Application Software Pack: ML-based State Monitor for FXOS8700CQ

Revision 9
January 2024

NXP Semiconductors
www.nxp.com

Contents

1 Lab Overview

This lab will cover how to use the ML-based fan state monitor application code in conjunction with the **Building and benchmarking Deep Learning models for Smart Sensing Appliances on MCUs** application note (AN13562). This document will cover how to gather the data that the model will be trained on, how to run the Jupyter notebook scripts that will train the model, and how to deploy the trained model to the board.

This lab was written for the i.MX RT1170 but can also be ran on the following boards:

- LPCXpresso55S69
- FRDM-K66F
- i.MX RT1170-EVK

This lab document is only for the FXOS8700CQ accelerometer that are on some EVK boards or if using the FRDM-STBC-AGM01 sensor board. If using the FXLS8974CF accelerometer which can be found on the ACCEL_4_CLICK board or on the FRDM-STBI-A8974 board please refer to the other lab document.

2 Software and Hardware Installation

This section will cover the steps needed to install the eIQ software and TensorFlow on your computer.

2.1 Hardware Requirements

- i.MX RT1170 EVK board (or other supported board like LPC55S69 or FRDM-K66F)
    - Some new i.MX RT1170 EVK boards do not have the accelerometer populated on U34. If this is the case, a FRDM-STBC-AGM01 board will need to be placed on the board to provide the accelerometer data required for this lab.
- microSD card (formatted for FAT32)
- microSD card reader to transfer data to a PC
- A fan, motor, or other vibration source to train the model on that can provide vibrations that will be gathered by the sensor. This fan for example.

2.2 TensorFlow Installation

1. Download and install Python 3.10. ****The 64-bit edition is required and it is highly recommended to use Python 64-bit 3.10.x to ensure compatibility with this lab****: https://www.python.org/downloads/
2. Open a Windows command prompt and verify that the python command corresponds to Python 3.10.x.
   python -V
3. Update the python installer tools:
   python -m pip install -U pip
   python -m pip install -U setuptools

4. Install the Tensorflow libraries and support for python. To ensure software compatibility, it should be these version numbers:
   python -m pip install tensorflow==2.8
   python -m pip install keras==2.8
   python -m pip install protobuf==3.20
   python -m pip install pandas==1.5.3
5. Install other useful python packages. Not all of these will be used for this lab but will be useful for other eIQ demos and scripts.
   python -m pip install numpy scipy matplotlib ipython jupyter sympy nose imageio
   python -m pip install netron seaborn west pyserial scikit-learn opencv-python pillow

2.3 Download Enablement Software

1. Install the latest version of MCUXpresso IDE
2. Install a terminal program like TeraTerm.
3. Install Git

2.4 eIQ Tool Installation

The model inference can be done with TensorFlow Lite for Microcontrollers, Glow, or DeepView RT. If using Glow or DeepViewRT, some additional PC software needs to be installed to convert the model to the necessary formats for those two inference engines.

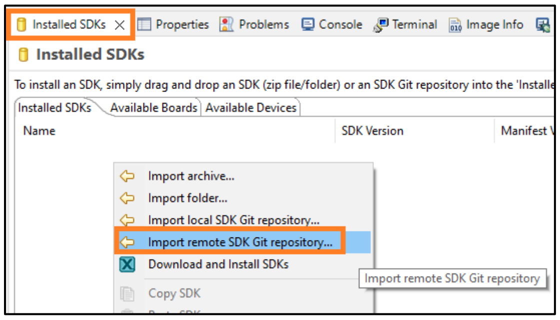
1. Install the latest Glow package. This will put the Glow compiler and helper programs into **C:\NXP\Glow** and add it to your executable path.
2. Install the latest eIQ Toolkit which will provide a tool to convert a .tflite model to the .rtm format needed for the DeepViewRT inference engine. **Note that the DeepViewRT inference engine is only available for i.MX RT Cortex-M7 devices at this time.**

3 Import the App Software Pack into MCUXpresso IDE

There are two methods to get the application software pack. This section will cover both options, but only one needs to be done. It is recommended to use the first option as it is more straight forward.

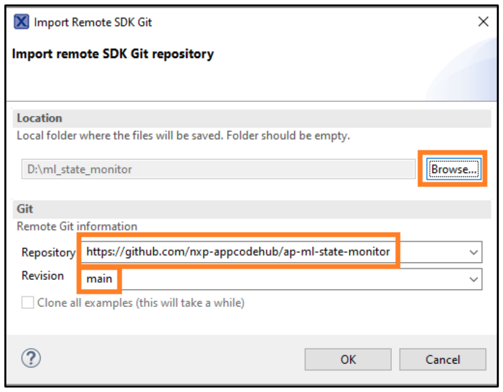
3.1 Option #1: Get the App Software Pack with MCUXpresso IDE

1. Open MCUXpresso IDE and select a workspace location in an empty directory.
2. Right click in the blank area of the **Installed SDKs** panel at the bottom and select **Import remote SDK Git repository…**


To install an SDK, simply drag and drop an SDK (zip file/folder) or an SDK Git repository into the 'Installe

3. In the dialog box that comes up:
   a) In the **Location** field, click on the Browse button and select or create an empty directory for the application software pack to be downloaded to. Make note of this location as it'll be used throughout this lab.
   b) In the **Repository** field put: **https://github.com/nxp-appcodehub/ap-ml-state-monitor**
   c) In the **Revision** field put: **main**

   Then hit OK to download the application software pack



Import Remote SDK Git

Import remote SDK Git repository

Location

Local folder where the files will be saved. Folder should be empty.

D:\ml_state_monitor

Git

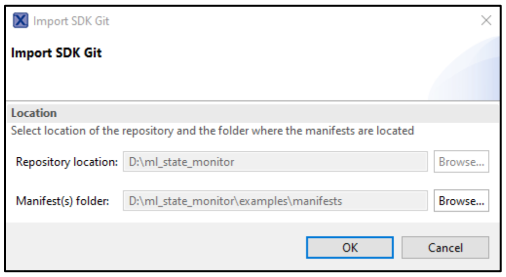
Remote Git information

Repository: https://github.com/nxp-appcodehub/ap-ml-state-monitor

Revision: main

Clone all examples (this will take a while)

4. A dialog box will then come up asking for the location of the Manifests folder. Make sure to point it to the **<repo_directory>\examples\manifests** folder if it is not already.



Import SDK Git

Import SDK Git

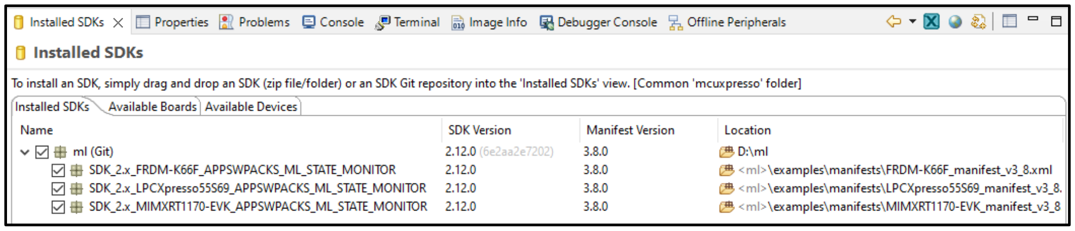
Location

Select location of the repository and the folder where the manifests are located

Repository location: D:\ml_state_monitor

Manifest(s) folder: D:\ml_state_monitor\examples\manifests

5. Once imported the Installed SDKs tab will look like this:



Installed SDKs

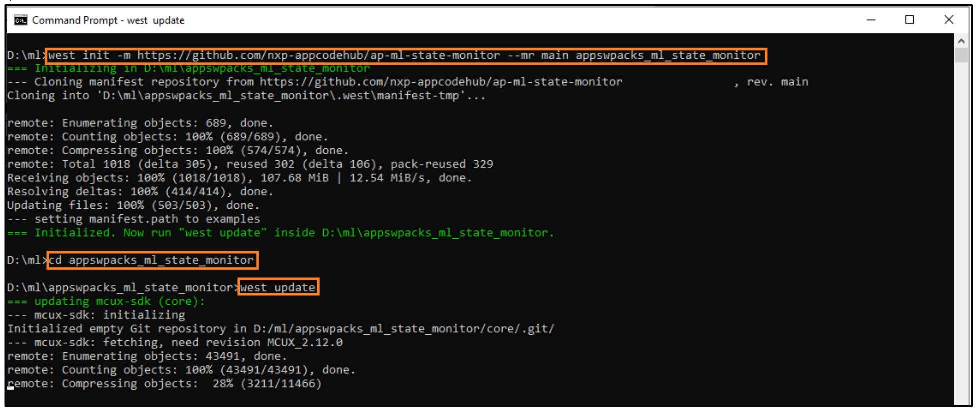
To install an SDK, simply drag and drop an SDK (zip file/folder) or an SDK Git repository into the 'Installed SDKs' view. [Common 'mcuxpresso' folder]

Name	SDK Version	Manifest Version	Location
✓ ml (Git)	2.12.0 (6e2aa2e7202)	3.8.0	D:\ml
✓ SDK_2.x_FRDM-K66F_APPSWPACKS_ML_STATE_MONITOR	2.12.0	3.8.0	<ml>\examples\manifests\FRDM-K66F_manifest_v3_8.xml
✓ SDK_2.x_LPCXpresso55S69_APPSWPACKS_ML_STATE_MONITOR	2.12.0	3.8.0	<ml>\examples\manifests\LPCXpresso55S69_manifest_v3_8.
✓ SDK_2.x_MIMXRT1170-EVK_APPSWPACKS_ML_STATE_MONITOR	2.12.0	3.8.0	<ml>\examples\manifests\MIMXRT1170-EVK_manifest_v3_8

3.2 Option #2: Use the Command Line

1. Open up a Windows command line and execute the following lines:

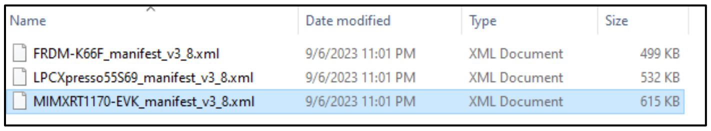
```
west init -m https://github.com/nxp-appcodehub/ap-ml-state-monitor --mr main appswpacks_ml_state_monitor
cd appswpacks_ml_state_monitor
west update
```



Command Prompt - west update

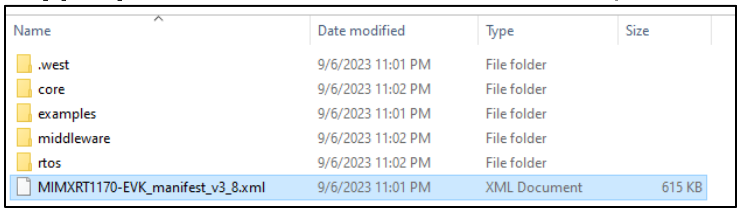
```
D:\ml>west init -m https://github.com/nxp-appcodehub/ap-ml-state-monitor --mr main appswpacks_ml_state_monitor
=== Initializing in D:\ml\appswpacks_ml_state_monitor
--- Cloning manifest repository from https://github.com/nxp-appcodehub/ap-ml-state-monitor            , rev. main
Cloning into 'D:\ml\appswpacks_ml_state_monitor\.west\manifest-tmp'...

remote: Enumerating objects: 689, done.
remote: Counting objects: 100% (689/689), done.
remote: Compressing objects: 100% (574/574), done.
remote: Total 1018 (delta 305), reused 302 (delta 106), pack-reused 329
Receiving objects: 100% (1018/1018), 107.68 MiB | 12.54 MiB/s, done.
Resolving deltas: 100% (414/414), done.
Updating files: 100% (503/503), done.
--- setting manifest.path to examples
=== Initialized. Now run "west update" inside D:\ml\appswpacks_ml_state_monitor.

D:\ml>cd appswpacks_ml_state_monitor

D:\ml\appswpacks_ml_state_monitor>west update
=== updating mcux-sdk (core):
--- mcux-sdk: initializing
Initialized empty Git repository in D:/ml/appswpacks_ml_state_monitor/core/.git/
--- mcux-sdk: fetching, need revision MCUX_2.12.0
remote: Enumerating objects: 43491, done.
remote: Counting objects: 100% (43491/43491), done.
remote: Compressing objects:  28% (3211/11466)
```

2. Go to the **\appswpacks_ml_state_monitor\examples\manifests** folder and find the **MIMXRT1170-EVK_manifest_v3_8.xml** file. If using a different board copy that board's .xml file instead.



