
Getting started with the STM32Cube function pack for STEVAL-PROTEUS1 evaluation kit for predictive maintenance application based on artificial intelligence (AI)

Introduction

FP-AI-PDMWBSOC is an [STM32Cube](#) function pack for the [STEVAL-PROTEUS1](#) evaluation kit. It acquires motion sensor data, process them for anomaly detection and fault classification, and sends the results to the [STBLESensor](#) mobile app and/or a PC terminal console.

The application is based on the vibrometer and accelerometer data, which are used to execute a machine learning algorithm to detect anomaly conditions and classify different kinds of failures on industrial assets. You can configure this application remotely through wireless or wired connectivity.

The monitoring of equipment conditions for predictive maintenance applications through the [STEVAL-PROTEUS1](#) allows reducing the loss of productivity, due to machine downtime, and unplanned maintenance costs, improving the performance of the production chain.

Using a binary data logging file, the raw data can be extracted from the sensors within the [STEVAL-PROTEUS1](#) evaluation board and provided into the [NanoEdgeAIStudio](#) software, which allows you to extract a machine learning library for anomaly detection and N-Class Classification purposes.

These NanoEdge AI libraries, built on the classified dataset, can be easily integrated into the proposed framework architecture, facilitating the early detection of an alert status or failure type recognition.

FP-AI-PDMWBSOC implements two different HMI communication tools at user level:

- A wired interactive CLI (USB CDC) to configure the node and manage the learning, detecting and classifying phases
- A wireless interactive connection based on the [STBLESensor](#) app to provide the same functionalities. The [STBLESensor](#) mobile app works also as a bridge to display data on a cloud service supported by Azure IoT Central.

To start/stop the learning, detecting and classifying phases, an additional control, indicated by LEDs, can be performed by just pressing the user button.

The NanoEdgeAI libraries functionalities are based on a set of application-level modules (Sensor Manager, Digital Processing Unit, EMDData, PnPLCompManager), useful to reuse and easily extendable to build other customized applications.

These application modules implement innovative software design models based on application layers built on an embedded light object-oriented framework (eLoom) middleware for embedded applications powered by STM32.

Related links

[Visit the STM32Cube ecosystem web page on \[www.st.com\]\(http://www.st.com\) for further information](#)

1 FP-AI-PDMWBSOC software expansion for STM32Cube

1.1 Overview

The **FP-AI-PDMWBSOC** software package features are:

- Firmware to develop a WPAN sensor node for predictive maintenance applications, featuring motion sensors and performing anomaly detection and classification controlled via Bluetooth® Low Energy connectivity
- STM32 wireless personal area network middleware developed within the STM32WB framework used to support Bluetooth® Low Energy 5
- Compatible with **NanoEdgeAIStudio** solution to enable AI-based applications
- On-board battery status monitor
- Compatible with **STBLESensor** app (Android and iOS) to enable AI libraries and sensors setting, and firmware update via fast FUOTA
- Based on accelerometer data up to 6 kHz bandwidth
- NanoEdge AI libraries generated to run in the STM32WB module
- Firmware modular example based on the embedded light object-oriented framework (eLooM) to enable code reusability at application level
- Application for datalogging in binary format
- Utilities: Python and CLI real-time control applications
- Free, user-friendly license terms

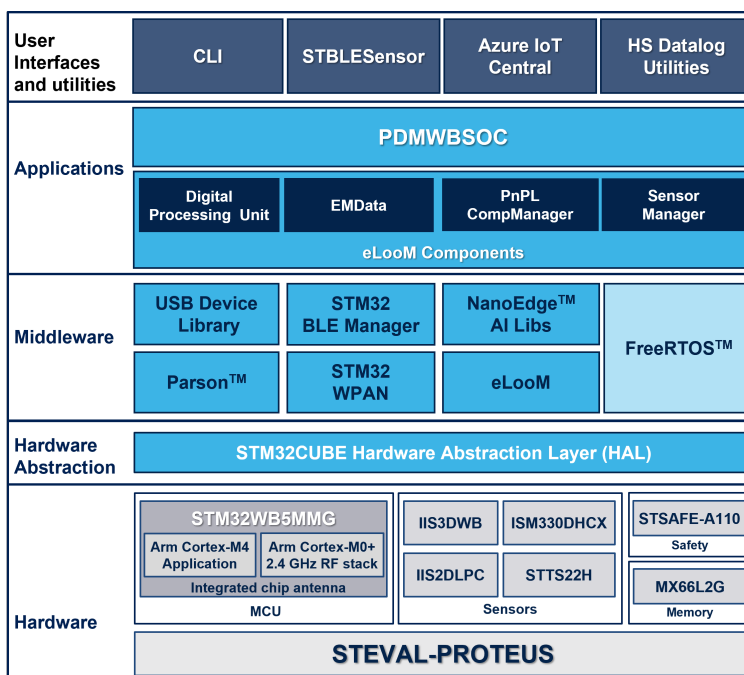
The software gathers:

- the 3-axis acceleration from the **IIS3DWB** or the **ISM330DHCX**
- the battery status from the **STBC02**
- green, red, and blue LEDs to indicate the connection status and NEAI phase

1.2 Architecture

The **FP-AI-PDMWBSOC** software is designed for the **STEVAL-PROTEUS1** development kit, respecting the compliance with the **STM32Cube** architecture, structured into a set of layers of increasing abstraction.

Figure 1. FP-AI-PDMWBSOC software architecture



The hardware abstraction layer (HAL) interfaces with the hardware and provides the low-level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries, and stacks). It provides APIs for the communication peripherals (I²C, SPI, UART, etc.) for initialization and configuration, data transfer, and communication errors.

There are two types of HAL driver APIs:

- generic APIs, which provide common and generic functions to the entire STM32 series
- extension APIs, which provide specific, customized functions for a particular family or a specific part number

The package extends [STM32Cube](#) by providing a set of APIs, which follows the hardware structure, including a component management layer as well as the specific layers of the boards used.

On top of such features, inherited from [STM32Cube](#), the [FP-AI-PDMWBSOC](#) is focused on code reusability, according to the eLoom-based application-level architecture.

It is based on several firmware modules that interface and offer data to other application modules according to well defined design patterns and specific APIs.

Each firmware module (Sensor Manager, Digital Processing Unit, EMDData, PnPLCompManager) is packed into a folder. They are independent from each other, and they can be added to your firmware application by just including the module folder inside your project.

Every firmware module implements or extends drivers, events, and services made available by the eLoom framework. More specifically, here you can find:

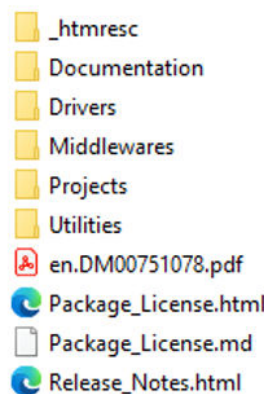
- drivers - objects that implement the base interface for any low-level subsystem that can be used into the firmware module (SPI, UART)
- events - objects that handle information about something that happened in the system at a given moment; these files implement the event and source/listener design patterns
- services - additional utilities for the firmware module

1.3

Folders

The [FP-AI-PDMWBSOC](#) firmware package folder structure follows the layer-based approach of the [STM32Cube](#) architecture.

Figure 2. FP-AI-PDMWBSOC package folder structure



The folders included in the software package are:

- **Documentation** - contains a compiled HTML file generated from the source code, which details the software components and APIs.
- **Drivers** - contains the HAL drivers and the board-specific drivers for each supported board or hardware platform, including the on-board components and the CMSIS vendor-independent hardware abstraction layer for ARM Cortex-M processor series.

- **Middlewares** - contains all the libraries released by ST or a third party, as listed below:
 - **STM32_WPAN**, wireless personal area network middleware developed within the STM32WB framework with protocols for Bluetooth® Low Energy connectivity
 - **STM32_BLE_Manager**, STM32 middleware providing an API to manage the Bluetooth® Low Energy service
 - **eLoom**, embedded light object-oriented framework for STM32 applications
 - **NanoEdge_AI_Library**, default (“stub”) NanoEdge libraries
 - **STM32_USB_Device_Library**, middleware for USB connectivity
 - **FreeRTOS™**, , third-party free real-time operating system
 - **Parson**, lightweight JSON library
- **Projects** - contains one sample application for the STEVAL-PROTEUS, in source code, for the following toolchains:
 - IAR Embedded Workbench for Arm ([IAR-EWARM](#)).
 - Keil® Microcontroller Development Kit for Arm ([MDK-ARM](#)).
 - ST integrated development environment for STM32 ([STM32CubeIDE](#)).
 - A binary folder that contains all the applications in a compiled format
- **Utilities** - includes three different folders to support the whole development:
 - the **Bootloader**, which allows updating the firmware by FUOTA; just download the binary files into your smartphone and update the firmware via the [STBLESensor](#) mobile app
 - the **HSDatalog** application that provides an easy way to save data from any combination of sensors configured at their maximum sampling rate

The STM32WB_Copro_Wireless_Binaries folder contains the coprocessor wireless firmware binary to be used.

1.4 APIs

Detailed technical information with full user API function and parameter description are in a compiled HTML file in the “Documentation” folder and in the “Doc” folder of each firmware module and eLoom middleware.

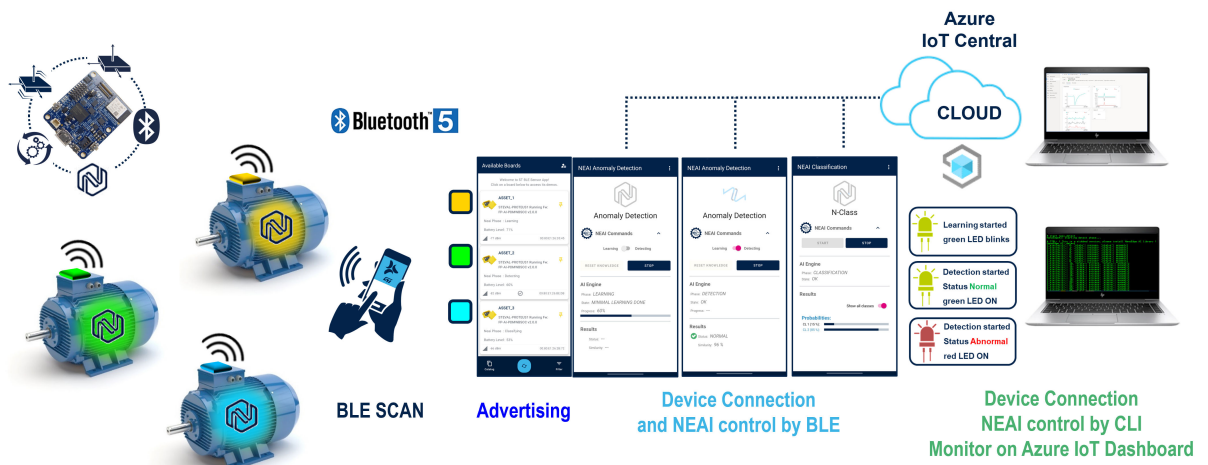
2 Application scenario

The condition monitoring by anomaly detection or fault classification, based on the STEVAL-PROTEUS, enables the early detection of fault conditions directly on the equipment. This facilitates the implementation of a predictive maintenance program, which reduces the loss of productivity in the production chain due to unplanned downtime. The STEVAL-PROTEUS evaluation board includes some sensors to acquire vibration data such as the [IIS3DWB](#) vibrometer (wide bandwidth up to 6 kHz), or the [ISM330DHCX](#) accelerometer (bandwidth up to 3.3 kHz), useful to be processed by the embedded Anomaly Detection (AD) or N-Class Classification (NCC) libraries.

Bluetooth® Low Energy permits a direct connection with a smartphone and facilitates the WPAN node configuration, data monitoring, equipment status, and firmware upgrade over the air (FUOTA).

The [STBLESensor](#) app includes all the commands for learning, detecting and classifying, in addition it includes dedicated setting pages to configure AI libraries and sensors. The CLI (USB) connectivity supports the same functionalities.

Figure 3. FP-AI-PDMWBSOC application overview



You can see the results of the Anomaly Detection (AD) or N-Class Classification (NCC) processing using the [STBLESensor](#) app or the CLI, which is accessible via a standard USB terminal console.

The example is provided in the “projects” directory, in source code, ready to be built for multiple IDEs.

The application provides:

- a stable connection via USB with the CLI terminal console and, at the same time, connectivity via Bluetooth® Low Energy in advertising mode
- the initialization of all the sensors data handling and the embedded NanoEdge AI libraries, including all the setting parameters
- the battery status according to the [STBC02](#) measurements

The package is based on the following mandatory key elements:

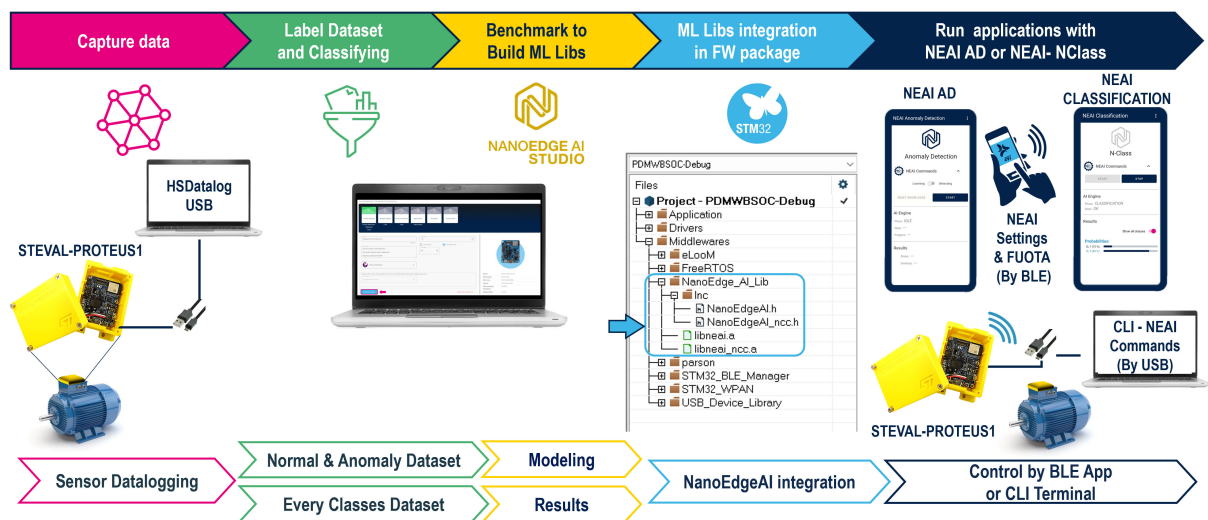
- the FreeRTOS™ operating system as a third-party middleware for the real-time execution

- the eLooM middleware, which implements the object-oriented methodology for the data management and processing with the AI library, easy to use at application level, thanks to the following reusable software modules:
 - The Sensor Manager component retrieves sensor data and sets the sensor parameters. It is implemented as an acquisition engine that:
 - Controls multiple task accesses to the sensor bus
 - Defines interfaces to avoid implementation dependencies
 - Dispatches events to notify when a certain amount of data is available
 - The Digital Processing Unit (DPU) component provides a set of processing blocks, which can be chained together, to create complex processing functions that use data stream coming from sensor manager. Available processing modules include:
 - Anomaly Detection (AD)
 - N-Class Classification (NCC)
 - The EMDData component defines a common data model among eLooM framework aligned to the standards used in data science tools. Around this data format, the component provides also some util classes and interfaces to address data driven use-cases.
 - The PnPLCompManager deals with PnP-Like messages parsing.

Further details are available in the "documentation" folder.

FP-AI-PDMWBSOC design and development follow a specific working flow. This flow is based on the use of different application firmware and different software tools oriented to datalogging and library generation.

Figure 4. FP-AI-PDMWBSOC working flow



3 System setup guide

3.1 STEVAL-PROTEUS1 industrial sensor evaluation kit

The **FP-AI-PDMWBSOC** software package is tailored for the STEVAL-PROTEUS evaluation board. This is the main board of the **STEVAL-PROTEUS1** kit, which embeds an ultra-low-power and small form factor certified 2.4 GHz wireless module (**STM32WB5MMG**). This module supports Bluetooth® Low Energy 5.0 and others 802.15.4 proprietary protocols (for further details, refer to **UM3000**).

Figure 5. STEVAL-PROTEUS1 evaluation kit



The module is protocol stack royalty-free. It is featured by a dual Arm® core, a dedicated Arm® Cortex®-M0+ for radio and security tasks and a dedicated Arm® Cortex®-M4 CPU with FPU and ART (adaptive real-time accelerator) up to 64 MHz speed. It also comes with 1-Mbyte Flash memory and 256-Kbyte SRAM. These features allow running complex algorithms directly through the board and deliver ready-to-use results. It includes ST MEMS sensors for battery-operated applications to target IoT as well as Industry 4.0.

Table 1. Hardware platform details

Features	STEVAL-PROTEUS	
Board		
Size (mm)	35 x 30 x 7	
Vibration analysis	Up to 6 kHz	
Connectivity		
Datalog and GUI	Bluetooth® Low Energy	STBLESensor app (Android/iOS)
	CLI	Debug terminal via UART
Development tools	STLINK-V3MINIE	

The evaluation board comes with three inertial modules:

- the [IIS3DWB](#) ultra-wide bandwidth, low-noise, 3-axis digital vibration sensor
- the [IIS2DLPC](#) high-performance ultra-low-power 3-axis accelerometer
- the [ISM330DHCX](#) 3-axis accelerometer and 3-axis gyroscope with machine learning core (MLC)

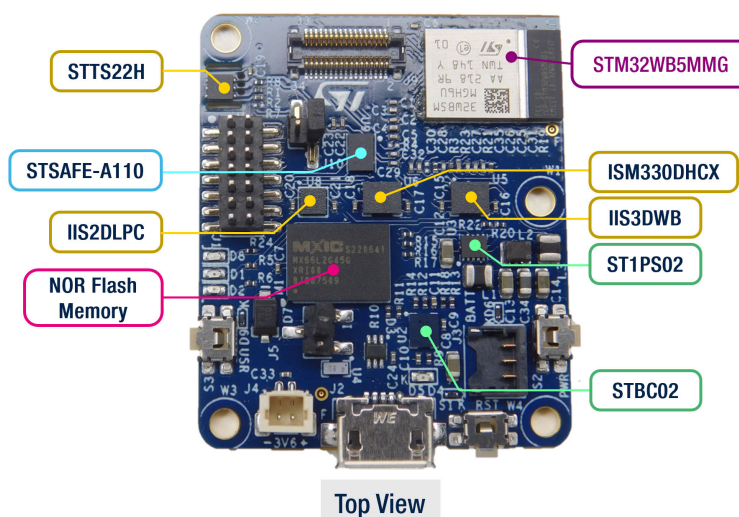
The [STTS22H](#) low-voltage, ultra-low-power, high accuracy temperature sensor has also been integrated in the board, far from the heat noise sources (the power management and the microcontroller) to provide a more precise temperature measurement. Its exposed pad and the PCB accurate design allow the temperature sensor to be in contact with the surface of the target equipment.

