16x8 means the activations are Int16 and the weights are Int8. This significantly reduces memory footprint and shortens inference time but also introduces model quality degradation. Furthermore, the NPU can only accelerate int8x8 or int16x8 layers so to leverage that quantization has to be used. Later chapters will explore how to validate the performance differences due to this change.



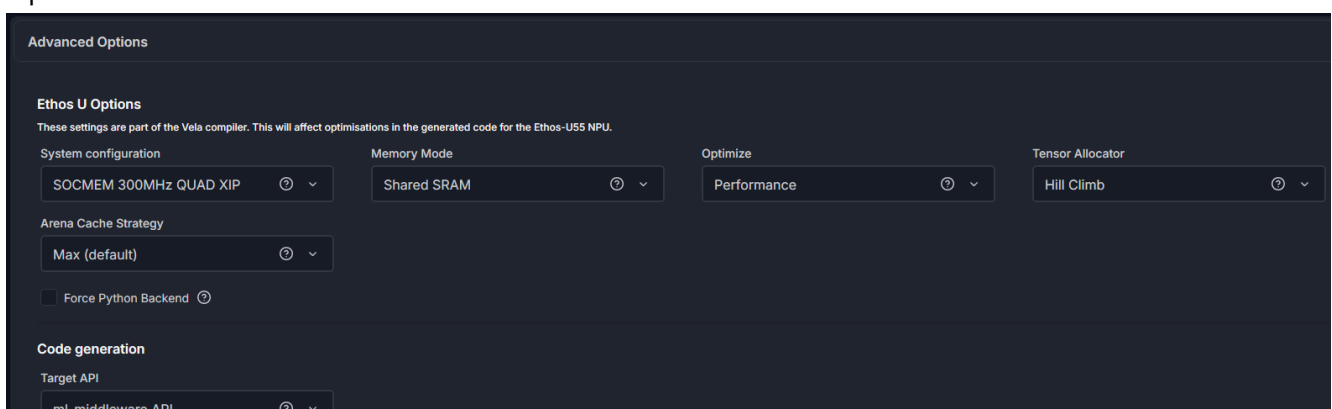
The screenshot shows the 'Optimization' section of the Vela compiler interface. It includes a header with a description: 'Optimization options for the code generation. Use these to reduce inference time and memory footprint. Some options might affect model accuracy, read more in the tool-tips of the options.' Below this are two checked checkboxes: 'Network Quantization' and 'Enable Sparsity'. There are two main sections: 'Quantization Type' and 'Max Samples'. 'Quantization Type' is set to 'Int8x8'. 'Max Samples' is set to '1000'. There are also sections for 'Calibration Data Type' (set to 'Compressed Numpy File (.npz)') and 'Calibration Data Path' (a text input field with a placeholder 'Select calibration data file (.npz, .csv, or folder)' and a red border indicating it is empty). A red text label 'Empty path' is visible below the path field.

In order to perform quantization, data is required to provide the algorithm a reference. This can be provided in the formats: Compressed Numpy File (.npz), CSV (.csv) or Recursive Folder Search. Alternatively, if no data is available and the goal is to gauge a model's inference time and memory, random data can be selected.

1. Check the Network Quantization box.
2. Select Int8x8 as Quantization Type
3. Select the .npz file 'resnet_inference_output.npz' from the GitHub repository.

2.5 Exercise 5: Advanced Options (Solved)

If the Ethos-U55 NPU is selected as target core, there are additional options available to configure from the Vela compiler in the **Advanced Options** section. These options are used to dictate how memory is allocated and shared between the Ethos-U. For this workshop, **leave them unchanged**. The following describes what each option does:



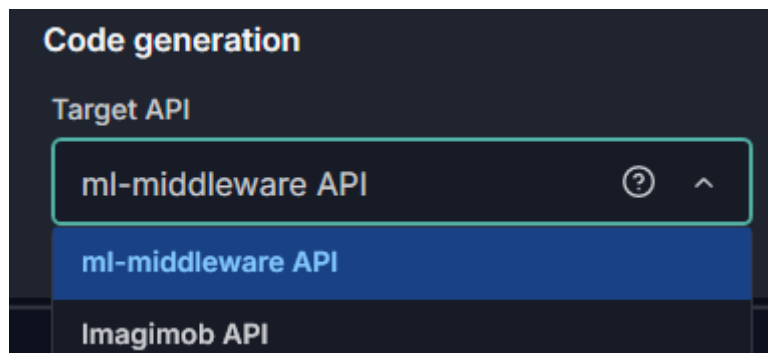
The screenshot shows the 'Advanced Options' section of the Vela compiler interface. It has a title 'Advanced Options' and a subtitle 'These settings are part of the Vela compiler. This will affect optimisations in the generated code for the Ethos-U55 NPU.' Below this are four sections: 'Ethos U Options', 'System configuration', 'Memory Mode', 'Optimize', and 'Tensor Allocator'. 'System configuration' is set to 'SOCMEM 300MHz QUAD XIP'. 'Memory Mode' is set to 'Shared SRAM'. 'Optimize' is set to 'Performance'. 'Tensor Allocator' is set to 'Hill Climb'. There is also an 'Arena Cache Strategy' section set to 'Max (default)' and a 'Force Python Backend' checkbox which is unchecked. At the bottom, there is a 'Code generation' section with a 'Target API' dropdown set to 'ml-middleware API'.

1. **System Configuration:** Select **SOCMEM 300MHz QUAD XIP** or **SOCMEM 200MHz QUAD XIP** as the system configuration of the Ethos-U NPU. Refer to Silicon documentation to know more.
2. **Memory Mode:** Select **Shared SRAM** or **SRAM Only** as the memory mode. Shared SRAM is shared between Ethos-U and Cortex-M software, whereas SRAM Only is dedicated to Ethos-U. The non-SRAM memory is considered read-only. By default, Shared SRAM is selected.
3. **Optimize:** Select the optimization strategy based on **Size** or **Performance**. The Size strategy ensures minimal RAM usage by avoiding the use of the arena cache memory area size whereas the

Performance strategy prioritizes maximal performance by utilizing the specified arena cache memory area size. By default, Performance strategy is selected.

4. **Tensor Allocator:** Select **Linear Alloc**, **Greedy** or **HillClimb** to specify the algorithm for allocating non-constant tensors on the NPU and CPU. By default, HillClimb is selected.
5. **Arena Cache Strategy:** Select the strategy to be used for determining how much arena cache is allocated for communication between the M55 and U55. Min results in the minimum memory required for the model. Max results in the maximum memory allowed by the target. Custom allows you to enter an exact specified value. Note that the arena cache is not the same as tensor arena. Tensor arena is the total memory needed by the model, not just the M55 -> U55 communication.
6. **Force Python Backend:**
Changes the backend to the deprecated Vela Python API which was used in older versions of DEEPCRAFT Studio. Activating this also reveals two new options for that API: Max Block Dependency and Recursion Limit.
7. **Recursion Limit (Force Python Backend):** Set the Python internal limit to depth of recursion. For very large networks, increasing the recursive limit may be necessary due to the recursive graph traversal algorithm. If generation fails with a RecursionError, increase the limit using this option. By default, the recursion limit is set as 1000.
8. **Max Block Dependency (Force Python Backend):** Set the maximum value that can be used for the block dependency delay between NPU kernel operations. A lower value may result in longer execution time. You may set: 0, 1, 2 or 3 as the value for NPU block dependency. By default, the value of block dependency is set as 3

More details of the options for Vela compiler can be found in [Arm's documentation](#).



