Integrating regression models on industrial embedded system

Introduction

The current success of AI involves an increasing number of applications and AI algorithms are growing in the industrial field to strongly improve the effectiveness of condition monitoring and predictive maintenance, taking the best of traditional processing methods (for example, FFT analysis) and combining it with different kinds of machine learning (ML) algorithms.

Classical problems like multiclass classification and regression can be solved using ML and they can find a strategic use in fault classification or fault severity estimation; problems dear to many world industries.

ML integration on embedded systems has accelerated over the past few years thanks to several advancements in microcontroller (MCU) and sensor technologies. New MCUs have made great strides in terms of clock speed, memories footprint and specialized processing IP such as advanced DSP unit; these improvements allowed to execute on the edge complex algorithms using MCUs increasingly powerful and smaller at the same time. Clearly, without the support of upgraded sensors which produce the fuel of AI that are data, ML integration could not become an effective solution. Sensors like the latest inertial module are more than sensors, they can compensate for temperature instabilities, buffer a huge quantity of samples, and sometimes preprocess data too. Finally, we have new automated ML software capable of training and validating hundreds of models on a common PC. The process of finding the best ML model is automatized to allow users that have no ML skills to integrate the best into industrial microcontrollers. At the end of the day, the available hardware and software tool ecosystem strongly facilitate and speed up a fine-grained monitoring inside industrial plants.

The main goal of this application note is to demonstrate how a regression problem can be established in an industrial context with particular focus on rotative machines monitoring. Typically, the monitoring of these kinds of assets is based on vibration analysis and it can involve the usage of power industrial PC and expansive sensors, the output of the latter is usually hard to understand to non-experts causing down-time and/or extra maintenance costs.

STMicroelectronics provides a complete hardware solution STEVAL-PROTEUS1 to quickly design your own product starting from a reference design which has all the fundamental devices you need for your predictive maintenance system based on MEMS vibration sensors, industrial MCU, and equipped with a wireless connectivity.

One of the most common rotative machines in an industrial environment is the fan. Fans are indispensable in industrial environments for maintaining air quality, cooling equipment and workspaces, controlling dust and humidity, and ensuring the safety and comfort of workers. Inside industrial plants, the obstruction of fans and turbines is very common. This document would demonstrate how it is possible to estimate the fan obstruction percentage thanks to a regression algorithm integrated in an industrial STM32 wireless SoC.

An automated ML tool like NanoEdgeAIStudio can generate different kinds of ML models for a wide application area without any constraint in terms of specs such as rotation speed, number and size of blades, and other specific application features.

Any industrial system needs connectivity to spread information about asset operation to gateways, concentrators, or HMI devices. In the industrial IoT field, one of the most typical wireless connectivities is the Bluetooth® Low Energy (BLE); its low-power consumption, fast connection and data throughput, fit industrial IoT node requirements and therefore it is used in a wide range of products.

To provide a real-use case example, a professional validation setup was realized, several models have been evaluated reporting at the end of the document some benchmark and useful suggestions to obtain the best from ML approach.

1 Industrial embedded systems

Embedded systems are widely used in the industrial field. They provide automation, control, condition monitoring, and much else to address specific applications such as preventive and predictive maintenance.

Building a solid network formed by many IIoT devices distributed in the field, embedded systems are used in different kinds of manufacturing from food and beverage to pharmaceutical to automative.

Any embedded systems can integrate different kinds of sensors, processing unit, and connectivity.

Embedded systems designed to make condition monitoring and predictive maintenance on rotative machines, typically includes motion sensors like accelerometer and gyroscope which can be differentiated according to their data rate, full scale, or specific additional features.

The processing unit might be an MCU or an MPU but generally, in tiny boards like the ones directly applied or integrated on or inside the rotative machines, MCUs are preferred because they embed all the necessary resources.

Sometimes MCU and connectivity circuitry are placed in a unique device named a wireless SoC. In recent years, ML algorithms can be run also in non-high performance MCUs thanks to improvements in model quantization and lightweight model architectures.

Connectivity can be short or long, wireless, or wired, according to application needs. Among the most used industrial protocols, we have Bluetooth® Low Energy, ZigBee, Wi-Fi as wireless connectivity, and Profinet, modbus, IO-Link as wired ones.

1.1 STEVAL-PROTEUS1

The STEVAL-PROTEUS1 is an evaluation tool designed to monitor the temperature and vibration of industrial equipment.

It is based on a 2.4 GHz multiprotocol wireless SoC to address machine or facility conditions monitoring industrial applications.

The board includes the STM32WB5MMG ultra-low-power and small form factor wireless radio module, based on the STM32WB55VGY wireless SoC, compliant with the Bluetooth® Low Energy SIG specification v5.2, ZigBee 3.0, and IEEE 802.15.4-2011.

The powerful Arm®-based Cortex®-M4 with FPU and large memory allow running the embedded algorithm at node level.

The board even integrates the IIS3DWB high bandwidth (up to 6 kHz) accelerometer, the IIS2DLPC ultra-low-power accelerometer (used as wakeup source), and the ISM330DHCX inertial module (accelerometer and gyroscope) with MLC, making the hardware ideal for a customized vibration monitoring development.

The STTS22H temperature sensor has been integrated in the board, providing high accuracy and low-power consumption.

An onboard external flash memory is connected via QSPI to the STM32WB5MMG module for data buffering and event storage.

The STEVAL-PROTEUS is LiPo rechargeable battery-powered, but it can also be powered via USB (5 V at 500 mA) or via a primary battery (which is not included in the kit).

2 Embedded system requirements

All the requirements needed to implement a reliable regression on an industrial embedded system are collected in this section.

Listed in the following paragraphs are the requirements to do a preliminary vibration data analysis, to acquire vibration data produced by motion sensors, and finally to integrate ML models in a ready-to-use firmware application.

The following table refers to the validation setup which is presented in Section 3: Validation set up requirements.

Figure 1. Requirements

Requirements		
Hardware	Firmware	Software
STEVAL-PROTEUS1	STSW-PROTEUS v1.1.1	STM32CubeIDE v1.14.0 or IAR-EWARM v9.20.1 or MDK-ARM v5.38.0
STLINK-V3MINIE (optional)	FP-SNS-DATAPRO1 v1.1.0	NanoEdgeAIStudio v4.5.0 or higher
NUCLEO-C031C6 (optional)	FP-AI-PDMWBSOC2 v2.0.0	STBLESensor v5.2.8 for Android or v5.3.4 for IOS
Resistor R1=13 kΩ		STM32CubeProgrammer (optional)
Capacitors C1 = 3.3 nF, C2 = 1 mF (optional)		Tera Term (optional)
Axial Brushless DC Fan Sanyo Denki (No. 9HVA0812P1G001)		
Power Supply 12 Vdc (at least 50 W)		
Cable 2x0.5mm^2, DuPont wires		
Matrix board (optional)		
Plexiglass or other to realize fan stoppers		
Smartphone (Android or IOS)		
PC (Windows v10 or higher)		

2.1 Hardware

2.1.1 STEVAL-PROTEUS1

Thanks to the completeness of the STEVAL-PROTEUS1 kit, it represents a reference solution which can be used in an industrial application like the one subject of this document. Its sensing, processing, and connectivity capabilities are fundamental to generate and deploy a regression model in the field.

In the following sections there is a particular focus on the ISM330DHCX inertial module to produce the necessary data, and the same focus is placed on the STM32WB5MMG, to use its capabilities to manage sensors, to execute ML algo, and spread processing results via Bluetooth® Low Energy.

See UM3000 for more details about the STEVAL-PROTEUS1 hardware architecture, to understand the product features, and to obtain some guidelines for the proper use of the kit.

2.2 Firmware

A set of firmware packages are essential to achieve the goal of the document. This section contains a brief description of the firmware materials which are used.

2.2.1 STSW-PROTEUS

The STSW-PROTEUS is an STM32Cube software package for vibration and temperature condition equipment monitoring through the STEVAL-PROTEUS1, over Bluetooth® Low Energy and ZigBee connectivity.

The application captures vibration and temperature data from MEMS sensors, and it uses them as inputs to perform complex algorithms such as frequency and time-domain vibration analysis.

It transfers the ready-to-use results such as FFT or acceleration peak via the above-mentioned connectivity.

The STSW-PROTEUS software package includes two different project examples respectively based on Bluetooth® Low Energy and ZigBee.

This document covers the Bluetooth® Low Energy application especially, which allows to directly connect a smartphone through the STBLESensor mobile app to facilitate node configuration, local monitoring equipment status, and other additional functionalities.

The UM3045 provides more information about this software package.

2.2.2 FP-SNS-DATAPRO1

FP-SNS-DATAPRO1 is an STM32Cube function pack that allows the user to store data from any combination of the sensors mounted on the STEVAL-PROTEUS1, configuring them up to the maximum sampling rate.

Sensor data can be stored in the FatFs volume created inside the onboard external NOR flash memory or streamed to a PC via USB.

The STEVAL-PROTEUS1 that runs the FP-SNS-DATAPRO1 acts as a Bluetooth® Low Energy device, and it is supported by the STBLESensor mobile app. In addition to configuring the sensors, the mobile app allows to control the start/stop acquisition as well as transfer the data stored in the onboard external NOR flash memory to the mobile device that is running the app.

Furthermore, the FP-SNS-DATAPRO1 can make the STEVAL-PROTEUS1 act as USB mass storage, connecting it to a USB port of a PC and using a file manager, the user is able to manage folders and files inside the FatFs volume stored in the onboard external NOR flash memory.

UM3396 provides more information about this software package.

2.2.3 FP-AI-PDMWBSOC2

FP-AI-PDMWBSOC2 is an STM32Cube function pack for the STEVAL-PROTEUS1, developed to get motion sensor data, process them for anomaly detection, faults classification, and fault severity estimation.

The processing results can be sent to the STBLESensor mobile app or a PC terminal console. The application uses sensors data as inputs to execute machine learning algorithms to detect anomaly behaviors, to classify operating modes or faults, and to estimate fault severity in an industrial environment.

The user can configure the industrial node by remote, using wireless or wired connectivity.

Anomaly detection, N-Class classification, and extrapolation models coming from NanoEdgeAIStudio software can be easily integrated into the proposed framework architecture.

See UM3208 for more information about this software package.

2.3 Software

In addition to NanoEdgeAIStudio, there are several software which are fundamental to perform some mandatory steps. A brief description of the software which is used was inserted inside this section.

2.3.1 IDE

All the firmware packages mentioned in Section 2.2: Firmware contain an application example, in source code, for the following IDEs:

- STM32CubeIDE v1.14.0 or higher
- IAR-EWARM v9.20.1
- MDK-ARM v5.37.0/v5.38.0

The user can find an IDE project for each of these IDE software.

2.3.2 NanoEdge AI Studio

NanoEdgeAIStudio is a free software provided by ST to easily add AI into any embedded project running on any Arm Cortex®-M MCU. It empowers embedded engineers, even those unfamiliar with AI, to almost effortlessly find the optimal AI model for their requirements through straightforward processes.

Operated locally on a PC, the software takes input data and generates the NanoEdge AI library that incorporates the model, its preprocessing, and functions for easy integration into new or existing embedded projects. The main strength of NanoEdgeAIStudio is its benchmark, which explores thousands of combinations of preprocessing, models, and parameters. This iterative process identifies the most suitable algorithm tailored to the user's needs based on their data.

See NanoEdgeAIStudio for more information about this software tool.

2.3.3 STBLESensor app

The STBLESensor application is available for Android and iOS devices and shows the data exported by a Bluetooth® Low Energy device using the BlueST protocol.

The app shows different panels based on the data types exported by the firmware, including plot sensors data, high-speed data logging, condition-based monitoring, NEAI anomaly detection, classification and extrapolation, and many others.

See DB3756 for more information about the mobile app.

3 Validation setup requirements

3.1 Axial brushless DC fan

The main part of the validation setup is the fan. There are no specific features that the fan must have, though a size greater than or equal to 80x80x38 mm is comfortable to place the STEVAL-PROTEUS1 on top of the fan.

The Sanyo Denki San Ace 80 9HVA was used, it is an ideal solution for efficiently cooling servers, data storage systems, and power supplies. The main features of the used fan are listed below:

- Power supply: 12 V dc
- Maximum current: 3.5 A
- Maximum power consumption: 42 W
- Size: 80x80x38 mm
- Fan type: 4 wire, PWR (2) + PWM (1) + Tacho (1)
- Bearing type: spherical roller
- Minimum rated speed: 4,200 RPM
- Maximum rated speed: 16,100 RPM

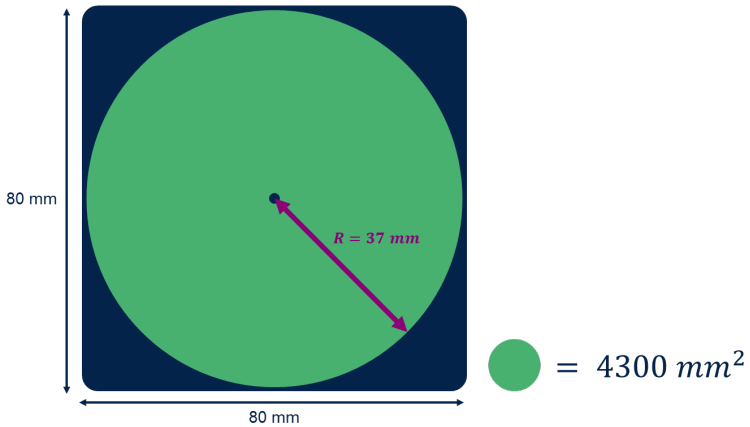
Usage of a 4-wire fan is recommended as it allows to measure rotational speed as well as to control it according to the duty cycle of a PWM control signal.

Figure 2. Sanyo Denki San Ace 80 9HVA

3.1.1 Fan stoppers

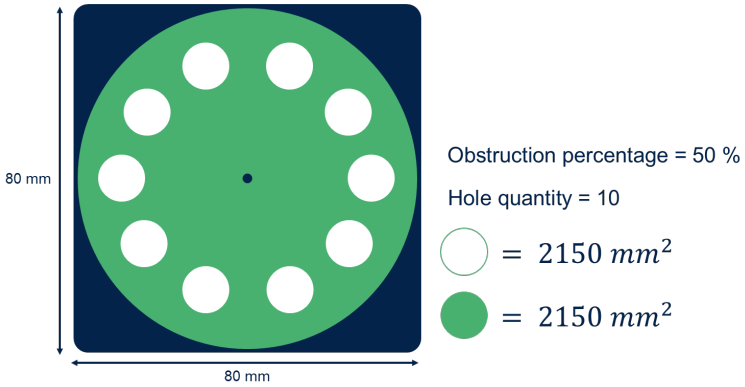
As explained in the introduction, the aim of this document is to prove the capability of a regressor model to estimate the percentage of fan obstruction. For this reason it is necessary to realize several stoppers suitably sized to inject different levels of obstruction. The following figure refers to an 80x80 mm fan and it represent the free area when no stopper is placed:

Figure 3. Fan free area



For example, to realize a 50% clogging stopper, we have:

Figure 4. 50% clogging stopper



With similar computations, we can define the following sizes for the stopper from 50% to 90% with a 5% step:

Figure 5. Stopper sizes

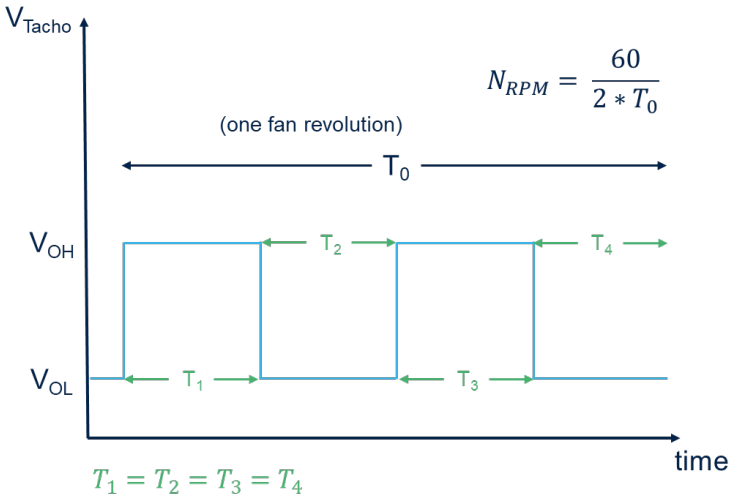
Clogging [%]	Free area [mm²]	Hole area [mm²]	Hole radius [mm]
50	2150.42	215.04	8.27
55	1935.38	193.54	7.85
60	1720.34	172.03	7.40
65	1505.29	150.53	6.92
70	1290.25	129.03	6.41
75	1075.21	107.52	5.85
80	860.17	86.02	5.23
85	645.13	64.51	4.53
90	430.08	43.01	3.70

3.2 STM32 Nucleo board (optional)

In case of fan with 4 wires, an additional STM32 Nucleo board can be useful to measure the tacho signal period and to tune the PWM control signal duty cycle.