Moreover, the evaluation board integrates the [STSAFE-A110](#) secure element that provides authentication and secure data management services to a local or remote host.

Furthermore, a NOR flash memory is soldered on the board. It is connected via QSPI to the [STM32WB5MMG](#) module for data buffering and event storage.

The [ST1PS02](#) 400 mA step down converter for low-power applications and the [STBC02](#) Li-Ion linear battery charger feature the power management.

Figure 6. STEVAL-PROTEUS main devices

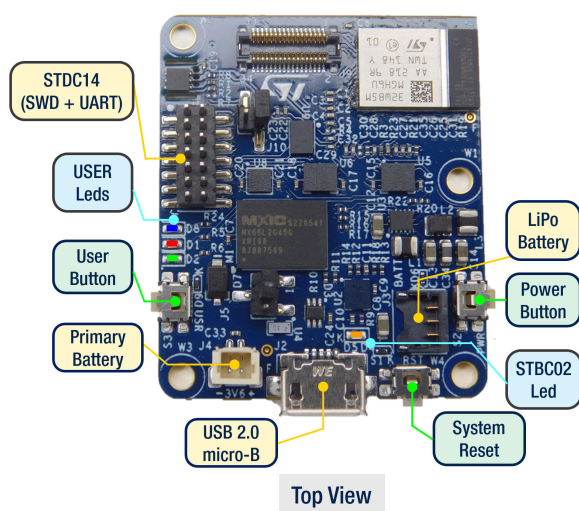


The STEVAL-PROTEUS can be powered via:

- a LiPo rechargeable battery
- USB (5 V at 500 mA)
- a primary battery (not included in the kit)

All components are mounted exclusively on the top side of the PCB to ensure easy mounting on other equipment, even when using the plastic case for the board and the LiPo battery (included in the kit).

Figure 7. STEVAL-PROTEUS power connector, SWD, user button, and LED



3.2 Hardware setup

The hardware prerequisites to prepare the kit are:

- two USB Type-A to Micro-B cables: one to monitor the debug console and program the board via the STDC14 connector, and one to use a CLI via USB
- an [STLINK-V3MINIE](#) (not included in the kit) for programming and debugging

The [STEVAL-PROTEUS1](#) kit includes a plastic case with its related screws to host the board and the LiPo battery, allowing an easy fixing (for demo purposes only).

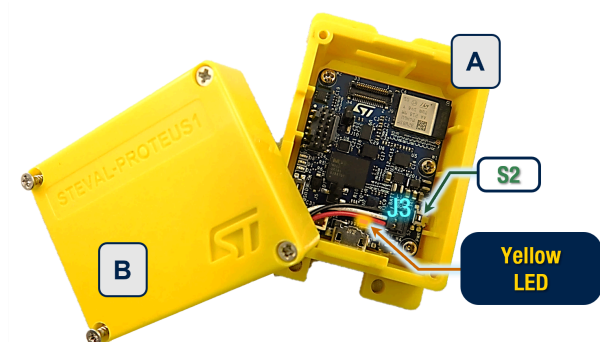
To assemble the kit, follow the procedure below.

1. Fix the main board to the case bottom (**A**) with the four screws included in the kit.

Note: Pay attention to the board orientation.

2. Put the LiPo battery in the top case (**B**). Then, insert the battery cable into the dedicated hole.
3. Put the cover on the battery and close it using two screws.
4. Plug the LiPo battery connector to the J3 connector.
5. Close the A and B case parts using four screws.
6. Use something thin (for example, a needle) to access the holes and press the push buttons.
7. Power on the board by pushing and hold the **S2** button for at least three seconds: the yellow LED turns on, then off. To power off, push and hold the **S2** button until the yellow LED turns on.

Figure 8. STEVAL-PROTEUS mounting and power-on

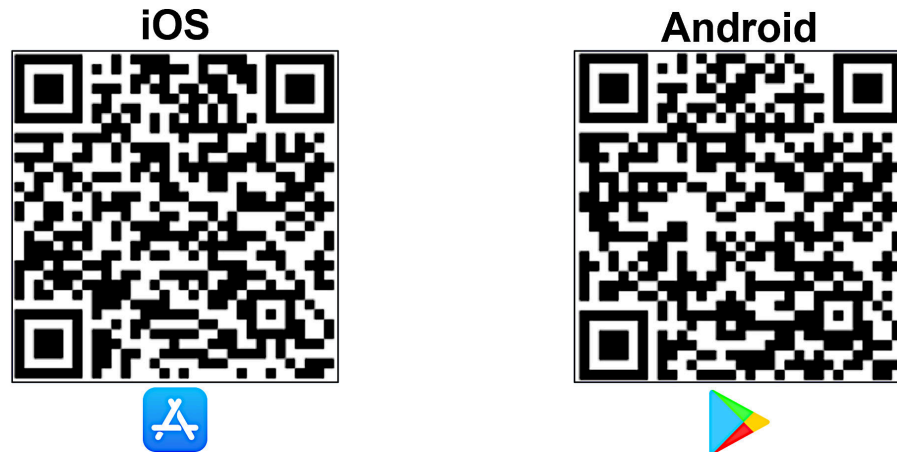


3.3 Software setup

To use the FP-AI-PDMWBSOC, update, and run the application, you need:

- the [STBLESensor](#) app, available for Android and iOS, to handle the data exported by a Bluetooth® Low Energy device using the BlueST protocol. The app allows running the application and performing the firmware update over-the-air (FUOTA).

Figure 9. QR codes for STBLESensor



- A Windows™ (v10 or higher) PC, with [TeraTerm](#) or another terminal emulator software
- [NanoEdgeAIStudio](#) (v4.5.0 or higher)
- [STM32CubeProgrammer](#) software
- A development toolchain and compiler. The STM32Cube expansion software supports the three following environments:
 - IAR
 - Keil®
 - [STM32CubeIDE](#)

3.3.1 STEVAL-PROTEUS flash memory mapping

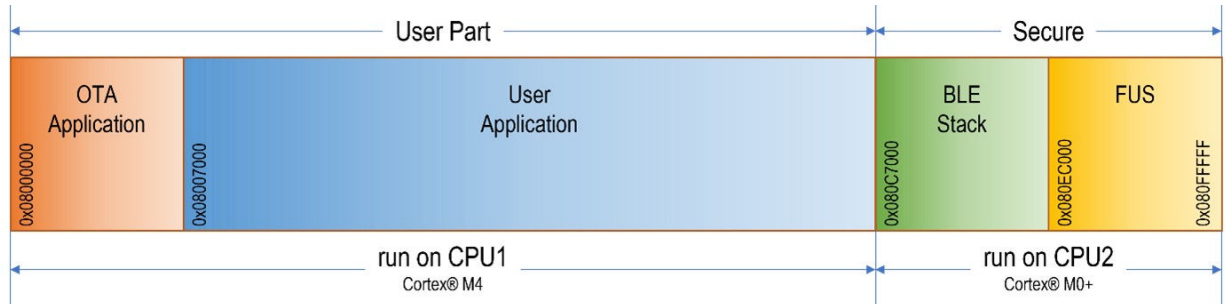
The [STM32WB5MMG](#) comes with a 1 Mbyte flash memory, shared between the two Arm® cores as the user part to be run into the CPU1 (Cortex® M4) and the secure part to be run into the CPU2 (Cortex® M0+). The user part starts from the beginning of the flash memory, whereas the secure part is placed at the end.

A specific memory mapping helps to develop an application easy to update.

To integrate the FUOTA features, design the firmware package as follows:

- the bootloader, placed in the first main memory address (0x08000000), must be an OTA application designed to handle the firmware update
- the user application, placed in the specific memory address (0x08007000), can be updated with the FUOTA and contains the main application
- the secure sectors include the application running in the CPU2 core; the secure part (FUS) is placed at the end and its size depends on the wireless Bluetooth® Low Energy stack to be used.

You can also update the Bluetooth® Low Energy stack using the same FUOTA functionality.

Figure 10. STEVAL-PROTEUS memory map for Bluetooth® Low Energy OTA applications


For further details on this topic, refer to [AN5247](#), which describes the procedure for over-the-air (OTA) firmware update on ST32WB devices with Bluetooth® Low Energy connectivity.

3.3.2 Toolchains settings for STM32WB memory mapping

Usually, the binary code generated with default settings, for any toolchain, considers the firmware starting address as the first address of the main memory (0x08000000).

So, to generate the project with OTA capabilities correctly, follow the steps below.

1. Make sure that the OTA application firmware has already been downloaded into the STEVAL-PROTEUS from the first main memory address.
2. Open `$PROJ_DIR$\Src\system_stm32wbxx.c` and:
 - Verify that the `USER_VECT_TAB_ADDRESS` define is uncommented
 - Verify that `VECT_TAB_OFFSET` is defined as 0x00007000U
3. Apply the following changes for each IDE toolchain:
 - **IAR EWARM**
Open [Project]>[Options]>[Linker] and use `$PROJ_DIR$\stm32wb5mxx_flash_cm4_ota.icf` as the linker configuration file
 - **Keil® MDK**
Open [Project]>[Options]>[Target] and set IROM1 as 0x8007000 for start and 0x79000 for size.
Open [Project]>[Options]>[Linker] and use `.\stm32wb5mxx_flash_cm4_ota.sct` as the scatter file.
Open [Project]>[Options]>[Linker] and add two lines in the misc controls:

```
--keep *.o (TAG_OTA_START)
--keep *.o (TAG_OTA_END)
```
 - **STM32CubeIDE**
Open [Properties]>[C/C++ Build]>[Settings]>[Tool Settings]>[MCU GCC Linker->[General] and use `${workspace_loc}/${ProjName}/STM32WB5MMGHX_FLASH_ota.ld` as the linker script
4. Download the generated firmware into the STEVAL-PROTEUS that you are using from the address 0x08007000

3.3.3 STEVAL-PROTEUS flash programming via Bluetooth® Low Energy OTA

The STEVAL-PROTEUS comes with a preloaded application firmware, supporting the condition-based maintenance via Bluetooth® Low Energy connectivity ([STSW-PROTEUS](#)).

This preloaded firmware supports the Bluetooth® Low Energy OTA. So, to use directly the [FP-AI-PDMWBSOC](#) application, update the application coprocessor binary or the wireless coprocessor binary through the [STBLESensor](#) app.

3.3.3.1 Application firmware update

To update the main application via Bluetooth® Low Energy OTA, follow the steps below.

1. Copy the PDMWBSOC.bin binary file, available in the `$PROJ_DIR$\Projects\STM32WB5MMG-PROTEUS\Applications\FP-AI-PDMWBSOC\Binary` folder, inside a mobile device folder.
2. Launch the mobile app and connect to the Proteus board.
3. Touching the gear icon, select *Firmware Update* option.
4. Tap on *Selected File* label, browse inside your smartphone memories and select PDMWBSOC.bin binary file.

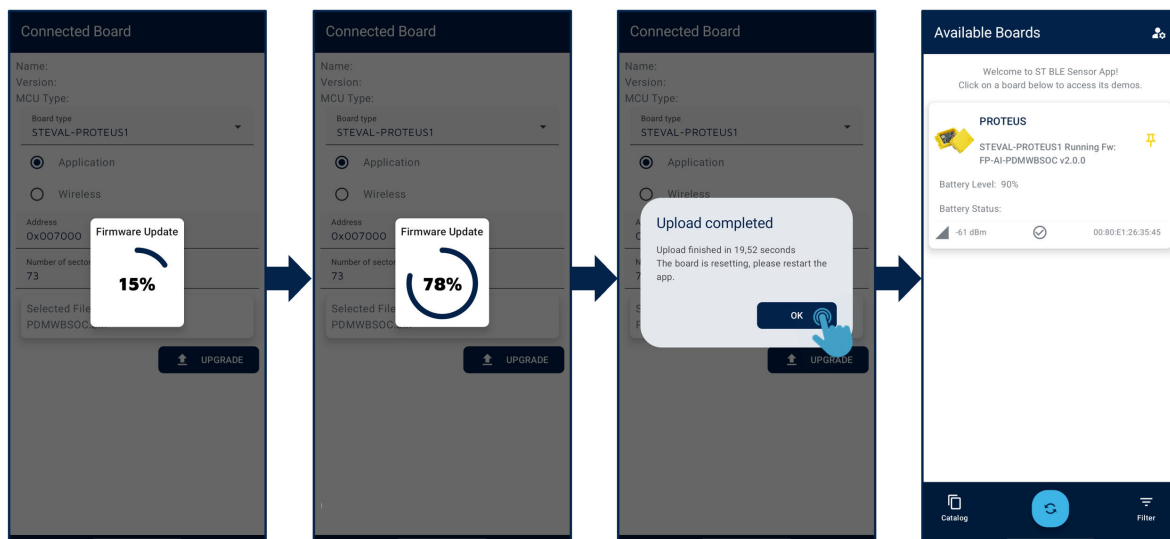
5. Tap on *Upgrade* button to launch OTA firmware upgrade.

Figure 11. Start FUOTA and select FP-AI-PDMWBSOC binary application



6. Wait a few dozen seconds until the completion message will be displayed.

Figure 12. FUOTA progress and completion



As the upload has been completed, the board reboots as *STEVAL-PROTEUS* according to the uploaded firmware. It is visible in the list of the Bluetooth® Low Energy connectable devices.

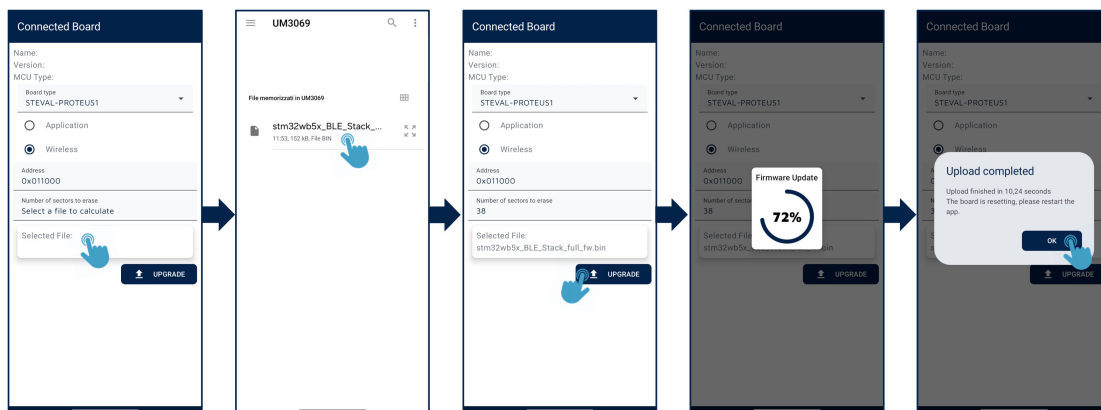
3.3.3.2 **Wireless stack firmware update**

To update the wireless stack via Bluetooth® Low Energy OTA, follow the steps below.

1. Copy the `stm32wb5x_BLE_Stack_full_fw.bin` binary file, available in the `$PROJ_DIR$\Utilities\STM32WB_Copro_Wireless_Binaries\STM32WB5x` folder, inside a mobile device folder.
2. Launch the mobile app and connect to the Proteus board.
3. Select Wireless Coprocessor Binary option, address will be updated automatically.
4. Tap on *Selected File* label, browse inside your smartphone memories and select `stm32wb5x_BLE_Stack_full_fw.bin` binary file.
5. Tap on *Upgrade* button to launch OTA firmware upgrade.

6. Wait a few dozen seconds until the completion message will be displayed.

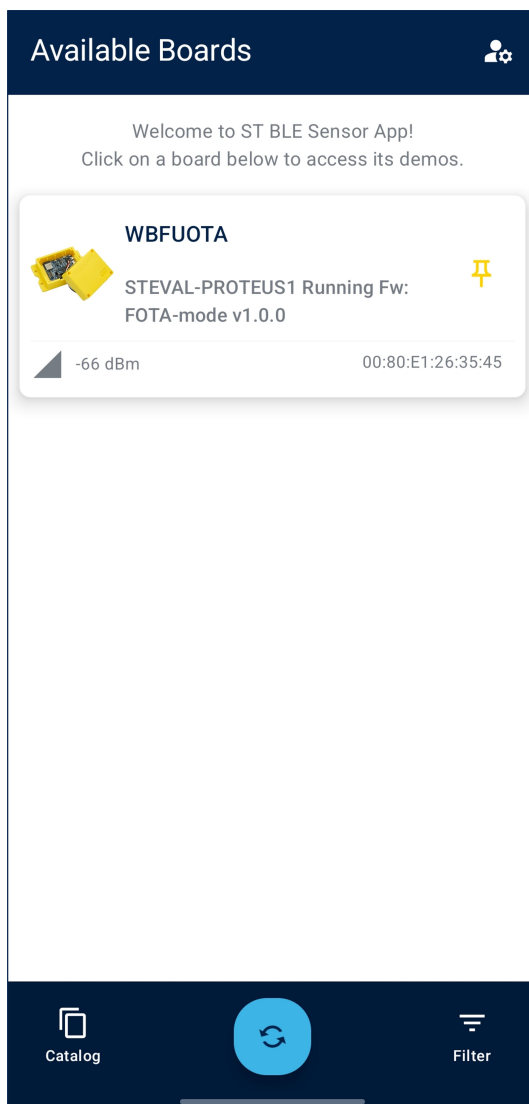
Figure 13. Start FUOTA and select wireless stack binary



Note: The process is not yet complete. The firmware is stored in an area of the flash memory buffer. Then, the FUS copies the binary into the right flash memory area. Do not disconnect the power from the board and do not press any buttons on it. This process can take up to 25 seconds. During this time, the board is no longer visible in the device list of the app. Just tap on the Search button until you see the WBFUOTA board in the list of the Bluetooth® Low Energy connectable devices.

Important: The wireless stack update procedure erases the application firmware. So, you must update the related user application sector just after the stack update (see [Section 3.3.3.1: Application firmware update](#)).

Figure 14. Searching the WBFUOTA in the device list



After tapping on the *WBFUOTA* board, update the application firmware as shown in [Section 3.3.3.1: Application firmware update](#).