Finally, the **Code Generation** option has two potential APIs. The ml-middleware API and Imagimob API which is a wrapper of the former. The wrapper doesn't do anything to models, as such the ml-middleware API is default. However, models from DEEPCRAFT™ Studio uses the Imagimob API because they include more than just the model. Learn more about the APIs [here](#).

2.6 Exercise 6: Preparing on-device Validation Data and Generating the Code

Enabling validation in the 'Code Generation' tab provides attaches a data folder to the generated output. This folder is then used in Chapter 2.7 to provide evaluation of the generated code. The optimized model is compared to the original baseline and a report summarizes how the model predictions match a provided reference output. The reference output can be provided in the same format as the quantization data: Recursive Folder Search, CSV or NPZ.

The screenshot shows the 'Validation Data' configuration window. It includes a text box for 'Max Samples' set to 1000, a dropdown for 'Validation Data Type' set to 'Compressed Numpy File (.npz)', and a text box for 'Validation Data Path' showing a file path. At the bottom, there are three radio buttons for 'Validation mode': 'Default' (selected), 'Use Original Model Reference', and 'Use Provided Reference'.

Furthermore, there are 2 modes available: 'Default' & 'Use Provided Reference'. If 'Default' is selected the validation uses the Validation Data as input to two inferences: the newly generated C code and the desktop LiteRT interpreter of the original model. These two results are then compared. If 'Use Provided Reference' is selected, there is no comparison done using the original model; instead, the comparison happens between the C code and outputs included in the Validation Data. This is useful when the dataset includes ground truth output which you want to compare to.

In our case, the 'resnet_inference_output.npz' contains a ground truth output, so we can use that.

1. Select the 'resnet_inference_output.npz' as Validation Data.
2. Select 'Use Provided Reference'
3. **Click Start to begin the Code Generation**

The screenshot shows the terminal output of the code generation process. It includes information about the generated C code path and a 'Model metrics summary' table. The table lists Data Type, Model Memory, Tensor Arena, CPU Cycles, and NPU Cycles for int8x8 data. A note at the bottom states that 'Unknown' cycle counts mean that estimation is not available for this target or configuration.

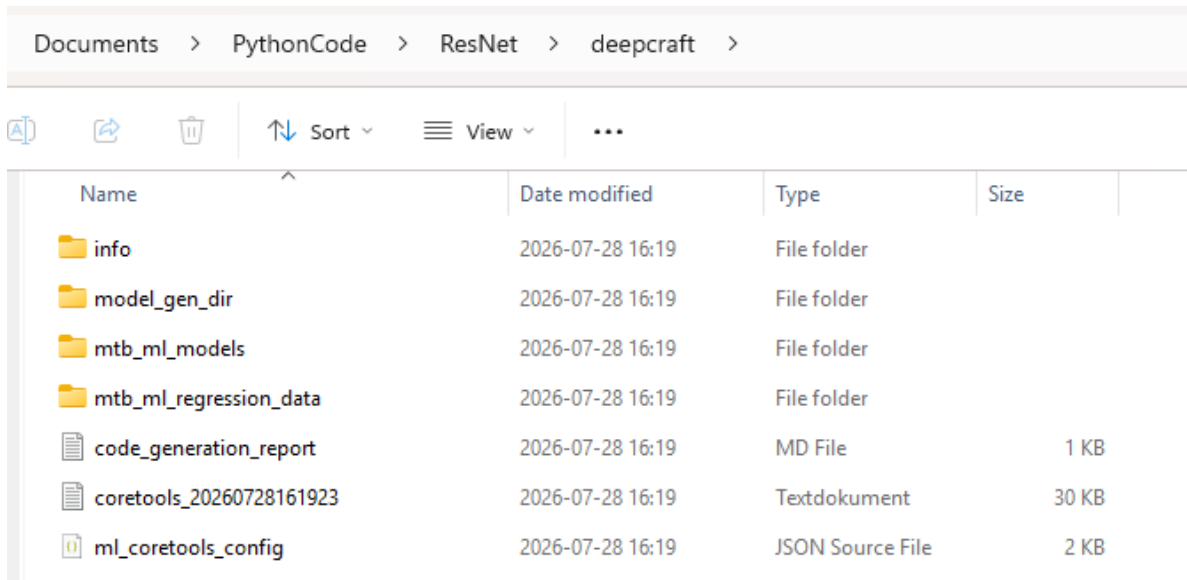
Data Type	Model Memory, bytes	Tensor Arena, bytes	CPU Cycles	NPU Cycles
int8x8	85088	57344	Unknown	291201

After the code generation, an overview of the Model Memory required, and the # of CPU/NPU Cycles required is given. In this case we see that around 85kB of Flash and 57kB of RAM memory is needed and the model requires 291201 cycles to run. We can use this to calculate an inference time by using the formula:

$$\text{Execution Time} = [\text{avg_cpu_cycles} / \text{cpu_freq_MHz}] + [\text{avg_npu_cycles} / \text{npu_freq}]$$

And since the frequency of the U55 is 400 MHz, the execution time estimate becomes $291201/400000000 = 0.000728$ or $728\mu s$. This is an estimate based only on the pure execution time of the model and not taking into account other peripherals and potential preprocessing.

The code that has been generated can be found in the output directory and consists of the model files, a code generation report and regression data based on the 'Validation' settings.



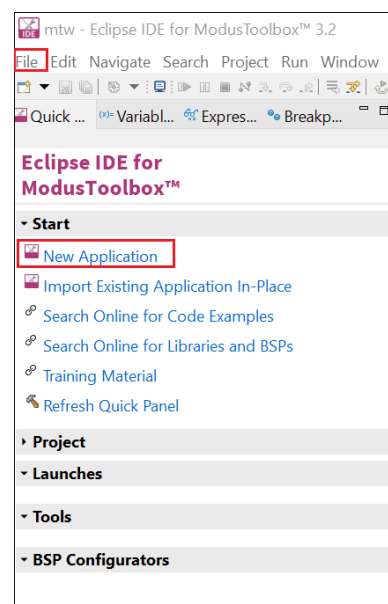
Name	Date modified	Type	Size
info	2026-07-28 16:19	File folder	
model_gen_dir	2026-07-28 16:19	File folder	
mtb_ml_models	2026-07-28 16:19	File folder	
mtb_ml_regression_data	2026-07-28 16:19	File folder	
code_generation_report	2026-07-28 16:19	MD File	1 KB
coretools_20260728161923	2026-07-28 16:19	Textdokument	30 KB
ml_coretools_config	2026-07-28 16:19	JSON Source File	2 KB