The figure below represents the typical operation of a fan tacho signal.

Figure 6. Fan tacho signal



Using a timer with input capture mode, $T_0/2$ can be easily measured, the time base must be computed according to the rotational speed range of the selected fan. Considering the minimum and maximum rated speed of the proposed fan, a time base of 1 µs is good, and the speed can be computed according to the following formula:

$$N_{RPM} = \frac{Timer\_Clock/_{PSC}}{\left(CCR1_{first\_rising} - CCR1_{second\_rising}\right)} * \frac{1}{2} * 60 \quad (1)$$

For example, if a NUCLEO-C031C6 is used, there are five 16-bit timers clocked at 48 MHz, we could have the following setup:

$$TIM3\_Clock = 48MHz; TIM3\_PSC = 48; TIM3\_ARR = 65535 \quad (2)$$

To mitigate some imprecision in the fan speed measurement, it is suggested to compute the arithmetic mean of a certain number of N_{RPM} measurements (for example, every 10 measurements could be enough).

Once you are able to measure the rotational speed, the PWM control signal can be generated using a separate timer. PWM control signal frequency depends on the specific fan being used, generally it is about 25 kHz.

For the proposed fan, a possible configuration is:

$$TIM1\_Clock = 48MHz; TIM1\_PSC = 0; TIM1\_ARR = 1919 \quad (3)$$

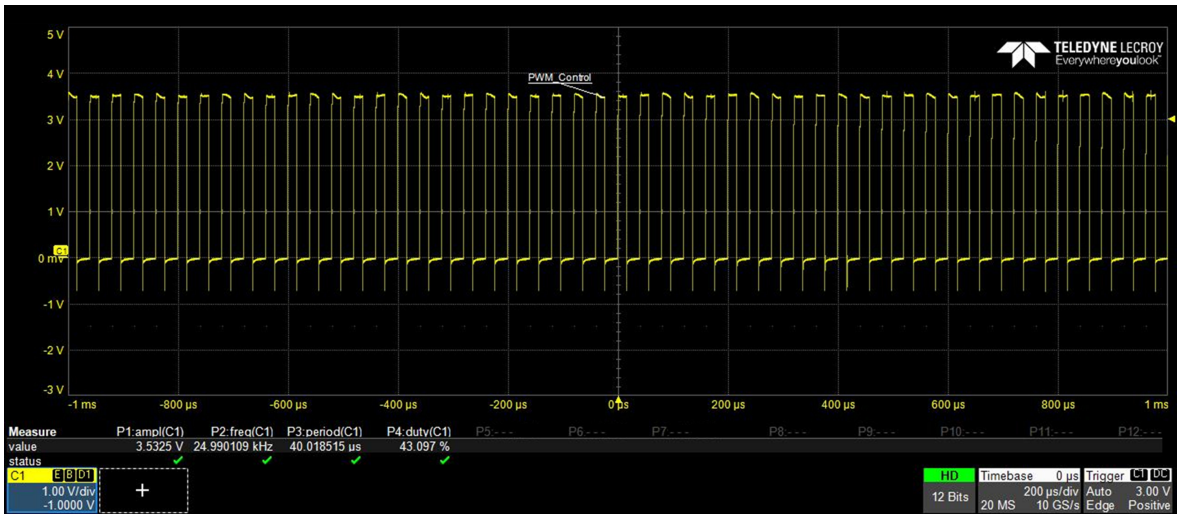
These values can be put in the following formula to verify itself:

$$PWM_{frequency} = \frac{Timer\_Clock}{(PSC + 1) * (ARR + 1)} = 25kHz \quad (4)$$

The duty cycle can be set putting in the CCRx register a percentage of the ARR, for example to obtain a rotational speed of 8000 RPM, the duty cycle must be ≈ 44 % and therefore the CCRx register is:

$$CCRx = TIM1_{ARR} * 0.44 = 844 \quad (5)$$

Figure 7. Fan PWM control signal

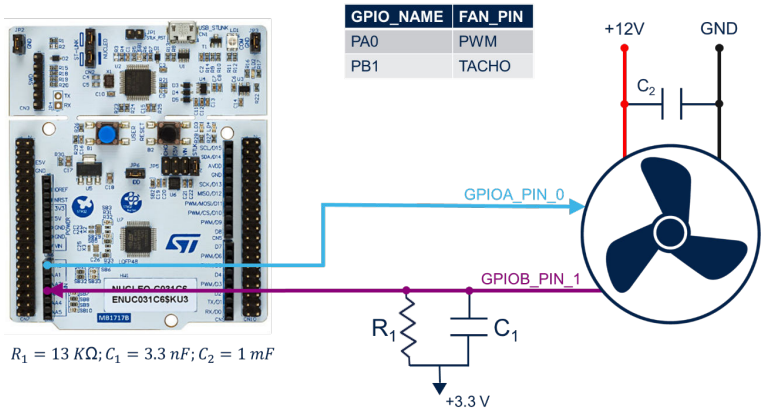


Note: the relationship between rotational speed and duty cycle is not perfectly linear, so you may need experimental measurements to find the right duty cycle value.

At the end of the day, any STM32 Nucleo board, that is any STM32 MCU, can be used to manage the fan because only two general-purpose timers are needed. Of course, the user may implement a PID controller or other type of feedback controller in the chosen STM32 MCU.

The following figure shows a reference schema to realize the validation setup.

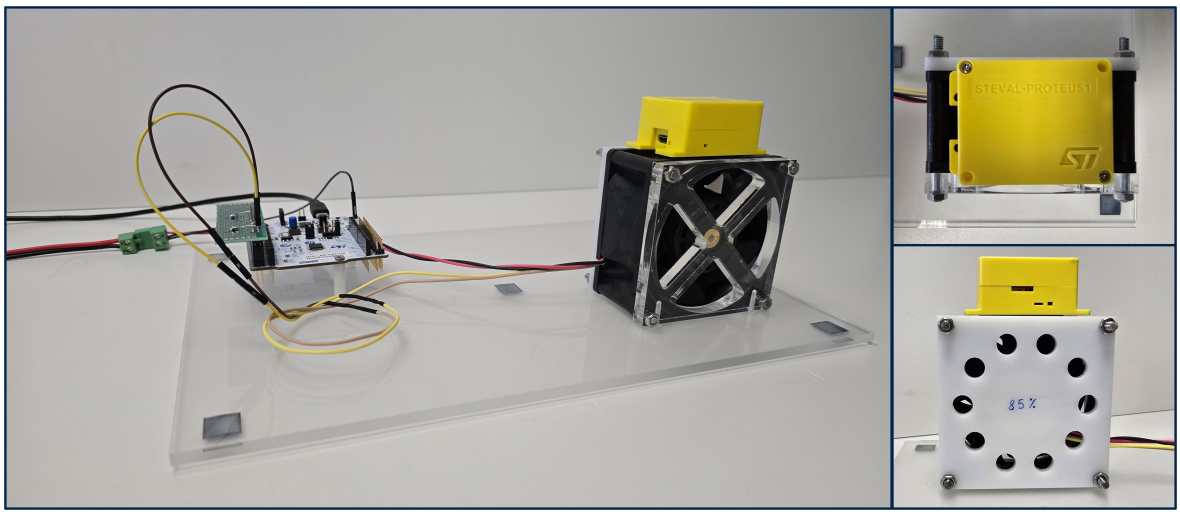
Figure 8. Validation setup schema



Typically, tacho signal requires a pull-up resistor because it is generated by an open collector circuit embedded in the fan and a capacitor might be useful to avoid some distortion on the line. The PWM signal generated by the MCU can be directly put on the fan PWM line.

An image of the setup is given below.

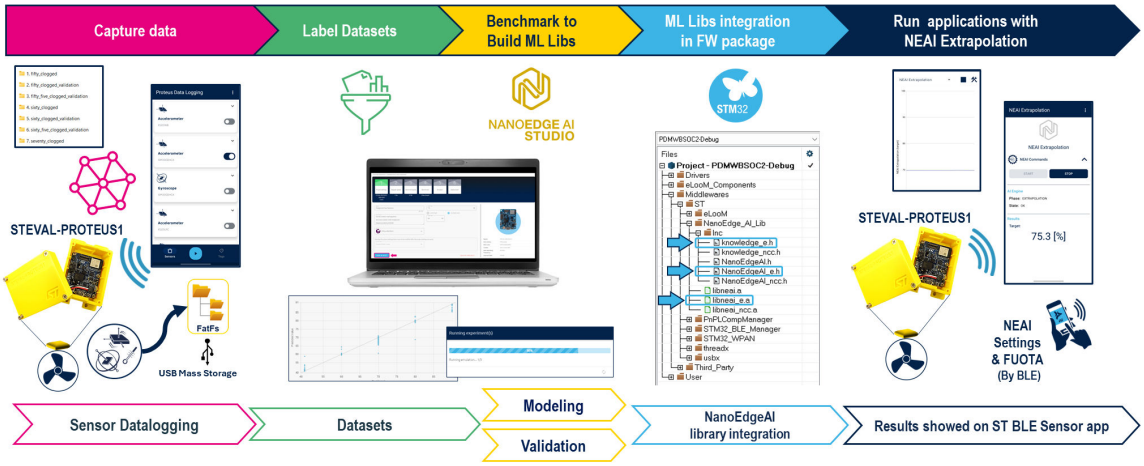
Figure 9. Validation setup photos



4 Regression model creation, deployment, and integration

The aim of this caption is to provide a complete description of the workflow that the user should follow to create an accurate regression model able to estimate the fan obstruction percentage.

Figure 10. Working flow



As you can see in the above figure, there are five mandatory steps to perform, starting with the data collection, then the usage of NanoEdge AI Studio to identify the best model and finally integrating and testing the generated extrapolation model on the validation setup.

4.1 Data log parameters selection and datasets acquisitions

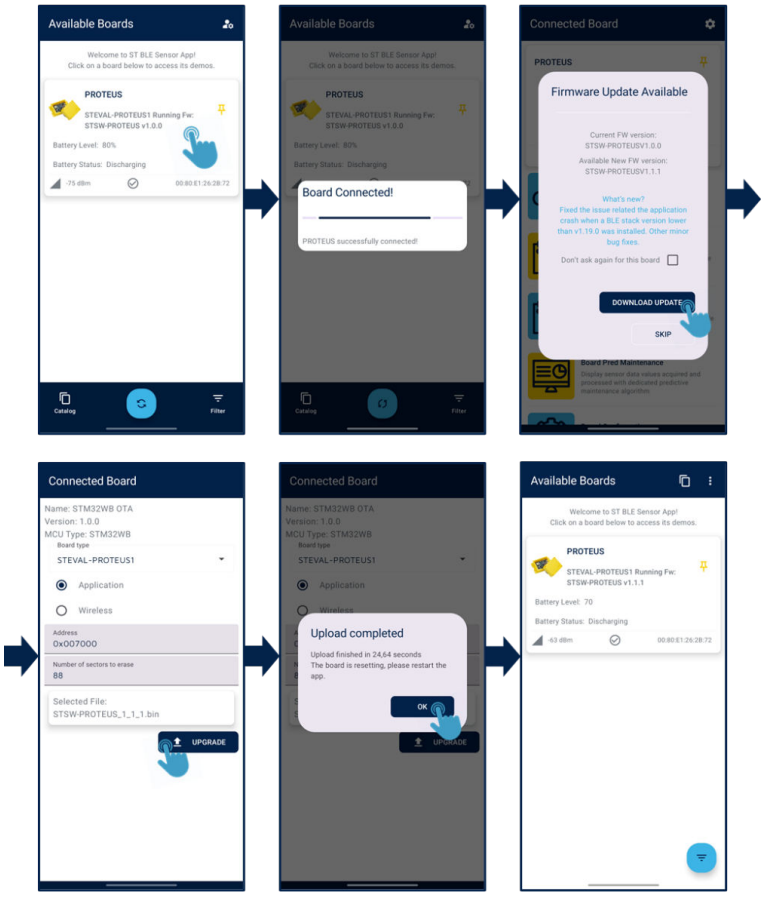
The dataset acquisition or its retrieval is the very first step of any ML approach. Data are the fundamental building blocks to create a reliable ML model, its quality, and generality are decisive for a high accuracy achievement, and they depend on the right data log parameters selection as well as the right sensor configuration. To estimate the fan obstruction percentage, it is suggested to use acceleration data. Accelerometer configuration mainly depends on two parameters: data rate and full scale.

Since we do not know in advance the spectrum of vibration generated in different obstruction conditions, one of the best strategies to select the right output data rate (ODR), and full scale (FS), is to observe the accelerometer data FFT in different obstruction conditions.

To evaluate the FFT of accelerometer data, STSW-PROTEUS v1.1.1 can be used. After unpacking your STEVAL-PROTEUS1 kit, you need to follow these steps:

1. Turn on the board pressing the PWR button until the yellow LED turns on.
2. Open the STBLESensor mobile app to find your PROTEUS board ready for the connection with your smartphone.
3. Tapping on the PROTEUS window, the connection between the PROTEUS and the smartphone takes place.
4. At this point, a firmware update pop-up appears and it is strongly suggested updating the firmware following the proposed steps.