3.3.4

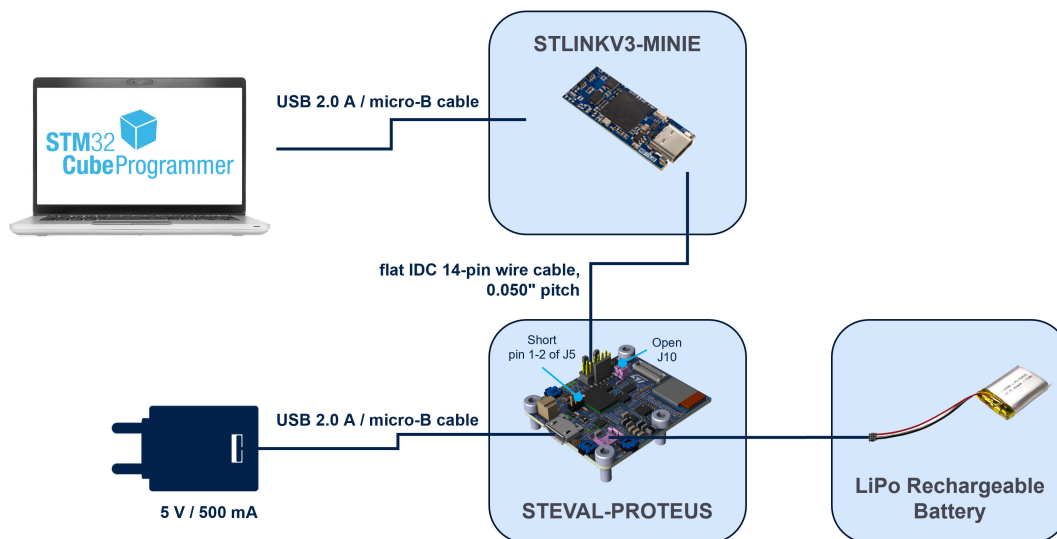
STEVAL-PROTEUS flash memory programming via ST-LINK

The STEVAL-PROTEUS flash memory programming methodology is stack independent.

There are different binaries files for the wireless stack and the application.

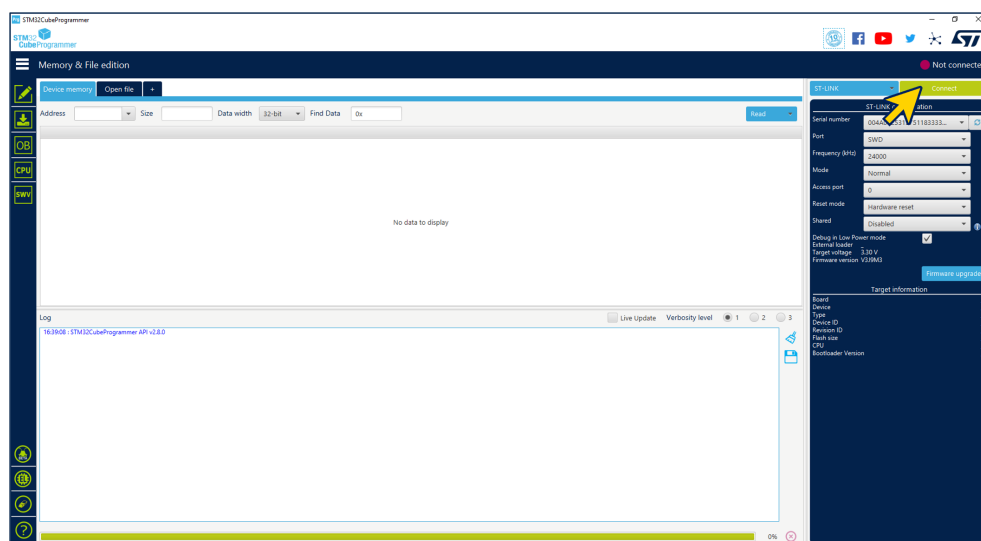
Install the [STM32CubeProgrammer](#) software on your PC and arrange the connection as shown below.

Figure 15. STEVAL-PROTEUS flash programming scenario



Note: You can choose to power the STEVAL-PROTEUS via USB or by a LiPo battery. You can also use both at the same time. When using only the battery, press the power button at least for three seconds to turn on the system. Launch the STM32CubeProgrammer software and connect the attached board, as highlighted below.

Figure 16. Connecting the board

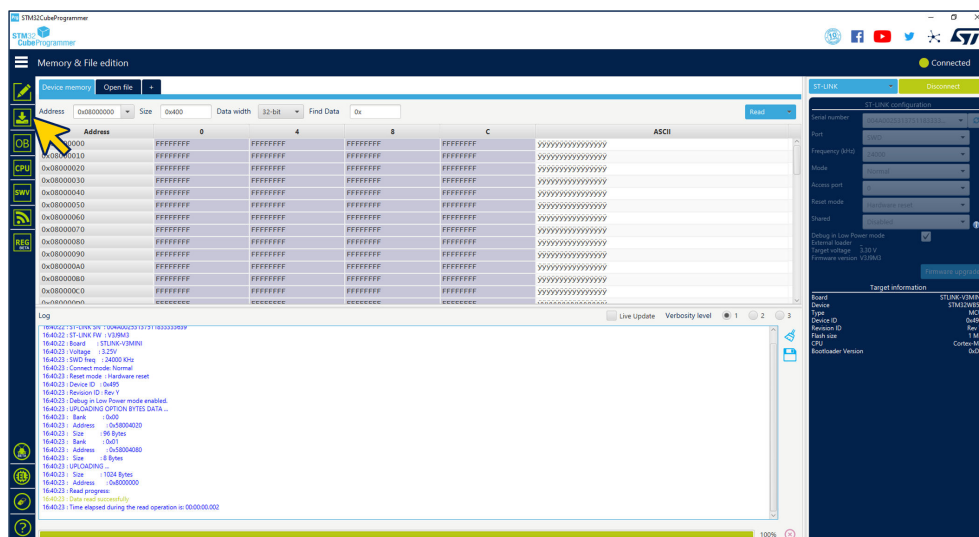


3.3.4.1

Application firmware update

- Step 1.** Launch STM32CubeProgrammer if not done already (see Figure 16).
- Step 2.** Select *Erasing & Programming*.

Figure 17. Erasing & Programming selection



- Step 3.** Browse to select the application binary file you are going to use (see Figure 18, arrow No. 1).
- Step 4.** Set the right start address to download the binary (see Figure 18, arrow No. 2).

Note: The table below shows the right start address to use with the chosen application binary.

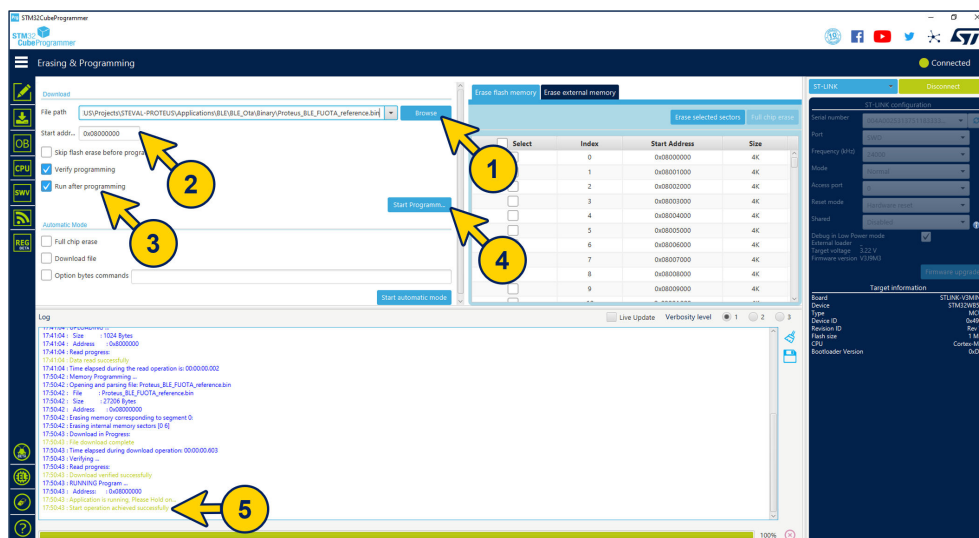
Table 2. STM32WB Application Binaries to use

Hardware	Connectivity	Application	Application binary	Install address
STEVAL-PROTEUS	Bluetooth® Low Energy	OTA	Proteus_BLE_FUOTA_reference.bin	0x08000000
		FP-AI-PDMWBSOC	PDMWBSOC.bin	0x08007000
		HSDatalog	Proteus_HS_Datalog_FUOTA_reference.bin	0x08007000

- Step 5.** Check both *Verify programming* and *Run after programming* (see Figure 18, arrow No. 3).
- Step 6.** Select *Start Programming* (see Figure 18, arrow No. 4).

Step 7. Wait for all operations to be successfully completed (see Figure 18, arrow No. 5).

Figure 18. Application firmware installation overview



If no error occurs, the application firmware has been correctly downloaded into the STEVAL-PROTEUS flash memory.

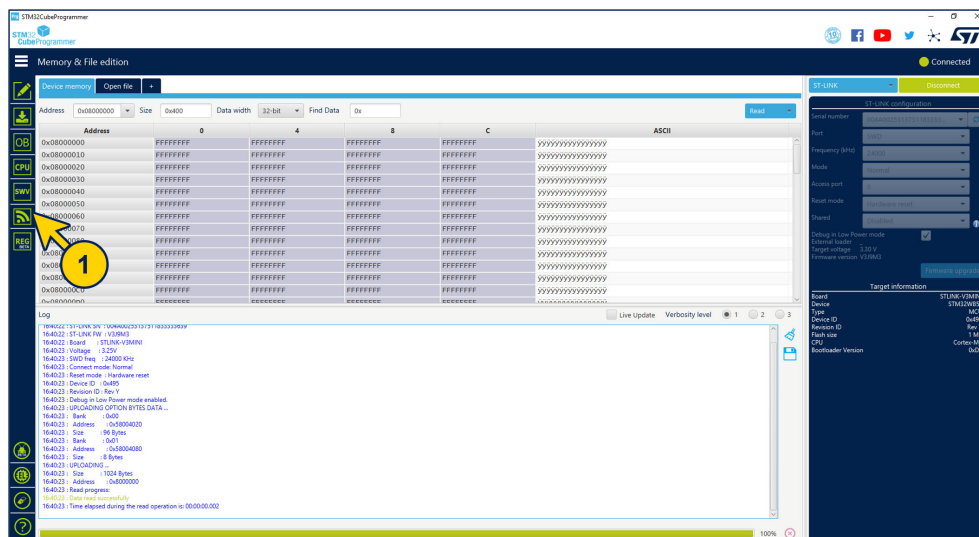
3.3.4.2

Wireless stack firmware update

Step 1. Launch STM32CubeProgrammer if not done already (see Figure 16) and connect the board.

Step 2. Select *Firmware Upgrade Services* (see Figure 20. Firmware upgrade overview, arrow No. 1).

Figure 19. Firmware upgrade services



Step 3. Select *Start FUS* (see Figure 20, arrow No. 2).

Note: If a failure message appears ("Start FUS failed!" or "Start FUS Operation Failure! Please, try again"), relaunch the command.

Step 4. Browse to select the stack binary file you are going to use (see Figure 20, arrow No. 3), available in \$PROJ_DIR\$\Utilities\STM32WB_Copro_Wireless_Binaries\STM32WB5x.

Step 5. Set the right start address to download the binary (see Figure 20, arrow No. 4).

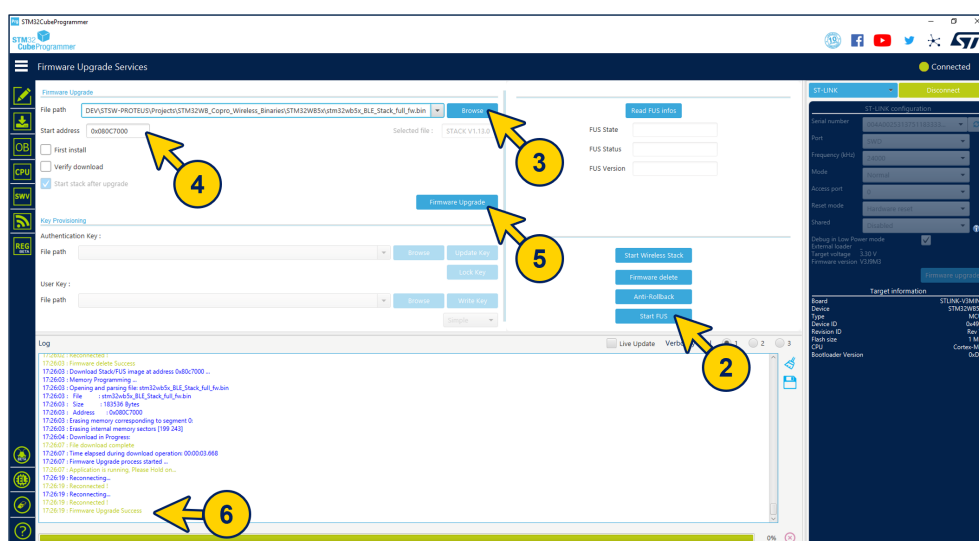
Table 3. STM32WB coprocessor wireless binaries to use

Hardware	Connectivity	Role	Wireless coprocessor binary	Install address
STEWAL-PROTEUS	BLE	Slave	stm32wb5x_BLE_Stack_full_fw.bin (V1.13.0)	0x080C7000

Step 6. Select *Firmware Upgrade* (see Figure 20, arrow No. 5).

Step 7. Wait for *Firmware Upgrade Success* (see Figure 20, arrow No. 6).

Figure 20. Firmware upgrade overview

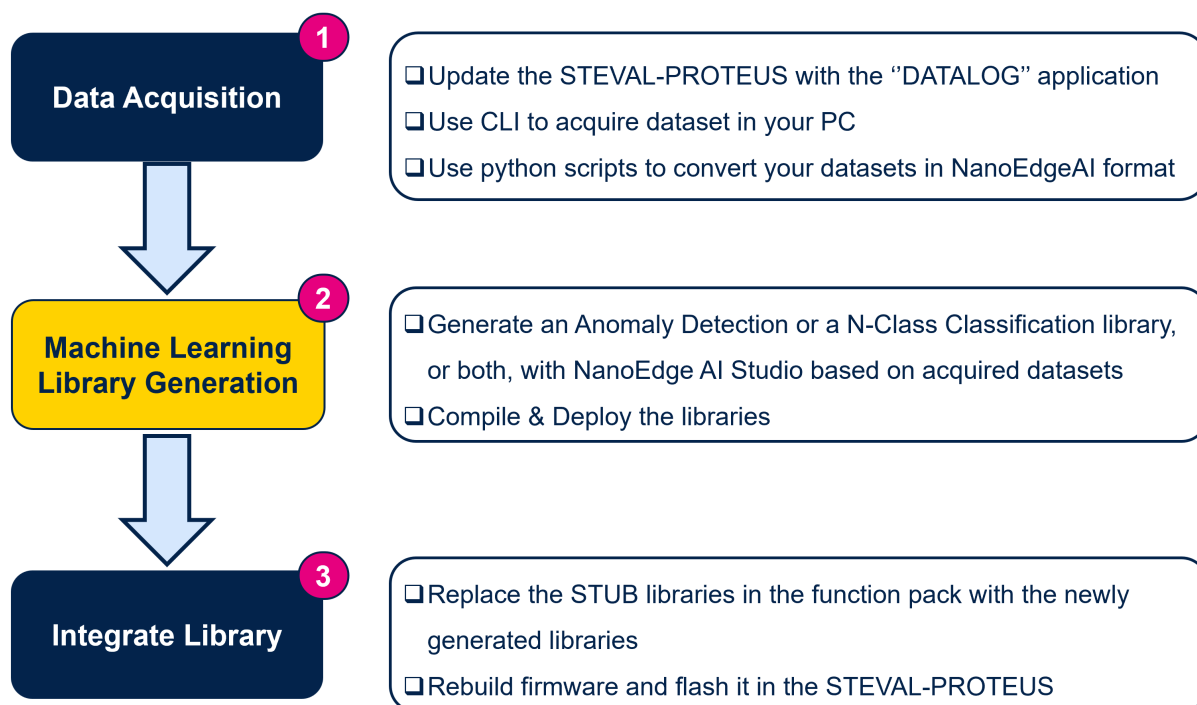


Note: The wireless coprocessor stack firmware update procedure corrupts the data already stored in the first part of the flash memory (OTA application). So, after the stack update procedure, you must download the OTA application as explained in Section 3.3.4.1.

4 Making and embedding the NEAI libraries

The following section describes how to generate the Anomaly Detection or N-Class Classification, library by using [NanoEdgeAIStudio](#), starting from the dataset acquisition, NEAI library creation, deployment, integration and testing on the [STEVAL-PROTEUS1](#), using the [FP-AI-PDMWBSOC](#).

Figure 21. Dataset acquisition, NEAI library creation, deployment, and integration



4.1 Data acquisition by data log

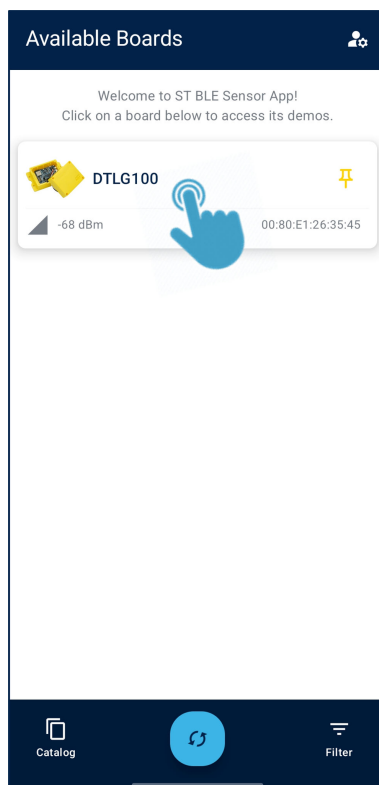
4.1.1 Uploading HS data log application firmware

The high-speed data log application firmware for STEVAL-PROTEUS comes with Bluetooth® Low Energy and FUOTA features.

STEVAL-PROTEUS1 kit is already programmed with the Bluetooth® Low Energy stack and OTA application. So, the high-speed data log application firmware can be downloaded into the MCU via OTA as explained in [Section 3.3.3: STEVAL-PROTEUS flash programming via Bluetooth® Low Energy OTA](#).

Via the [STBLESensor](#), a smartphone can discover the new device.

Figure 22. DTLG100 discovered by STBLESensor app



The available demos are limited. The Bluetooth® Low Energy feature just helps you to change the application firmware via OTA smoothly.