2.7 Exercise 7: Deploying & Profiling the Model on PSOC™ Edge Kit

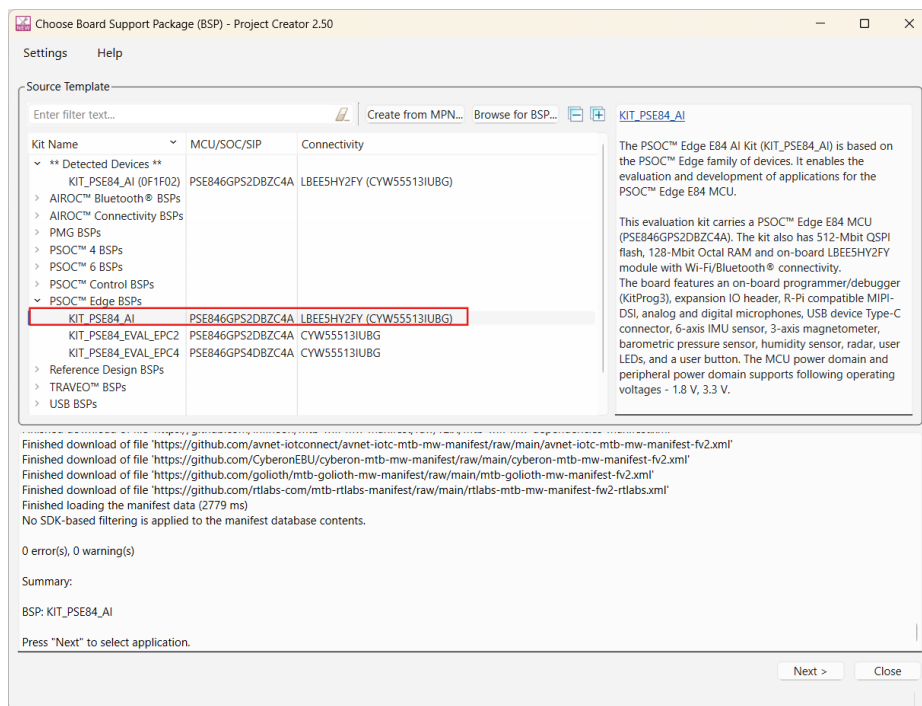
This task will guide you through creating the profiling project in ModusToolbox™ that will be needed later to deploy onto the device.

Step 1: Create the ModusToolbox™ Project

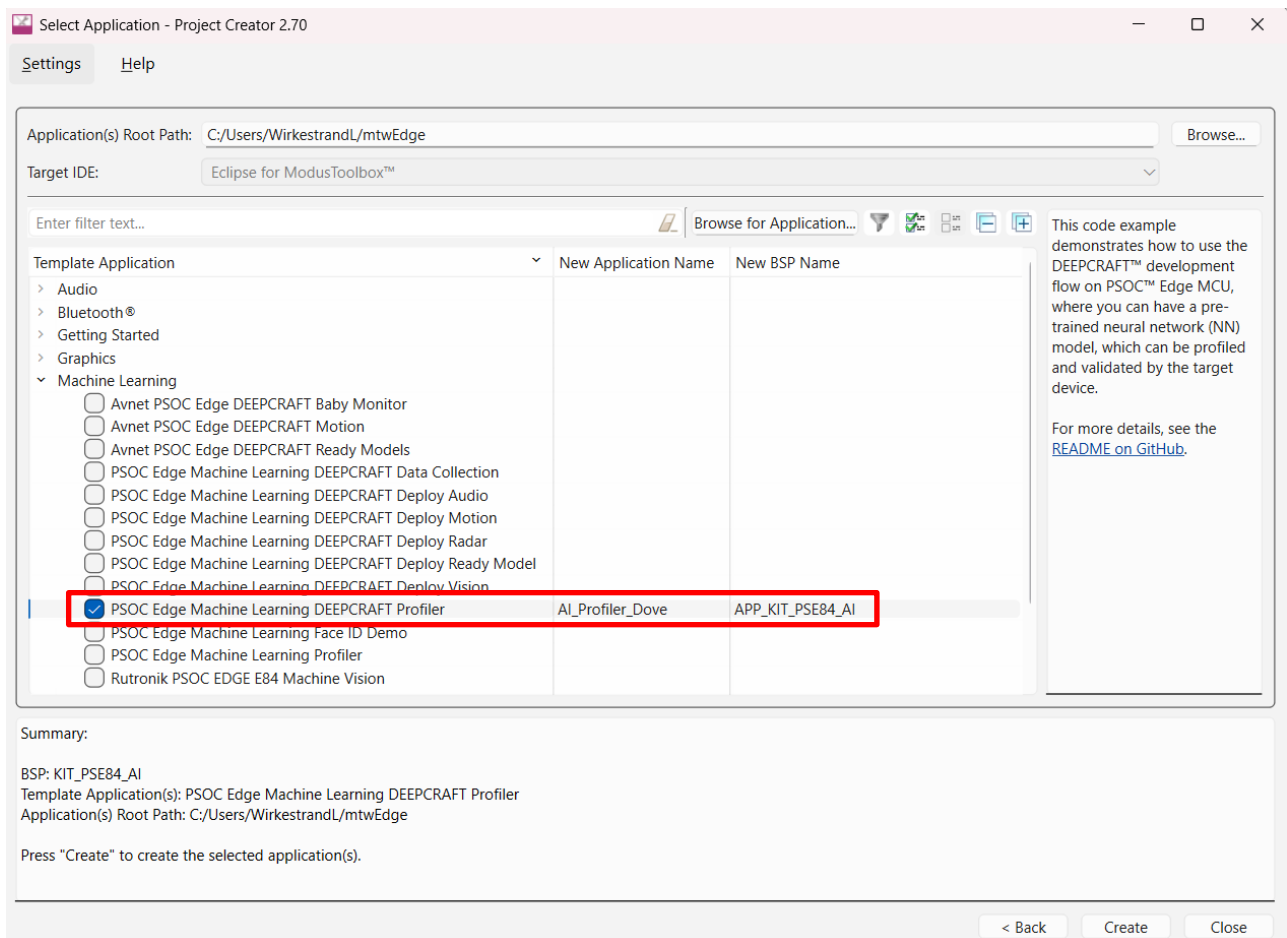
1. Open **ModusToolbox™ > Eclipse IDE for ModusToolbox™** from the **ModusToolbox™ Dashboard** software. The Eclipse IDE for ModusToolbox™ window appears.
2. Browse and select the workspace directory for your project and click **Launch** to open the ModusToolbox™ workspace.
Suggestion: Use a workspace with a short name (ex: mtb_ai) and store it close to your C:\ root folder.
3. Select **New Application** from the Quick Panel or navigate to **File> New> Modus Toolbox™ Application** to open the Project Creator Tool.



4. Expand the Kit Name PSoC™ Edge BSPs from the list and select the KIT_PSE84_AI as BSP for your board and click **Next**. The Select Application window appears.