Figure 11. **STSW-PROTEUS - firmware update**



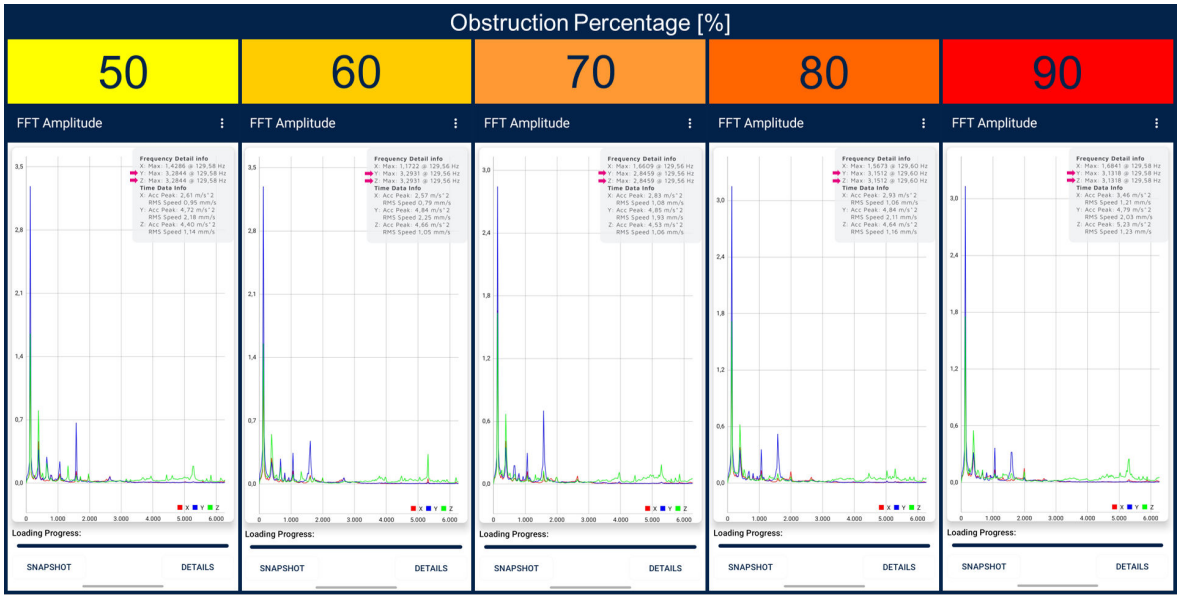
Take care: After the firmware update, you need to power off the PROTEUS by a long press on the PWR button, in addition you have to update the Bluetooth® Low Energy stack just in case you want to use low-power functionalities of the STSW-PROTEUS v1.1.1; the usage of these functionalities is not mandatory for the scope of this document.

Once STSW-PROTEUS v1.1.1 is flashed, you can connect to the PROTEUS board, select the *FFT amplitude* demo, and analyze the FFT results.

Figure 12. **STSW-PROTEUS - FFT analysis**

The following figure shows the FFT generated by STSW-PROTEUS firmware in the specified obstruction conditions.

Figure 13. FFT among different obstruction conditions



We can observe how the signal frequency spectrum is contained within 2.5 kHz and the maximum amplitude among recorded frequency components is about 3 m/s^2. According to these measurements, we can use the ISM330DHCX sensor, setting nominal ODR equals to 6.6 kHz and FS equals to 2 g. The use of IIS3DWB is not suggested because of its wide bandwidth that is useless according to the FFT spectrum observed, while the use of IIS2DLPC is not technically feasible because of its insufficient bandwidth.

Once the sensor configuration has been identified, the user can proceed to compute the number $N_{dataset}$ of datasets to acquire. $N_{dataset}$ can be computed using the following empiric formula:

$$N_{datasets} = N_{train\_examples} + N_{test\_examples\_on\_training\_conditions} + N_{test\_examples\_out\_of\_training\_conditions} \quad (6)$$

where:

$$N_{train\_examples} \geq 3 \quad (7)$$

$$N_{test\_examples\_out\_off\_training\_conditions} \geq 2 \quad (8)$$

For this document, we set: $N_{train\_examples} = 5$, $N_{test\_examples\_on\_training\_conditions} = 5$, $N_{test\_examples\_out\_of\_training\_conditions} = 4$.

The following tables aim to summarize what we have just defined:

Figure 14. Summary of dataset to collect

No. Dataset	Target	Train example
1	50.0	fifty_clogged
2	60.0	sixty_clogged
3	70.0	seventy_clogged
4	80.0	eighty_clogged
5	90.0	ninety_clogged

No. Dataset	Target	Test example on training condition
1	50.0	fifty_clogged_validation
2	60.0	sixty_clogged_validation
3	70.0	seventy_clogged_validation
4	80.0	eighty_clogged_validation
5	90.0	ninety_clogged_validation

No. Dataset	Target	Test example out off training condition
1	55.0	fifty_five_clogged_validation
2	65.0	sixty_five_clogged_validation
3	75.0	seventy_five_clogged_validation
4	85.0	eighty_five_clogged_validation

Notice that each row of these tables corresponds to a singular dataset which has to be acquired. In addition, for each row, the target value and the name of the dataset is specified: there are no setup differences between “fifty_clogged” and “fifty_clogged_validation,” between "sixty_clogged” and “sixty_clogged_validation,” and so on. Train examples, test examples in training conditions and test examples out of training conditions, are used at different times. The last statement is clearer in the following paragraphs.

Another parameter that the user has to compute is the buffer length, in fact, every dataset row has a length equal to the buffer length.

First, assuming to set the rotational speed equal to 8000 RPM, the fundamental frequency which we want to sample is equal to $f_s = 133Hz$. The corresponding period is $T_s = 7.5 \quad ms$.

We can apply the following formula to compute the buffer length:

$$Buffer\_lenght \geq (ODR_{nominal}{*}T_s){*}3{\sim}150 samples/buffer \rightarrow 256 samples/buffer \quad (9)$$

Since we must choose a multiple of 2 as buffer length, we can approximate to the higher multiple which is 256 samples/buffer.

Frequently, NanoEdgeAIStudio models, and in general ML approaches, are preceded by FFT computation and the more samples/buffer increases the more FFT resolution increases as well. Because of this relation, a good practice is to create several projects starting from a buffer length equal to the previous result (buffer length = 256) and going to higher values as much as memory footprint, inference time, and power consumption are sustainable.

For the use-case in question, considering STM32WB memory capabilities, its clock at 64 MHz, and abstracting from specific power consumption constraints, we choose to create three models respectively based on 256-512-1024 samples/buffer.

To avoid wasting time in dataset acquisition, we can set the acquisition time considering the highest buffer length, that is 1024, and assuming that we want to collect 1000 buffer/dataset:

$$samples = buffer\_length{*}buffers = 1024{*}1000 = 1,024,000 \quad (10)$$

$$acquisition\_time = \frac{samples}{ODR_{real}} = \frac{1024{,}000}{7200{*}} \approx 150s \quad (11)$$

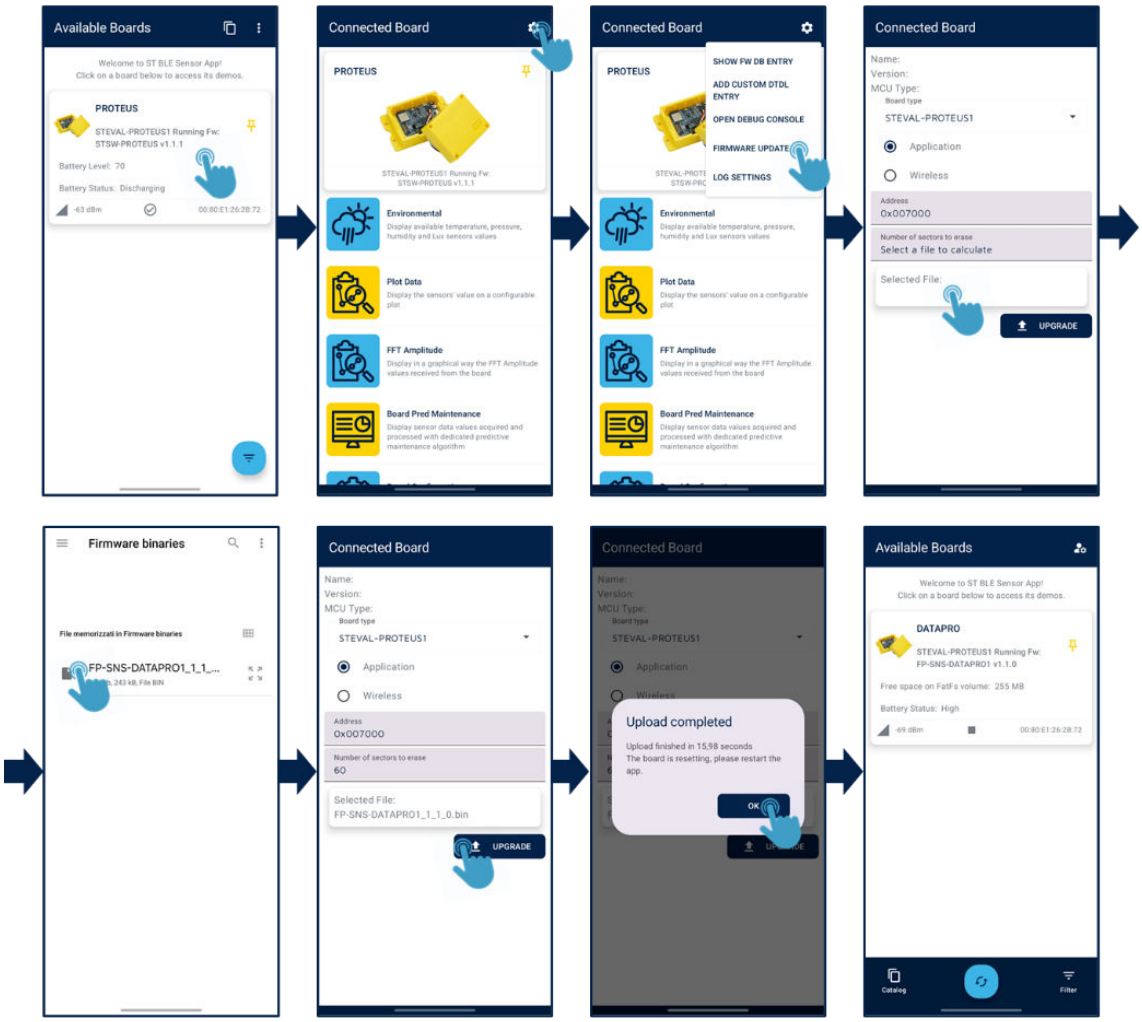
The above-mentioned acquisition time regards only the datasets used to train the ML model (train examples), for the rest of the datasets the acquisition time can be set to 10% of the computed acquisition time for training examples collection, that is 15 seconds.

To generate the $N_{train\_examples}$, $N_{test\_examples\_on\_training\_condition}$, and $N_{test\_examples\_out\_off\_training\_condition}$, the user can use FP-SNS-DATAPRO1 v1.1.0.

Following these steps, you can flash the DATAPRO1 application by FUOTA:

1. Connect to the PROTEUS board
2. Tap on the gear icon and the *FIRMWARE UPDATE*
3. Select DATAPRO1 v1.1.0 binary and tap on *UPGRADE* button

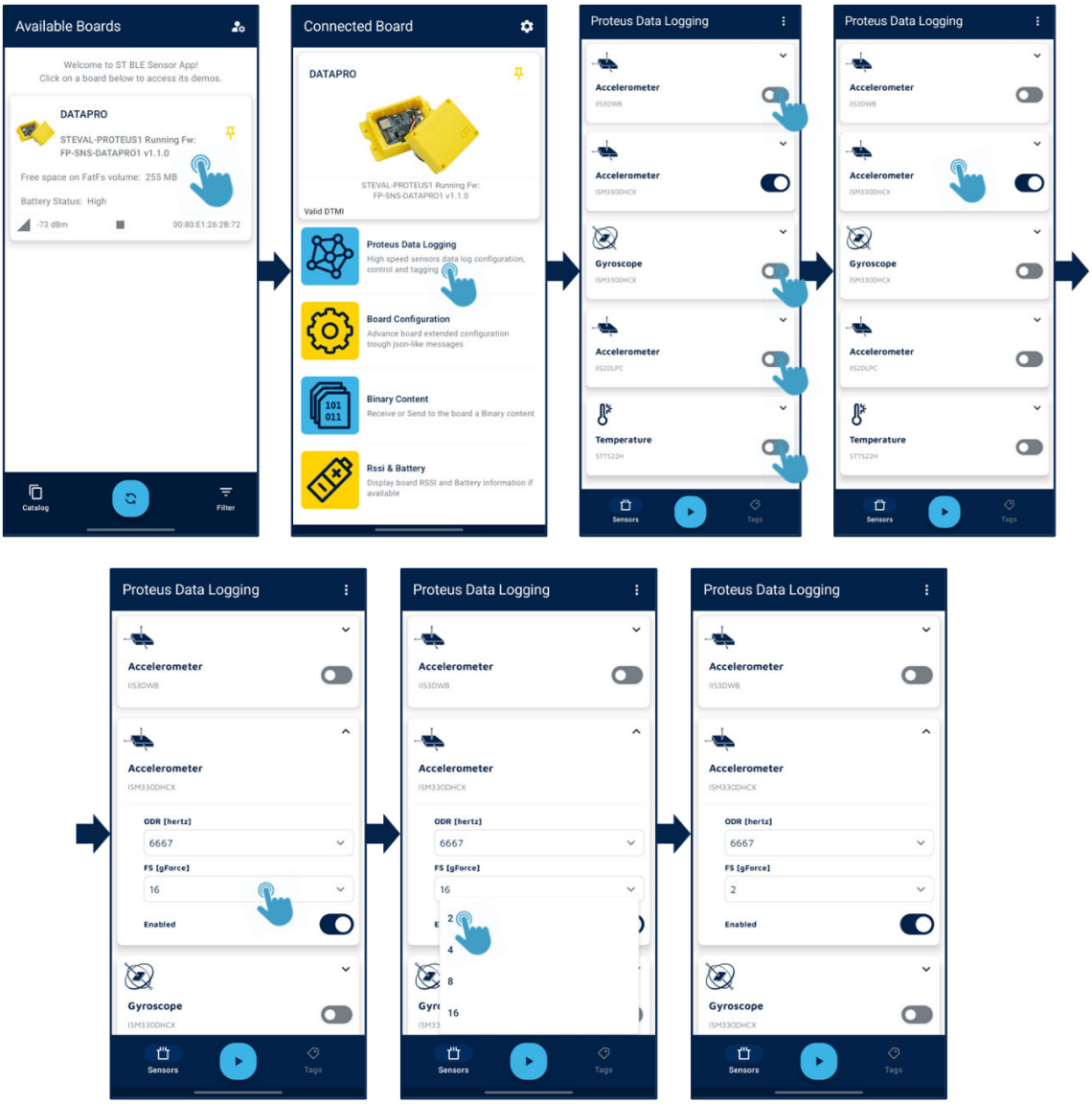
Figure 15. **FP-SNS-DATAPRO1 - firmware update**



As explained in Section 2.2.2: FP-SNS-DATAPRO1, the DATAPRO1 application allows to acquire datasets in different ways. In the following we describe how data logging can be configured and started via Bluetooth® Low Energy connectivity through the STBLESensor app.