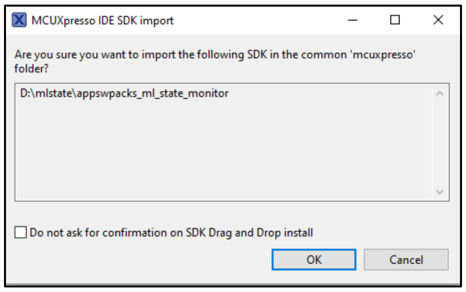
Name	Date modified	Type	Size
FRDM-K66F_manifest_v3_8.xml	9/6/2023 11:01 PM	XML Document	499 KB
LPCXpresso55S69_manifest_v3_8.xml	9/6/2023 11:01 PM	XML Document	532 KB
MIMXRT1170-EVK_manifest_v3_8.xml	9/6/2023 11:01 PM	XML Document	615 KB

3. Copy that **MIMXRT1170-EVK_manifest_v3_8.xml** file into the root **\appswpacks_ml_state_monitor** directory. It should look like the following when done:



Name	Date modified	Type	Size
.west	9/6/2023 11:01 PM	File folder	
core	9/6/2023 11:02 PM	File folder	
examples	9/6/2023 11:01 PM	File folder	
middleware	9/6/2023 11:02 PM	File folder	
rtos	9/6/2023 11:02 PM	File folder	
MIMXRT1170-EVK_manifest_v3_8.xml	9/6/2023 11:01 PM	XML Document	615 KB

6. Next open MCUXpresso IDE and select a workspace location in an empty directory.
7. Drag-and-drop the \appswpacks_ml_state_monitor directory that was created in the previous step into the **Installed SDKs** window, located on a tab at the bottom of the screen named "Installed SDKs". You will get the following pop-up, so hit OK.



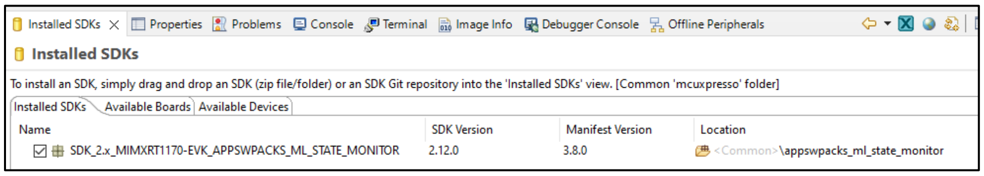
MCUXpresso IDE SDK import

Are you sure you want to import the following SDK in the common 'mcuxpresso' folder?

D:\mlstate\appswpacks_ml_state_monitor

☐ Do not ask for confirmation on SDK Drag and Drop install

8. Once imported, the Installed SDK panel will look something like this



Installed SDKs

To install an SDK, simply drag and drop an SDK (zip file/folder) or an SDK Git repository into the 'Installed SDKs' view. [Common 'mcuxpresso' folder]

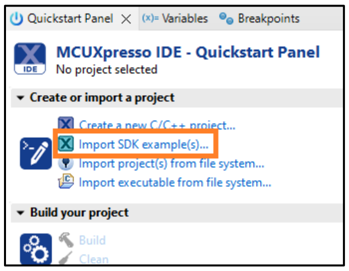
Name	SDK Version	Manifest Version	Location
☑ SDK_2.x_MIMXRT1170-EVK_APPSWPACKS_ML_STATE_MONITOR	2.12.0	3.8.0	<Common>\appswpacks_ml_state_monitor

4 Gather the Data

To train a model, sensor data must be collected. For this lab the on-board accelerometer will be used and the data that is collected will be stored on a microSD card.

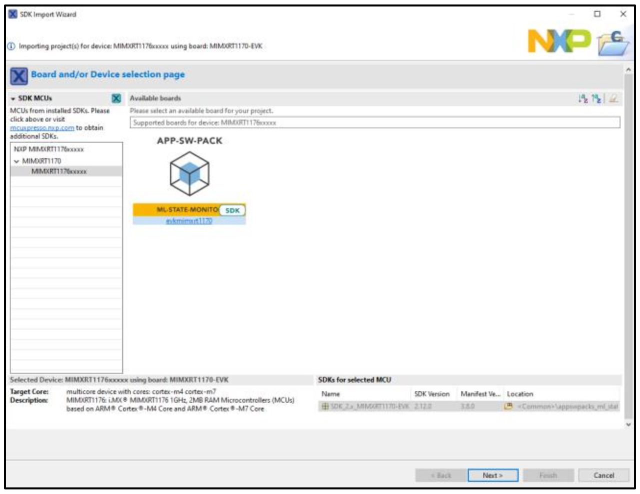
4.1 Import the ML-based System State Monitor project

1. In the Quickstart Panel in the lower left corner, click on **Import SDK examples(s)…**



Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
No project selected
Create or import a project
Create a new C/C++ project...
Import SDK example(s)...
Import project(s) from file system...
Import executable from file system...
Build your project
Build
Clean

2. Select the **APP-SW-PACK ML-State-Monitor** for evkmimxrt1170 and click on **Next**


SDK Import Wizard

Importing project(s) for device: MIMXRT1176xxxxx using board: MIMXRT1170-EVK

Board and/or Device selection page

SDK MCUs — MCUs from installed SDKs. Please click above or visit mcuxpresso.nxp.com to obtain additional SDKs.

NXP MIMXRT1176xxxxx
MIMXRT1170
MIMXRT1176xxxxx

Available boards — Please select an available board for your project.

Supported boards for device: MIMXRT1176xxxxx

APP-SW-PACK
ML-STATE-MONITO SDK
evkmimxrt1170

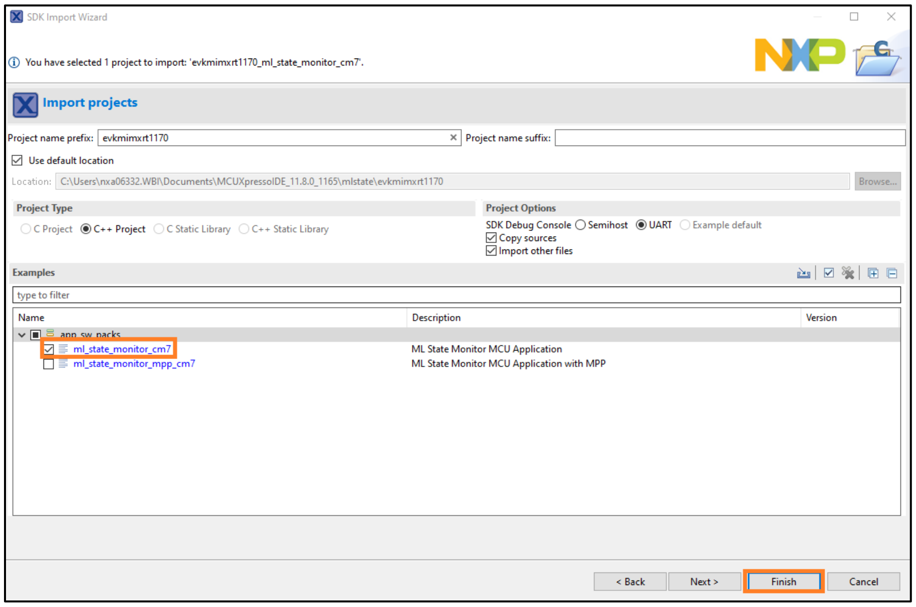
Selected Device: MIMXRT1176xxxxx using board: MIMXRT1170-EVK

Target Core: multicore device with cores: cortex-m4 cortex-m7
Description: MIMXRT1176: i.MX® MIMXRT1176 1GHz, 2MB RAM Microcontrollers (MCUs) based on ARM® Cortex®-M4 Core and ARM® Cortex®-M7 Core

SDKs for selected MCU

Name	SDK Version	Manifest Ve...	Location
SDK_2.x_MIMXRT1170-EVK	2.12.0	3.8.0	<Common>\appswpacks_ml_stat

3. Expand the **app_sw_packs** category and select the **ml_state_monitor_cm7** example which is compatible with the accelerometer on the i.MX RT1170 EVK or the FRDM-STBC-AGM01 sensor board. Click on **Finish**.


SDK Import Wizard

You have selected 1 project to import: 'evkmimxrt1170_ml_state_monitor_cm7'.

Import projects

Project name prefix: evkmimxrt1170
Project name suffix:

☑ Use default location

Location: C:\Users\nxa06332.WBI\Documents\MCUXpressoIDE_11.8.0_1165\mlstate\evkmimxrt1170

Project Type: C Project [✓ C++ Project] C Static Library C++ Static Library

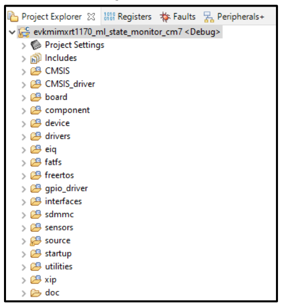
Project Options:
SDK Debug Console: Semihost [✓ UART] Example default
☑ Copy sources
☑ Import other files

Examples

type to filter

Name	Description	Version
app_sw_packs		
☑ ml_state_monitor_cm7	ML State Monitor MCU Application	
☐ ml_state_monitor_mpp_cm7	ML State Monitor MCU Application with MPP	

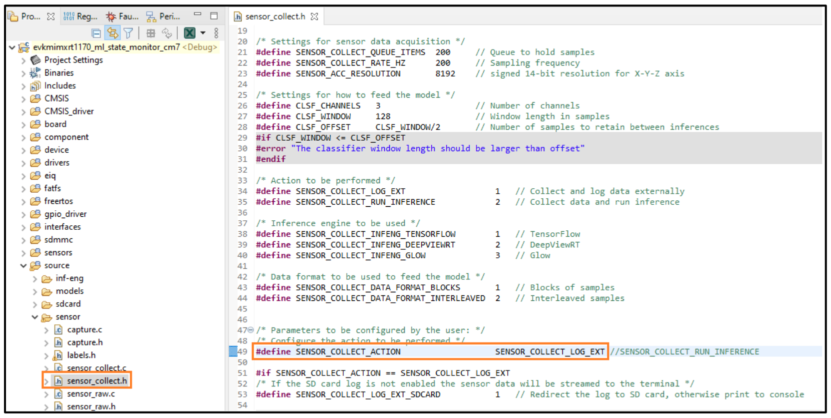
4. Once imported it should look like the following:



4.2 Collect data

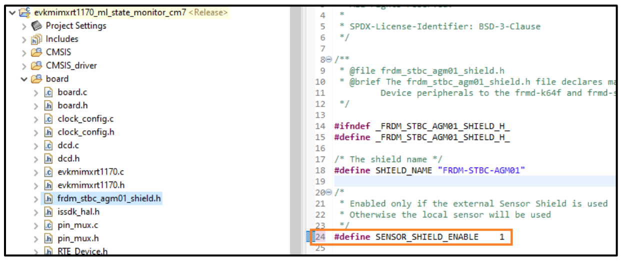
The project will need to be modified to collect accelerometer data and store it on an SD card.