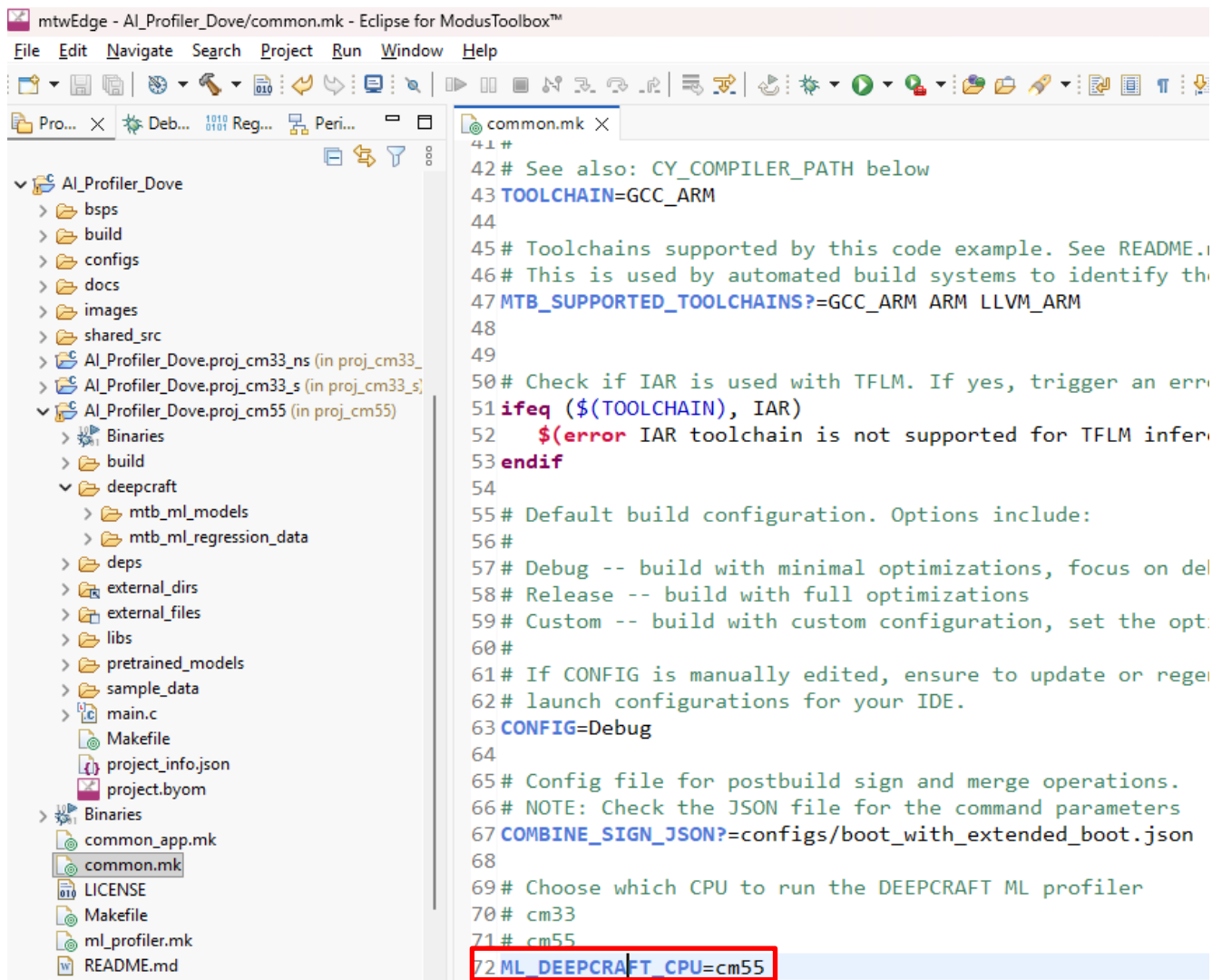


5. In **Template Application**, expand **Machine Learning** and select the **PSOC_Edge_Machine_Learning_DEEPCRAFT_Profiler** code example.



Rename it with a shorter name (ex: AI_Profiler_Dove) and click **Create**.

Creating the project takes a couple of minutes; when it is complete, expand it in the Project Explorer. There are some settings that will commonly be changed.

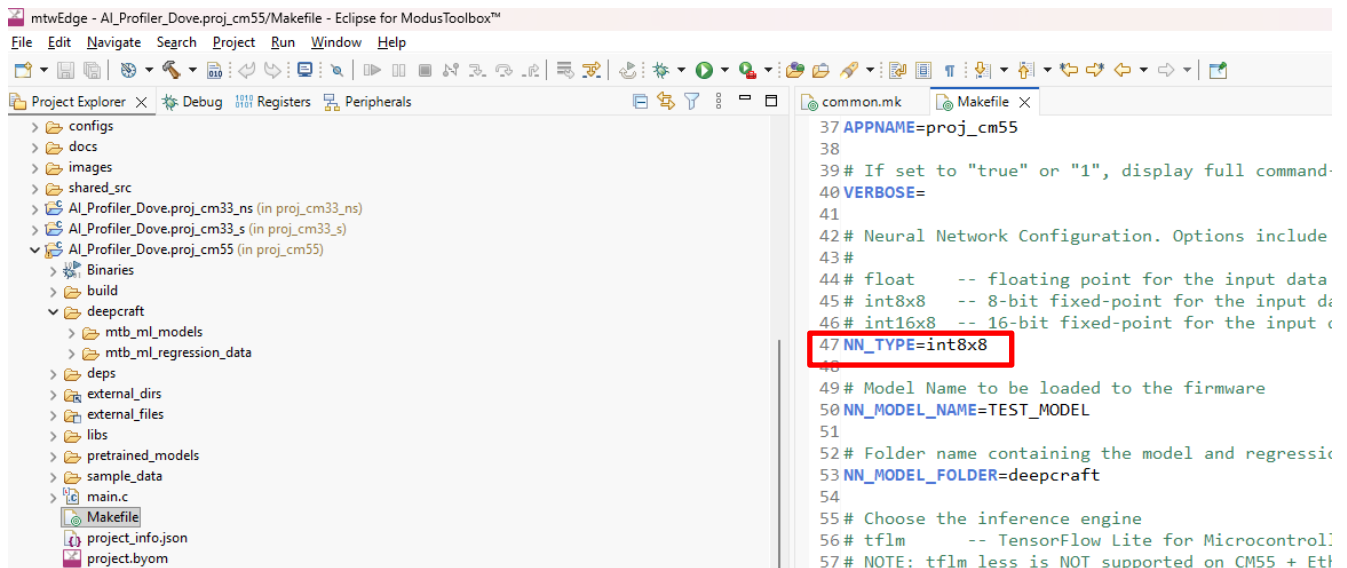


```
mtwEdge - AI_Profiler_Dove/common.mk - Eclipse for ModusToolbox™
File Edit Navigate Search Project Run Window Help

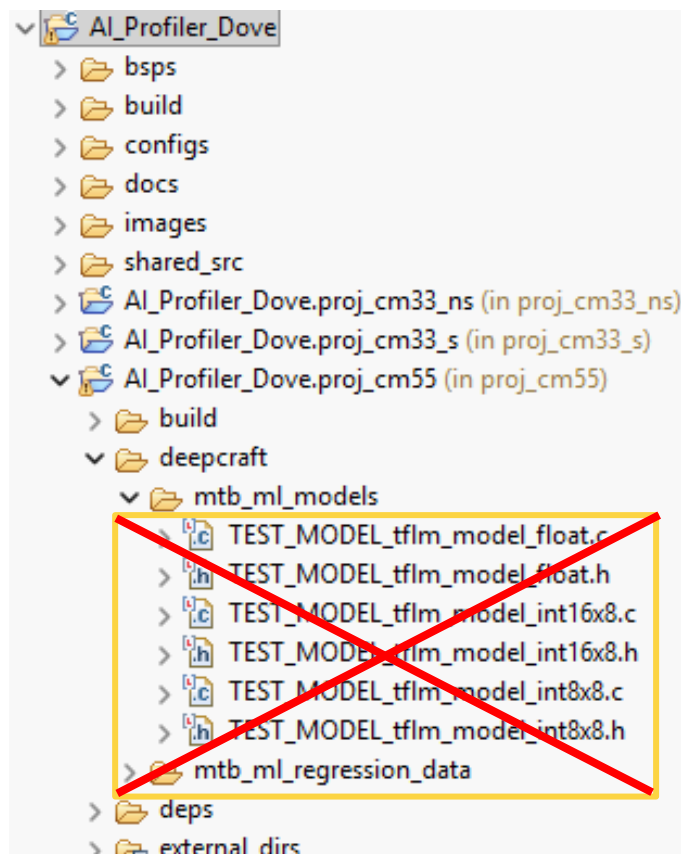
41 #
42 # See also: CY_COMPILER_PATH below
43 TOOLCHAIN=GCC_ARM
44
45 # Toolchains supported by this code example. See README.1
46 # This is used by automated build systems to identify the
47 MTB_SUPPORTED_TOOLCHAINS?=GCC_ARM ARM LLVM_ARM
48
49
50 # Check if IAR is used with TFLM. If yes, trigger an error
51 ifeq ($(TOOLCHAIN), IAR)
52     $(error IAR toolchain is not supported for TFLM inference)
53 endif
54
55 # Default build configuration. Options include:
56 #
57 # Debug -- build with minimal optimizations, focus on debug
58 # Release -- build with full optimizations
59 # Custom -- build with custom configuration, set the options
60 #
61 # If CONFIG is manually edited, ensure to update or regenerate
62 # launch configurations for your IDE.
63 CONFIG=Debug
64
65 # Config file for postbuild sign and merge operations.
66 # NOTE: Check the JSON file for the command parameters
67 COMBINE_SIGN_JSON?=configs/boot_with_extended_boot.json
68
69 # Choose which CPU to run the DEEPCRAFT ML profiler
70 # cm33
71 # cm55
72 ML_DEEPCRAFT_CPU=cm55
```