Sensor configuration can be customized using the *Proteus data logging* demo as shown in the figure below:

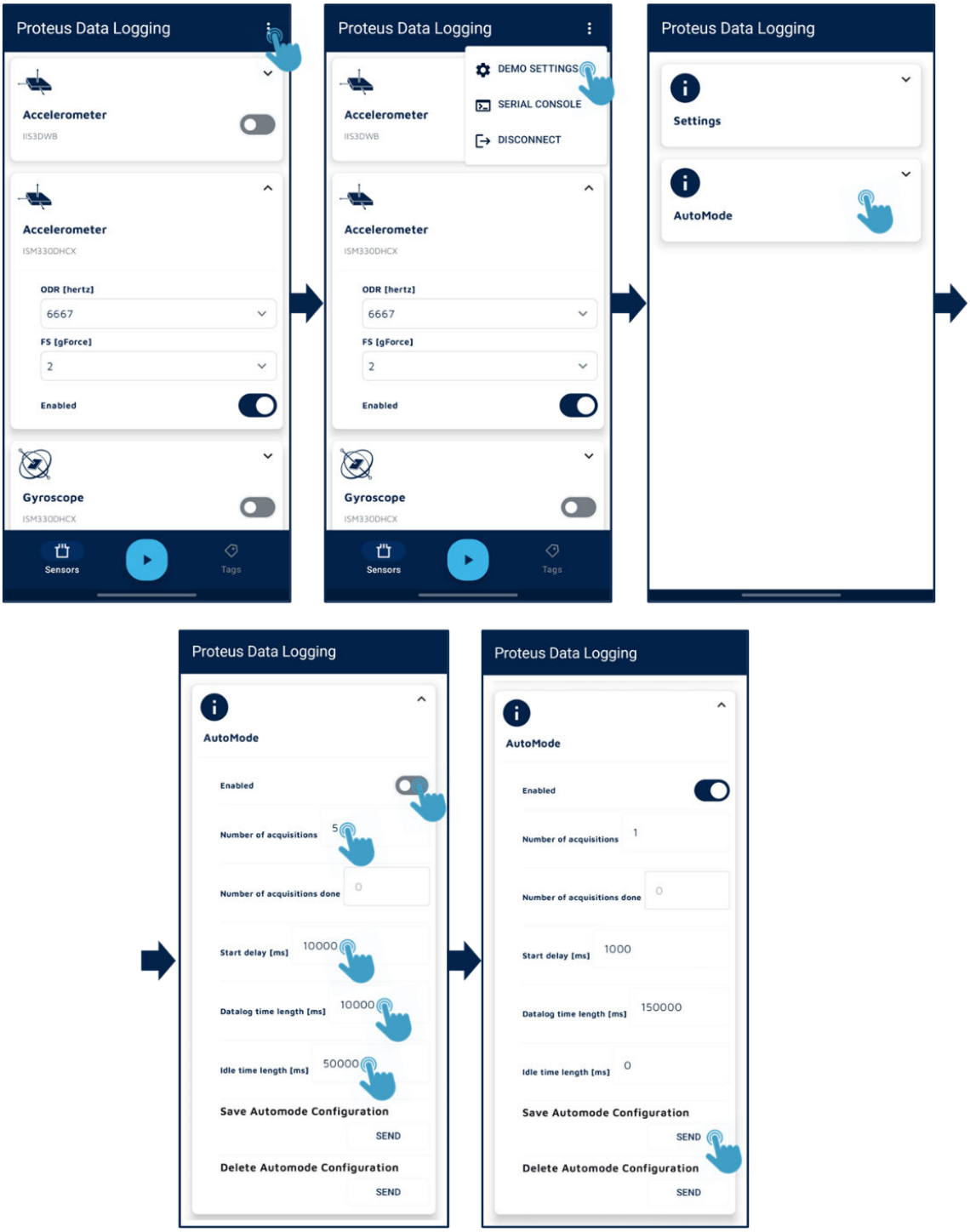
Figure 16. **FP-SNS-DATAPRO1 - sensors configuration**



ISM330DHCX has been configured according to the parameters computed before (Nominal ODR = 6667 Hz, FS = 2 g), while all other sensors have been disabled.

To automatically stop data acquisition after 150 seconds, the user can enable and set the *Automode* functionality using the dedicated page inside the *Proteus data logging* demo.

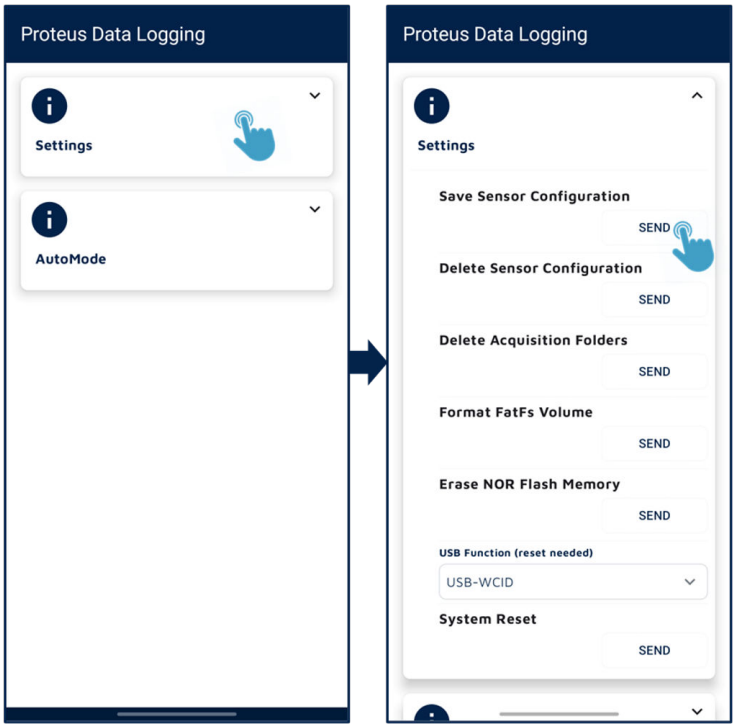
Figure 17. **FP-SNS-DATAPRO1 - automode configuration**



The *Save Automode Configuration* button allows to permanently store the automode parameters inside the onboard external NOR flash memory.

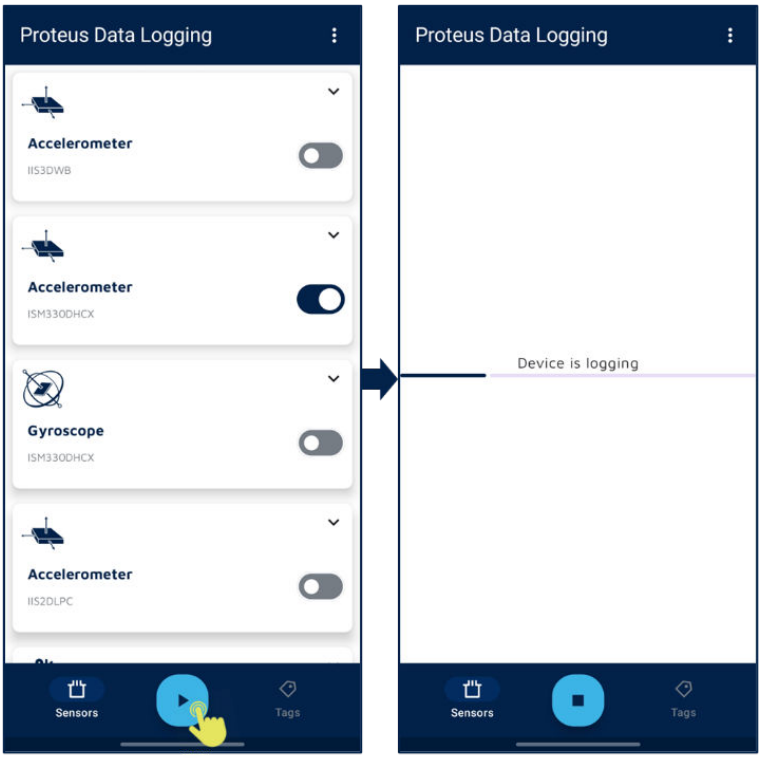
Furthermore, the user can permanently store the sensor configuration too, using *Settings* properties shown in the figure below:

Figure 18. **FP-SNS-DATAPRO1 - saving sensors configuration**



At this point, through the play button on the *Proteus data logging* demo, the user can launch the dataset acquisition which is started after the *Start delay*. It is automatically stopped after the *Datalog time length* and repeated according to the *Number of acquisitions* spaced out of *Idle time length*.

Figure 19. **FP-SNS-DATAPRO1 - starting sensor data collection**

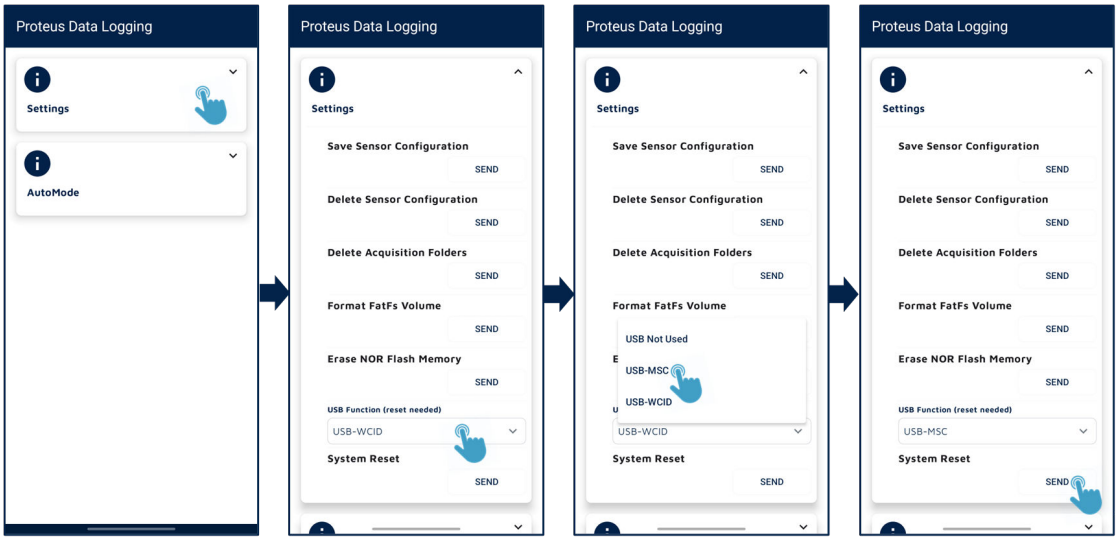


At the end of the acquisition campaign, the external NOR flash memory of the STEVAL-PROTEUS1 contains the generated acquisition folders.

As explained in Section 2.2.2: FP-SNS-DATAPRO1, the DATAPRO1 application can make the STEVAL-PROTEUS1 act as a USB mass storage and allow to retrieve acquisition folders stored inside the external NOR flash memory.

The figure below shows the steps to be performed before connecting the STEVAL-PROTEUS1 to a PC and copying/pasting the generated datasets.

Figure 20. FP-SNS-DATAPRO1 – enabling USB Mass Storage Class (USB-MSC)

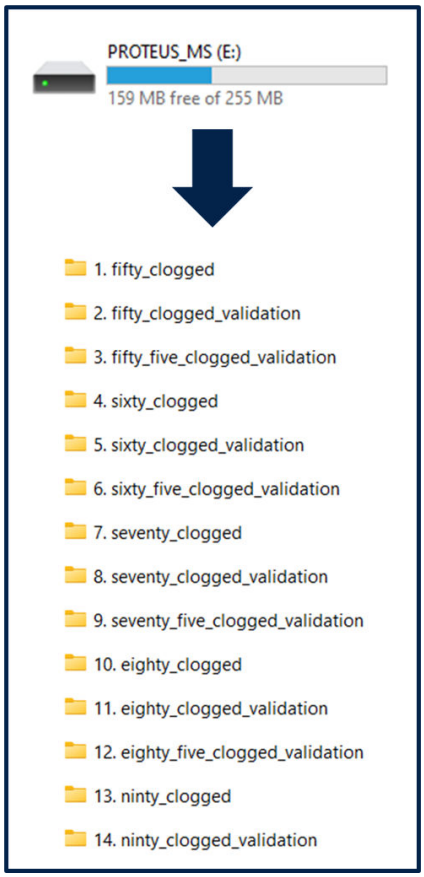


Pay attention to the last step: to execute the *System Reset* is mandatory to enable mass storage functionality.

To know more about the possibility to send the collected datasets via Bluetooth® Low Energy, refer to UM3396.

According to Figure 14. Summary of dataset to collect, we obtained the following folders:

Figure 21. FP-SNS-DATAPRO1 – acquired dataset folders on file manager



Note: the folders reported in the figure above have already been labeled (go to Section 4.2: Data labeling and shaping for more information about data labelling).

Each of the obtained folders contain, other than a .dat file with acquired sensor data, two JSON files which describe, respectively, the acquisition time and date, and the sensor configuration during the data acquisition, they are very useful when doing a lot of acquisitions, and need to verify the acquisition setup of every single one.

The validation acquisition folders correspond to 15-second data acquisition, while all the other folders correspond to 150-second data acquisition.

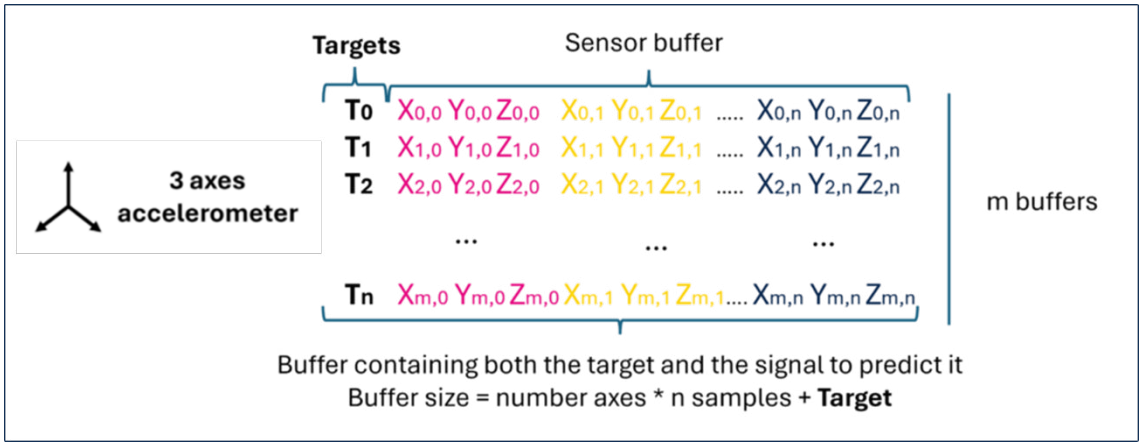
4.2 Data labeling and shaping

The first thing to do at the end of data acquisition is to label data, that is, to rename all the acquisition folders according to the related setup condition.

Renaming acquisition folders is as simple as it is important because issues around renaming cause bad model performance. Therefore, pay close attention during the labeling process.

Since NanoEdgeAIStudio does not accept *.dat files as input datasets, when all the folders have been labeled, the user must convert them into *.csv files formatted as shown below:

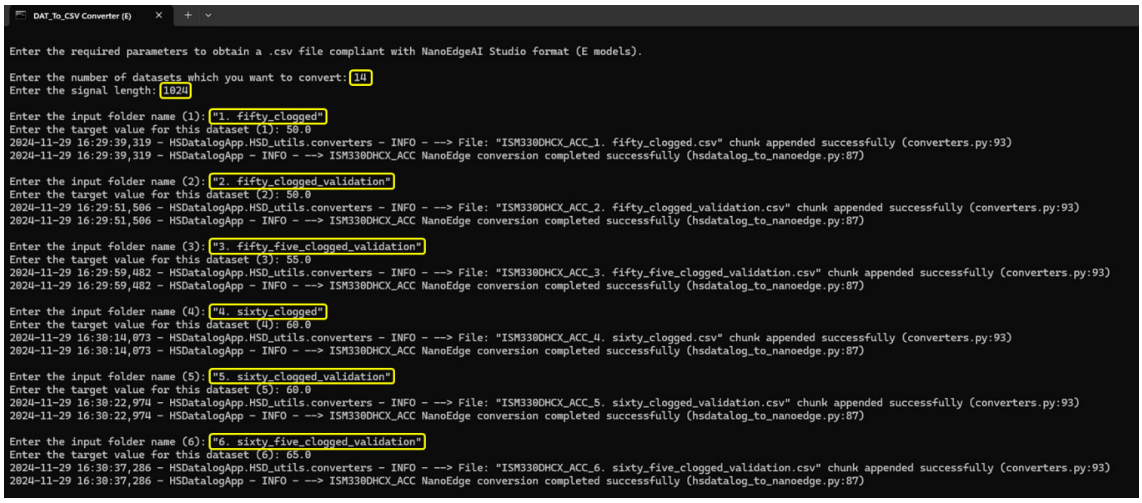
Figure 22. **NanoEdgeAI extrapolation dataset format**



The easiest way to convert your datasets to make them compliant to the NanoEdgeAI format is to apply the ready-to-use batch file located in FP-AI-PDMWBSOC2 v2.0.0 inside the folder:
$PROJ_DIR$\Utilities\SwUtilities\HS_Datalog\PROTEUS_batch_file_examples,
the *HS-DL_NanoEdge_Conversion_E.bat* is placed here.
After launching the *HS-DL_NanoEdge_Conversion_E.bat*, the user needs to enter the following information:

1. The quantity of acquisition folders that they want to convert in *.csv file.
2. The signal length, which is the buffer size.
3. The input folders' names.
4. The target value that corresponds to the dataset specified under the previous point.