Figure 23. DTLG100 Bluetooth® Low Energy available demos for STBLESensor app

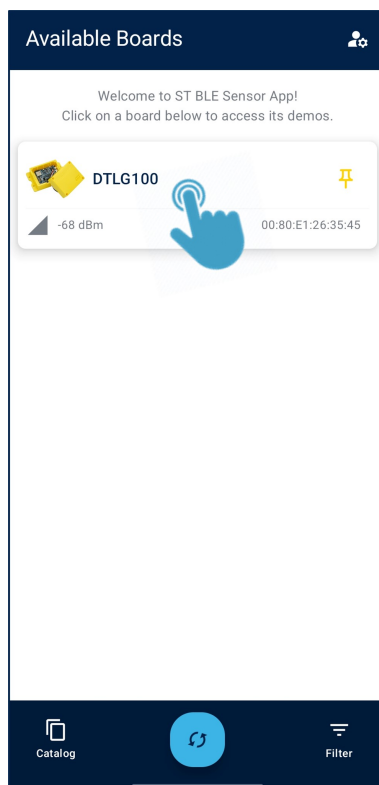
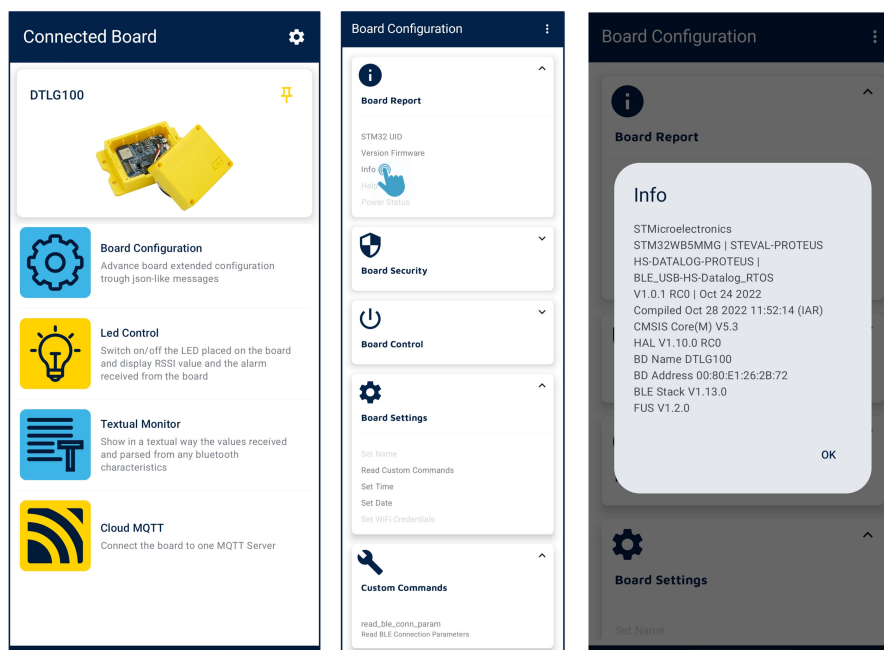


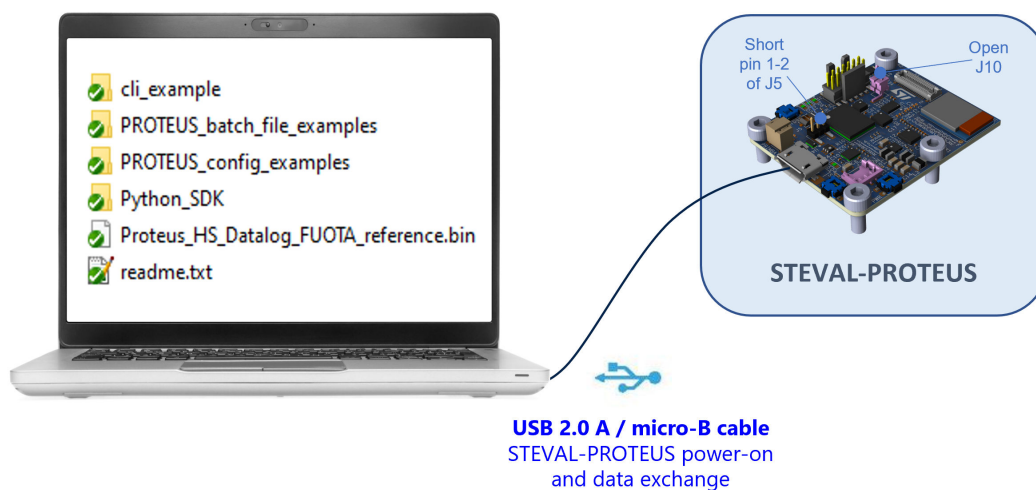
Figure 24. DTLG100 info showed by STBLESensor app



4.1.2 Creating HS data log by CLI

The STEVAL-PROTEUS has been programmed with the high-speed data log application firmware. The following figure shows how to connect the board to retrieve the data log we need.

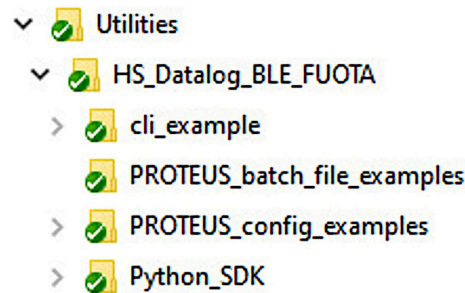
Figure 25. STEVAL-PROTEUS connected to a PC



4.1.2.1 High-speed data log utility

To retrieve data from the STEVAL-PROTEUS, firstly, create a dedicated folder that contains the utility to use.

Figure 26. High-speed data log utility folder



- **cli_example:** contains the binary executables
- **PROTEUS_config_examples:** contains a few device configuration files in json format to be used as parameter for the binary executable
- **PROTEUS_batch_file_examples:** contains a few ready-to-use batch files to call the binary executable with a specific device configuration file to obtain particular datasets
- **Python_SDK:** contains different Python scripts and classes that can be used to handle the obtained datasets

4.1.2.2 Dataset generation

Proceed to obtain the needed dataset.

Note: Before starting any new acquisition, if the blue LED of the STEVAL-PROTEUS is not blinking, press the reset button.

You can obtain the dataset by using the executable file running into a cmd.exe Windows shell.

You can locate it in the `.cli_example\bin` folder for both x86 and x64 OS system or, only for x64 OS system, in the `.cli_example\bin_64` directory.

Open a cmd.exe shell from any of the previous folders and launch the executable file entering `cli_example.exe [-COMMAND [ARGS]]`.

The possible commands are:

- the device configuration file: `-f <filename>`: Device Configuration file (JSON)
- the duration of the current acquisition (seconds): `-t <seconds>`: Duration of the current acquisition (seconds)

All commands are optional. Without any device configuration file, the default configuration stored in the firmware is used. If you do not enter the duration time, the current acquisition can be stopped by pressing the esc key.

The best way to start is to use the ready-to-use batch files located in the folder `.\PROTEUS_batch_file_examples\`.

Launch the `HS-DL_Run_IIS3DWB.bat` to obtain a dataset coming from the IIS3DWB 3-axis digital vibration sensors.

Note: During an acquisition, the blue LED turns off while the green LED blinks.

Figure 27. USB HS-Datalog CLI ready to start

```
C:\WINDOWS\system32\cmd.exe
USB HS-Datalog Command Line Interface example
Version: 2.5.0
Based on : ST USB Data Log 2.0.0
Device information:
{
  "deviceInfo": {
    "URL": "www.st.com",
    "alias": "PROTEUS",
    "dataFileExt": ".dat",
    "dataFileFormat": "HSD_1.0.0",
    "fwName": "USB-HS DATALOG",
    "fwVersion": "1.0.0",
    "nSensor": 3,
    "partNumber": "STEVAL-PROTEUS",
    "serialNumber": "004A00423550501820323642"
  }
}
Configuration imported from json file
Press any key to start logging
█
```

Press the return key.

Figure 28. USB HS-Datalog CLI acquisition running

```
C:\WINDOWS\system32\cmd.exe
+-----HSDatalog CLI-----+
| Streaming from:          PROTEUS |
| Elapsed: 12s           Remaining: 108s |
+-----Received Data-----+
| IIS3DWB_ACC           1791000 Bytes |
+-----+
+-----Tag labels-----+
| -0- ( ) SW_TAG_0 |
| -1- ( ) SW_TAG_1 |
| -2- ( ) SW_TAG_2 |
| -3- ( ) SW_TAG_3 |
| -4- ( ) SW_TAG_4 |
+-----+
Press the corresponding number to activate/deactivate a tag. ESC to exit!
█
```

While the acquisition is ongoing, a folder is created and updated. Its name is based on the date and time of acquisition. The folder contains a few files:

- *<SensorName_subSensorType>.dat* (*IIS3DWB_ACC.dat*), which contains raw sensor data coupled with timestamps
- *DeviceConfig.json*, which contains specific information about the device configuration that is necessary for the correct data interpretation
- *AcquisitionInfo.json*, which contains information about the acquisition and the data labelling

Figure 29. USB HS-Datalog CLI acquisition completed

```

C:\WINDOWS\system32\cmd.exe
-----HSDatalog CLI-----
| Streaming from:          PROTEUS |
| Elapsed:    120s        Remaining:  0s |
|-----Received Data-----|
| IIS3DWB_ACC          19134000 Bytes |
|-----Tag labels-----|
| -0- ( ) SW_TAG_0      |
| -1- ( ) SW_TAG_1      |
| -2- ( ) SW_TAG_2      |
| -3- ( ) SW_TAG_3      |
| -4- ( ) SW_TAG_4      |
|-----|
Press the corresponding number to activate/deactivate a tag. ESC to exit!
Press any key to continue . . . █
  
```

Note: As soon as the acquisition has been correctly acquired, the green LED turns off while the blue LED blinks.

4.1.2.3 Dataset acquisition folder

When an acquisition is performed, the HSDatalog generates a folder, named according to the date and time of acquisition, in which you can find different files: *DeviceConfig.json* and *AcquisitionInfo.json*.

DeviceConfig.json

It contains specific information about the device configuration, necessary for the correct data interpretation.

Figure 30. DeviceConfig.json - device attributes

```

object ▶ device ▶
  ▼ object {3}
    JSONVersion : 1.1.0
    UUIDAcquisition : 072f8dee-1dcf-4710-8b24-a2c233a3d95f
    ▼ device {3}
      ▶ deviceInfo {9}
      ▶ sensor [3]
      ▶ tagConfig {3}
  
```

The *device* consists of three attributes: *deviceInfo*, *sensor*, and *tagConfig*.
The *deviceInfo* attribute identifies the device.

Figure 31. DeviceConfig.json - device info

```
object ▶ device ▶ deviceInfo ▶
▼ object {3}
  JSONVersion : 1.1.0
  UUIDAcquisition : 072f8dee-1dcf-4710-8b24-a2c233a3d95f
  ▼ device {3}
    ▼ deviceInfo {9}
      URL : www.st.com
      alias : PROTEUS
      dataFileExt : .dat
      dataFileFormat : HSD_1.0.0
      fwName : USB-HS DATALOG
      fwVersion : 1.0.0
      nSensor : 3
      partNumber : STEVAL-PROTEUS
      serialNumber : 004A00423550501820323642
    ▶ sensor [3]
    ▶ tagConfig {3}
```

The *sensor* attribute is an array of attributes to describe all the sensors available on the board. Each sensor has a *unique ID*, a *name*, *sensorDescriptor*, and *sensorStatus* attributes.

Figure 32. DeviceConfig.json - sensor

```

object ▶ device ▶ sensor ▶
▼ object {3}
  JSONVersion : 1.1.0
  UUIDAcquisition : 072f8dee-1dcf-4710-8b24-a2c233a3d95f
  ▼ device {3}
    ▶ deviceInfo {9}
    ▼ sensor [3]
      ▼ 0 {4}
        id : 0
        name : IIS3DWB
        ▶ sensorDescriptor {1}
        ▶ sensorStatus {1}
      ▼ 1 {4}
        id : 1
        name : ISM330DHCX
        ▶ sensorDescriptor {1}
        ▶ sensorStatus {1}
      ▼ 2 {4}
        id : 2
        name : STTS22H
        ▶ sensorDescriptor {1}
        ▶ sensorStatus {1}
    ▶ tagConfig {3}

```

The *sensorDescriptor* attribute describes the main information about the single sensors through the list of its *subSensorDescriptor*. Each element of *subSensorDescriptor* describes the main information about the single subsensor (name, data type, sensor type, ODR, and full scale available, samples per unit of time supported, unit of measurement, etc.).

Figure 33. DeviceConfig.json - sensor descriptor

```

object ▶ device ▶ sensor ▶ 0 ▶ sensorDescriptor ▶
▼ object {3}
  JSONVersion : 1.1.0
  UUIDAcquisition : 072f8dee-1dcf-4710-8b24-a2c233a3d95f
  ▼ device {3}
    ▶ deviceInfo {9}
    ▼ sensor [3]
      ▼ 0 {4}
        id : 0
        name : IIS3DWB
        ▼ sensorDescriptor {1}
          ▼ subSensorDescriptor [1]
            ▼ 0 {9}
              ▼ FS [4]
                0 : 2
                1 : 4
                2 : 8
                3 : 16
              ▼ ODR [1]
                0 : 26667
                dataType : int16_t
                dimensions : 3
              ▼ dimensionsLabel [3]
                0 : x
                1 : y
                2 : z
                id : 0
              ▼ samplesPerTs {3}
                dataType : int16_t
                max : 1000
                min : 0
                sensorType : ACC
                unit : g
              ▶ sensorStatus {1}
              ▶ 1 {4}
              ▶ 2 {4}
            ▶ tagConfig {3}

```

The *sensorStatus* attribute describes the actual configuration of the related sensor through the list of its *subSensorStatus*. Each element of *subSensorStatus* describes the actual configuration of the single subsensor (whether the sensor is active or not, the actual ODR, time offset, data transmitted per unit of time, full scale, etc.).

Figure 34. DeviceConfig.json - sensor status

```
object ▶ device ▶ sensor ▶ 0 ▶ sensorStatus ▶
  ▼ object {3}
    JSONVersion : 1.1.0
    UUIDAcquisition : 072f8dee-1dcf-4710-8b24-a2c233a3d95f
    ▼ device {3}
      ▶ deviceInfo {9}
      ▼ sensor [3]
        ▼ 0 {4}
          id : 0
          name : IIS3DWB
          ▶ sensorDescriptor {1}
          ▼ sensorStatus {1}
            ▼ subSensorStatus [1]
              ▼ 0 {12}
                FS : 16
                ODR : 26667
                ODRMeasured : 26771.67969
                comChannelNumber : -1
                initialOffset : 0.019986
                isActive : true
                samplesPerTs : 1000
                sdWriteBufferSize : 32768
                sensitivity : 0.000488
                ucfLoaded : false
                usbDataPacketSize : 3000
                wifiDataPacketSize : 0
          ▶ 1 {4}
          ▶ 2 {4}
      ▶ tagConfig {3}
```

The *tagConfig* attribute describes the labels activated by the user.

Figure 35. DeviceConfig.json - tag config

```

object ▶ device ▶ tagConfig ▶
  ▼ object {3}
    JSONVersion : 1.1.0
    UUIDAcquisition : 072f8dee-1dcf-4710-8b24-a2c233a3d95f
    ▼ device {3}
      ▶ deviceInfo {9}
      ▶ sensor [3]
      ▼ tagConfig {3}
        ▼ hwTags [5]
          ▼ 0 {4}
            enabled : false
            id : 0
            label : HW_TAG_0
            pinDesc : value
          ▶ 1 {4}
          ▶ 2 {4}
          ▶ 3 {4}
          ▶ 4 {4}
          maxTagsPerAcq : 100
        ▼ swTags [5]
          ▼ 0 {2}
            id : 0
            label : SW_TAG_0
          ▶ 1 {2}
          ▶ 2 {2}
          ▶ 3 {2}
          ▶ 4 {2}

```

Note: The tag attribute is not used for the NEAI library generation.

AcquisitionInfo.json

It contains complementary information regarding the acquisition and the list of selected labels and tags (if labeling is enabled by the user).

Figure 36. AcquisitionInfo.json

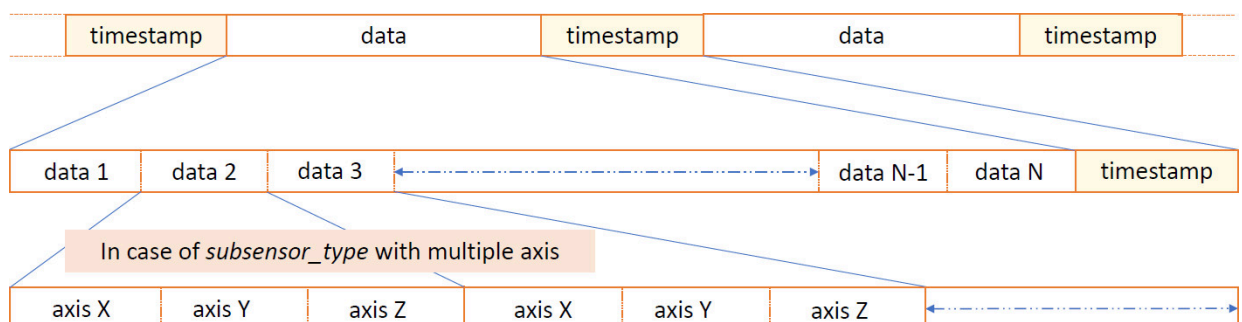
```
object ▶
  ▼ object {6}
    Description : descriptionTest
    Name : testName
    ▼ Tags [0]
      (empty array)
    UUIDAcquisition : 072f8dee-1dcf-4710-8b24-a2c233a3d95f
    end_time : value
    start_time : value
```

Raw data, saved in the *SensorName_subSensorType.dat* files, contain sensor data coupled with timestamps. The name of the file describes the sensor part number and the sensor type, as follows:

- Name: <sensor_name>_<subsensortype>.dat
 - <sensor_name>: component part number
 - <subsensortype>: ACC, GYRO, TEMP

A *filename.dat* file contains raw data and their timestamps. The related sensor configuration information is available in the *DeviceConfig.json* file.

Figure 37. filename.dat file - data stream structure



The data stream structure consists of:

- “Data k” ($k = 1 \dots N$) that represents a sample generated by a *subsensortype*.
If the *subsensortype* has multiple axes, such as acceleration sensors (IIS3DWB) each “data k” packet is one sample for each axis, as in the following schema: | axis X | axis Y | axis Z |
- The data length, in bytes (1, 2, or 4), which is defined in the *dataType* field available in the *subSensorDescriptor* attribute of the *DeviceConfig.json*.
- N, which corresponds to the value of the *samplesPerTs* field available in the *subSensorStatus* attribute of the *DeviceConfig.json*.
- The timestamp, which is a float value calculated in seconds.

4.1.2.4

Custom dataset generation

To customize the *DeviceConfig.json* file and pass it as a parameter for the CLI executable file, follow the procedure below.