For example, ML_DEEPCRAFT_CPU in *common.mk* has to be updated when another core than the M55 is targeted for profiling.

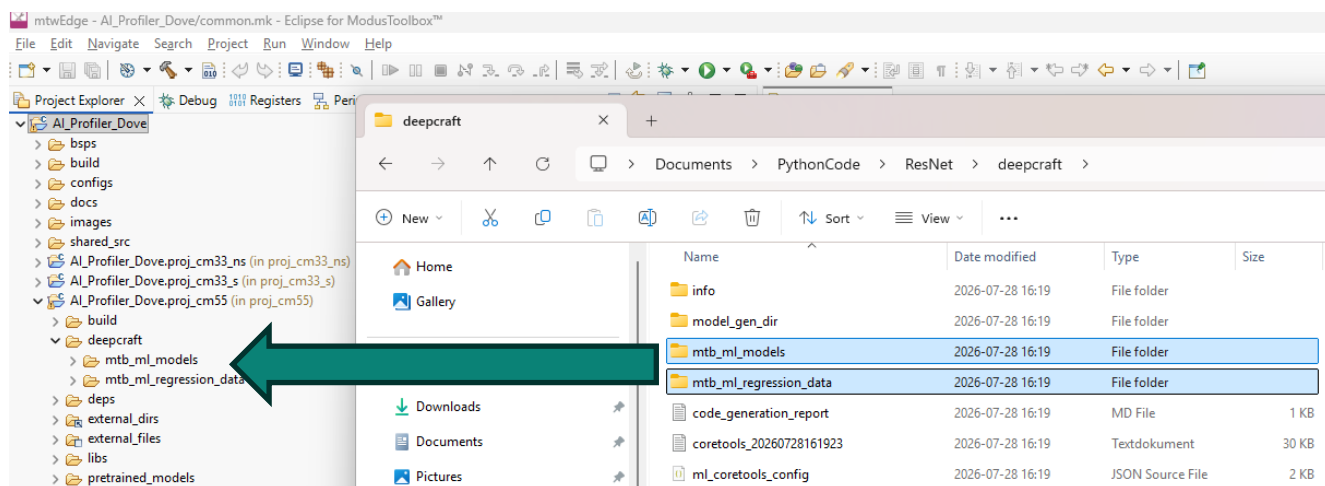
Additionally, when another quantization level than int8x8 is used another change is required in the Makefile of the core used. For example profiling with int16x8 or float32 on the M55 requires updating NN_TYPE in the makefile inside the proj_cm55 folder.



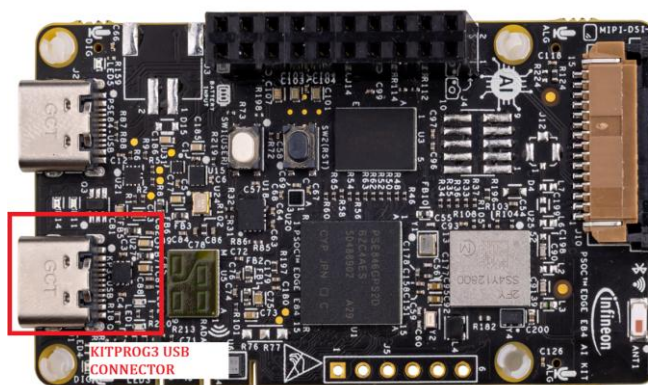
For this first deployment we will use the default settings. However, we want to use our own model, so expand the cm55 subfolder in the project and delete the existing files in the 'deepcraft' folder:



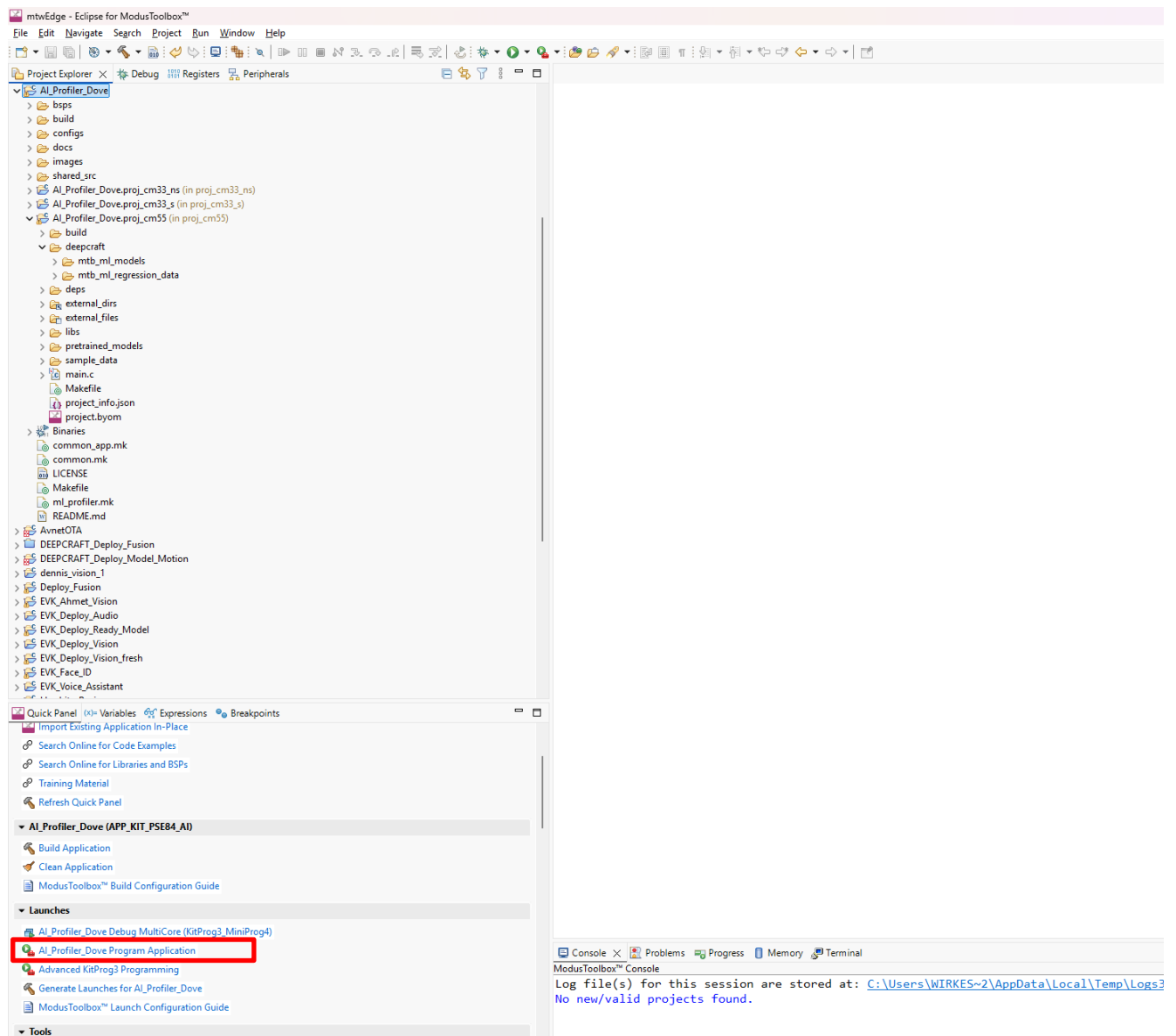
Replace these with the files generated by Model Converter by dragging and dropping them into the deepcraft folder.