Figure 23. **Running the DAT to CSV format converter**



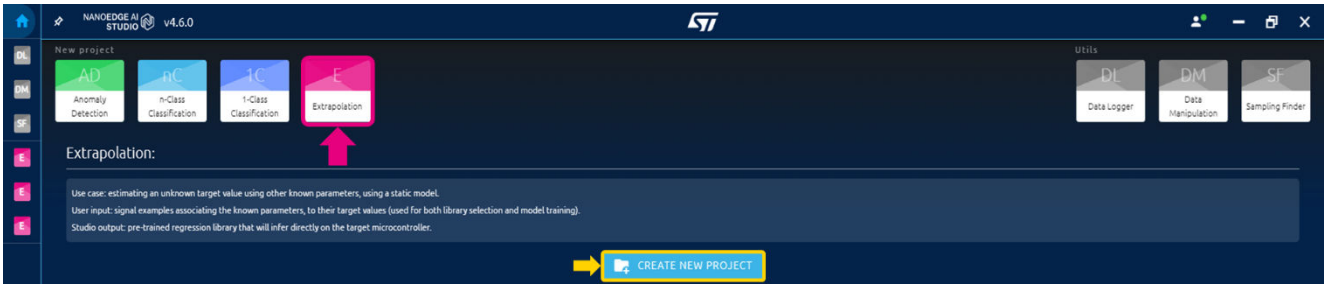
When conversions are completed, the user finds a *.csv file inside each folder acquisition, and they can be directly imported into the NanoEdgeAIStudio.

4.3 Training and cross-validation

When all the datasets have been acquired, the user can proceed with the generation of their own ML model. If you do not have the specific skills, designing an ML model could be very difficult and time consuming. NanoEdgeAIStudio is an ST free software tool which allows to design and to find the best model typology without any skills in machine learning. NanoEdgeAIStudio can generate different types of models, and according to the aim of this document, extrapolation ones are treated in the following paragraphs. Extrapolation models, commonly known as regression models, predict discrete values of a dependent variable *Y* processing one or more independent variables in the input. For our application use-case, the dependent variable is the fan obstruction percentage while the independent variables are the buffers filled with acceleration on the x-y-z axes.

To generate the extrapolation model, open NanoEdgeAIStudio and start by creating a new extrapolation project as shown in the following figure.

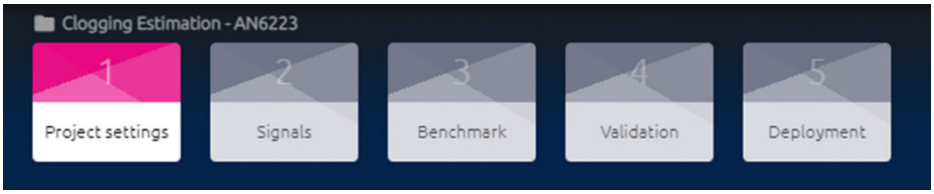
Figure 24. NanoEdge AI Studio - starting new extrapolation project



The text shows the information on what the extrapolation is good for. The project can be created by clicking on the *CREATE NEW PROJECT* button.

The process to generate the extrapolation library consists of five steps as shown below.

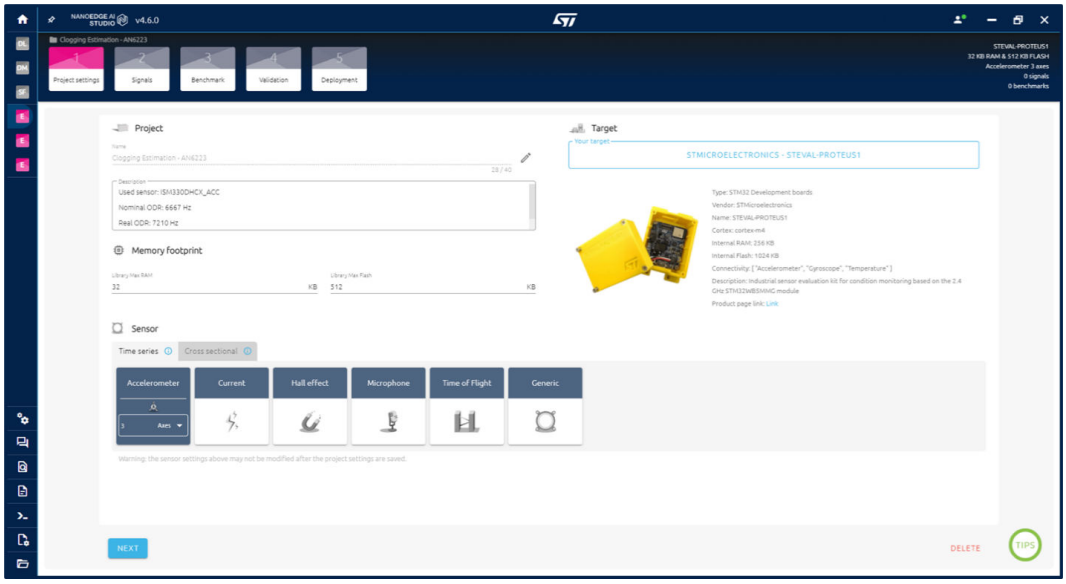
Figure 25. NanoEdge AI Studio - extrapolation project steps



In the first one, the user has to enter some preliminary information which is summarized in the following points:

- Enter the project name and description.
- Select the *STEVAL-PROTEUS* from the list provided in the dropdown menu under Target.
- Enter the maximum amount of RAM to be allocated for the library. Usually, a few Kbytes are enough, but we can set 32 Kbytes to support models which require additional RAM: this choice is feasible because we are focused on extrapolation avoiding the memory consumption of other kinds of models.
- Enter the maximum amount of FLASH to be allocated for the library. Usually, 100 Kbytes are enough, but we can set 512 Kbytes to support models which require additional FLASH: this choice is feasible because we are focused on extrapolation avoiding the memory consumption of other kinds of models.
- Select the sensor type through the list in the dropdown menu: according to our data collection, the 3-axis accelerometer was selected.

Figure 26. NanoEdge AI Studio - extrapolation project setting

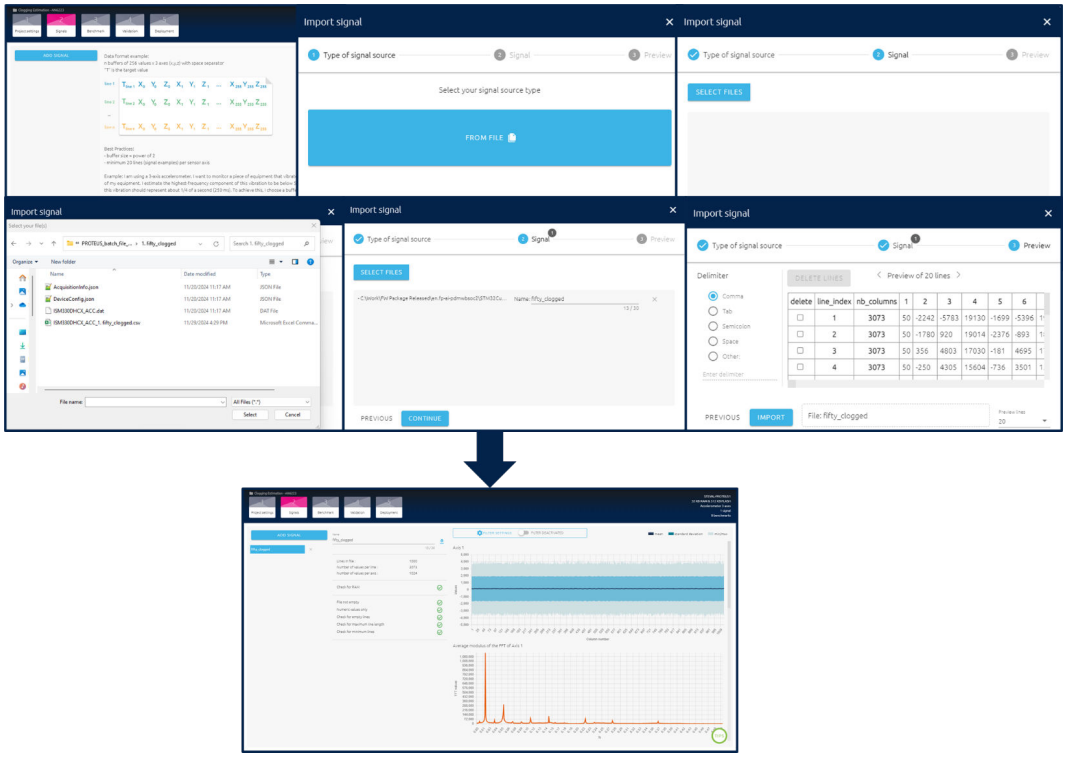


Clicking on the next button, a new page is displayed, and the user can add their own signals.

The figures which are shown in the following pages are intended as reference for the extrapolation model creation and they refer to one of the projects which has been created, the one based on 1024 buffer length. In the last section dedicated to the results, some comparisons between 256/512/1024 buffer length-based projects are placed.

The figure below shows how a dataset (a.k.a. signal in NanoEdgeAIStudio) can be imported in the NanoEdgeAIStudio project; the procedure must of course be repeated to load all the five train examples.

Figure 27. NanoEdge AI Studio - signals importing

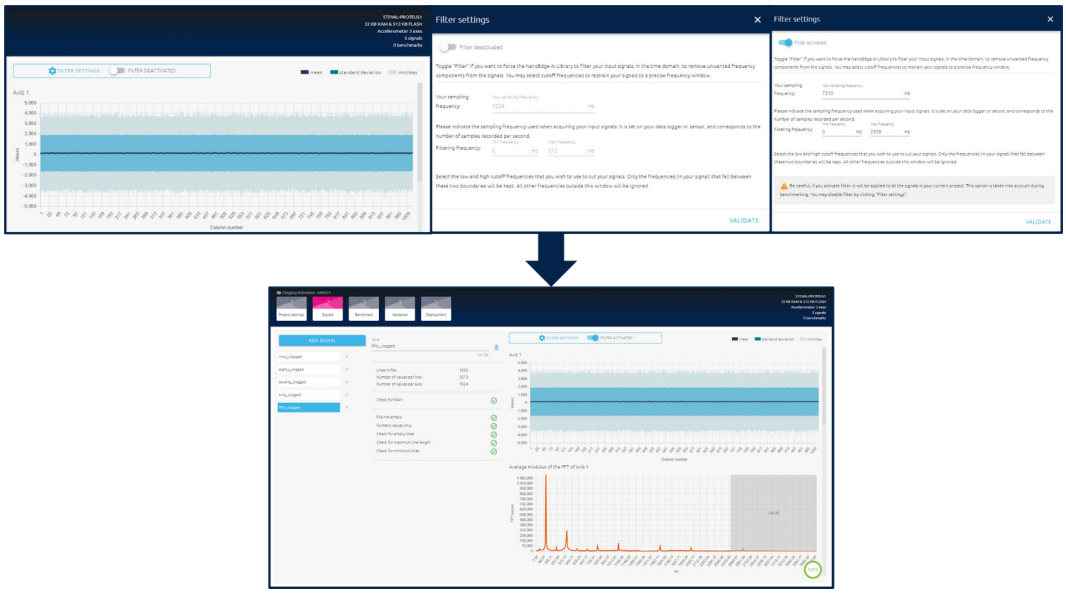


When a signal is imported, NanoEdgeAIStudio performs some check to ensure that the signal has no issues.

Check for RAM and the next five checks are blocking, if one of these checks fails, an error is returned, and the user is not able to use the signal as it is. Furthermore, the user can execute some additional checks to find duplicate, equal consecutive values, outliers, or other bad data inside the imported signal. Take care regarding the number of lines which signals have because data should be balanced among your training examples, therefore you should have the same number of lines for each signal.

For each axis, a time-domain and a frequency-domain graphs are displayed, which are useful to evaluate the signal bandwidth, allowing to properly set a frequency filter if needed. Observing the collected signals, the harmonic content above 2.5 kHz is very low, as already verified in the performed preliminary analysis (see 4.1), and because of that, we can set the frequency filter as shown in the figure below.

Figure 28. NanoEdge AI Studio - signals filter setting



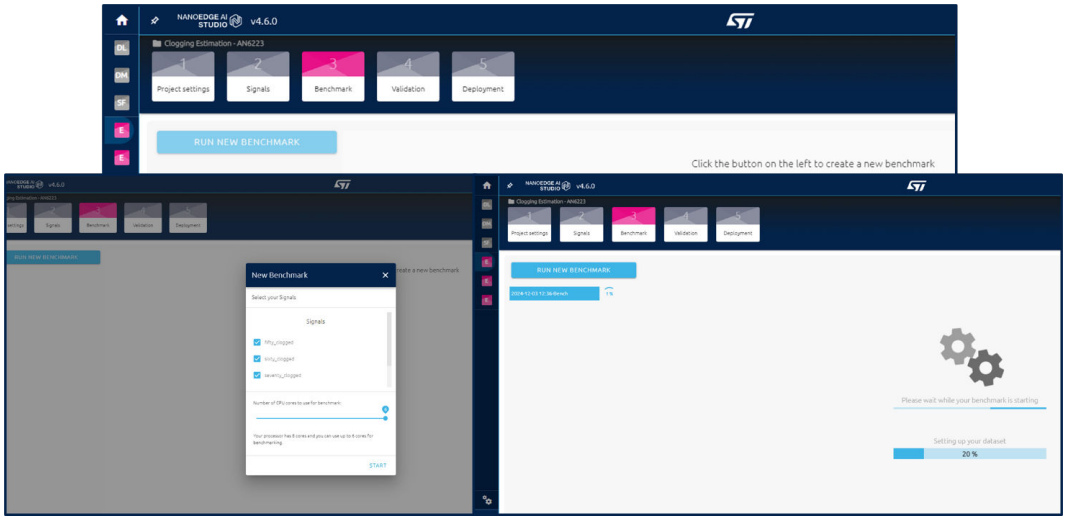
Specifying the cut-off frequency is mandatory to know the real sampling frequency. The user can find it, that is the measured ODR, in the *DeviceConfig.json* placed on every acquisition folder.

After signals import, the user can proceed to run the benchmark process. The benchmark is the most important phase of ML model generation. NanoEdgeAIStudio uses the signal examples imported in the previous step to automatically search for the best possible model according to several performance indicators (for example, R-Squared, RAM, and FLASH consumption).