1. Put in a microSD card into the board and set the fan on top of the board. Also make sure nothing is in the M.2 slot in the board as it shares data lines with the SD card.
2. In MCUXpresso IDE, open the **sensor_collect.h** file and on line 49 change the #define for **SENSOR_COLLECT_ACTION** to **SENSOR_COLLECT_LOG_EXT** as shown in the image below.



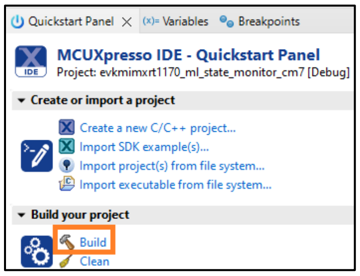
```
19
20 /* Settings for sensor data acquisition */
21 #define SENSOR_COLLECT_QUEUE_ITEMS  200     // Queue to hold samples
22 #define SENSOR_COLLECT_RATE_HZ      200     // Sampling frequency
23 #define SENSOR_ACC_RESOLUTION       8192    // signed 14-bit resolution for X-Y-Z axis
24
25 /* Settings for how to feed the model */
26 #define CLSF_CHANNELS   3                   // Number of channels
27 #define CLSF_WINDOW     128                 // Window length in samples
28 #define CLSF_OFFSET     CLSF_WINDOW/2       // Number of samples to retain between inferences
29 #if CLSF_WINDOW <= CLSF_OFFSET
30 #error "The classifier window length should be larger than offset"
31 #endif
32
33 /* Action to be performed */
34 #define SENSOR_COLLECT_LOG_EXT                  1   // Collect and log data externally
35 #define SENSOR_COLLECT_RUN_INFERENCE            2   // Collect data and run inference
36
37 /* Inference engine to be used */
38 #define SENSOR_COLLECT_INFENG_TENSORFLOW        1   // TensorFlow
39 #define SENSOR_COLLECT_INFENG_DEEPVIEWRT        2   // DeepViewRT
40 #define SENSOR_COLLECT_INFENG_GLOW              3   // Glow
41
42 /* Data format to be used to feed the model */
43 #define SENSOR_COLLECT_DATA_FORMAT_BLOCKS       1   // Blocks of samples
44 #define SENSOR_COLLECT_DATA_FORMAT_INTERLEAVED  2   // Interleaved samples
45
46
47 /* Parameters to be configured by the user: */
48 /* Configure the action to be performed */
49 #define SENSOR_COLLECT_ACTION                   SENSOR_COLLECT_LOG_EXT //SENSOR_COLLECT_RUN_INFERENCE
50
51 #if SENSOR_COLLECT_ACTION == SENSOR_COLLECT_LOG_EXT
52 /* If the SD card log is not enabled the sensor data will be streamed to the terminal */
53 #define SENSOR_COLLECT_LOG_EXT_SDCARD           1   // Redirect the log to SD card, otherwise print to console
54
```

3. If using the FRDM-STBC-AGM01 add-on board, open up the **frdm_stbc_agm01_shield.h** file in the board directory, and change the **#define SENSOR_SHIELD_ENABLE** to **1**

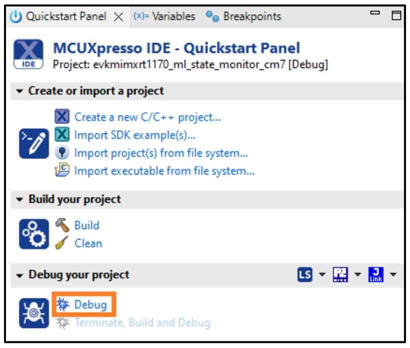


```
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

/**
 * @file frdm_stbc_agm01_shield.h
 * @brief The frdm_stbc_agm01_shield.h file declares ma
        Device peripherals to the frmd-k64f and frmd-
 */

#ifndef _FRDM_STBC_AGM01_SHIELD_H_
#define _FRDM_STBC_AGM01_SHIELD_H_

/* The shield name */
#define SHIELD_NAME "FRDM-STBC-AGM01"

/*
 * Enabled only if the external Sensor Shield is used
 * Otherwise the local sensor will be used
 */
#define SENSOR_SHIELD_ENABLE    1
```

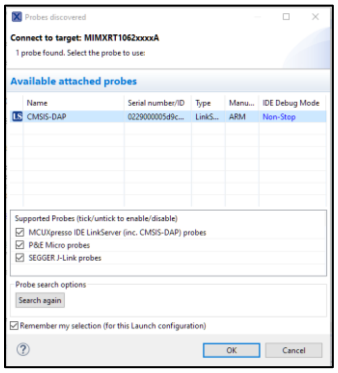
4. Next build the project by clicking on "Build" in the Quickstart Panel and make sure there are no errors.


Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
Project: evkmimxrt1170_ml_state_monitor_cm7 [Debug]
Create or import a project
Create a new C/C++ project...
Import SDK example(s)...
Import project(s) from file system...
Import executable from file system...
Build your project
Build
Clean

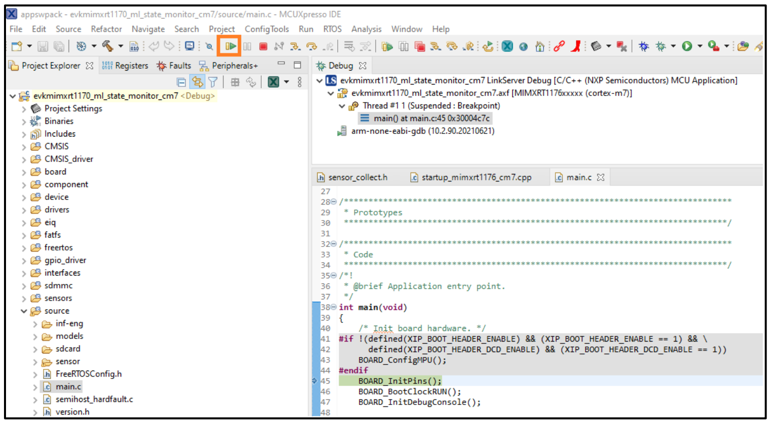
4. Plug the micro-B USB cable into the board at J11 on the i.MXRT1170 board.
5. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
6. Debug the project by clicking on "Debug" in the Quickstart Panel.


Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
Project: evkmimxrt1170_ml_state_monitor_cm7 [Debug]
Create or import a project
Create a new C/C++ project...
Import SDK example(s)...
Import project(s) from file system...
Import executable from file system...
Build your project
Build
Clean
Debug your project
Debug
Terminate, Build and Debug

7. It will ask what interface to use. Select CMSIS-DAP.

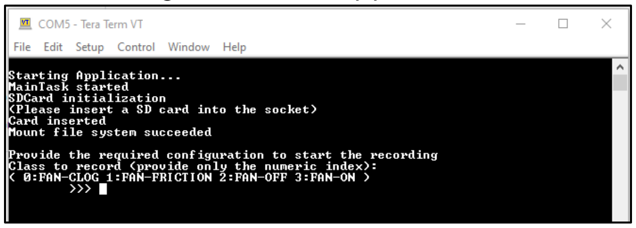

Probes discovered
Connect to target: MIMXRT1062xxxxA
1 probe found. Select the probe to use:
Available attached probes
Name
Serial number/ID
Type
Manu...
IDE Debug Mode
CMSIS-DAP
0229000005d9c...
LinkS...
ARM
Non-Stop
Supported Probes (tick/untick to enable/disable)
MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
P&E Micro probes
SEGGER J-Link probes
Probe search options
Search again
Remember my selection (for this Launch configuration)
OK
Cancel

8. The debugger will download the firmware and open up the debug view. Click on the Resume button to start running.


appswpack - evkmimxrt1170_ml_state_monitor_cm7/source/main.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run RTOS Analysis Window Help
Project Explorer
Registers
Faults
Peripherals+
Debug
evkmimxrt1170_ml_state_monitor_cm7 <Debug>
Project Settings
Binaries
Includes
CMSIS
CMSIS_driver
board
component
device
drivers
eiq
fatfs
freertos
gpio_driver
interfaces
sdmmc
sensors
source
inf-eng
models
sdcard
sensor
FreeRTOSConfig.h
main.c
semihost_hardfault.c
version.h
evkmimxrt1170_ml_state_monitor_cm7 LinkServer Debug [C/C++ (NXP Semiconductors) MCU Application]
evkmimxrt1170_ml_state_monitor_cm7.axf [MIMXRT1176xxxxx (cortex-m7)]
Thread #1 1 (Suspended : Breakpoint)
main() at main.c:45 0x30004c7c
arm-none-eabi-gdb (10.2.90.20210621)
sensor_collect.h
startup_mimxrt1176_cm7.cpp
main.c

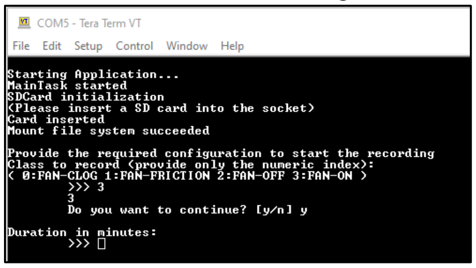
```
/*******************************************************************************
 * Prototypes
 ******************************************************************************/

/*******************************************************************************
 * Code
 ******************************************************************************/
/*!
 * @brief Application entry point.
 */
int main(void)
{
    /* Init board hardware. */
#if !(defined(XIP_BOOT_HEADER_ENABLE) && (XIP_BOOT_HEADER_ENABLE == 1) && \
      defined(XIP_BOOT_HEADER_DCD_ENABLE) && (XIP_BOOT_HEADER_DCD_ENABLE == 1))
    BOARD_ConfigMPU();
#endif
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
```

9. The following text should appear in the terminal program:



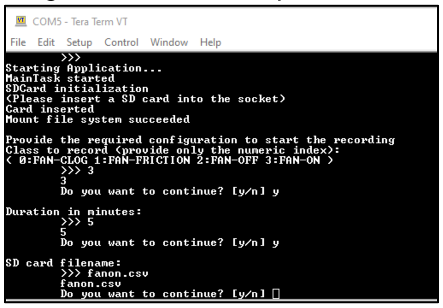
```
Starting Application...
MainTask started
SDCard initialization
(Please insert a SD card into the socket)
Card inserted
Mount file system succeeded

Provide the required configuration to start the recording
Class to record (provide only the numeric index):
< 0:FAN-CLOG 1:FAN-FRICTION 2:FAN-OFF 3:FAN-ON >
        >>>
```

10. For this demo there are four fan states that are going to be detected:
    - FAN-OFF
    - FAN-ON (normal operation)
    - FAN-CLOG (something covering the fan so it does not get the normal airflow)
    - FAN-FRICTION (something creating drag on the blades like a piece of tape or cardboard)
11. Physically put the fan in the desired state. For this first run, lets put the fan into the normal ON state.
12. In the terminal, type of the number of the fan state that corresponds with what the physical fan state is. So for this example, type '3' for FAN-ON.
13. The next selection is how many minutes to gather data. There is no hard and fast rule on how much data needs to be collected, but at a minimum several minutes worth will be required. For this simple example we'll select 5 minutes, but for more complex examples it could be significantly longer to ensure good training results. The ML-based System State Monitor Dataset Creation Guide provides a much deeper exploration into the optimal data collection techniques, but the key goal is to cover every corner case as much as possible and to not use the same data for training and validation.