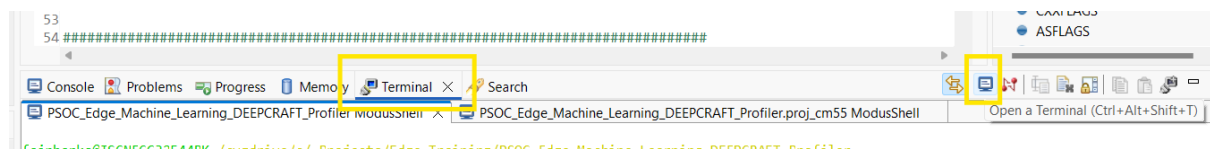
Next, we must make sure our board is connected to the computer. Connect to the J1 port, which is the KitProg3 port used for flashing the board.



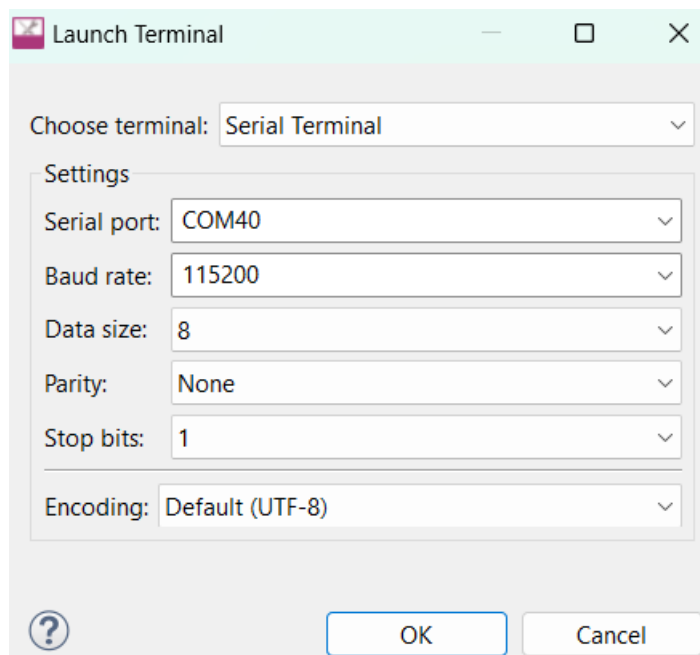
Finally, program the board using the Quick Panel in the image above.



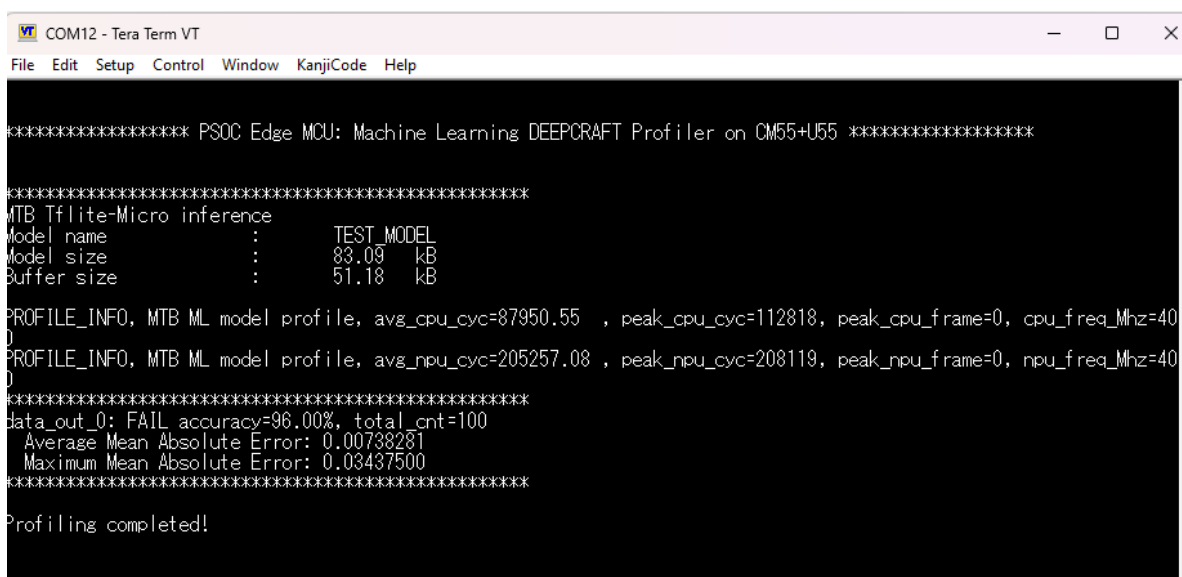
After the project is built and programmed, open a serial terminal connection. This can be done via ModusToolbox™ through the below buttons. Alternatively, it can be done via TeraTerm.



Ensure the settings are correct as per below.



From the serial terminal we can see the results of the model inference on the M55 + U55, giving us model sizes again and the results of the model inference on-device!