1. Save the previous *DeviceConfig.json* file as *DeviceConfigCustom.json* (this name is not mandatory but is only as an example).
2. Open the *DeviceConfigCustom.json* file with a text editor and move to the attribute: `device -> sensor[id_s] -> sensorStatus -> subSensorStatus[id_ss]`
3. Look at the fields:

```
- FS
- ODR
- isActive
```

You can change **FS** (full scale) and **ODR** (output data rate) for the subsensor, choosing among the possible values that you can find in the corresponding fields of the attribute: `Device -> sensor[id_s] -> sensorDescriptor -> subSensorDescriptor[id_ss]`

The subsensor can be activated (that is, its dataset is stored) or deactivated (that is, its dataset is not stored) through the **isActive** Boolean field.

4. Save the *DeviceConfigCustom.json* modified file.
5. As already shown in [Section 4.1.2.2: Dataset generation](#), open a cmd.exe shell from the *.cli_example\bin* folder or *.cli_example\bin_64*, only for x64 OS system.
6. Move the *DeviceConfigCustom.json* file to this latter folder.
7. Inside the shell, launch the executable file entering: `cli_example.exe -f DeviceConfigCustom.json -t 120`

The dataset is acquired according to the custom device configuration as you can see from the *DeviceConfigCustom.json* file, in the related dataset acquisition folder, looking at: `device -> sensor[id_s] -> sensorStatus -> subSensorStatus[id_ss]`

4.1.2.5

Plot the datasets acquired

After acquiring and storing the dataset in the related folder, you can easily plot the data.

In the *.Python_SDK* directory, you can find some useful Python scripts, including one to plot the acquired dataset, *hsdatalog_plot.py*.

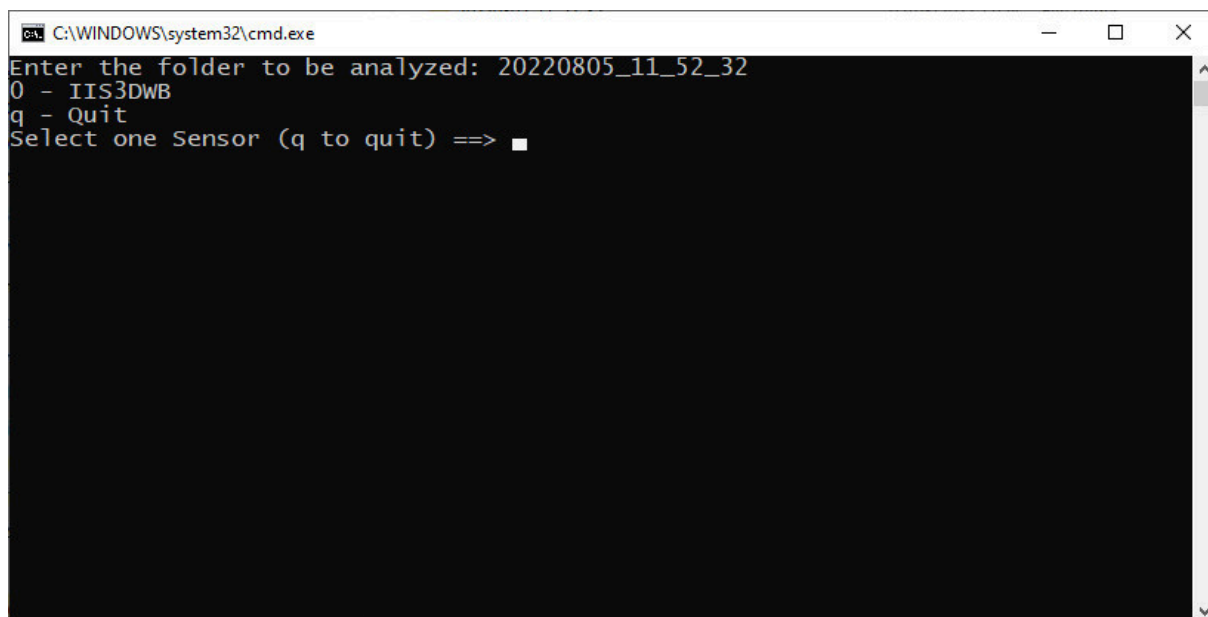
Python 3.7 must be already installed as well as the following modules:

- `asciimatics`
- `click`
- `colorama`
- `matplotlib`
- `numpy`
- `pandas`

You can manually install the above modules (through the `pip` command) or you can exploit the available *requirements.txt* file that describes all the libraries used by our SDK with their dependencies and versions, then launch the `pip install -r requirements.txt` command from the *.Python_SDK* folder.

In the *.PROTEUS_batch_file_examples* folder, you can find a ready-to-use batch file, *HS-DL_Plot.bat*, that launches a cmd.exe shell to enter the dataset folder to be analyzed.

Figure 38. HS-DL_Plot.bat shell



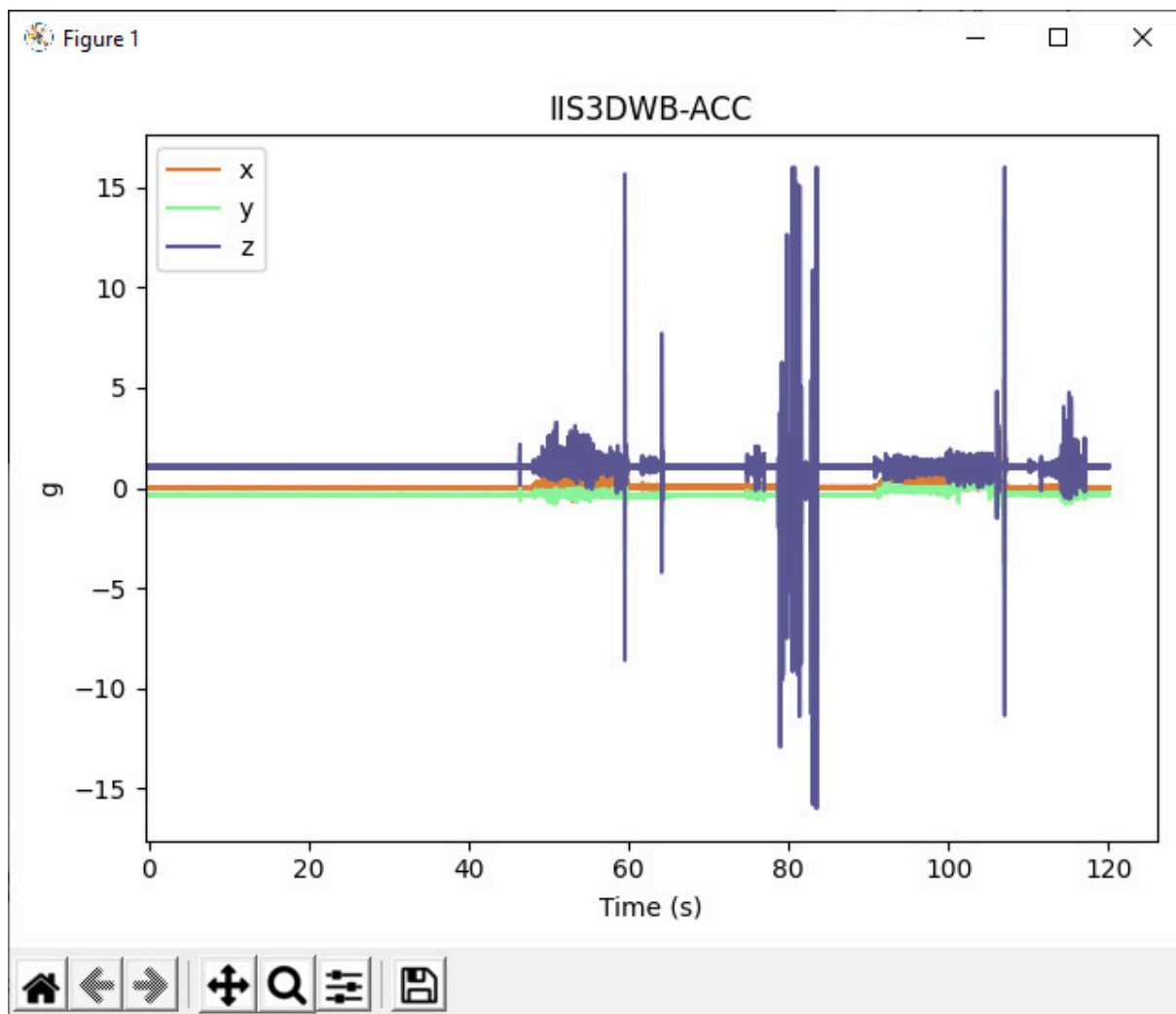
```

C:\WINDOWS\system32\cmd.exe
Enter the folder to be analyzed: 20220805_11_52_32
0 - IIS3DWB
q - Quit
Select one Sensor (q to quit) ==>

```

Enter the number related to the sensor whose data you want to plot (for example, '0') to display the graph.

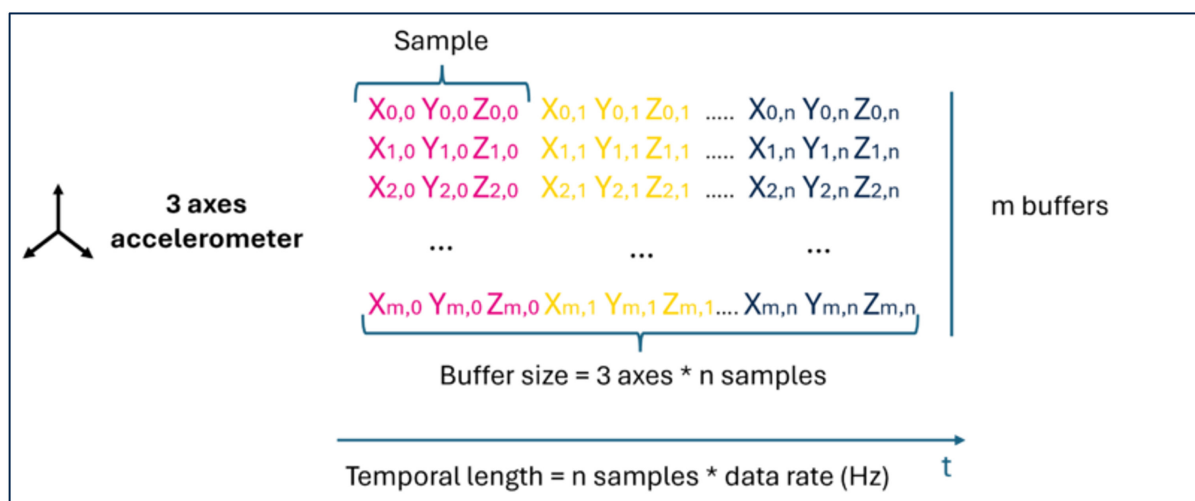
Figure 39. IIS3DWB data plot example



4.1.2.6 Convert the dataset in NanoEdgeAI format

As explained in [Section 4.1.2.2: Dataset generation](#), the acquired sensors data are collected in .dat files but this format is not accepted by NanoEdgeAI Studio. NanoEdgeAI Studio requires .csv file formatted as shown below.

Figure 40. NanoEdgeAI dataset format



The easiest way to convert your datasets to make it compliant to NanoEdgeAI format is to use the ready-to-use batch files located in the folder `$PROJ_DIR$\Utilities\HS_Datalog\PROTEUS_batch_file_examples`, here it is placed the `HS-DL_NanoEdge_Conversion.bat`.

Figure 41. Datasets converter batch example

```

DAT_To_CSV Converter
Enter the required parameters to obtain a .csv file compliant with NanoEdgeAI Studio format.
Enter the input folder name: My_Dataset
Enter the signal length: 1024
Enter the output folder name: My_Dataset

0 - ISM330DHCX
q - Quit
Select one Sensor (q to quit) ==> 0
2024-07-31 09:14:15,064 - HSDatalogApp.HSD_utils.converters - INFO - --> File: "ISM330DHCX_ACC_NanoEdge_segments_0.csv" chunk a
ppended successfully (converters.py:91)
2024-07-31 09:14:15,064 - HSDatalogApp - INFO - --> ISM330DHCX_ACC NanoEdge conversion completed successfully (hsdatalog_to_nan
oedge.py:86)
0 - ISM330DHCX
q - Quit
Select one Sensor (q to quit) ==> q
Bye!
Press any key to continue . . .
  
```

After launching batch file, you need to enter the following information:

1. The input folder name (e.g. `My_Dataset`)
2. The signal length, which is the buffer size
3. The output folder name (e.g. `My_Dataset`)

Then, you will be required to choose which sensor data you want to convert. Finally, you can close the batch entering the character `q`, as shown in the figure above.

Figure 42. Datasets converter batch example

Name	Date modified	Type	Size
Today			
ISM330DHCX_ACC.dat	7/31/2024 7:11 AM	DAT File	2,604 KB
DeviceConfig.json	7/31/2024 7:11 AM	JSON File	7 KB
AcquisitionInfo.json	7/31/2024 7:11 AM	JSON File	1 KB
_Exported	7/31/2024 9:14 AM	File folder	

When conversion will be completed, a new folder named **_Exported** will be created inside the output folder earlier specified, inside the new folder you will find the .csv file compliant with NanoEdgeAI format.

Note: Consecutive samples collected by a sensor form buffer and concatenating a certain number of buffers you get a dataset (or signal). Inside .csv converted dataset you will find a certain number of lines, which are the buffers, and its length is the signal length.

Note: Buffer size must be a power of 2 and dataset should contain at least 300 lines.

4.2 Library generation using NanoEdgeAI Studio

The NanoEdge AI Studio can generate libraries of four types:

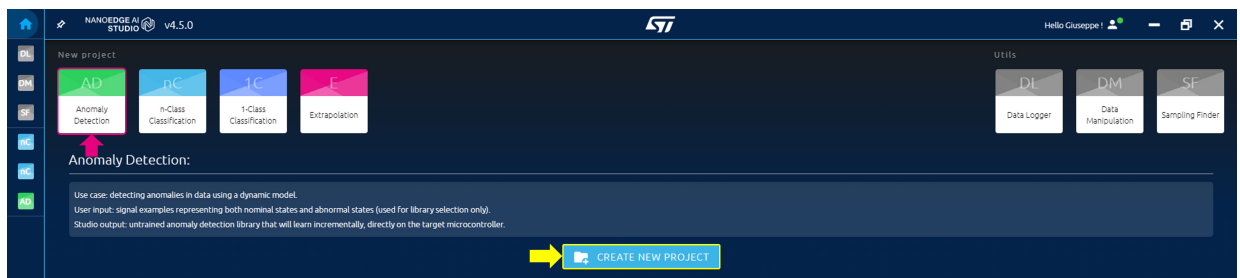
- Anomaly detection
- 1-class classification
- n-class classification
- Extrapolation

The user must be careful to collect the dataset correctly, during the different working conditions to be monitored, and then use it to generate the desired ML model.

4.2.1 NEAI anomaly detection library

To generate the anomaly detection library using the NanoEdgeAI tool, you need to start by creating the new project using the AD button as shown in the following figure.

Figure 43. Start a project (AD)



The text shows the information on what the anomaly detection is good for. You can create the project by clicking on the **CREATE NEW PROJECT** button.

The process to generate the anomaly detection library consists of six steps as shown below.

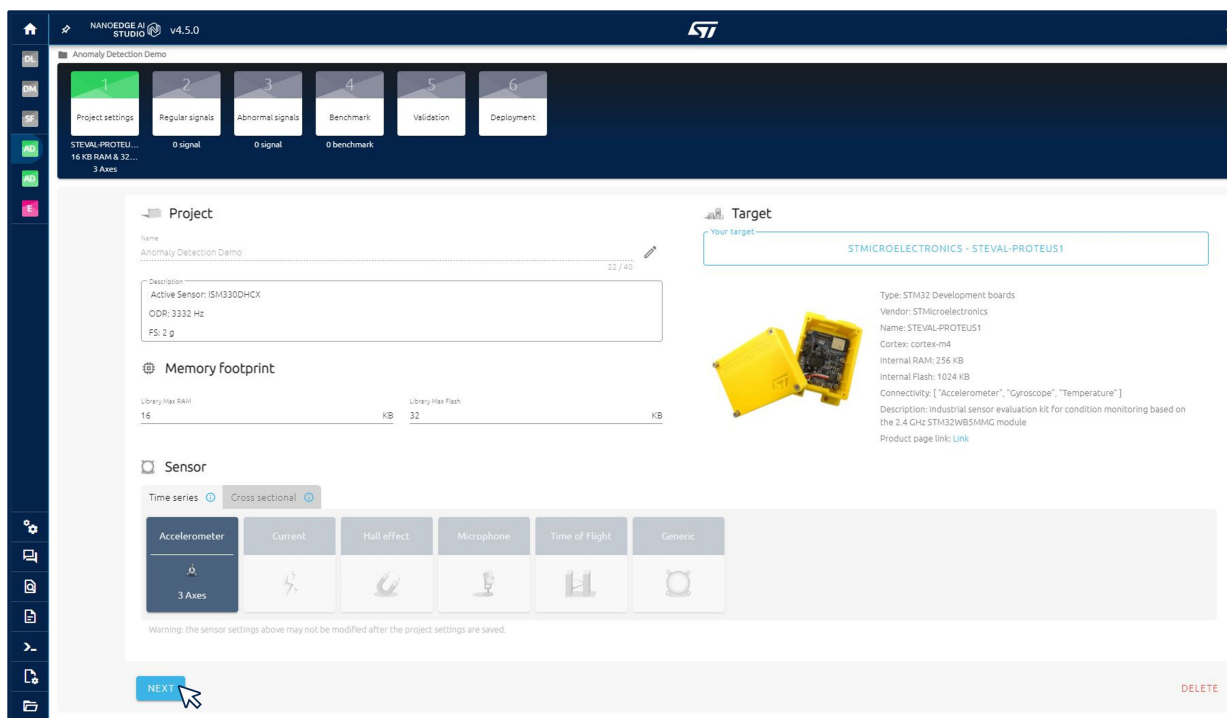
Figure 44. Six-step workflow



4.2.1.1 Project settings (AD)

1. Choose the settings for the library generation:
 - a project name and description
 - the right STM32 platform or a microcontroller type
2. Select the STEVAL-PROTEUS from the list provided in the drop-down menu under *Target*.
3. Enter the maximum amount of RAM to be allocated for the library. Usually, a few Kbytes is enough (but it depends on the data frame length used in the process of data preparation; 16 Kbytes is a good starting point).
4. Enter the maximum amount of FLASH to be allocated for the library. Usually, a few Kbytes is enough (but it depends on the data frame length used in the process of data preparation; 32 Kbytes is a good starting point).
5. Select the sensor type, that is the 3-axis accelerometer from the list in the drop-down menu.

Figure 45. Project settings (AD)



4.2.1.2 Insert the labeled data: regular and abnormal signals

To insert the labeled dataset, follow the procedure below.