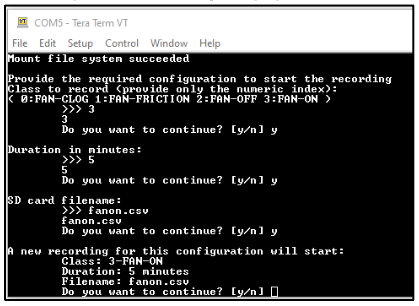
```
Starting Application...
MainTask started
SDCard initialization
(Please insert a SD card into the socket)
Card inserted
Mount file system succeeded

Provide the required configuration to start the recording
Class to record (provide only the numeric index):
( 0:FAN-CLOG 1:FAN-FRICTION 2:FAN-OFF 3:FAN-ON )
        >>> 3
        3
        Do you want to continue? [y/n] y

Duration in minutes:
        >>>
```

14. Next give a filename where this data will be stored in CSV (Comma Separated Value) format. It can be any name but should be descriptive and cannot be more than 12 characters in length. For this example lets use fanon.csv



```
        >>>
Starting Application...
MainTask started
SDCard initialization
(Please insert a SD card into the socket)
Card inserted
Mount file system succeeded

Provide the required configuration to start the recording
Class to record (provide only the numeric index):
( 0:FAN-CLOG 1:FAN-FRICTION 2:FAN-OFF 3:FAN-ON )
        >>> 3
        3
        Do you want to continue? [y/n] y

Duration in minutes:
        >>> 5
        5
        Do you want to continue? [y/n] y

SD card filename:
        >>> fanon.csv
        fanon.csv
        Do you want to continue? [y/n]
```

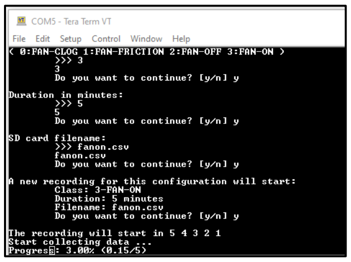
15. Finally, it will display your selections. Type 'y' to then start the data collection:



```
Mount file system succeeded

Provide the required configuration to start the recording
Class to record (provide only the numeric index):
( 0:FAN-CLOG 1:FAN-FRICTION 2:FAN-OFF 3:FAN-ON )
        >>> 3
        3
        Do you want to continue? [y/n] y

Duration in minutes:
        >>> 5
        5
        Do you want to continue? [y/n] y

SD card filename:
        >>> fanon.csv
        fanon.csv
        Do you want to continue? [y/n] y

A new recording for this configuration will start:
        Class: 3-FAN-ON
        Duration: 5 minutes
        Filename: fanon.csv
        Do you want to continue? [y/n]
```

16. You will see the status as it collects the data.

17. Once completed, then you can either remove the SD card to look at the data on your computer or hit the Reset button on the board to start the process again to collect validation data for FAN-ON or to collect the other fan states. You'll want to collect data for all 4 conditions and collect both Training data and Validation data so there should be at least 8 runs. The same amount of data collection time should be used for all data sets.



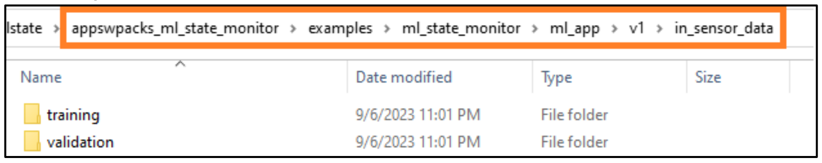
```
< 0:FAN-CLOG 1:FAN-FRICTION 2:FAN-OFF 3:FAN-ON >
        >>> 3
        3
        Do you want to continue? [y/n] y

Duration in minutes:
        >>> 5
        5
        Do you want to continue? [y/n] y

SD card filename:
        >>> fanon.csv
        fanon.csv
        Do you want to continue? [y/n] y

A new recording for this configuration will start:
        Class: 3-FAN-ON
        Duration: 5 minutes
        Filename: fanon.csv
        Do you want to continue? [y/n] y

The recording will start in 5 4 3 2 1
Start collecting data ...
Progress: 3.00% (0.15/5)
```

5 Train Data and create a model

5.1 Copy Training Data to PC

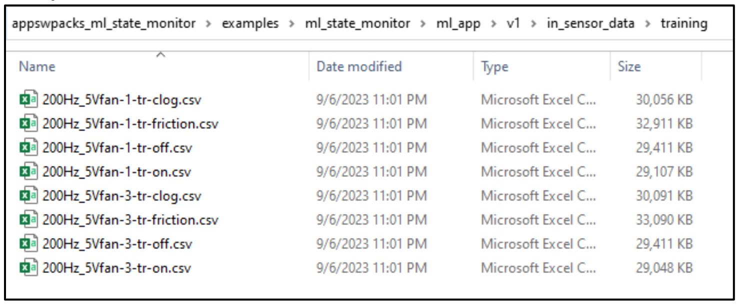
Once all the data required has been saved to the microSD card, remove it from the RT1170 EVK and put it into a card reader for your PC so the data can be copied to the hard drive.

1. Inside the project folder there is a folder named **\appswpacks_ml_state_monitor\examples \ml_state_monitor\ml_app\v1\in_sensor_data**. Locate that directory. Make sure to use the **v1** path.


lstate › appswpacks_ml_state_monitor › examples › ml_state_monitor › ml_app › v1 › in_sensor_data

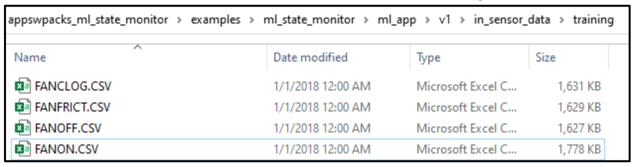
Name	Date modified	Type	Size
training	9/6/2023 11:01 PM	File folder	
validation	9/6/2023 11:01 PM	File folder	

2. Inside that folder there are two subfolders, one named training and the other validation. By default each folder contains pre-created data stored in .csv files that were gathered from a simple 5V fan.


appswpacks_ml_state_monitor › examples › ml_state_monitor › ml_app › v1 › in_sensor_data › training

Name	Date modified	Type	Size
200Hz_5Vfan-1-tr-clog.csv	9/6/2023 11:01 PM	Microsoft Excel C...	30,056 KB
200Hz_5Vfan-1-tr-friction.csv	9/6/2023 11:01 PM	Microsoft Excel C...	32,911 KB
200Hz_5Vfan-1-tr-off.csv	9/6/2023 11:01 PM	Microsoft Excel C...	29,411 KB
200Hz_5Vfan-1-tr-on.csv	9/6/2023 11:01 PM	Microsoft Excel C...	29,107 KB
200Hz_5Vfan-3-tr-clog.csv	9/6/2023 11:01 PM	Microsoft Excel C...	30,091 KB
200Hz_5Vfan-3-tr-friction.csv	9/6/2023 11:01 PM	Microsoft Excel C...	33,090 KB
200Hz_5Vfan-3-tr-off.csv	9/6/2023 11:01 PM	Microsoft Excel C...	29,411 KB
200Hz_5Vfan-3-tr-on.csv	9/6/2023 11:01 PM	Microsoft Excel C...	29,048 KB

3. These files should be deleted, and the newly collected data placed inside these folders in order to create a new custom model. The file name does not matter but should end in .csv. The files should cover all the possible classes and should be updated for both the **training** and **validation** folders with their respective training and validation data.

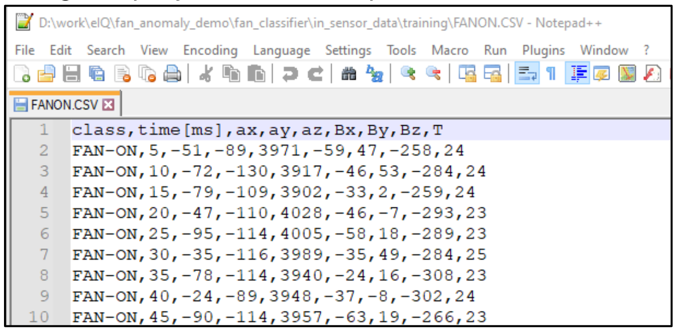


appswpacks_ml_state_monitor › examples › ml_state_monitor › ml_app › v1 › in_sensor_data › training

Name	Date modified	Type	Size
FANCLOG.CSV	1/1/2018 12:00 AM	Microsoft Excel C...	1,631 KB
FANFRICT.CSV	1/1/2018 12:00 AM	Microsoft Excel C...	1,629 KB
FANOFF.CSV	1/1/2018 12:00 AM	Microsoft Excel C...	1,627 KB
FANON.CSV	1/1/2018 12:00 AM	Microsoft Excel C...	1,778 KB

If you did not collect new data, you can use the pre-existing .csv files as that has data collected with a simple 5V brushless DC motor.

4. The Python scripts that will be used in the next section will assume the files will be in the following Comma Separated Value format, which is how the data was written to the files by using the projects from the previous section, so no modifications should be needed.

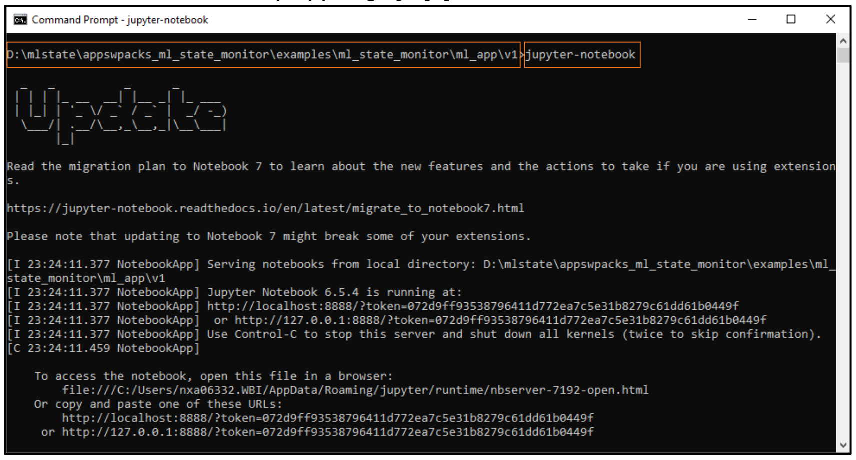

D:\work\eIQ\fan_anomaly_demo\fan_classifier\in_sensor_data\training\FANON.CSV - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
FANON.CSV

```
class,time[ms],ax,ay,az,Bx,By,Bz,T
FAN-ON,5,-51,-89,3971,-59,47,-258,24
FAN-ON,10,-72,-130,3917,-46,53,-284,24
FAN-ON,15,-79,-109,3902,-33,2,-259,24
FAN-ON,20,-47,-110,4028,-46,-7,-293,23
FAN-ON,25,-95,-114,4005,-58,18,-289,23
FAN-ON,30,-35,-116,3989,-35,49,-284,25
FAN-ON,35,-78,-114,3940,-24,16,-308,23
FAN-ON,40,-24,-89,3948,-37,-8,-302,24
FAN-ON,45,-90,-114,3957,-63,19,-266,23
```

5.2 Train The Model

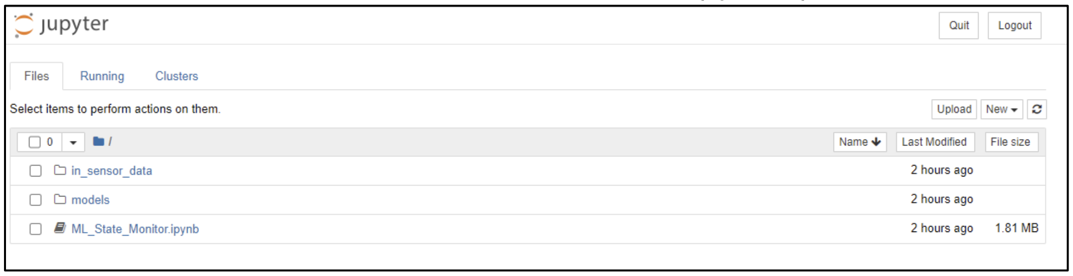
With the data ready, now use the Jupytr notebook to train the model.