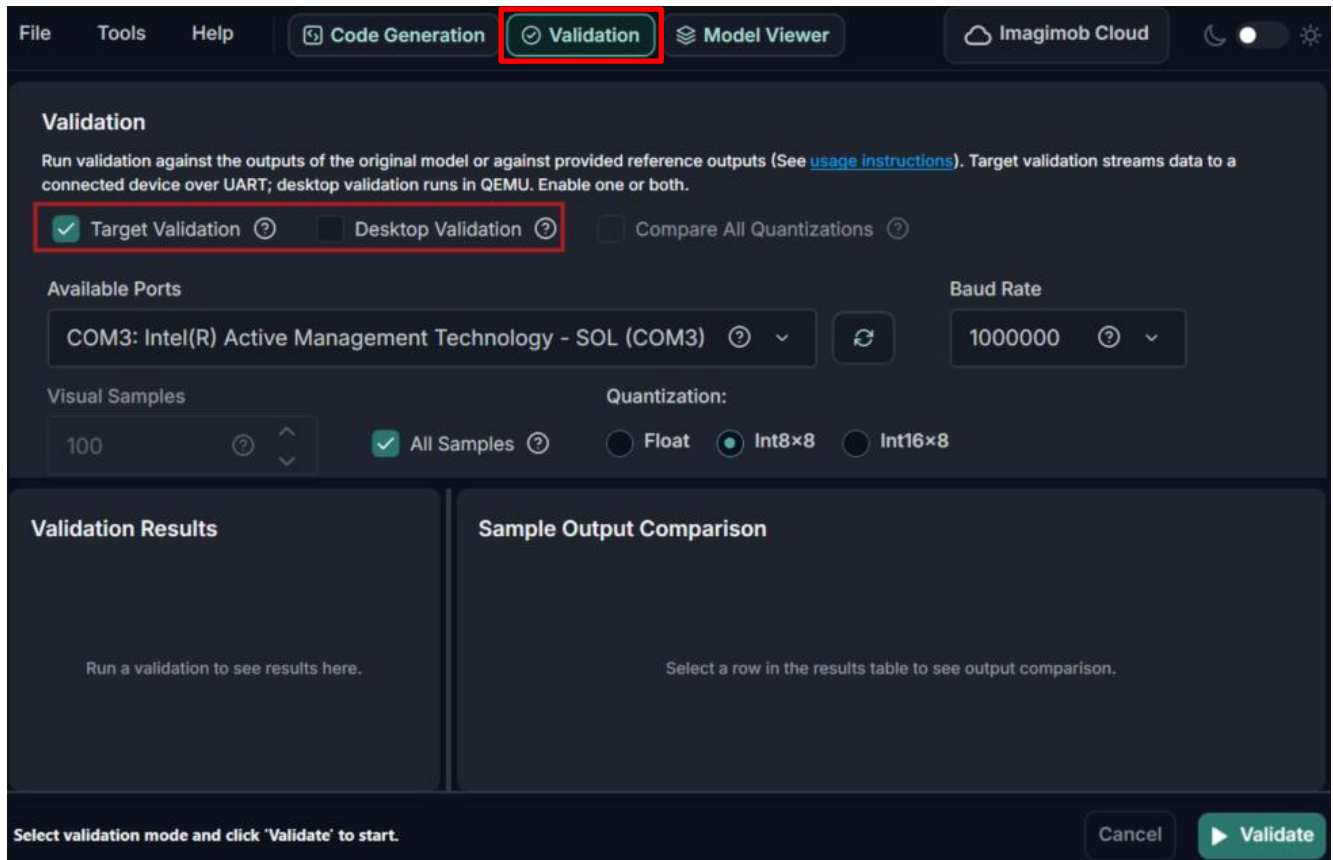
```
***** PSOC Edge MCU: Machine Learning DEEPCRAFT Profiler on CM55+U55 *****  
  
*****  
MTB Tflite-Micro inference  
Model name      :      TEST_MODEL  
Model size      :      83.09   kB  
Buffer size     :      51.18   kB  
  
PROFILE_INFO, MTB ML model profile, avg_cpu_cyc=87950.55 , peak_cpu_cyc=112818, peak_cpu_frame=0, cpu_freq_Mhz=40  
0  
PROFILE_INFO, MTB ML model profile, avg_npu_cyc=205257.08 , peak_npu_cyc=208119, peak_npu_frame=0, npu_freq_Mhz=40  
0  
*****  
data_out_0: FAIL accuracy=96.00%, total_cnt=100  
Average Mean Absolute Error: 0.00738281  
Maximum Mean Absolute Error: 0.03437500  
*****  
Profiling completed!
```

This time we see something else in addition, a relative accuracy of the new model. This is based on the settings we used for the Validation and so we see 96% relative accuracy. This means that compared to the ground truth included in the .npz data file, the newly quantized models were correct only in that many cases.

With that we have successfully ran our model on a microcontroller! What follows in this workshop are additional ways to use the Model Converter.

2.8 Exercise 8: Validating the Model in Model Converter (Optional)

At the top of the Model Converter is another whole tab dedicated to validation. Here, there are two options to run validation: **Target Validation** and **Desktop Validation**.



2.8.1 Target Validation

Target Validation streams the validation data to a connected device over UART but is limited to one quantization level at a time. One major benefit of running Target Validation is that large amounts of data, that might not be able to fit in the memory of the MCU can be used to validate the model. Additionally, it more closely resembles the data-flow that might be expected of a final deployment with data streaming in from off the board.

In order to run the **Target Validation** the target board first has to be flashed with the Code Example in the streaming validation model.

1. Update the common.mk file with `ML_VALIDATION_SOURCE = stream`.
2. Program the board again

- Return to the Model Converter and connect to the board's port. Make sure any other connections like ModusToolbox or TeraTerm are closed.

Validation

Run validation against the outputs of the original model or against provided reference outputs (See [usage instructions](#)). Target validation streams data to

☒ Target Validation ☐ Desktop Validation ☐ Compare All Quantizations

Available Ports

COM12: KitProg3 USB-UART (COM12)

Baud Rate

1000000

Visual Samples

100

Quantization:

☒ All Samples ☐ Float ☒ Int8×8 ☐ Int16×8

- Press **Validate**

After validating, detailed results are given. Each sample is sorted by their Mean Average Error (MAE) and can be clicked on to see a comparison between the models:



Additionally, a non-estimate buffer size and memory requirement for Weights & Bias are provided alongside accuracy numbers.

```
Runtime buffer size      :    51.18 kB

Weights & biases        :    83.09 kB

Accuracy for 8-bit fixed-point implementation : 96.00
```

```
Starting target validation...
2026-07-28 16:50:08.858811: I tensorflow/core/util/port.
2026-07-28 16:50:10.103339: I tensorflow/core/util/port.
[INFO]
ml-coretools 3.2.0.9646 based on TensorFlow 2.19.0 and e
ml-coretools-middleware-streaming 3.3.0.16726
ml-coretools-tflm-qemu-images 3.3.0.16726
[INFO] Base data folder: C:\Users\WirkestrandL\Documents
[INFO] Validation data: C:/Users/WirkestrandL/Documents/
[INFO] Base data folder: C:\Users\WirkestrandL\Documents
[INFO] Validation data: C:/Users/WirkestrandL/Documents/
[INFO] Deployment skipped.
[INFO] Data directory: C:\Users\WirkestrandL\Documents\P
[INFO] Running cross-domain validation:
```

Quantization	Action
float	Skip
int8x8	Validate
int16x8	Skip

Metric	Actual	Required	Status
Relative Accuracy	96	0	PASS
Mismatch Error	0.034	1.000	PASS
Scratch Memory	56	999	PASS

```
[INFO] Saving CDV results to C:\Users\WirkestrandL\Documents\Pythor
[INFO] Overall CDV Results (task: classification):
```

Quantization	Status
int8x8	PASS

```
[INFO] The cross-domain validation finished successfully.
[INFO] The log file is located at C:/Users/WirkestrandL/Documents/P
```

```
Validation on target completed successfully.
```

2.8.2 Desktop Validation (Takes 20 minutes to run with 1000 samples)

Desktop Validation runs the validation in QEMU and can compare the results of many different quantization levels. It is, however, not available for M55 + U55 so for this section we will have to generate a model optimized for another core.

1. Go back to 'Code Generation' and change Target Device to PSOC Edge M33 with NNLite
2. Return back to the 'Validation' Tab and select 'Desktop Validation' and uncheck 'Target Validation' since it cannot be combined with 'Compare All Quantizations'.
3. Tick the box for 'Compare All Quantizations' so we can obtain maximum comparisons. **This can only be used with a Keras model like .h5 or .keras.**
4. **Validate.**

This runs three different model conversions, one for each quantization level. It then performs validation of each of those 3 models and compares it to the original provided model – as such it might take a while. If you want to have it go faster, you can reduce the Max Samples to 100.