To start the benchmark, the user must:

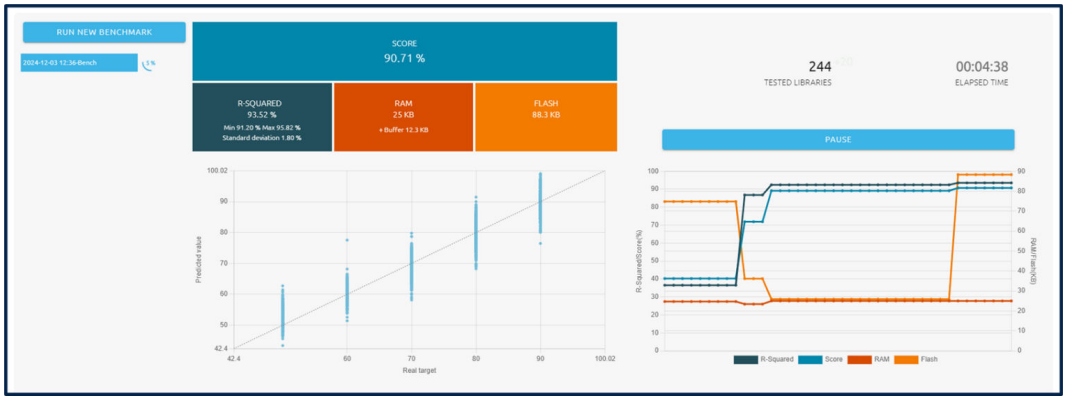
- Click on *Run New Benchmark*
- A new window appears that allows you to select input files (signal examples) to use, and to change the number of CPU cores to use.
- Start and wait until you see the perspective shown below.

Figure 29. **NanoEdge AI Studio - starting new extrapolation benchmark**



When the benchmark is ongoing, performance indicators of the best model found so far are displayed as shown in the figure below.

Figure 30. **NanoEdge AI Studio - extrapolation benchmark indicators**



The following are reported:

- R-Squared, that is the coefficient of determination. It provides a measurement of how well the observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model.
- RAM, that is the maximum amount of RAM memory used by the library when it is integrated into the target microcontroller.
- FLASH, that is the maximum amount of flash memory used by the library when it is integrated into the target microcontroller.
- Score, which depends on the value of the other indicators.

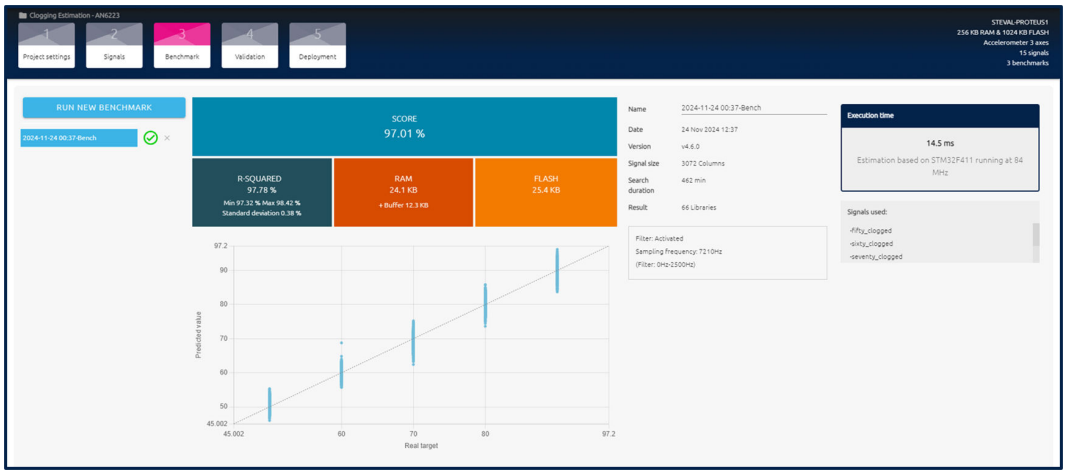
During the benchmark, NanoEdgeAIStudio performs a cross-validation of several models to reduce the overfitting phenomena.

Take care to the following aspects:

- Benchmarks may take a long time (several hours) to complete and to find a fully optimized library.
- Benchmarks can be paused/resumed, or stopped at any time, without canceling the process (the best library found is not lost).
- Benchmark progress in percentage is displayed on the left side of the screen, next to the name/ID of the benchmark.

In this specific application use-case, the benchmark took almost 8 hours to complete. The time required to complete it is influenced by several factors such as the quantity of lines for each dataset example and the computational capabilities of the PC used. The benchmark result is reported below:

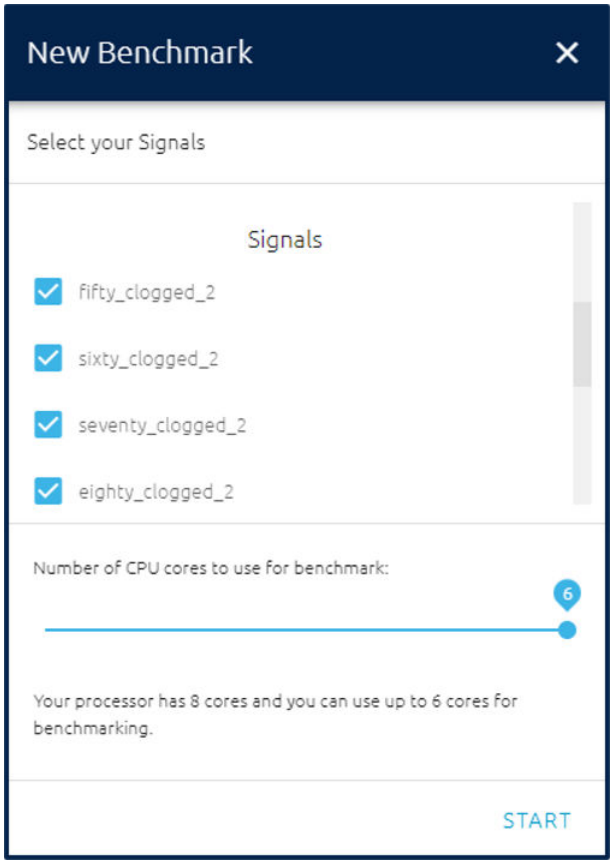
Figure 31. NanoEdge AI Studio - extrapolation benchmark results



A common solution to improve the result coming from the first benchmark performed is to run other benchmarks using new datasets which are referred to the same setup conditions (50-60-70-80-90 % fan clogged).

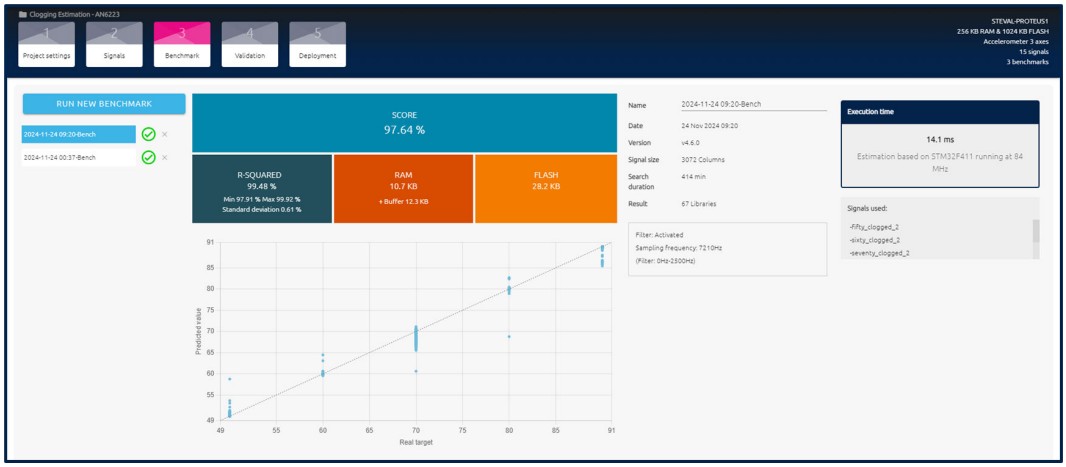
Therefore, the user needs to acquire new datasets, going back to the signal import step, enter the new datasets collected and then start a new benchmark selecting the 50-60-70-80-90 % fan obstruction signals just imported.

Figure 32. NanoEdge AI Studio - starting the second extrapolation benchmark



After the second benchmark execution, we obtained the following result:

Figure 33. **NanoEdge AI Studio - second extrapolation benchmark results**

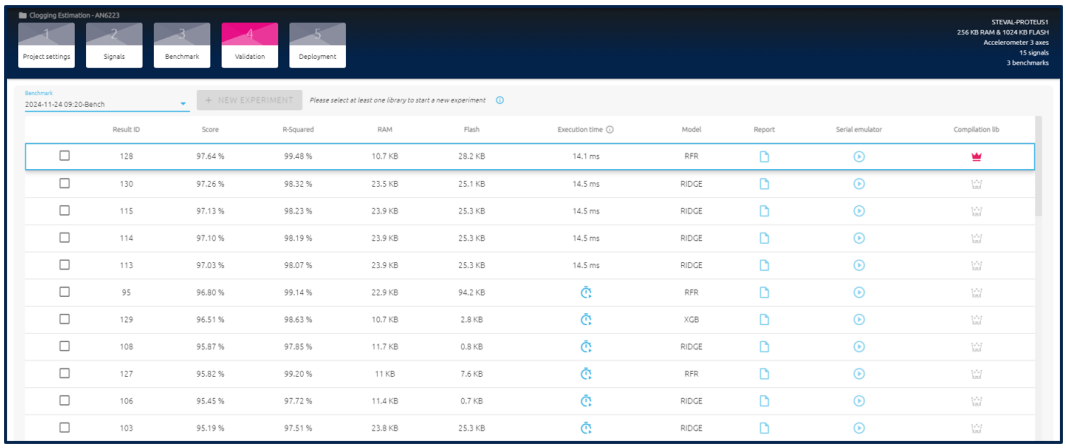


The R-Squared of the best-found model increased from 97.78% to 99.48% while the RAM memory footprint decreased from 24.1 Kbyte to 10.7 Kbyte and the FLASH footprint has remained almost unchanged. Furthermore, the benchmark process took more than 6.5 hours.

4.4 Best models validation

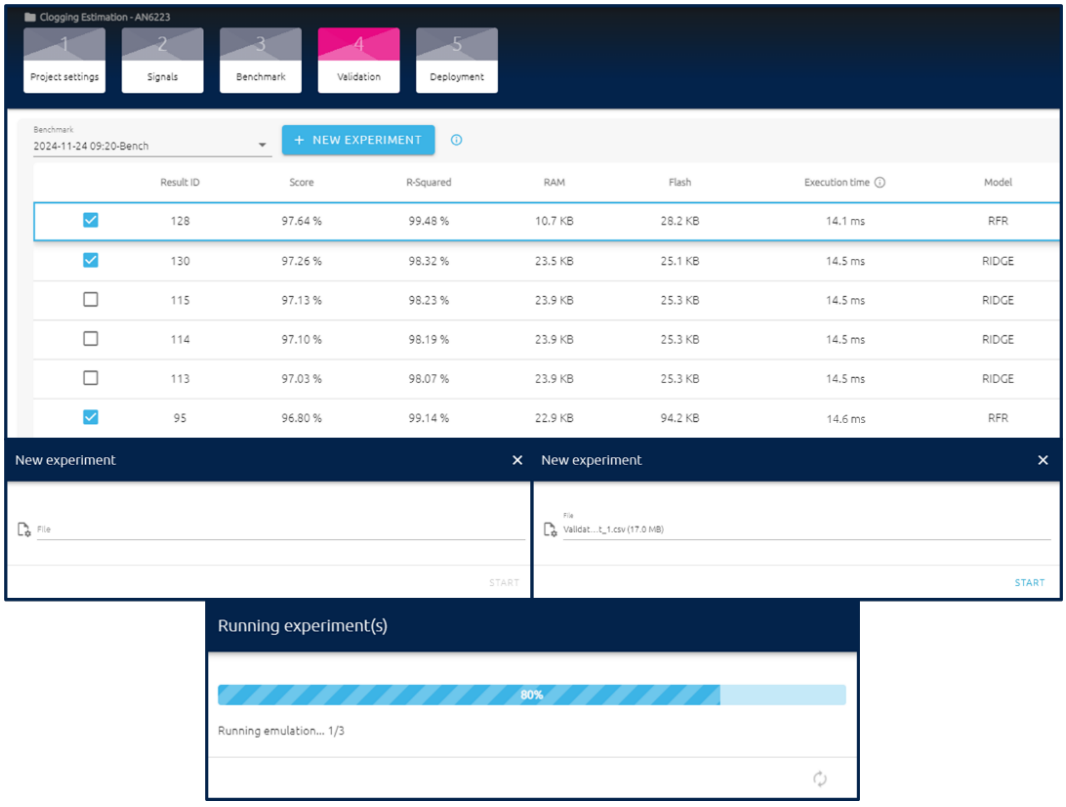
When the benchmark ends, even if the user accepts the best library proposed and directly integrates it on a firmware project, to perform one or more validation experiments is more than recommended. Moving to the fourth step of the NanoEdge AI Studio workflow, a model ranking appears giving the best models performance summary.

Figure 34. **NanoEdge AI Studio - models ranking**



Here the user can select up to 10 models and compare their performances on the validation datasets previously acquired. We have selected the model identified by the IDs 128-130-95.

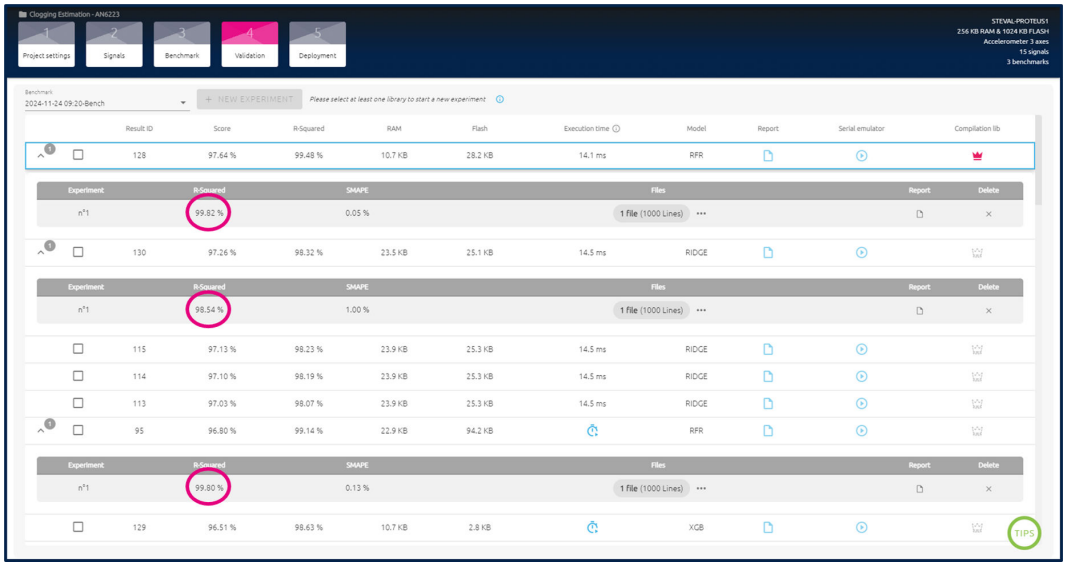
Figure 35. NanoEdge AI Studio - starting validation experiment



The selected models are Random Forest Regressor (RFR) and RIDGE regressor (RIDGE).

Clicking on the *NEW EXPERIMENT* button a new window appears allowing to choose your validation dataset. The validation dataset has to contain a certain number of lines corresponding to different target values. We enter on the validation dataset a 200 lines/clogging condition obtaining 1000 lines on the entire .csv file including 50-60-70-80-90% obstruction percentage data.

Figure 36. NanoEdge AI Studio - validation experiment results



After the experiment execution, new gray lines appear with the performance indicators re-computed on the inserted validation dataset. These results can confirm or disapprove the performance estimated at the end of the benchmark process according to the fact that the model has generalized enough or not, in the last case an overfitting occurred.