1. Open a Windows command prompt
2. Navigate to the **\appswpacks_ml_state_monitor\examples\ml_state_monitor\ml_app\v1** directory
3. Start the notebook by typing "**jupyter-notebook**" into the command prompt



```
D:\mlstate\appswpacks_ml_state_monitor\examples\ml_state_monitor\ml_app\v1>jupyter-notebook

 _   _           _       _
| | | |_ __   __| | __ _| |_ ___
| | | | '_ \ / _` |/ _` | __/ _ \
| |_| | |_) | (_| | (_| | ||  __/
 \___/| .__/ \__,_|\__,_|\__\___|
      |_|

Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extension
s.

https://jupyter-notebook.readthedocs.io/en/latest/migrate_to_notebook7.html

Please note that updating to Notebook 7 might break some of your extensions.

[I 23:24:11.377 NotebookApp] Serving notebooks from local directory: D:\mlstate\appswpacks_ml_state_monitor\examples\ml_
state_monitor\ml_app\v1
[I 23:24:11.377 NotebookApp] Jupyter Notebook 6.5.4 is running at:
[I 23:24:11.377 NotebookApp] http://localhost:8888/?token=072d9ff93538796411d772ea7c5e31b8279c61dd61b0449f
[I 23:24:11.377 NotebookApp]  or http://127.0.0.1:8888/?token=072d9ff93538796411d772ea7c5e31b8279c61dd61b0449f
[I 23:24:11.377 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 23:24:11.459 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/nxa06332.WBI/AppData/Roaming/jupyter/runtime/nbserver-7192-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=072d9ff93538796411d772ea7c5e31b8279c61dd61b0449f
     or http://127.0.0.1:8888/?token=072d9ff93538796411d772ea7c5e31b8279c61dd61b0449f
```

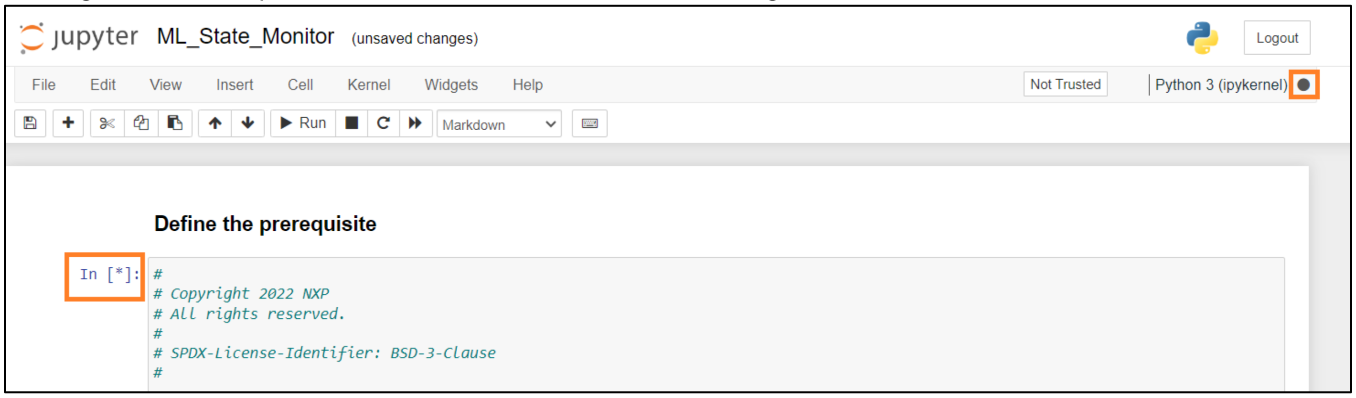
4. A web browser window should automatically open up and look like the following. If it does not, there are links in the terminal window that can be copy and pasted into a browser.



Select items to perform actions on them.

```
in_sensor_data          2 hours ago
models                  2 hours ago
ML_State_Monitor.ipynb  2 hours ago   1.81 MB
```

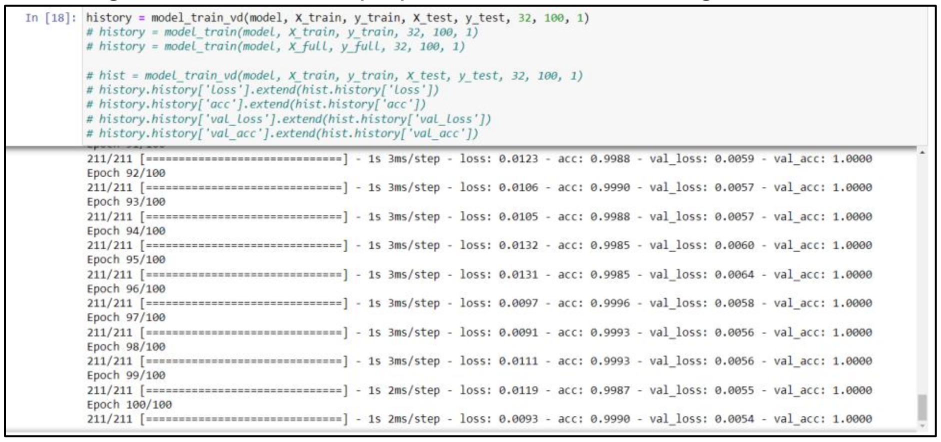
5. Click on **ML_State_Monitor.ipynb** to open the notebook
6. It will look like the following. Press the **Run** button to execute each section of the script. While executing an asterisk (*) will appear and the dot in the upper right will be dark. Don't press Run again until the previous section has finished executing.



Define the prerequisite

```
In [*]: #
        # Copyright 2022 NXP
        # All rights reserved.
        #
        # SPDX-License-Identifier: BSD-3-Clause
        #
```

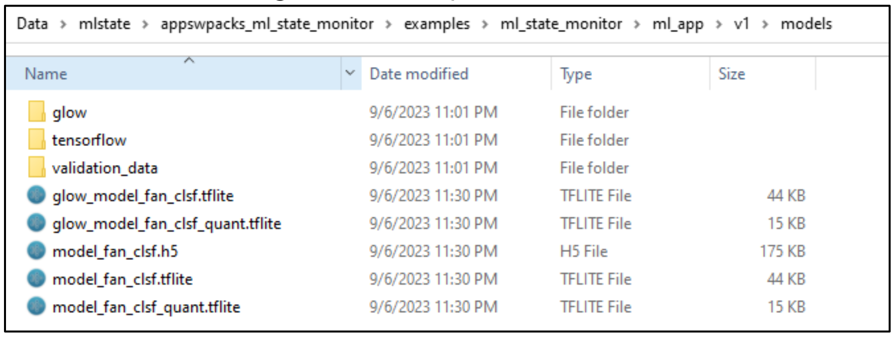
7. You may see some warning messages when executing certain blocks of the script but you should not get any error/failure messages.
8. It is in cell 18 that the model will actually be trained. This step may take a while. You should see a high (>90%) accuracy by the end of the training.



```
In [18]: history = model_train_vd(model, X_train, y_train, X_test, y_test, 32, 100, 1)
         # history = model_train(model, X_train, y_train, 32, 100, 1)
         # history = model_train(model, X_full, y_full, 32, 100, 1)

         # hist = model_train_vd(model, X_train, y_train, X_test, y_test, 32, 100, 1)
         # history.history['loss'].extend(hist.history['loss'])
         # history.history['acc'].extend(hist.history['acc'])
         # history.history['val_loss'].extend(hist.history['val_loss'])
         # history.history['val_acc'].extend(hist.history['val_acc'])
```

```
211/211 [==============================] - 1s 3ms/step - loss: 0.0123 - acc: 0.9988 - val_loss: 0.0059 - val_acc: 1.0000
Epoch 92/100
211/211 [==============================] - 1s 3ms/step - loss: 0.0106 - acc: 0.9990 - val_loss: 0.0057 - val_acc: 1.0000
Epoch 93/100
211/211 [==============================] - 1s 3ms/step - loss: 0.0105 - acc: 0.9988 - val_loss: 0.0057 - val_acc: 1.0000
Epoch 94/100
211/211 [==============================] - 1s 3ms/step - loss: 0.0132 - acc: 0.9985 - val_loss: 0.0060 - val_acc: 1.0000
Epoch 95/100
211/211 [==============================] - 1s 3ms/step - loss: 0.0131 - acc: 0.9985 - val_loss: 0.0064 - val_acc: 1.0000
Epoch 96/100
211/211 [==============================] - 1s 3ms/step - loss: 0.0097 - acc: 0.9996 - val_loss: 0.0058 - val_acc: 1.0000
Epoch 97/100
211/211 [==============================] - 1s 3ms/step - loss: 0.0091 - acc: 0.9993 - val_loss: 0.0056 - val_acc: 1.0000
Epoch 98/100
211/211 [==============================] - 1s 3ms/step - loss: 0.0111 - acc: 0.9993 - val_loss: 0.0056 - val_acc: 1.0000
Epoch 99/100
211/211 [==============================] - 1s 2ms/step - loss: 0.0119 - acc: 0.9987 - val_loss: 0.0055 - val_acc: 1.0000
Epoch 100/100
211/211 [==============================] - 1s 2ms/step - loss: 0.0093 - acc: 0.9990 - val_loss: 0.0054 - val_acc: 1.0000
```

9. In cell 29 it will export the trained model to generate two .tflite files in **\appswpacks_ml_state_monitor\examples\ml_state_monitor\ml_app\v1\models** that are named **model_fan_clsf.tflite** and **model_fan_clsf_quant.tflite**. There will also be C array versions of those two models in the **tensorflow** directory and Glow compiled versions of those models in the **glow** directory. Those files will be used in the next section.