After the validation is done the results are portrayed similarly to running the Target Validation:

```
[INFO] Cross-domain validation for the int8x8 (dense):
+-----+-----+-----+-----+
| Metric | Actual | Required | Status |
+-----+-----+-----+-----+
| Relative Accuracy | 98 | 0 | PASS |
+-----+-----+-----+-----+
| Mismatch Error | 0.058 | 1.000 | PASS |
+-----+-----+-----+-----+
| Scratch Memory | 58 | 999 | PASS |
+-----+-----+-----+-----+

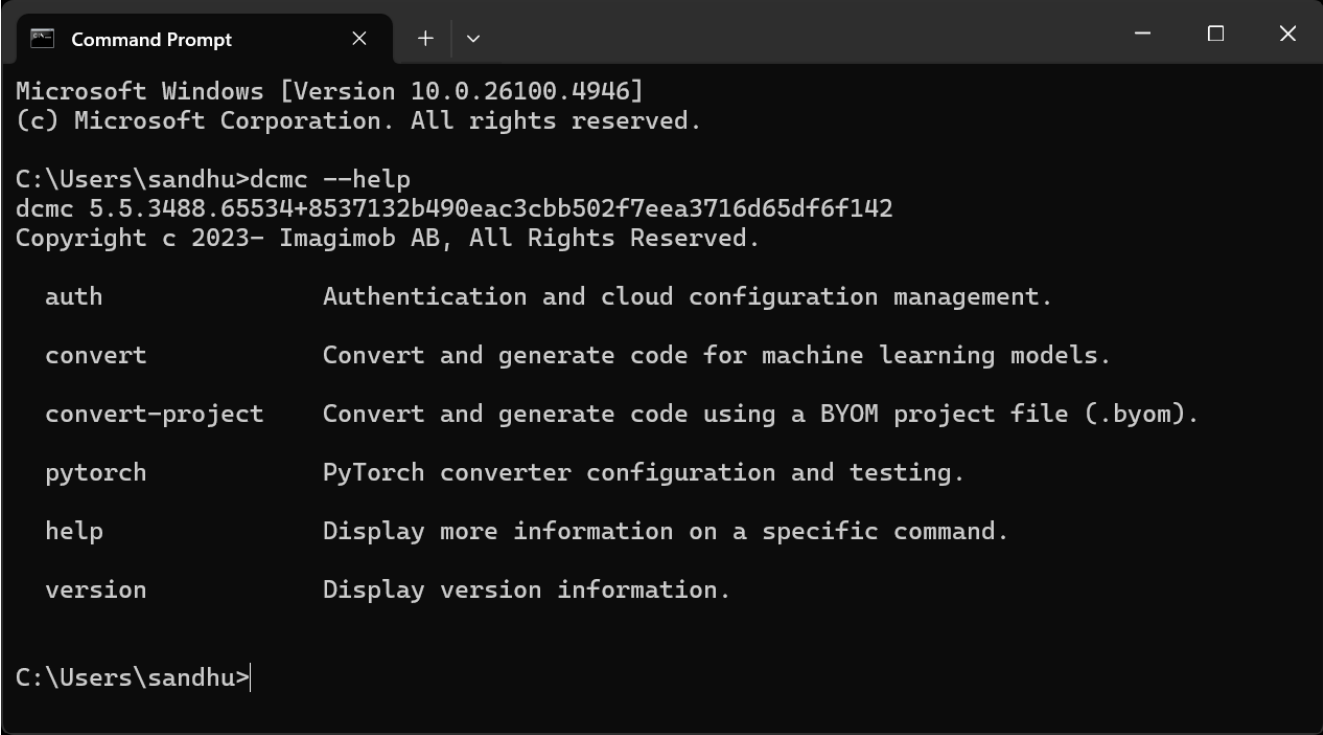
[INFO] Saving CDV results to C:\Users\WirkestrandL\Documents\Pyth
[INFO] Evaluating quantization: int16x8
[INFO] Parsing CDV file: C:\Users\WirkestrandL\Documents\PythonCo
[INFO] Output 0: loaded 100 samples with 10 values each
[INFO] Output 0: Accuracy=100.0, Max MAE=0.02092537097632885, Avg
[INFO] CPU cycles count estimation is not available for NNLite
[INFO] Cross-domain validation for the int16x8 (dense):
+-----+-----+-----+-----+
| Metric | Actual | Required | Status |
+-----+-----+-----+-----+
| Relative Accuracy | 100 | 0 | PASS |
+-----+-----+-----+-----+
| Mismatch Error | 0.021 | 1.000 | PASS |
+-----+-----+-----+-----+
| Scratch Memory | 110 | 999 | PASS |
+-----+-----+-----+-----+

[INFO] Saving CDV results to C:\Users\WirkestrandL\Documents\Pyth
[INFO] Evaluating quantization: float
[INFO] Parsing CDV file: C:\Users\WirkestrandL\Documents\PythonCo
[INFO] Output 0: loaded 100 samples with 10 values each
[INFO] Output 0: Accuracy=100.0, Max MAE=3.371154946307797e-07, A
[INFO] CPU cycles count estimation is not available for NNLite
[INFO] Cross-domain validation for the float (dense):
+-----+-----+-----+-----+
| Metric | Actual | Required | Status |
+-----+-----+-----+-----+
| Relative Accuracy | 100 | 0 | PASS |
+-----+-----+-----+-----+
| Mismatch Error | 0.000 | 1.000 | PASS |
+-----+-----+-----+-----+
| Scratch Memory | 212 | 999 | PASS |
+-----+-----+-----+-----+
```

- **Relative Accuracy:** Shows the percentage of samples where the maximum output index matches between the reference output and the target output. This metric is most useful for classification models.
- **Mismatch Error:** Shows the maximum Mean Absolute Error (MAE) across all validated samples.
- **Scratch Memory:** Shows the memory required for intermediate computations during inference in KB. This memory is also known as tensor arena or working memory.

2.9 Exercise 9: Using the Command-Line Interface (CLI) (Optional)

When Model Converter is installed, open a command prompt and type `dcmc --help` to get a list of all commands.



```
Microsoft Windows [Version 10.0.26100.4946]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sandhu>dcmc --help
dcmc 5.5.3488.65534+8537132b490eac3cbb502f7eea3716d65df6f142
Copyright c 2023- Imagimob AB, All Rights Reserved.

auth            Authentication and cloud configuration management.
convert         Convert and generate code for machine learning models.
convert-project Convert and generate code using a BYOM project file (.byom).
pytorch        PyTorch converter configuration and testing.
help           Display more information on a specific command.
version        Display version information.

C:\Users\sandhu>
```

Using the CLI you have two options, either to leverage a project file (.byom) exported from the GUI or issuing the commands manually. Refer to the Chapters above to set up the .byom file and run the code generation with: **dcmc convert-project -p <project file path>**. Open the CLI and run the command

An in-depth guide of using the CLI can be found [here](#).

3. Final Considerations and Additional Documentation

As this workshop is intended for a relatively novice audience, at least in our tools – there are plenty more advanced considerations not covered in detail here.

To optimize runtime on device there is plenty of documentation on memory allocation in ModusToolbox™: <https://www.infineon.com/design-resources/development-tools/sdk/modustoolbox-software>

Further details on deployment:

<https://developer.imagimob.com/deepcraft-model-converter/deploy-model-using-code-example>

For running the Model Converter on a PyTorch model, with or without a local container:

<https://developer.imagimob.com/deepcraft-model-converter/additional-resources/container-setup>

Common troubleshooting questions:

<https://developer.imagimob.com/deepcraft-model-converter/troubleshooting>

Validation for RNN models:

<https://developer.imagimob.com/deepcraft-model-converter/additional-resources/streaming-rnn>

Legal:

<https://developer.imagimob.com/legal>