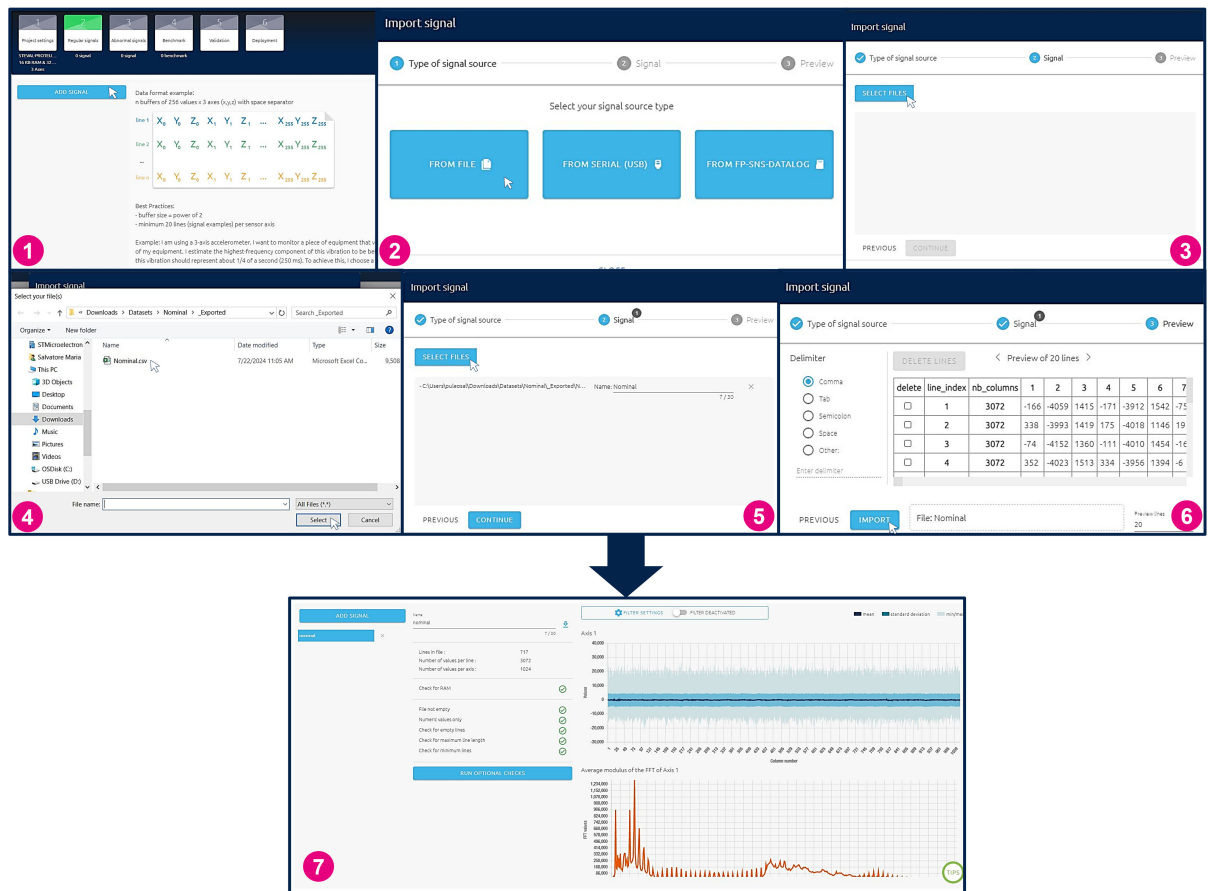
1. Start the data samples selection, for "Regular" class, pushing the "ADD SIGNAL" button.
2. Select the signal source.
Even if three options are available, we strongly recommend to use the first one, by selecting "FROM FILE" option.
3. Open the dialog panel to select the directory including the files *.csv to use.
4. Select the folder generated in the [Section 4.1.2.6: Convert the dataset in NanoEdgeAI format](#), with dataset previously classified: `./My_Dataset/ filename.csv`.

Note: Assign a class label to the imported data in the **Name** field once the data is imported. This label is used to differentiate the classes and is used in the emulation to show the predicted class.

5. Verify that the "Comma" delimiter is asserted and then click on the **IMPORT** button.
6. Check the data import in the last panel, where are exposed the time domain and frequency domain plot, plus some filter settings.

Repeat all the steps to import the dataset for abnormal class.

Figure 46. Classified data import

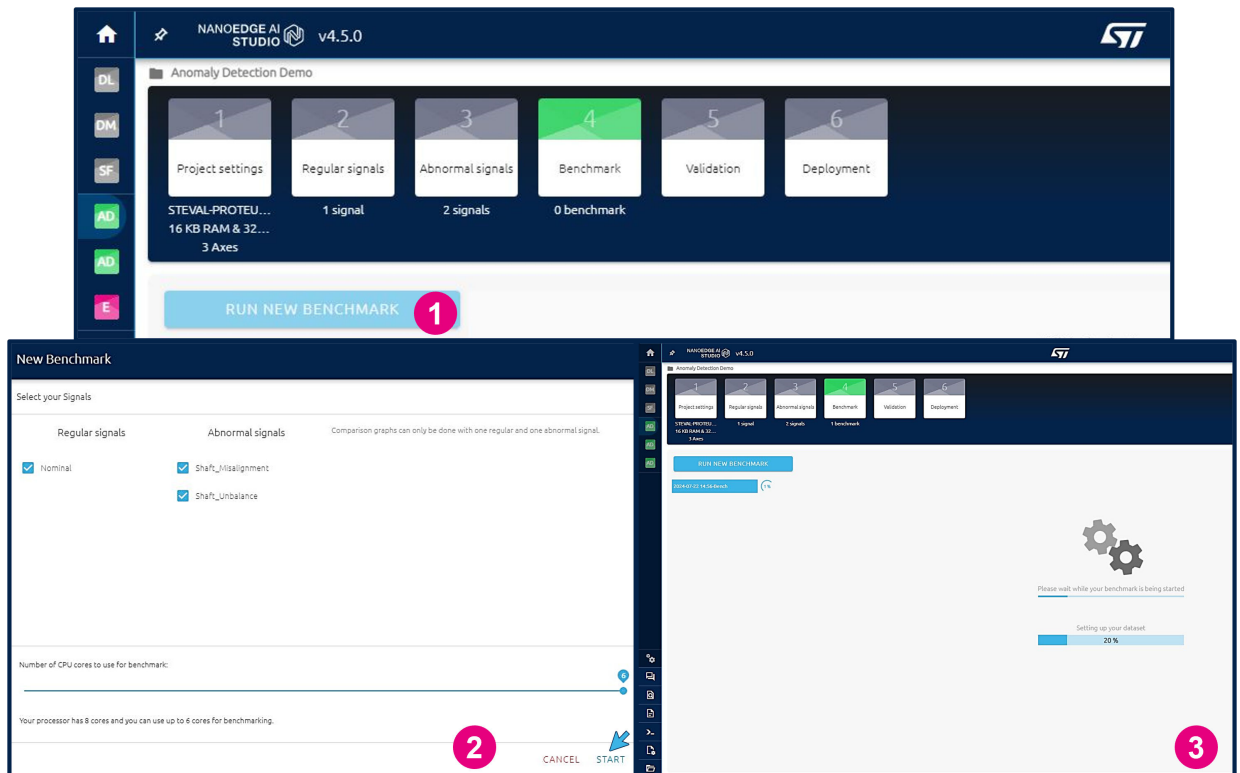


4.2.1.3 Run the benchmark to create the NEAI AD libraries

Benchmark the available models and choose the one that complies with the requirements and provides the best performance.

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

Figure 47. Start benchmark (AD)



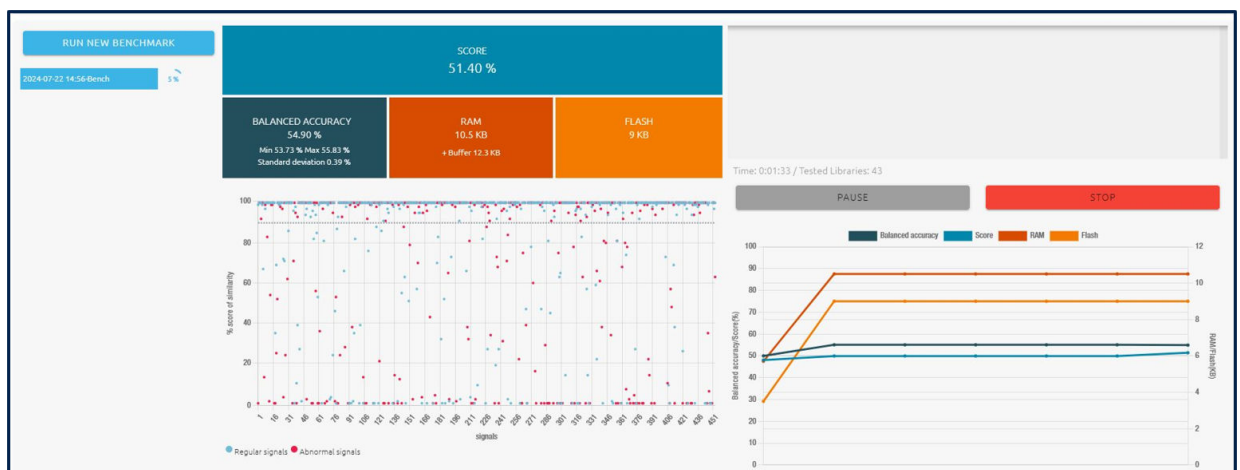
This perspective allows you to monitor some indicators like:

- Score
- Balanced accuracy
- RAM
- Flash memory

Keep in mind that:

- Benchmarks may take a long time (several hours) to complete and 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 the benchmark list under the RUN NEW BENCHMARK button.

Figure 48. Benchmark AD execution (1 of 2)



After some hours, you should obtain a result similar to the one of the figures below.

Figure 49. Benchmark AD execution (2 of 2)



You can choose whether to continue searching an improved library or pause/stop the benchmark to see the results.

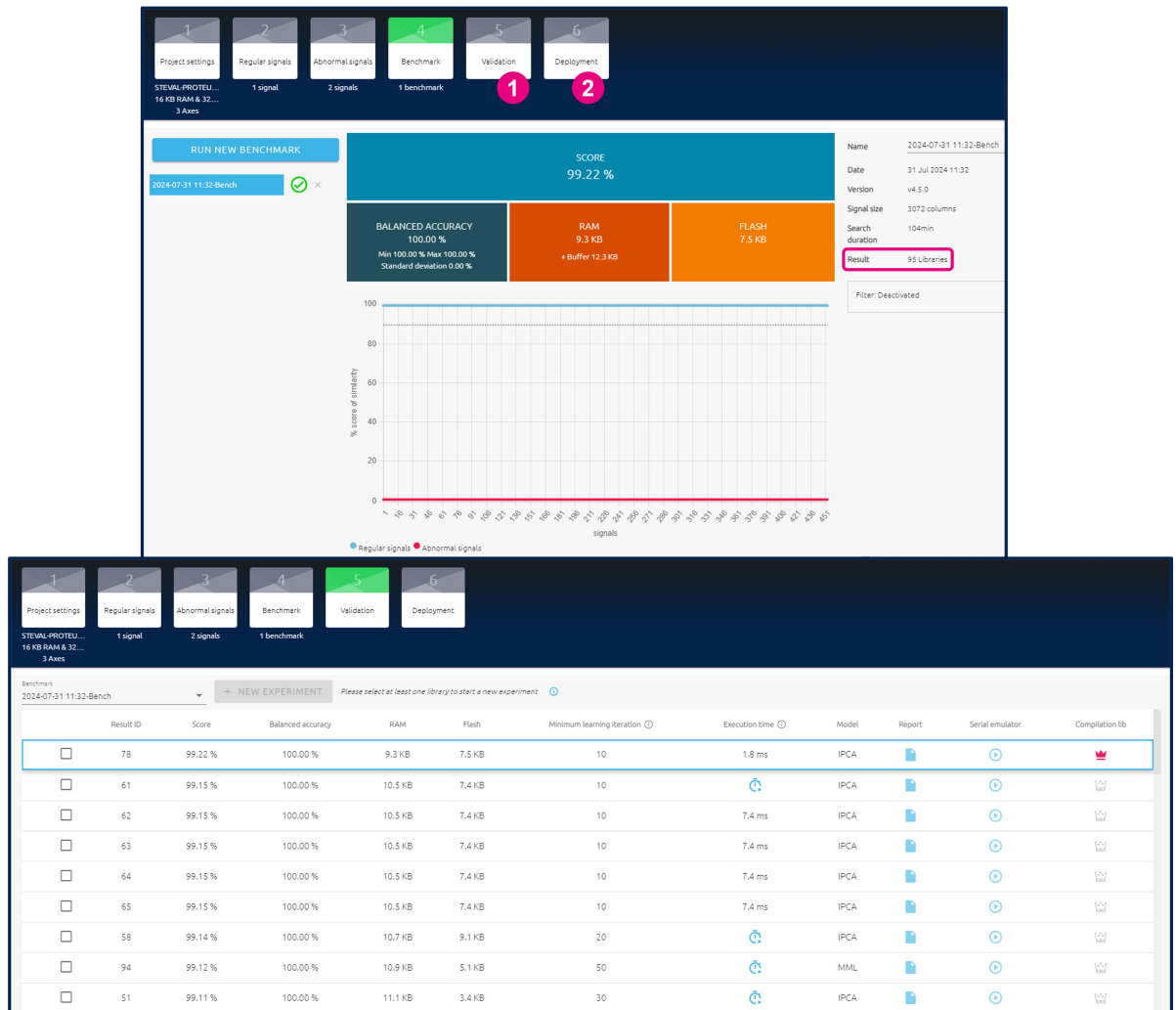
4.2.1.4

NEAI AD libraries evaluation

Now the user can proceed in two different ways:

- Start with a further models evaluation, pushing on **Validation** button. The tool shows a full list of the libraries evaluated, and allows to validate one or more libraries in the same time, selecting the libraries more suitable with your own final application.
- Accept the best library proposed by the tool, pushing on **Deployment** button, and follow the step described in the next paragraph.

Figure 50. Benchmark results and models summary (AD)



In this page is possible to compare the several library models generated by benchmark, in terms of performance index, RAM and Flash occupancy, execution time and model. To validate and testing one or more libraries, the user must proceed as follow:

1. Select at least one library (after the first selection the button **NEW EXPERIMENT** will be activated).
2. After that select the dataset acquired during the datalogging phase, for each classes condition to evaluate, and different from the data used for the benchmarking.
3. Run the evaluation experiment.

Figure 51. Models validation (AD)

The screenshot displays the 'Validation' tab of the software interface. At the top, there are six tabs: Project settings, Regular signals, Abnormal signals, Benchmark, Validation (active), and Deployment. Below the tabs, a 'Benchmark' section shows a table with columns: Result ID, Score, Balanced accuracy, RAM, Flash, Minimum learning iteration, Execution time, and Model. The table contains several rows of data, with the first row highlighted. A red arrow points from the '+ NEW EXPERIMENT' button to the 'New experiment' dialog box. The dialog box has two sections: 'File Learn' and 'File Regular'. The 'File Regular' section has a 'START' button. A red arrow points from the 'START' button to the 'Running experiment(s)' dialog box. The 'Running experiment(s)' dialog box shows a progress bar at 23% and the text 'Downloading libraries...'. A red arrow points from the 'START VALIDATION' button to the 'Running experiment(s)' dialog box.

Result ID	Score	Balanced accuracy	RAM	Flash	Minimum learning iteration	Execution time	Model
78	99.22 %	100.00 %	9.3 KB	7.5 KB	10	1.8 ms	IPCA
61	99.15 %	100.00 %	10.5 KB	7.4 KB	10	7.4 ms	IPCA
62	99.15 %	100.00 %	10.5 KB	7.4 KB	10	7.4 ms	IPCA
63	99.15 %	100.00 %	10.5 KB	7.4 KB	10	7.4 ms	IPCA
64	99.15 %	100.00 %	10.5 KB	7.4 KB	10	7.4 ms	IPCA
65	99.15 %	100.00 %	10.5 KB	7.4 KB	10	7.4 ms	IPCA

New experiment

File Learn

File Regular

File Abnormal

START

File Learn

Learnin...ion.csv (1.2 MB)

File Regular

Nominal...ion.csv (1.2 MB)

File Abnormal

Shaft_M...ion.csv (1.2 MB)

START VALIDATION

START

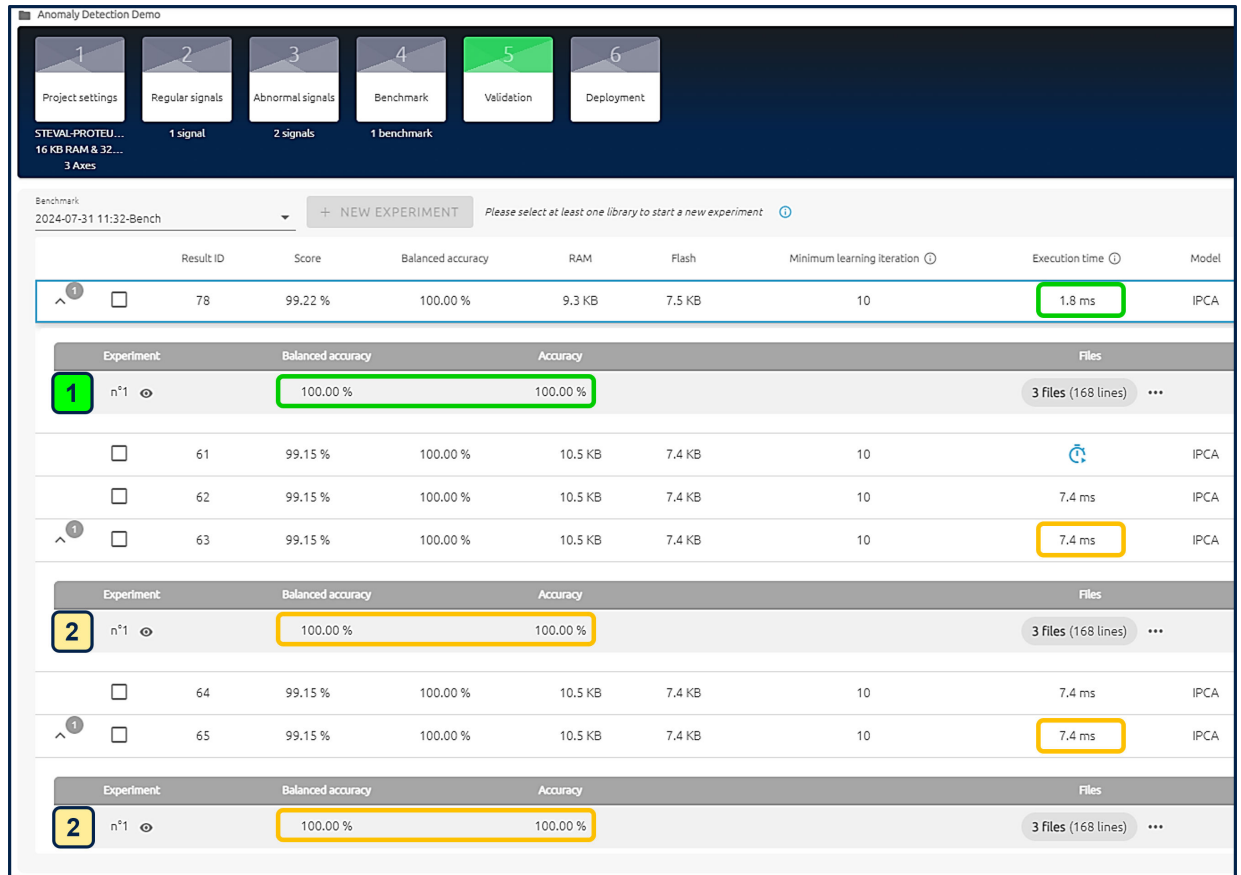
Running experiment(s)

23%

Downloading libraries...