Data > mlstate > appswpacks_ml_state_monitor > examples > ml_state_monitor > ml_app > v1 > models

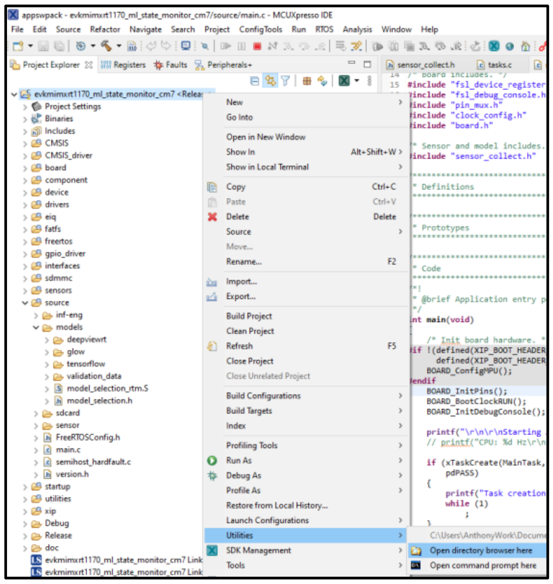
Name	Date modified	Type	Size
glow	9/6/2023 11:01 PM	File folder	
tensorflow	9/6/2023 11:01 PM	File folder	
validation_data	9/6/2023 11:01 PM	File folder	
glow_model_fan_clsf.tflite	9/6/2023 11:30 PM	TFLITE File	44 KB
glow_model_fan_clsf_quant.tflite	9/6/2023 11:30 PM	TFLITE File	15 KB
model_fan_clsf.h5	9/6/2023 11:30 PM	H5 File	175 KB
model_fan_clsf.tflite	9/6/2023 11:30 PM	TFLITE File	44 KB
model_fan_clsf_quant.tflite	9/6/2023 11:30 PM	TFLITE File	15 KB

10. **For informational purposes:** The last section of the Jupyter notebook allows you to run the model using accelerometer data sent over a UART in real-time. It is not required for this lab, but if you want to use that to verify your model, you can modify the COM port to match the number that the board enumerated as, and then set the #define SENSOR_COLLECT_LOG_EXT in sensor_collect.h to 0 so the data is sent out over the UART instead of written to an SD card. Start the project and then once the accelerometer data starts printing to the UART, close the serial terminal on your PC and then run that section of the Jupyter notebook. But again this step is not needed for this lab and can be skipped.

6 Update Model for Inference Engines

It is now possible to use all 3 eIQ inference engine options for microcontrollers (TFLM, Glow, and DeepViewRT) to run this newly created model on the i.MX RT1170 EVK.

1. First find the location on your PC where the ML-based system state monitor application resides after it was imported into MCUXpresso IDE in the previous section. So inside MCUXpresso IDE, right click on the **evkmimxrt1170_ml_state_monitor_cm7** project name and go down to **Utilities** and select **Open directory browser here** to open a Windows Explorer window at the location.

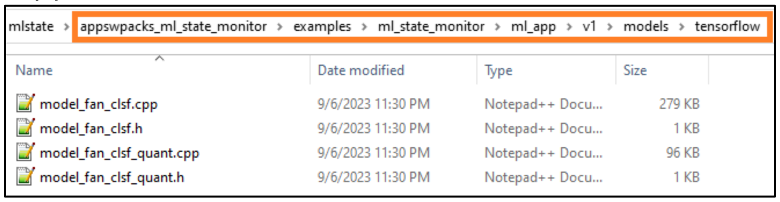


```
/* Board includes. */
#include "fsl_device_register
#include "fsl_debug_console.h
#include "pin_mux.h"
#include "clock_config.h"
#include "board.h"

/* Sensor and model includes.
#include "sensor_collect.h"

/*****************************
 * Definitions
 *****************************

/*****************************
 * Prototypes
 *****************************

/*****************************
 * Code
 *****************************
/*!
 * @brief Application entry p
 */
int main(void)
{
    /* Init board hardware. *
#if !(defined(XIP_BOOT_HEADER
      defined(XIP_BOOT_HEADER
    BOARD_ConfigMPU();
#endif
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();

    printf("\r\n\r\nStarting
    // printf("CPU: %d Hz\r\n

    if (xTaskCreate(MainTask,
        pdPASS)
    {
        printf("Task creation
        while (1)
            ;
    }
```

6.1 TensorFlow Lite for Microcontrollers

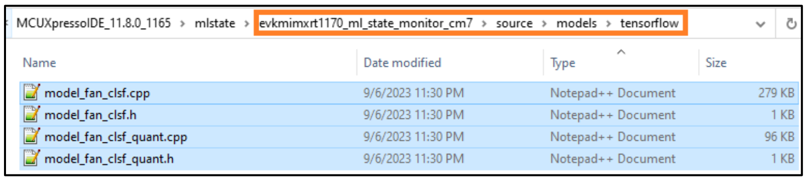
1. The Jupyter notebook already has created the C files needed by the TFLM inference engine, so those newly generated files just need to replace the default files in the project
2. Navigate to the <MCUXpresso IDE Workspace>**/source/models/tensorflow** directory.
3. Copy the generated .h and .cpp files from the previous section that were placed in the **\examples\ml_state_monitor\ml_app\v1\models\tensorflow\** folder and paste them into this directory.

Copy from:


mlstate › appswpacks_ml_state_monitor › examples › ml_state_monitor › ml_app › v1 › models › tensorflow

Name	Date modified	Type	Size
model_fan_clsf.cpp	9/6/2023 11:30 PM	Notepad++ Docu...	279 KB
model_fan_clsf.h	9/6/2023 11:30 PM	Notepad++ Docu...	1 KB
model_fan_clsf_quant.cpp	9/6/2023 11:30 PM	Notepad++ Docu...	96 KB
model_fan_clsf_quant.h	9/6/2023 11:30 PM	Notepad++ Docu...	1 KB

Paste into:


MCUXpressoIDE_11.8.0_1165 › mlstate › evkmimxrt1170_ml_state_monitor_cm7 › source › models › tensorflow

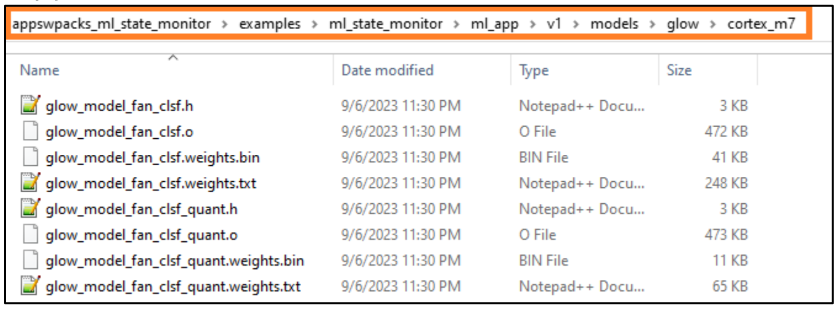
Name	Date modified	Type	Size
model_fan_clsf.cpp	9/6/2023 11:30 PM	Notepad++ Document	279 KB
model_fan_clsf.h	9/6/2023 11:30 PM	Notepad++ Document	1 KB
model_fan_clsf_quant.cpp	9/6/2023 11:30 PM	Notepad++ Document	96 KB
model_fan_clsf_quant.h	9/6/2023 11:30 PM	Notepad++ Document	1 KB

6.2 Glow

1. The Jupyter notebook already has created the Glow bundled needed by the Glow inference engine, so those newly generated files just need to replace the default files in the project. For details on the exact commands used to generate the Glow bundle see cell 29 in that Jupyter notebook.
2. Navigate to the <MCUXpresso IDE Workspace>**/source/models/glow** directory.

3. Copy the Glow bundle files from the previous section that were placed in the **\examples \ml_state_monitor\ml_app\v1\models\glow\cortex_m7** folder and paste them into this directory. Note that you should select the directory in that folder that corresponds with the core architecture in the device that will be used.

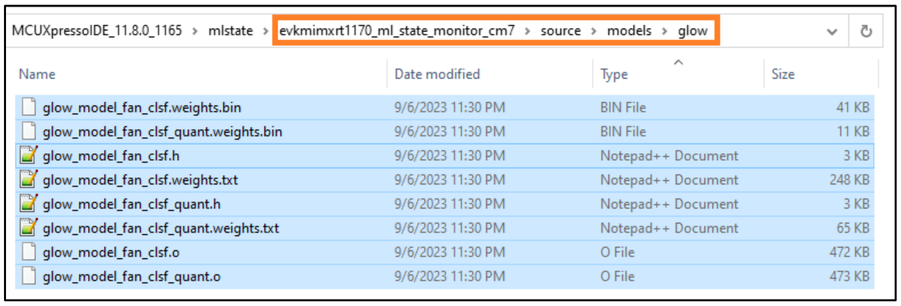
Copy from:



appswpacks_ml_state_monitor › examples › ml_state_monitor › ml_app › v1 › models › glow › cortex_m7

Name	Date modified	Type	Size
glow_model_fan_clsf.h	9/6/2023 11:30 PM	Notepad++ Docu...	3 KB
glow_model_fan_clsf.o	9/6/2023 11:30 PM	O File	472 KB
glow_model_fan_clsf.weights.bin	9/6/2023 11:30 PM	BIN File	41 KB
glow_model_fan_clsf.weights.txt	9/6/2023 11:30 PM	Notepad++ Docu...	248 KB
glow_model_fan_clsf_quant.h	9/6/2023 11:30 PM	Notepad++ Docu...	3 KB
glow_model_fan_clsf_quant.o	9/6/2023 11:30 PM	O File	473 KB
glow_model_fan_clsf_quant.weights.bin	9/6/2023 11:30 PM	BIN File	11 KB
glow_model_fan_clsf_quant.weights.txt	9/6/2023 11:30 PM	Notepad++ Docu...	65 KB

Paste into:



MCUXpressoIDE_11.8.0_1165 › mlstate › evkmimxrt1170_ml_state_monitor_cm7 › source › models › glow

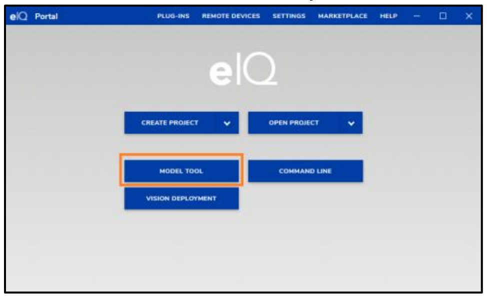
Name	Date modified	Type	Size
glow_model_fan_clsf.weights.bin	9/6/2023 11:30 PM	BIN File	41 KB
glow_model_fan_clsf_quant.weights.bin	9/6/2023 11:30 PM	BIN File	11 KB
glow_model_fan_clsf.h	9/6/2023 11:30 PM	Notepad++ Document	3 KB
glow_model_fan_clsf.weights.txt	9/6/2023 11:30 PM	Notepad++ Document	248 KB
glow_model_fan_clsf_quant.h	9/6/2023 11:30 PM	Notepad++ Document	3 KB
glow_model_fan_clsf_quant.weights.txt	9/6/2023 11:30 PM	Notepad++ Document	65 KB
glow_model_fan_clsf.o	9/6/2023 11:30 PM	O File	472 KB
glow_model_fan_clsf_quant.o	9/6/2023 11:30 PM	O File	473 KB

6.3 DeepViewRT

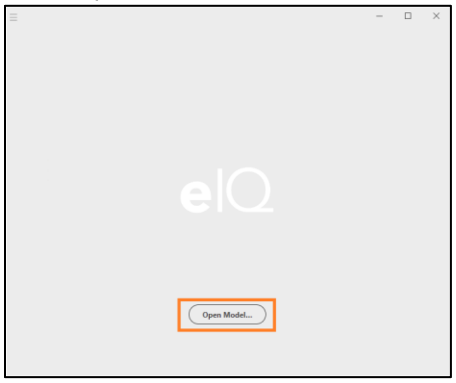
To generate an RTM file that the DeepViewRT inference engine uses, make sure that you have already installed the latest version of eIQ Toolkit.

Note that using the DeepViewRT inference engine is only supported on i.MX RT1170 and other i.MX RT devices.