The figure below demonstrates that the selected libraries are robust in terms of R-Squared indicator and the 128 ID model is the best. For a deeper validation, a good solution is to perform a second validation experiment using test examples out of training conditions, that are 55-65-75-85% obstruction percentages datasets. The following figures show the model selection, experiment execution, and results using a new validation dataset made up of test examples out of training conditions. This second validation dataset contains 200 lines/clogging condition obtaining 800 lines on the entire .csv file including 55-65-75-85% obstruction percentage data.

Figure 37. NanoEdge AI Studio - starting second validation experiment

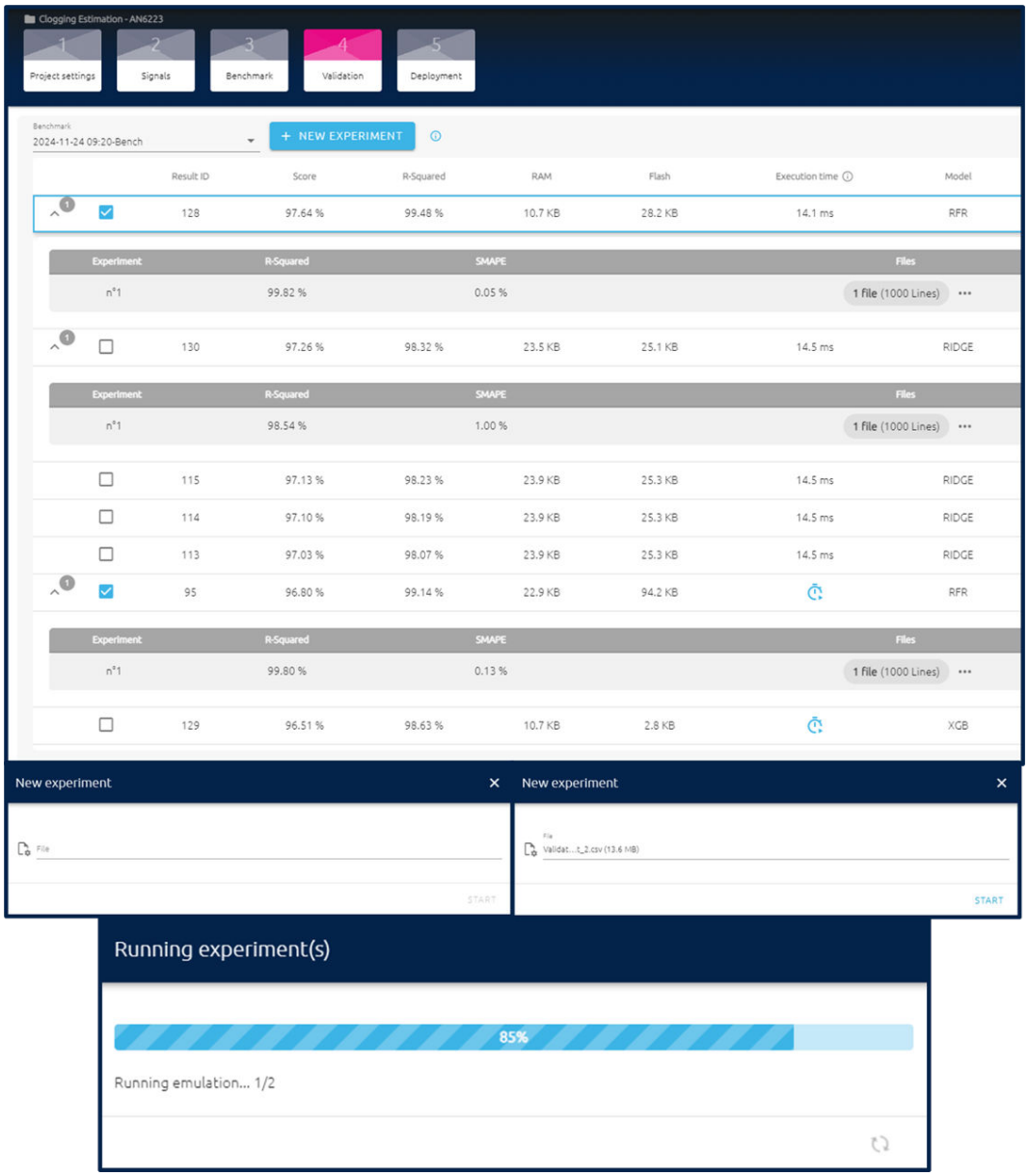
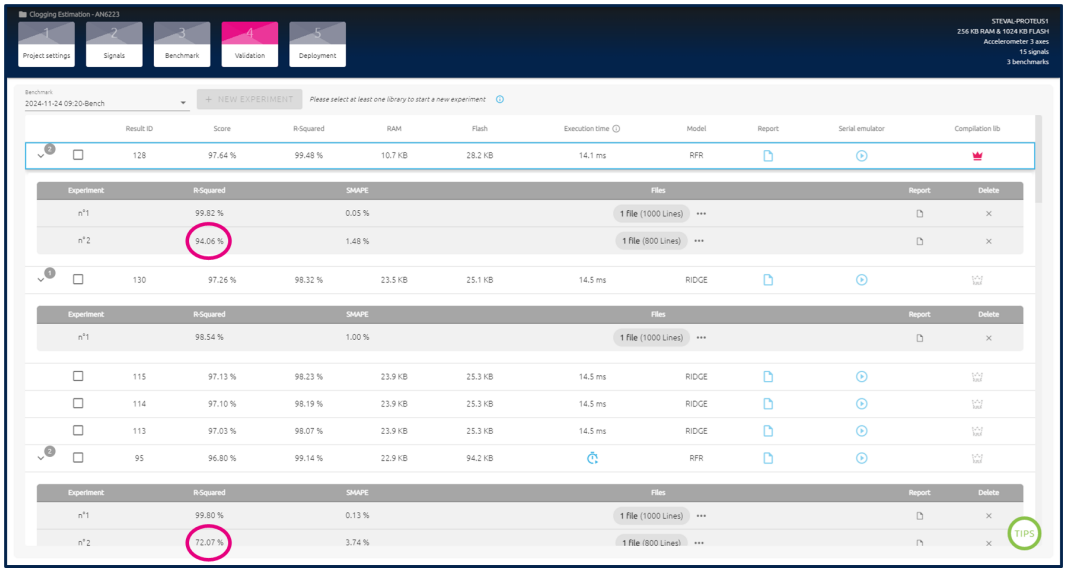


Figure 38. **NanoEdge AI Studio - second validation experiment results**



The second experiment is aligned with the results coming from the first one: 128 lib model is the best also for the estimation of out-of-training target values; it has been chosen to deploy on STM32WB5MMG.

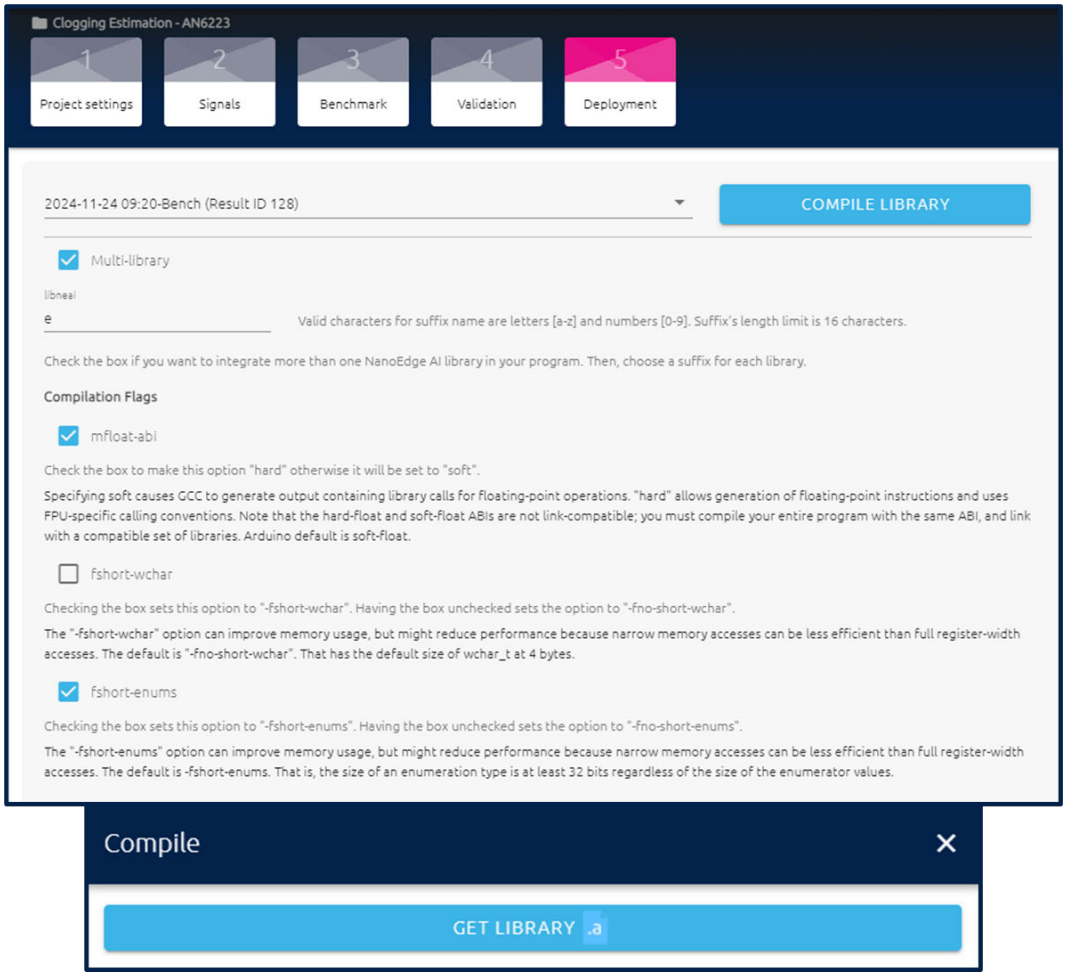
4.5 Library deployment

The last step for the NanoEdgeAIStudio tool is to compile and to release the selected library in a unique ZIP file containing the library model in compiled format, plus other files to complete the integration. Moving to the deployment phase, the user can generate the libraries' files.

For the extrapolation model deployment:

- Set the multi-library flag, to enable the suffix line insertion, and insert “e” just to identify these extrapolation files from others to insert in the application. Make sure to check the fshort-wchar option if the library is integrated in a Keil® project and leave the other flags in their default state.
- Push the *Compile Library* button.
- Push the *Get Library .a* button.

Figure 39. **NanoEdge AI Studio - extrapolation model deployment**



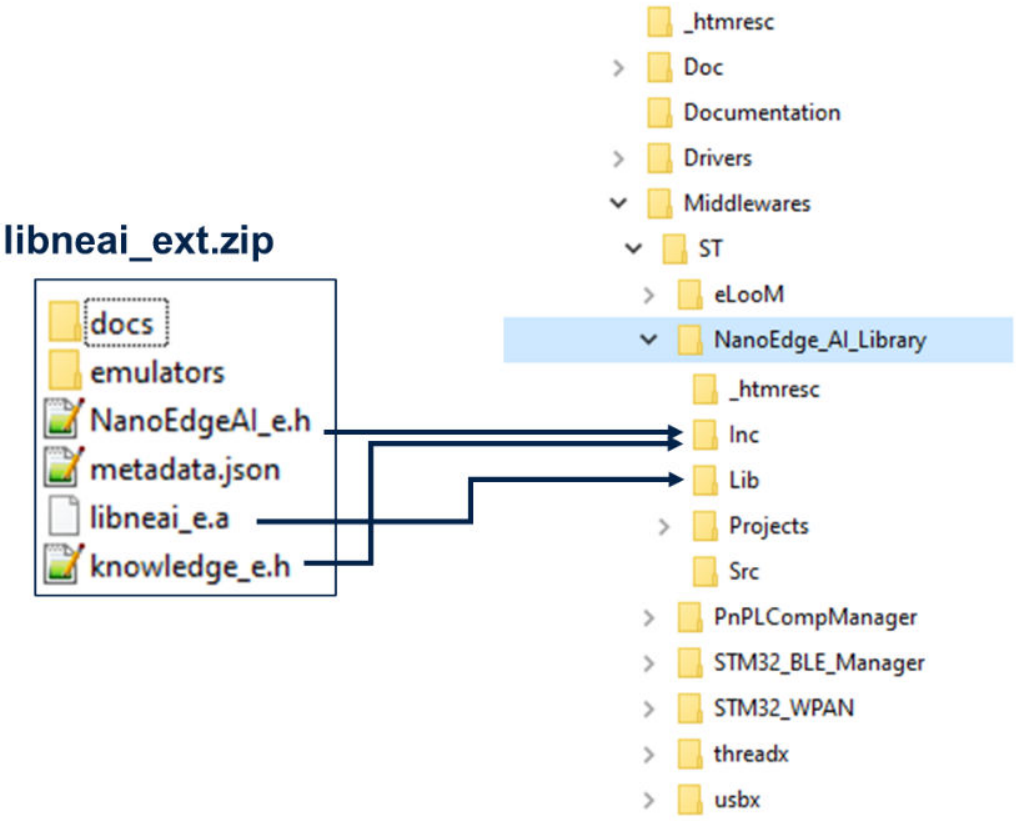
4.6 Library integration on FP-AI-PDMWBSOC2

The PDMWBSOC2 firmware example released on st.com contains three “stub” demonstration libraries, useful just to verify the functionality and behavior of, respectively, Anomaly Detection, N-Class Classification and Extrapolation. These libraries must be replaced by the libraries generated through NanoEdge AI Studio by the user.

To integrate its extrapolation library, the user needs to open the ZIP file generated after the deployment phase, then to follow these steps:

- Open the following path in the project folder: $PROJ_DIR$\Middlewares\ST\NanoEdge_AI_Library\Lib
- Replace the existing stub library with the deployed library:
  - libneai_e.a (for IAR and STM32CubeIDE projects)
  - libneai_e.lib (for Keil® projects)
- Open the path: $PROJ_DIR$\Middlewares\ST\NanoEdge_AI_Library\Inc
- Replace header files with new deployed:
  - NanoEdgeAI_e.h
  - knowledge_e.h
- Compile again the firmware application using one of the supported IDE in order to update the application binary with the new generated library.
- Update the STEVAL-PROTEUS application firmware via Bluetooth® Low Energy OTA or using the STLINK-V3MINIE.

Figure 40. **Extrapolation library integration on FP-AI-PDMWBSOC2**



5 Summary results, observations, and conclusions

As anticipated in the previous section, to generate different NanoEdge AI Studio projects based on different buffer length, it is recommended to have a larger number of models to compare with each other and therefore to find accurate models as much as possible. The following table summarizes the results coming from three NanoEdge AI Studio projects and it is fundamental to choose the best performing ML model.

Figure 41. Summary results

	Buffer Length = 256	Buffer Length = 512	**Buffer Length = 1024**
Fan Speed [RPM]	8000	8000	**8000**
Active Sensor	ISM330DHCX	ISM330DHCX	**ISM330DHCX**
Nominal ODR [Hz]	6667	6667	**6667**
Real ODR [Hz]	7210	7210	**7210**
Training examples	5	5	**5**
Test examples on training conditions	5	5	**5**
Test examples out of training conditions	4	4	**4**
Lines/training example	1000	1000	**1000**
Lines/test example	800	800	**800**
Model type	RIDGE	RFR	**RIDGE**
Benchmark performed	2	2	**2**
R-Squared Validation Dataset 1 [%]	95.37	98.66	**99.82**
R-Squared Validation Dataset 2 [%]	85.4	93.06	**94.06**
RMSE Validation Dataset 1	3.04	1.63	**0.60**
RMSE Validation Dataset 2	4.27	2.94	**2.73**
Execution Time [ms]	3.2	7.0	**14.5**
RAM [Kbyte]	6.1	11.6	**10.7**
FLASH [Kbyte]	6.5	250.5	**28.2**

The table contains the features and performances of the best models found for the specified buffer lengths. Of course, for a good comparison several parameters must be the same: fan speed, active sensor, ODR, number of example datasets used, and so on.