- Evaluate and compare the model performances. For each library model will be provided the **Balanced Accuracy** and **Accuracy** related to validation dataset used, and the results could be different from the ranking outcomes from the benchmark.

Figure 52. Validation results (AD)



In the example shown above, the validation is performed between the models in the **1st**, **4th** and **6th** ranking position. After running the validation, the performance accuracy index has reached the maximum (100%) for all the libraries selected, but considering the execution time (**1.8 ms**) the best choice is the **1st**. Before proceeding with the final deployment the user must select this library pushing on the relates crown on the right side.

4.2.1.5 NEAI AD library deployment

The last step for the NanoEdgeAI tool is to compile and release the libraries in a unique ZIP file containing the library model in compiled format, plus other files to complete the integration.

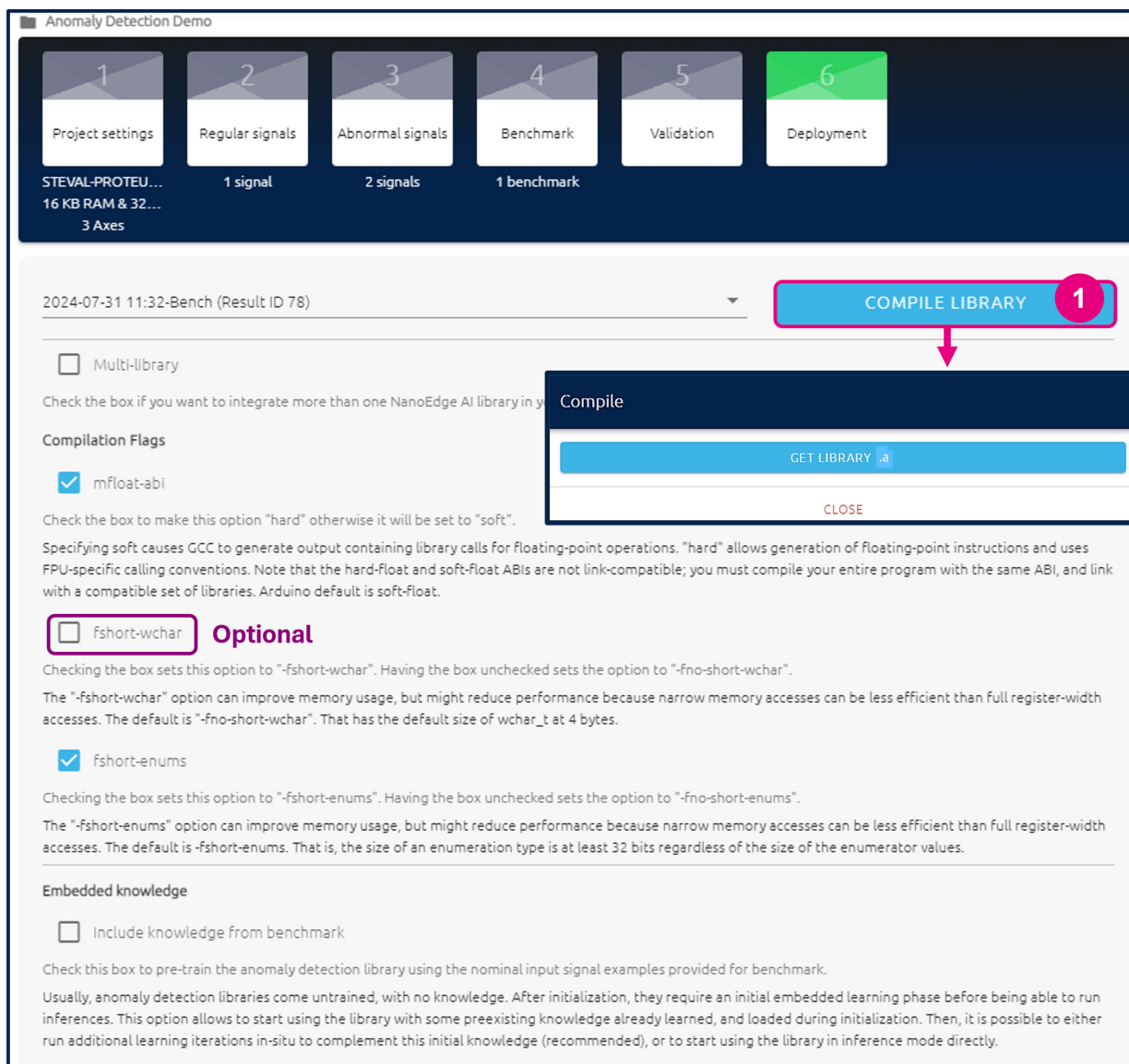
Pushing the last right button **Deployment**, to generate all the libraries files embedding the library models.

In case of classification models:

- Take care to check the **fshort-wchar** option if the library will be integrated in a KEIL® project and leave the other flags in their default state.
- Push the **Compile Library** button.

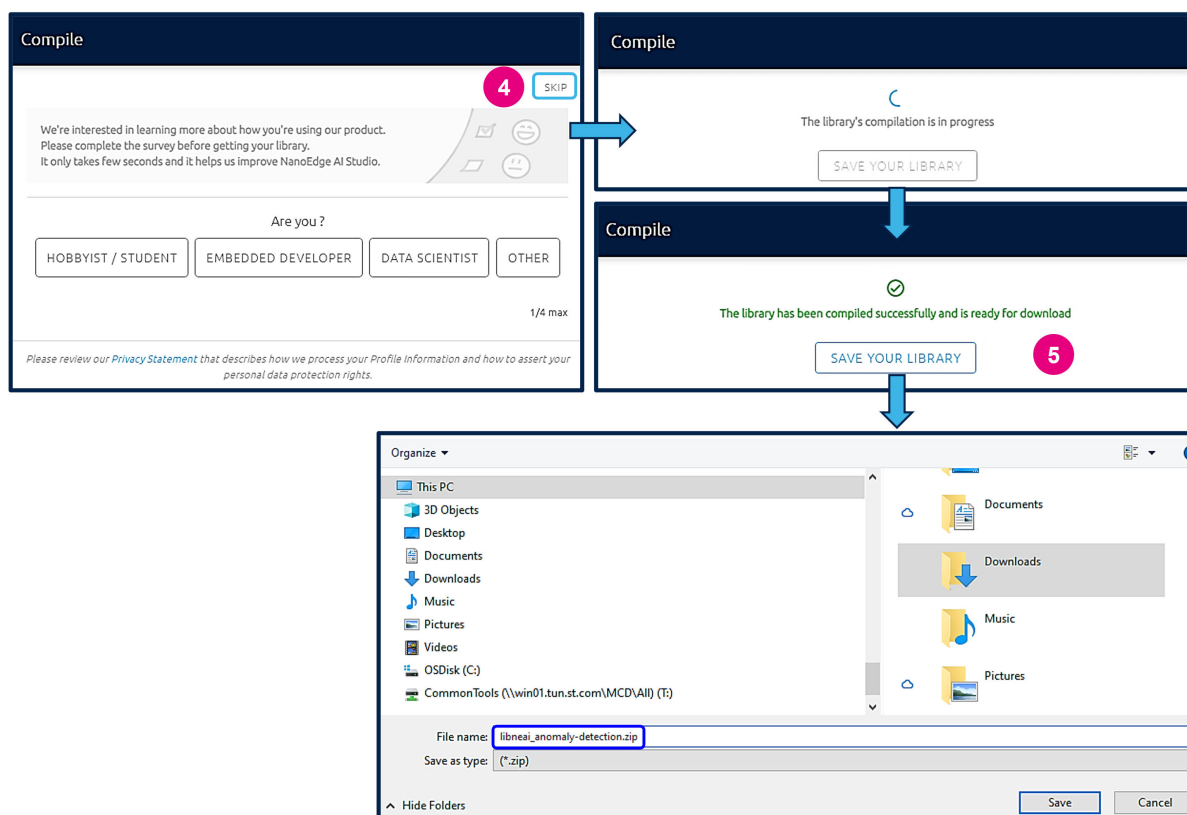
3. Push the **Get Library .a** button.

Figure 53. Deployment setting (AD)



4. Complete the survey or select **SKIP** option in the window to start the compilation and ZIP file creation.
5. Push on **SAVE YOUR LIBRARY** to store the ZIP file.

Figure 54. Library compilation and ZIP file saving (AD)



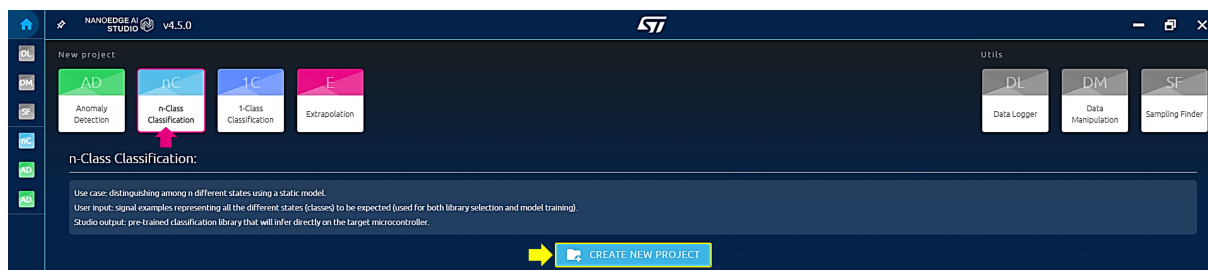
For further details about NanoEdgeAI Studio, see [NanoEdge AI Studio Documentation](#).

4.2.2

NEAI N-Class Classification library

To generate the classification library using the NanoEdgeAI tool, you need to start by creating the new project using the **N-Class Classification** button as shown in the following figure.

Figure 55. Start an N-Class Classification project



The text shows the information on what the classification is good for. You can create the project by clicking on the **CREATE NEW PROJECT** button. The process to generate the N-Class Classification library consists of five steps as shown below.

Figure 56. Five-step workflow

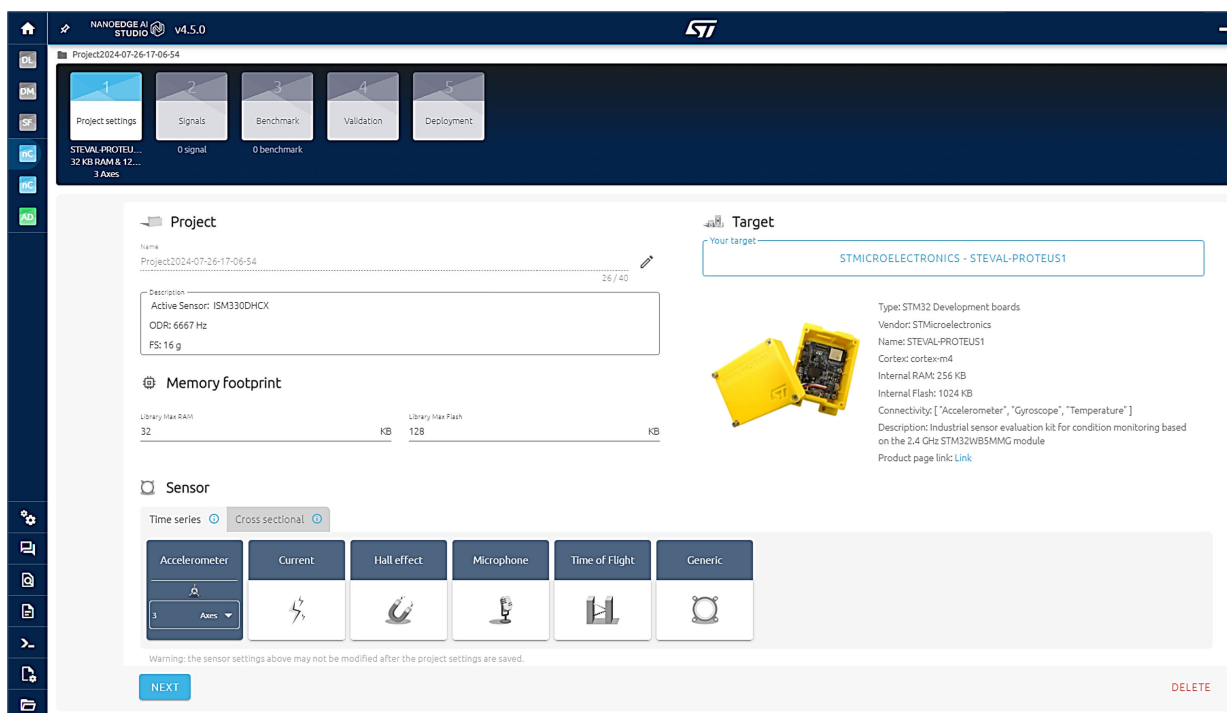


4.2.2.1

Project settings (NCC)

- Choose the settings for the library generation:
 - A project name and description.
 - The right STM32 platform or a microcontroller type
- Select the STEVAL-PROTEUS from the list provided in the drop-down menu under Target.
- Enter the maximum amount of RAM to be allocated for the library. Usually, a few Kbytes is enough (but it can depends on the data frame length used in the process of data preparation; 16 Kbytes is a good starting point).
- Enter the maximum amount of FLASH to be allocated for the library. Usually, a hundred Kbytes is enough (but it depends on the data frame length used in the process of data preparation; 128 Kbytes is a good starting point).
- Select the sensor type, that is the 3-axis accelerometer from the list in the drop-down menu.

Figure 57. Project settings (NCC)



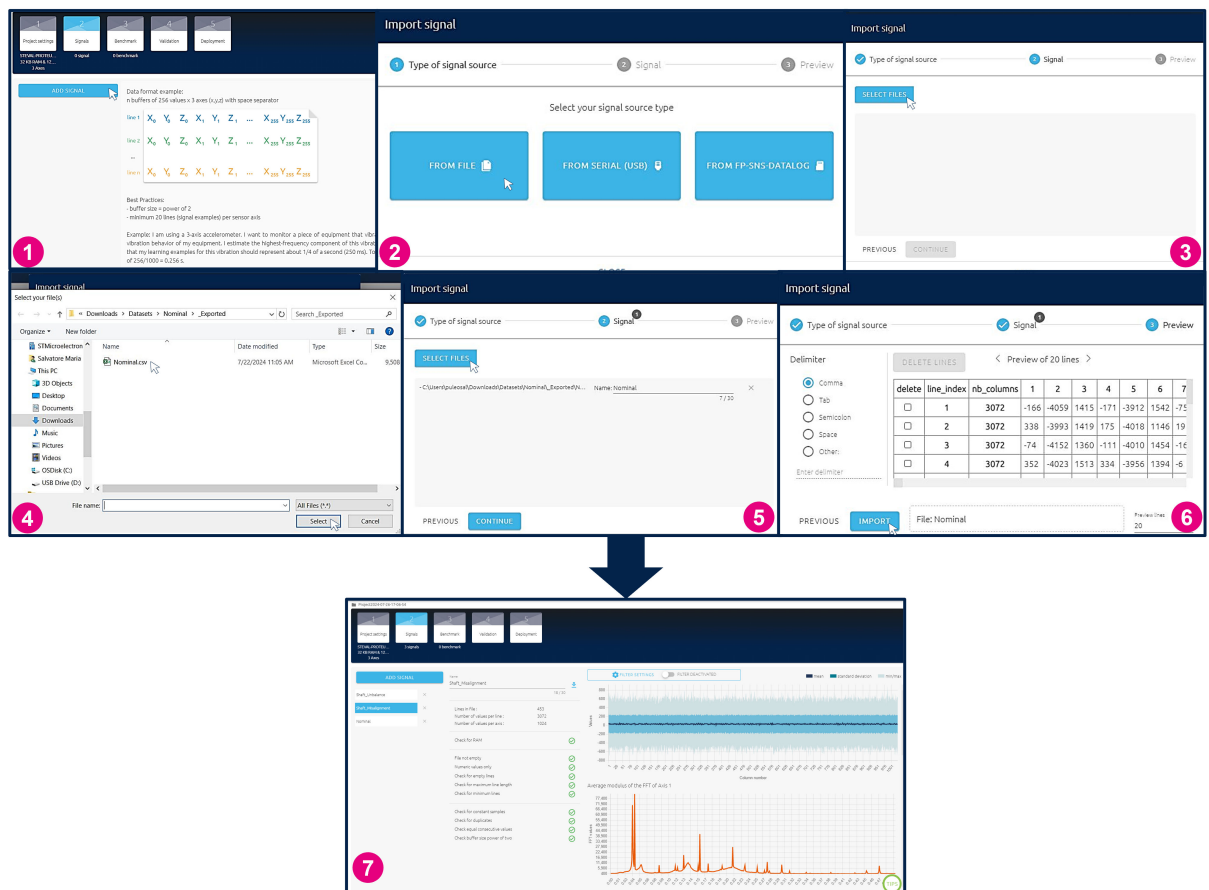
4.2.2.2

Insert the classified data

To insert the classified dataset, follow the procedure below.

1. Start the data samples selection pushing the “ADD SIGNAL” button.
2. Select the signal source. Even if three options are available, we strongly recommend using the first one, by selecting the “FROM FILE” option.
3. Open the dialog panel to select the directory including the files *.csv to use.
4. Select the folder generated in the Section 4.1.2.6, with dataset previously converted: `./My_Dataset/ filename.csv`.
5. Verify that the “Comma” delimiter is asserted and then click on the IMPORT button.
6. Check the data import in the last panel, where are exposed the time domain and frequency domain plot, plus some filter settings.
7. Repeat all the steps to import the dataset for other classes.