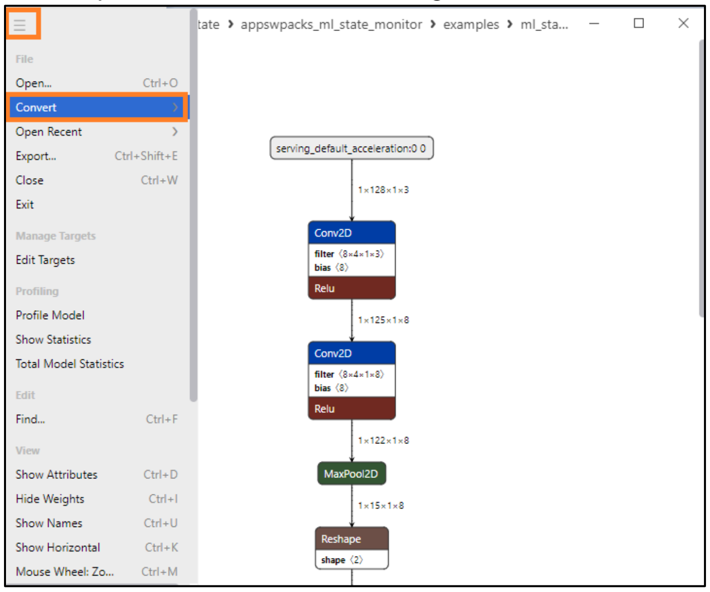
1. Open up eIQ Portal
2. Select the Model Tool option



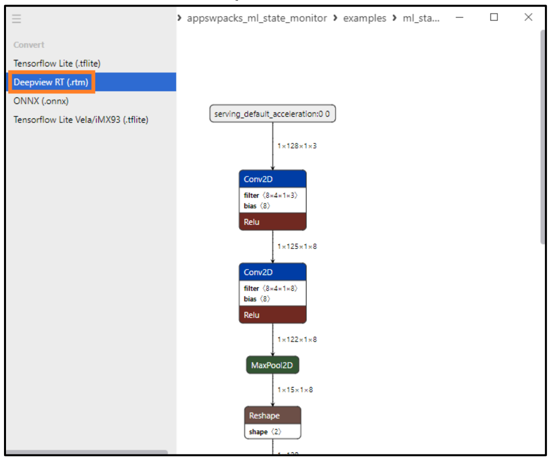
3. Click Open Model…



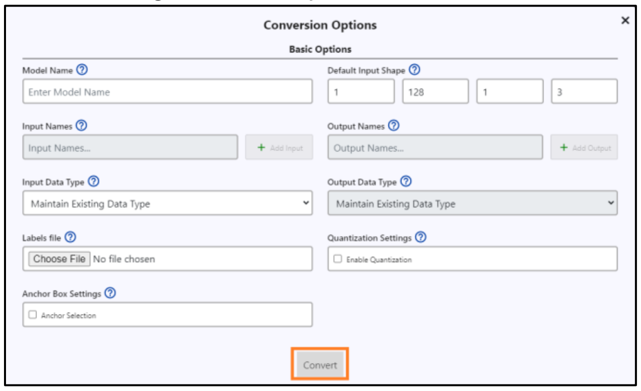
4. Navigate to the **\appswpacks_ml_state_monitor\examples\ml_state_monitor\ml_app\v1\models** folder to select the **model_fan_clsf.tflite** file
5. Once open, click on the Hamburger menu icon in the top left and then click on Convert



6. Then click on Deepview RT (.trm)


› appswpacks_ml_state_monitor › examples › ml_sta...
Convert
Tensorflow Lite (.tflite)
Deepview RT (.rtm)
ONNX (.onnx)
Tensorflow Lite Vela/iMX93 (.tflite)
serving_default_acceleration:0 0
1×128×1×3
Conv2D
Relu
1×125×1×8
Conv2D
Relu
1×122×1×8
MaxPool2D
1×15×1×8
Reshape

7. In the dialog box that opens, leave all the fields as default and then click on **Convert**



Conversion Options

Basic Options

Model Name: Enter Model Name

Default Input Shape: 1 | 128 | 1 | 3

Input Names: Input Names...

Output Names: Output Names...

Input Data Type: Maintain Existing Data Type

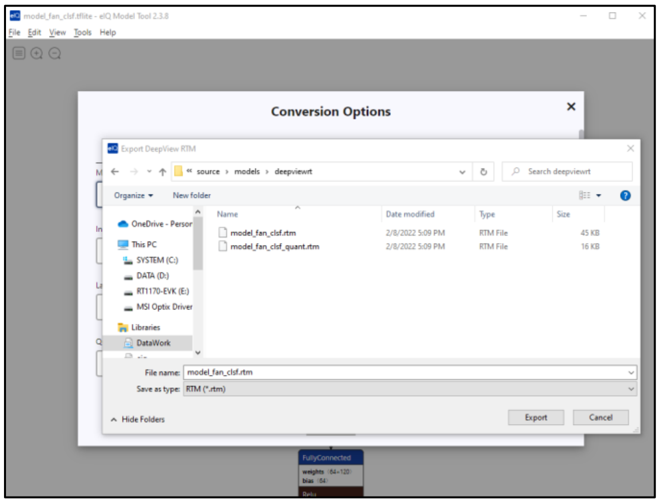
Output Data Type: Maintain Existing Data Type

Labels file: Choose File No file chosen

Quantization Settings: ☐ Enable Quantization

Anchor Box Settings: ☐ Anchor Selection

8. Save the generated file into the folder, overwriting the default original .rtm file, at **<MCUXpresso IDE Workspace>\evkmimxrt1170_ml_state_monitor_cm7\source\models\deepviewrt**



model_fan_clsf.tflite - eIQ Model Tool 2.3.8

Conversion Options

Export DeepView RTM

« source › models › deepviewrt

Name	Date modified	Type	Size
model_fan_clsf.rtm	2/8/2022 5:09 PM	RTM File	45 KB
model_fan_clsf_quant.rtm	2/8/2022 5:09 PM	RTM File	16 KB

File name: model_fan_clsf.rtm

Save as type: RTM (*.rtm)

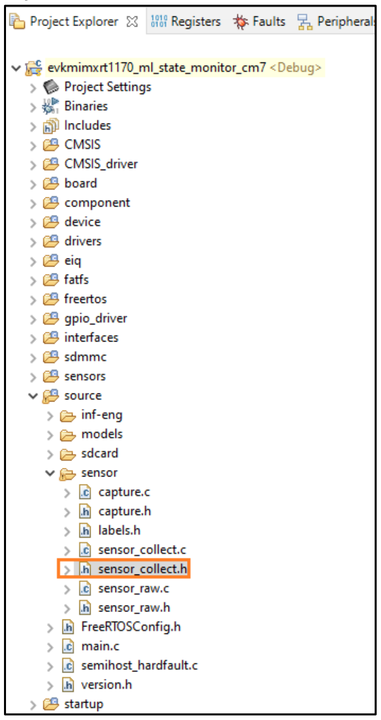
9. You can do the same for the pre-quantized version of the .tflite file as well by opening up **model_fan_clsf_quant.tflite** and performing the same conversion steps.

7 Deploy Model To Board

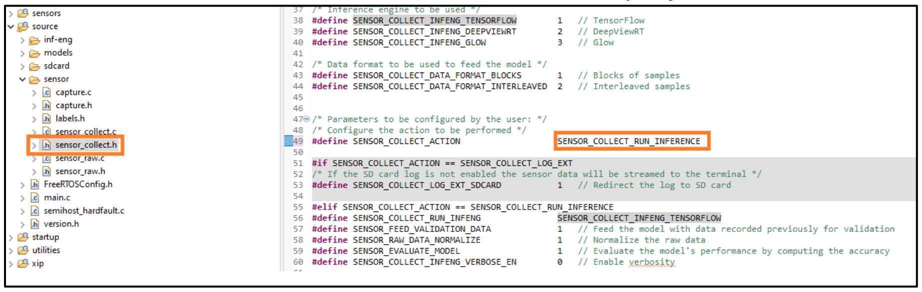
Now the models can be executed on the i.MX RT1170 EVK.

1. In MCUXpresso IDE, the **evkmimxrt1170_ml_state_monitor_cm7** project should already be imported into your workspace from the previous data collection section.

2. Open the **sensor_collect.h** file under **source\sensor** folder by double clicking on it

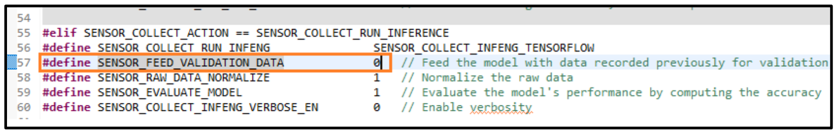


3. Modify line 49 to change the #define **SENSOR_COLLECT_ACTION** to **SENSOR_COLLECT_RUN_INFERENCE** to tell the project to now run the inference of the model



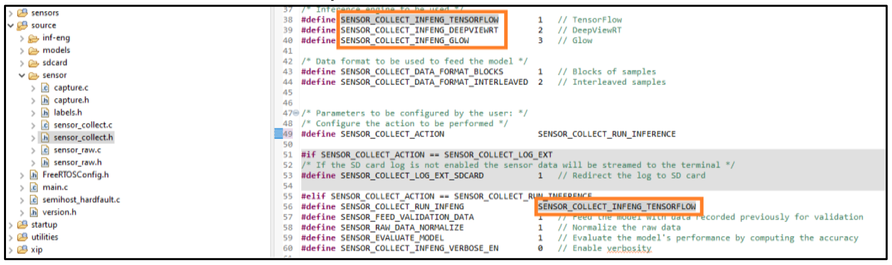
```
37 /* Inference engine to be used */
38 #define SENSOR_COLLECT_INFENG_TENSORFLOW       1   // TensorFlow
39 #define SENSOR_COLLECT_INFENG_DEEPVIEWRT       2   // DeepViewRT
40 #define SENSOR_COLLECT_INFENG_GLOW             3   // Glow
41
42 /* Data format to be used to feed the model */
43 #define SENSOR_COLLECT_DATA_FORMAT_BLOCKS      1   // Blocks of samples
44 #define SENSOR_COLLECT_DATA_FORMAT_INTERLEAVED 2   // Interleaved samples
45
46
47 /* Parameters to be configured by the user: */
48 /* Configure the action to be performed */
49 #define SENSOR_COLLECT_ACTION                  SENSOR_COLLECT_RUN_INFERENCE
50
51 #if SENSOR_COLLECT_ACTION == SENSOR_COLLECT_LOG_EXT
52 /* If the SD card log is not enabled the sensor data will be streamed to the terminal */
53 #define SENSOR_COLLECT_LOG_EXT_SDCARD          1   // Redirect the log to SD card
54
55 #elif SENSOR_COLLECT_ACTION == SENSOR_COLLECT_RUN_INFERENCE
56 #define SENSOR_COLLECT_RUN_INFENG              SENSOR_COLLECT_INFENG_TENSORFLOW
57 #define SENSOR_FEED_VALIDATION_DATA            1   // Feed the model with data recorded previously for validation
58 #define SENSOR_RAW_DATA_NORMALIZE              1   // Normalize the raw data
59 #define SENSOR_EVALUATE_MODEL                  1   // Evaluate the model's performance by computing the accuracy
60 #define SENSOR_COLLECT_INFENG_VERBOSE_EN       0   // Enable verbosity
```

4. Then modify line 57 change the #define **SENSOR_FEED_VALIDATION_DATA** to **0** in order to use the accelerometer data collected on the board to feed to the inference engine



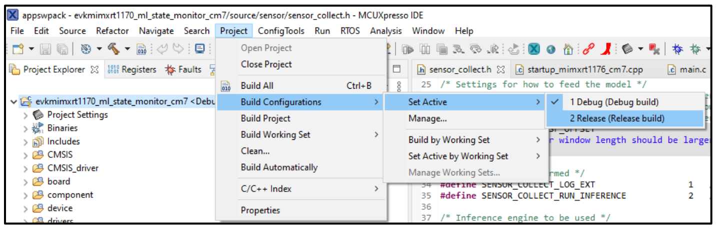
```
54
55 #elif SENSOR_COLLECT_ACTION == SENSOR_COLLECT_RUN_INFERENCE
56 #define SENSOR_COLLECT_RUN_INFENG              SENSOR_COLLECT_INFENG_TENSORFLOW
57 #define SENSOR_FEED_VALIDATION_DATA            0   // Feed the model with data recorded previously for validation
58 #define SENSOR_RAW_DATA_NORMALIZE              1   // Normalize the raw data
59 #define SENSOR_EVALUATE_MODEL                  1   // Evaluate the model's performance by computing the accuracy
60 #define SENSOR_COLLECT_INFENG_VERBOSE_EN       0   // Enable verbosity
```

5. Optionally modify line 56 if desired to change the #define for **SENSOR_COLLECT_RUN_INFENG** to the desired inference engine to use. By default the project will use TensorFlow Lite for Microcontrollers but you can see the other inference options as well. For this lab we'll keep it with TFLM.