Projects generated converge to two kinds of regression models:

- RIDGE regressor, it is a particular type of linear regressor that includes a regularization term to prevent overfitting
- Random Forest Regressor (RFR), it is an ensemble learning method which combines multiple decision trees to improve the performance and generalization ability of the model.

The table shown above triggers several observations:

- The most suitable model typology is the RIDGE regressor
- R-Squared grows with the buffer length
- Root Mean Square Error (RMSE) lowers when the buffer length increases
- Execution Time grows with the buffer length, which is almost linear depending on the buffer length
- RAM and FLASH footprints have no linear dependencies from the buffer length as they are influenced by model typology, hyperparameters quantity, but higher accuracy does not necessarily require higher memory consumption.

One final observation can be added to the above. Ultimately, the user can prioritize one aspect which is not necessarily the model accuracy but other parameters such as the execution time, the memory footprint, or the power consumption which can be critical for specific applications.

However, for the application use-case that is the subject of this document, high priority was given to the RMSE indicator to estimate the target obstruction percentage with the best available accuracy.

Another aspect to consider is the variability of results coming from the NanoEdge AI library. For this reason, most commonly an arithmetic mean, modal, or median is computed on a certain number of library results. For the application in question, a median of one hundred estimated target values, coming from the selected library, was computed. Computation can be performed modifying the *AppController.c* code placed under

$PROJ_DIR$\Project\STM32WB5MMG-PROTEUS\Applications\PDMWBSOC2\Core\Src
Editing the *ACDataEvtListener_vtblOnNewDataReady* function is sufficient to achieve our goal:

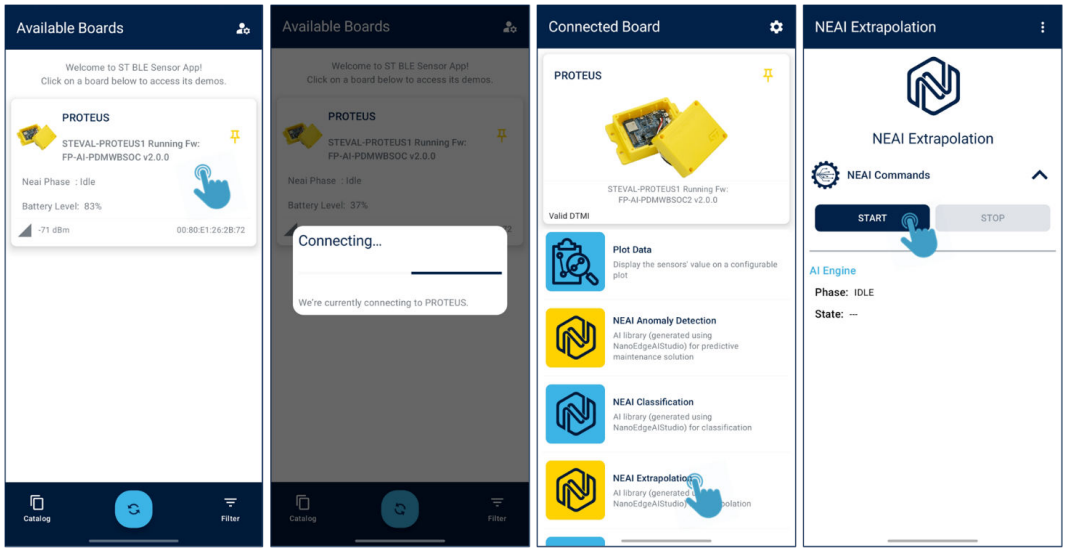
```
/**
  * Specifies the amount of NanoEdgeAI Extrapolation results on which to compute the median
  */
#define NEAI_NB_CHECKS 100U

/**
  * Private variables used to compute median of N=NEAI_NB_CHECKS extrapolation results
  */
static uint8_t s_inferences = 0;
static uint8_t s_median_counter = NEAI_NB_CHECKS;
static float s_ext_collection[NEAI_NB_CHECKS] = {0};
static float s_median_ext_results = 0;

sys_error_code_t ACDataEvtListener_vtblOnNewDataReady(IEventListener *_this, const DataEvent_
t *p_evt)
{
  assert_param(_this != NULL);
  sys_error_code_t res = SYS_NO_ERROR_CODE;
  AppController_t *p_if_owner = (AppController_t *)(((size_t)_this) - offsetof(AppController_
t, listener_if));
  AppMsg_t msg;

  msg.msg_id = APP_MESSAGE_ID_CTRL;

  if (p_evt->tag == E_NEAI_ANOMALY_LEARN)
  {
    /* this result come from NanoEdge AI process. We know the format of the data */
    uint32_t proc_res = (uint32_t)(*((float *) EMD_1dDataAt((EMData_t *)p_evt->p_data, 0)));
    msg.ctrl_msg.cmd_id = CTRL_CMD_NEAI_PROC_RES;
    msg.ctrl_msg.neai_state = proc_res;
    msg.ctrl_msg.sparam = CTRL_CMD_PARAM_NEAI_LEARN;

    if (TX_SUCCESS != tx_queue_send(&p_if_owner->in_queue, &msg, AMT_MS_TO_TICKS(100)))
    {
      res = SYS_CTRL_IN_QUEUE_FULL_ERROR_CODE;
      SYS_SET_SERVICE_LEVEL_ERROR_CODE(SYS_CTRL_IN_QUEUE_FULL_ERROR_CODE);
    }
  }
  else if (p_evt->tag == E_NEAI_ANOMALY_DETECT)
  {
    /* this result come from NanoEdge AI process. We know the format of the data */
    float *p_data = (float *) EMD_1dDataAt((EMData_t *)p_evt->p_data, 0);
    msg.ctrl_msg.cmd_id = CTRL_CMD_NEAI_PROC_RES;
    msg.ctrl_msg.neai_state = (uint32_t)p_data[0]; /* neai state*/
    msg.ctrl_msg.sparam = CTRL_CMD_PARAM_NEAI_DETECT;
    msg.ctrl_msg.neai_similarity = (uint8_t)p_data[1]; /* neai similarity*/

    if (TX_SUCCESS != tx_queue_send(&p_if_owner->in_queue, &msg, AMT_MS_TO_TICKS(100)))
    {
      res = SYS_CTRL_IN_QUEUE_FULL_ERROR_CODE;
      SYS_SET_SERVICE_LEVEL_ERROR_CODE(SYS_CTRL_IN_QUEUE_FULL_ERROR_CODE);
    }
  }
  else if (p_evt->tag == E_NEAI_CLASSIFICATION)
  {
    /* this result come from NanoEdge AI process. We know the format of the data */
    float *p_data = (float *) EMD_1dDataAt((EMData_t *)p_evt->p_data, 0);
    msg.ctrl_msg.cmd_id = CTRL_CMD_NEAI_PROC_RES;
    msg.ctrl_msg.neai_state = (uint32_t)p_data[0]; /* neai state*/
    msg.ctrl_msg.sparam = CTRL_CMD_PARAM_NEAI_CLASS;
    msg.ctrl_msg.neai_mpc = (uint8_t)p_data[1]; /* neai class id*/
    msg.ctrl_msg.neai_probabilities = (float *)&p_data[2]; /* classification probabilities */

    if (TX_SUCCESS != tx_queue_send(&p_if_owner->in_queue, &msg, AMT_MS_TO_TICKS(100)))
    {
      res = SYS_CTRL_IN_QUEUE_FULL_ERROR_CODE;
      SYS_SET_SERVICE_LEVEL_ERROR_CODE(SYS_CTRL_IN_QUEUE_FULL_ERROR_CODE);
```

```
    }
  }
  else if (p_evt->tag == E_NEAI_EXTRAPOLATION)
  {
    /* this result come from NanoEdge AI process. We know the format of the data */
    float *p_data = (float *) EMD_1dDataAt((EMData_t *)p_evt->p_data, 0);

    if(s_inferences < NEAI_NB_CHECKS) {
      s_ext_collection[s_inferences] += (float)p_data[1];
      s_inferences++;
    }
    else
    {
    bubbleSort(s_ext_collection, NEAI_NB_CHECKS);
    s_median_ext_results = (s_ext_collection[((NEAI_NB_CHECKS/2)-2)]+s_ext_collection[((NEAI_
NB_CHECKS/2)-1)])/2;

    msg.ctrl_msg.cmd_id = CTRL_CMD_NEAI_PROC_RES;
    msg.ctrl_msg.neai_state = (uint32_t)p_data[0]; /* neai state*/
    msg.ctrl_msg.sparam = CTRL_CMD_PARAM_NEAI_EXTRAPOLATE;
    msg.ctrl_msg.extrapolated_value = s_median_ext_results; /* neai extrapolated value*/

    if (TX_SUCCESS != tx_queue_send(&p_if_owner->in_queue, &msg, AMT_MS_TO_TICKS(100)))
    {
      res = SYS_CTRL_IN_QUEUE_FULL_ERROR_CODE;
      SYS_SET_SERVICE_LEVEL_ERROR_CODE(SYS_CTRL_IN_QUEUE_FULL_ERROR_CODE);
    }
    s_inferences = 0;
    memset(s_ext_collection, 0x00, sizeof(s_ext_collection));
    s_median_ext_results = 0;
    }
  }
  else
  {
    SYS_DEBUGF(SYS_DBG_LEVEL_VERBOSE, ("CTRL: unexpected TAG ID:0x%x\r\n", p_evt->tag));
    return SYS_INVALID_PARAMETER_ERROR_CODE;
  }

  return res;}
```

After model integration as explained in Section 4.6: Library integration on FP-AI-PDMWBSOC2, the user can proceed to start validation on the mechanical setup.

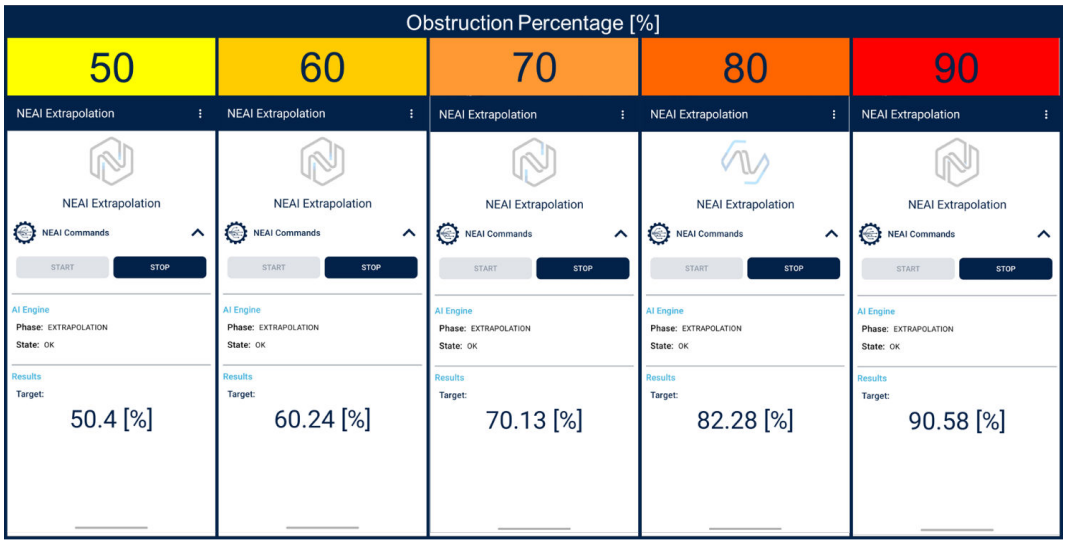
After flashing the FP-AI-PDMWBSOC2 firmware application, with your own NanoEdgeAI model, open the STBLESensor app and follow the steps below to run extrapolation processing.

Figure 42. FP-AI-PDMWBSOC2 - starting extrapolation processing



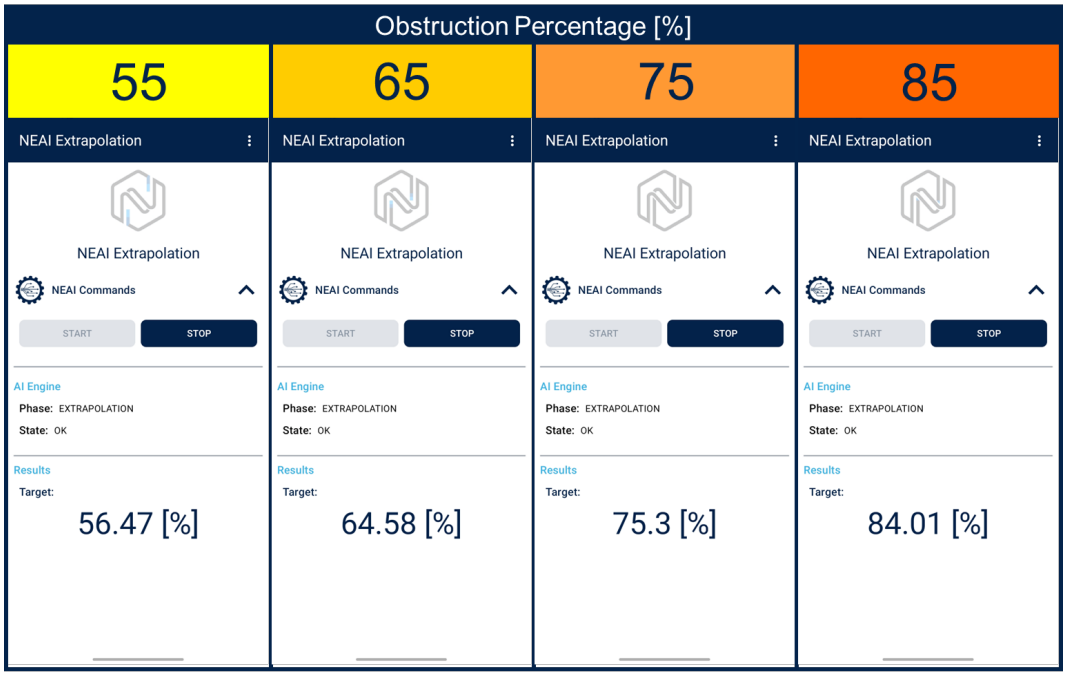
These are the results corresponding to the training obstruction conditions:

Figure 43. Results on validation setup injecting training conditions



Finally, these are the results corresponding to the out-of-training obstruction conditions:

Figure 44. Results on validation setup injecting out of training conditions



The table below compares the target expected value and the estimated ones with the corresponding absolute and relative error:

Figure 45. Extrapolation model accuracy performance

Obstruction percentage [%]	Median of estimated percentage [%]	Absolute Error [%]	Relative Error [%]
50.0	50.04	-0.04	0.08
55.0	56.47	-1.47	2.67
60.0	60.24	-0.24	0.40
65.0	64.58	0.42	-0.65
70.0	70.13	-0.13	0.19
75.0	75.30	-0.30	0.40
80.0	**82.28**	**-2.28**	**2.85**
85.0	84.01	0.99	-1.16
90.0	90.58	-0.58	0.64

In conclusion, using the best extrapolation model we obtained $R^2 = 99.50\%$ and $RMSE = 1.0\%$.

Revision history

Table 1. **Document revision history**

Date	Revision	Changes
08-Apr-2025	1	Initial release.

Contents

List of tables

List of figures