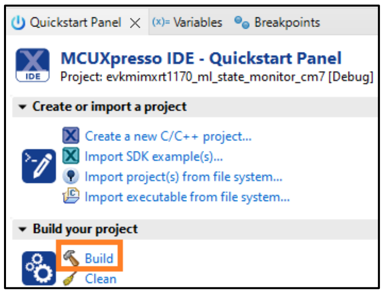
```
37 /* Inference engine to be used */
38 #define SENSOR_COLLECT_INFENG_TENSORFLOW       1   // TensorFlow
39 #define SENSOR_COLLECT_INFENG_DEEPVIEWRT       2   // DeepViewRT
40 #define SENSOR_COLLECT_INFENG_GLOW             3   // Glow
41
42 /* Data format to be used to feed the model */
43 #define SENSOR_COLLECT_DATA_FORMAT_BLOCKS      1   // Blocks of samples
44 #define SENSOR_COLLECT_DATA_FORMAT_INTERLEAVED 2   // Interleaved samples
45
46
47 /* Parameters to be configured by the user: */
48 /* Configure the action to be performed */
49 #define SENSOR_COLLECT_ACTION                  SENSOR_COLLECT_RUN_INFERENCE
50
51 #if SENSOR_COLLECT_ACTION == SENSOR_COLLECT_LOG_EXT
52 /* If the SD card log is not enabled the sensor data will be streamed to the terminal */
53 #define SENSOR_COLLECT_LOG_EXT_SDCARD          1   // Redirect the log to SD card
54
55 #elif SENSOR_COLLECT_ACTION == SENSOR_COLLECT_RUN_INFERENCE
56 #define SENSOR_COLLECT_RUN_INFENG              SENSOR_COLLECT_INFENG_TENSORFLOW
57 #define SENSOR_FEED_VALIDATION_DATA            1   // Feed the model with data recorded previously for validation
58 #define SENSOR_RAW_DATA_NORMALIZE              1   // Normalize the raw data
59 #define SENSOR_EVALUATE_MODEL                  1   // Evaluate the model's performance by computing the accuracy
60 #define SENSOR_COLLECT_INFENG_VERBOSE_EN       0   // Enable verbosity
```

5. If using TensorFlow Lite for Microcontrollers, make sure to set the Build Configuration to the "Release" target which does a high compiler optimization. This setting has a minor effect when using Glow or DeepViewRT but significantly reduces inference time when using TFLM. Click on the project name and then go to **Project->Build Configurations->Set Active->Release**



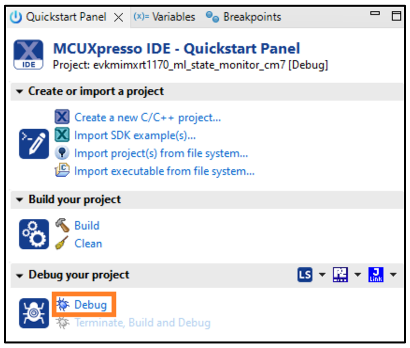
```
/* Settings for how to feed the model */
#define SENSOR_COLLECT_RUN_INFERENCE         2

/* Inference engine to be used */
```

6. Next build the project by clicking on "Build" in the Quickstart Panel and make sure there are no errors.



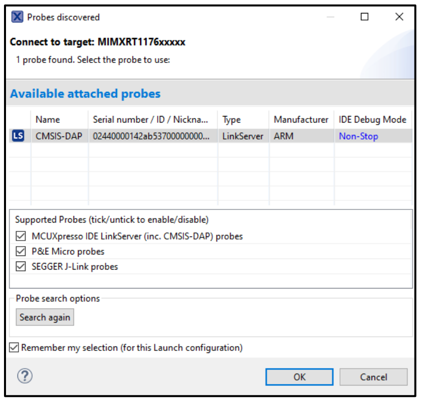
Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
Project: evkmimxrt1170_ml_state_monitor_cm7 [Debug]
Create or import a project
Create a new C/C++ project...
Import SDK example(s)...
Import project(s) from file system...
Import executable from file system...
Build your project
Build
Clean

5. Plug the micro-B USB cable into the board at J11 on the i.MXRT1170 board.
6. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
7. Debug the project by clicking on "Debug" in the Quickstart Panel.



Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
Project: evkmimxrt1170_ml_state_monitor_cm7 [Debug]
Create or import a project
Create a new C/C++ project...
Import SDK example(s)...
Import project(s) from file system...
Import executable from file system...
Build your project
Build
Clean
Debug your project
Debug
Terminate, Build and Debug

8. It will ask what interface to use. Select CMSIS-DAP.



Connect to target: MIMXRT1176xxxxx

1 probe found. Select the probe to use:

Available attached probes

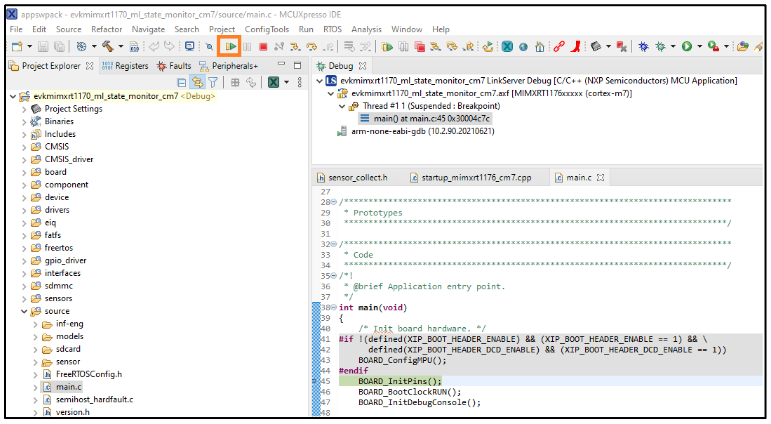
	Name	Serial number / ID / Nickna...	Type	Manufacturer	IDE Debug Mode
LS	CMSIS-DAP	02440000142ab53700000000...	LinkServer	ARM	Non-Stop

Supported Probes (tick/untick to enable/disable)
- ☑ MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
- ☑ P&E Micro probes
- ☑ SEGGER J-Link probes

Probe search options

☑ Remember my selection (for this Launch configuration)

9. The debugger will download the firmware and open up the debug view. Click on the Resume button to start running.

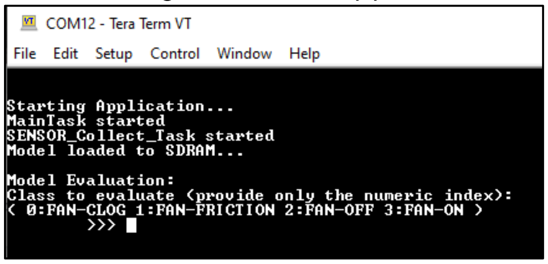

appswpack - evkmimxrt1170_ml_state_monitor_cm7/source/main.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run RTOS Analysis Window Help
Project Explorer
Registers
Faults
Peripherals+
Debug
sensor_collect.h
startup_mimxrt1176_cm7.cpp
main.c

evkmimxrt1170_ml_state_monitor_cm7 <Debug>
- Project Settings
- Binaries
- Includes
- CMSIS
- CMSIS_driver
- board
- component
- device
- drivers
- eiq
- fatfs
- freertos
- gpio_driver
- interfaces
- sdmmc
- sensors
- source
  - inf-eng
  - models
  - sdcard
  - sensor
  - FreeRTOSConfig.h
  - main.c
  - semihost_hardfault.c
  - version.h

evkmimxrt1170_ml_state_monitor_cm7 LinkServer Debug [C/C++ (NXP Semiconductors) MCU Application]
- evkmimxrt1170_ml_state_monitor_cm7.axf [MIMXRT1176xxxxx (cortex-m7)]
  - Thread #1 1 (Suspended : Breakpoint)
    - main() at main.c:45 0x30004c7c
  - arm-none-eabi-gdb (10.2.90.20210621)

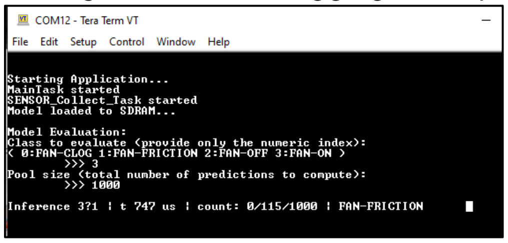
```c

/*******************************************************************************
 * Prototypes
 ******************************************************************************/

/*******************************************************************************
 * Code
 ******************************************************************************/
/*!
 * @brief Application entry point.
 */
int main(void)
{
    /* Init board hardware. */
#if !(defined(XIP_BOOT_HEADER_ENABLE) && (XIP_BOOT_HEADER_ENABLE == 1) && \
      defined(XIP_BOOT_HEADER_DCD_ENABLE) && (XIP_BOOT_HEADER_DCD_ENABLE == 1))
    BOARD_ConfigMPU();
#endif
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();

```

10. The following text should appear in the terminal program:



```
Starting Application...
MainTask started
SENSOR_Collect_Task started
Model loaded to SDRAM...

Model Evaluation:
Class to evaluate (provide only the numeric index):
< 0:FAN-CLOG 1:FAN-FRICTION 2:FAN-OFF 3:FAN-ON >
        >>> 
```

11. Input a target class
12. Then input the pool size (how many samples to take). Put in a large number (like 1000) to get a continuously updating status in the terminal.
13. It will then respond with its predictions based on the state of the fan. Try turning off the fan, adding friction, and clogging it and you should see the state change:



```
Starting Application...
MainTask started
SENSOR_Collect_Task started
Model loaded to SDRAM...

Model Evaluation:
Class to evaluate (provide only the numeric index):
< 0:FAN-CLOG 1:FAN-FRICTION 2:FAN-OFF 3:FAN-ON >
        >>> 3
Pool size (total number of predictions to compute):
        >>> 1000

Inference 3?1 | t 747 us | count: 0/115/1000 | FAN-FRICTION
```

8 Conclusion

This lab demonstrated how to gather and train sensor data to a classification model and deploy it on an i.MX RT device.

This same lab can be used for other types of sensor data models that detect vibrations. By enabling machine learning in embedded systems, there's a wide world of opportunity for new smarter applications.