Figure 58. NCC classified data import



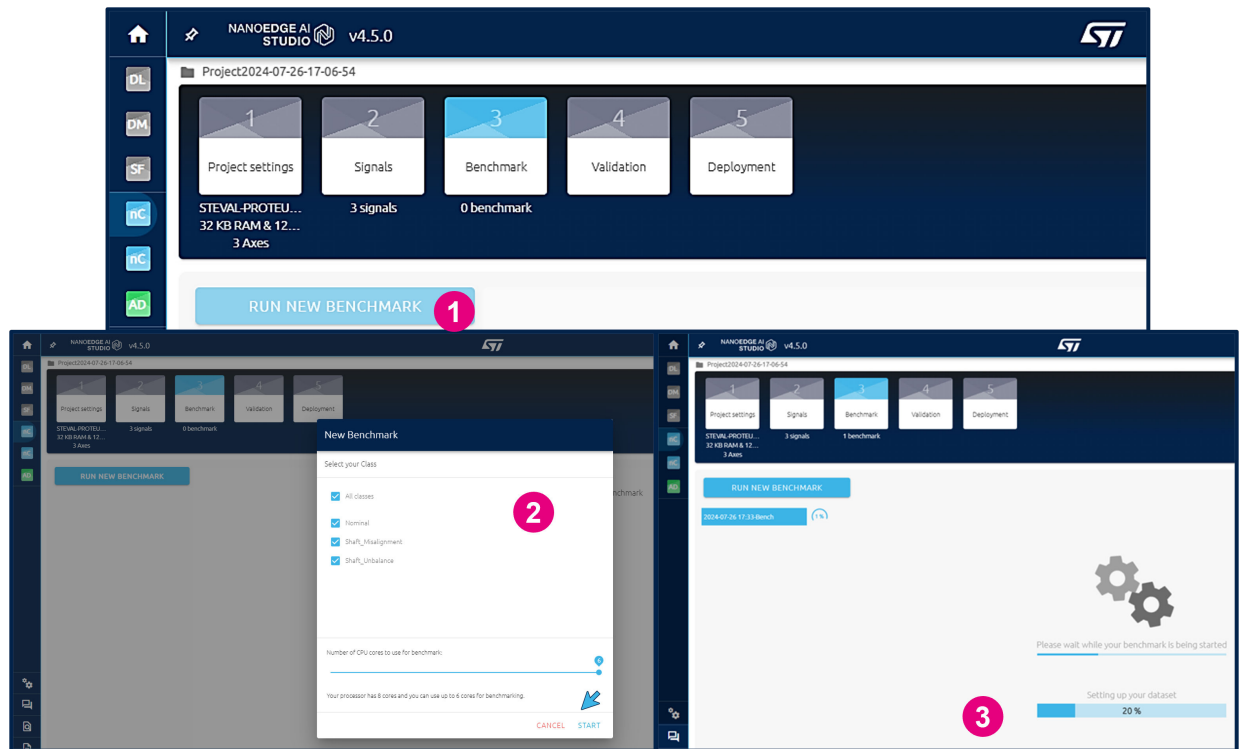
4.2.2.3

Run the benchmark to create the NEAI N-Class Classification libraries

Benchmark the available models and choose the one that complies with the requirements and provides the best performance.

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

Figure 59. Start benchmark (NCC)



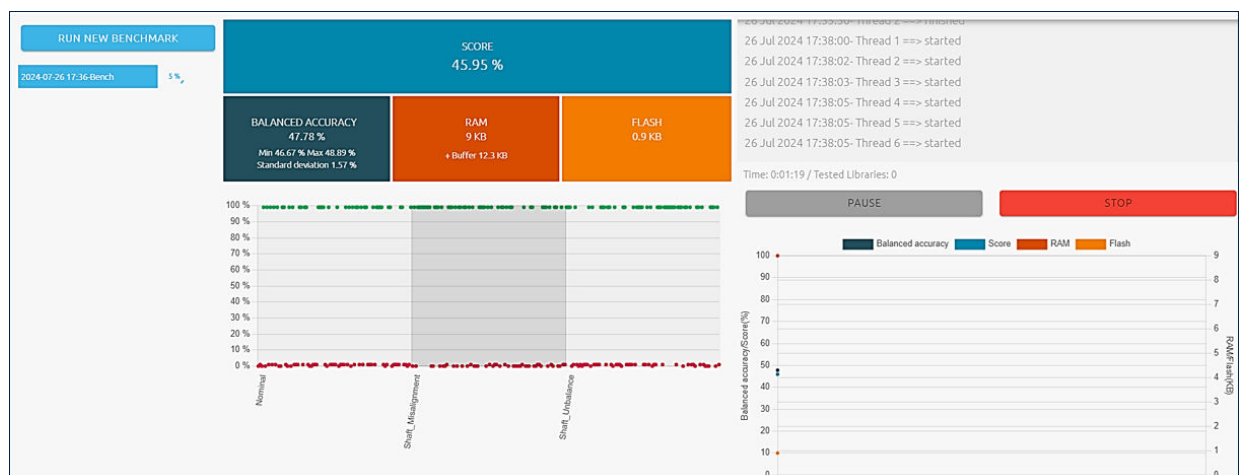
This perspective allows you to monitor some indicators like:

- Score
- Balanced accuracy
- RAM
- Flash memory

Keep in mind that:

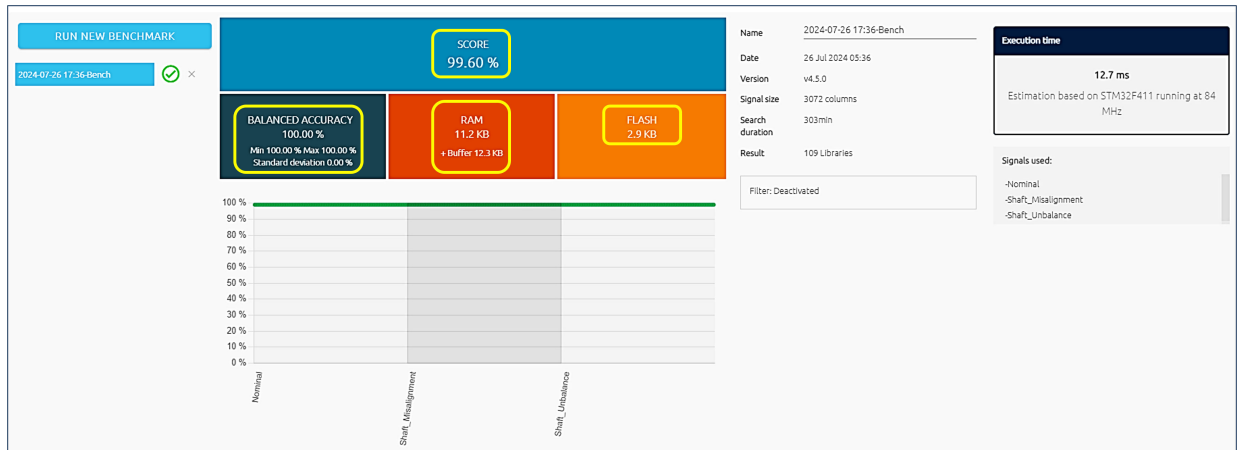
- Benchmarks may take a long time (several hours) to complete and find a fully optimized library
- Benchmarks can be paused/resumed, or stopped at any time, without canceling the process (the best library found is not lost)
- Benchmark progress in percentage is displayed on the left side of the screen,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

Figure 60. Benchmark NCC execution (1 of 2)



You can choose whether to continue searching an improved library or pause/stop the benchmark to see the results. After some hours (5 hours for this example), you should obtain a result similar to the one of the figures below.

Figure 61. Benchmark NCC execution (2 of 2)



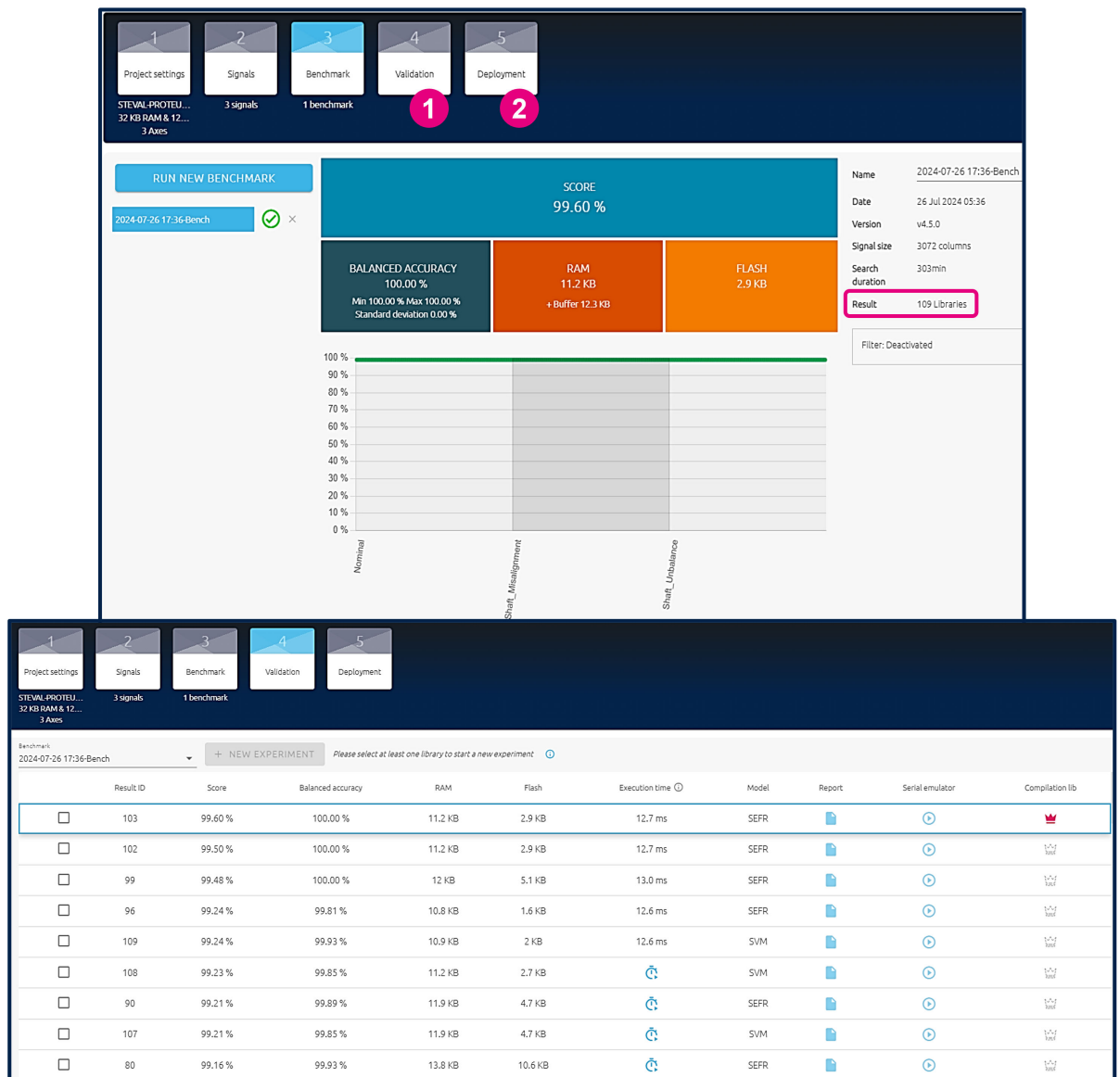
Where are summarized the library model more suitable for the tool, in terms of indicators detailed before.

4.2.2.4 NEAI N-Class Classification libraries evaluation

Now the user can proceed in two different ways:

- Start with a further models evaluation, pushing on **Validation** button. The tool shows a full list of the libraries evaluated, and allows to validate one or more libraries in the same time, selecting the libraries more suitable with your own final application.
- Accept the best library proposed by the tool, pushing on **Deployment** button, and follow the step described in the next paragraph.

Figure 62. Benchmark results and models summary (NCC)



In this page is possible to compare the several library models generated by benchmark, in terms of performance index, RAM and Flash occupancy, execution time and model. To validate and testing one or more libraries, the user must proceed as follow:

1. Select at least one or more libraries (after the first one selection the button **NEW EXPERIMENT** will be activated).
2. After that select the dataset acquired during the datalogging phase, for each classes condition to evaluate, and different from the data used for the benchmarking.
3. Run the evaluation experiment.

Figure 63. Models validation (NCC)

The interface shows a navigation bar with five tabs: 1 Project settings, 2 Signals, 3 Benchmark, 4 Validation (active), and 5 Deployment. Below the tabs, the 'Validation' tab is active, displaying a table of benchmark results and a 'New experiment' dialog box.

Benchmark Results Table:

Result ID	Score	Balanced accuracy	RAM	Flash	Execution time	Model
103	99.60 %	100.00 %	11.2 KB	2.9 KB	12.7 ms	SEFR
102	99.50 %	100.00 %	11.2 KB	2.9 KB	12.7 ms	SEFR
99	99.48 %	100.00 %	12 KB	5.1 KB	13.0 ms	SEFR
96	99.24 %	99.81 %	10.8 KB	1.6 KB	12.6 ms	SEFR
109	99.24 %	99.93 %	10.9 KB	2 KB	12.6 ms	SVM
108	99.23 %	99.85 %	11.2 KB	2.7 KB	12.6 ms	SVM

New experiment dialog box:

The dialog box is titled 'New experiment' and contains a 'START' button. A blue box with the text 'Insert new dataset for each class to validate' is overlaid on the dialog. The dialog also shows a list of files to be used for validation:

- File Nominal: Nominal...ion.csv (1.2 MB)
- File Shaft_Misalignment: Shaft_M...ion.csv (1.2 MB)
- File Shaft_Unbalance: Shaft_U...ion.csv (1.2 MB)

A 'START VALIDATION' button is located at the bottom right of the dialog box.

Running experiment(s) dialog box:

The dialog box is titled 'Running experiment(s)' and shows a progress bar at 19% with the text 'Downloading libraries...' and a refresh icon.

4. Evaluate and compare the model performances.
For each library model will be provided the **Balanced Accuracy** and **Accuracy** related to validation dataset used, and the results could be different from the ranking outcomes from the benchmark.

Figure 64. Validation results (NCC)

Result ID	Score	Balanced accuracy	RAM	Flash	Execution time	Model
103	99.60 %	100.00 %	11.2 KB	2.9 KB	12.7 ms	SEFR
102	99.50 %	100.00 %	11.2 KB	2.9 KB	12.7 ms	SEFR
99	99.48 %	100.00 %	12 KB	5.1 KB	13.0 ms	SEFR
96	99.24 %	99.81 %	10.8 KB	1.6 KB	12.6 ms	SEFR
109	99.24 %	99.93 %	10.9 KB	2 KB	12.6 ms	SVM
1	100.00 %	100.00 %				

In the example shown above, the validation is performed between the models in the 1st, 3rd and 5th ranking position but, after running the validation, the performance accuracy index is changed and the best is the 5th. So, using the validation procedure the user was able to find out that the 5th library is the most accurate one. Before proceeding with the final deployment, the user must select this library pushing on the relates crown on the right side

4.2.2.5 NEAI N-Class library deployment

The last step for the NanoEdgeAI tool is to compile and release the libraries in a unique ZIP file containing the library model in compiled format, plus other files to complete the integration.

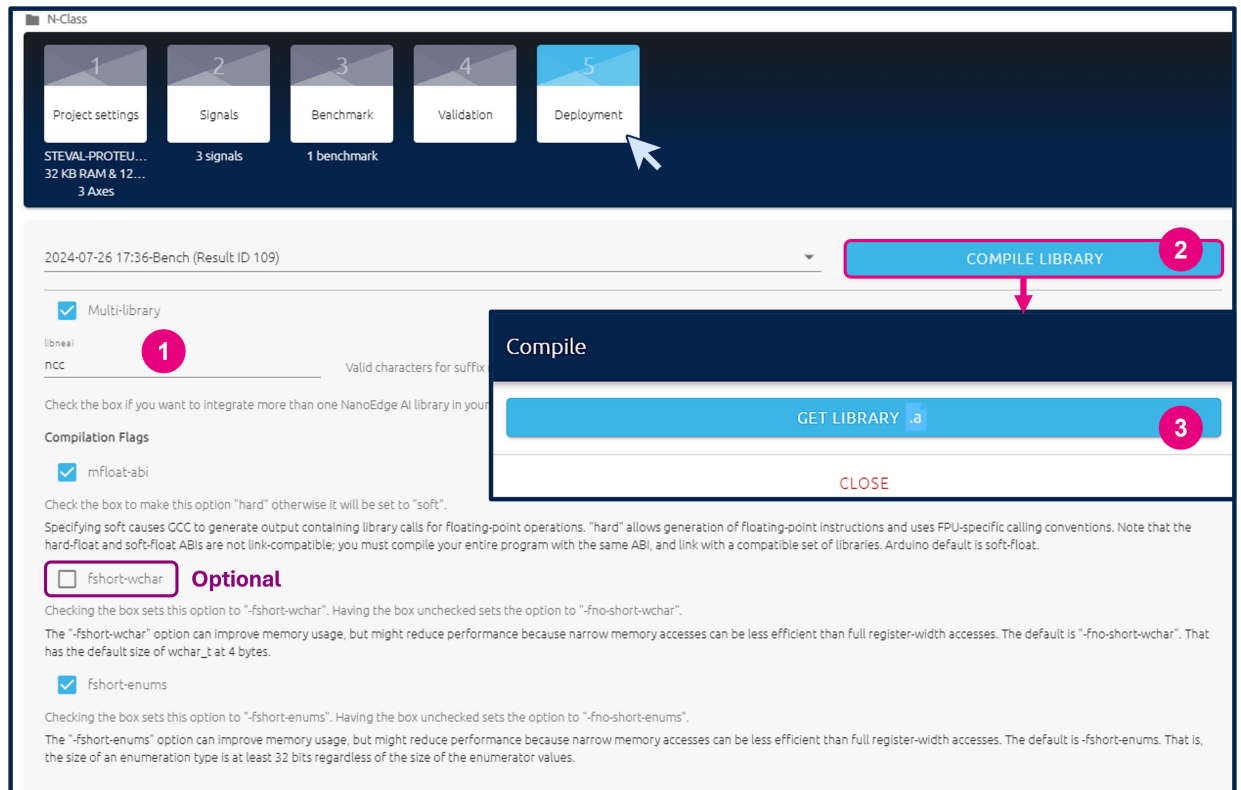
Pushing the last right button **Deployment**, to generate all the libraries files embedding the library models.

In case of classification models:

1. Set the **Multi-library** flag, to enable the suffix line insertion, and insert “**ncc**” just to identify these classification files from others to insert in the application. Take care to check the **fshort-wchar** option if the library will be integrated in a KEIL® project and leave the other flags in their default state.
2. Push the **Compile Library** button.

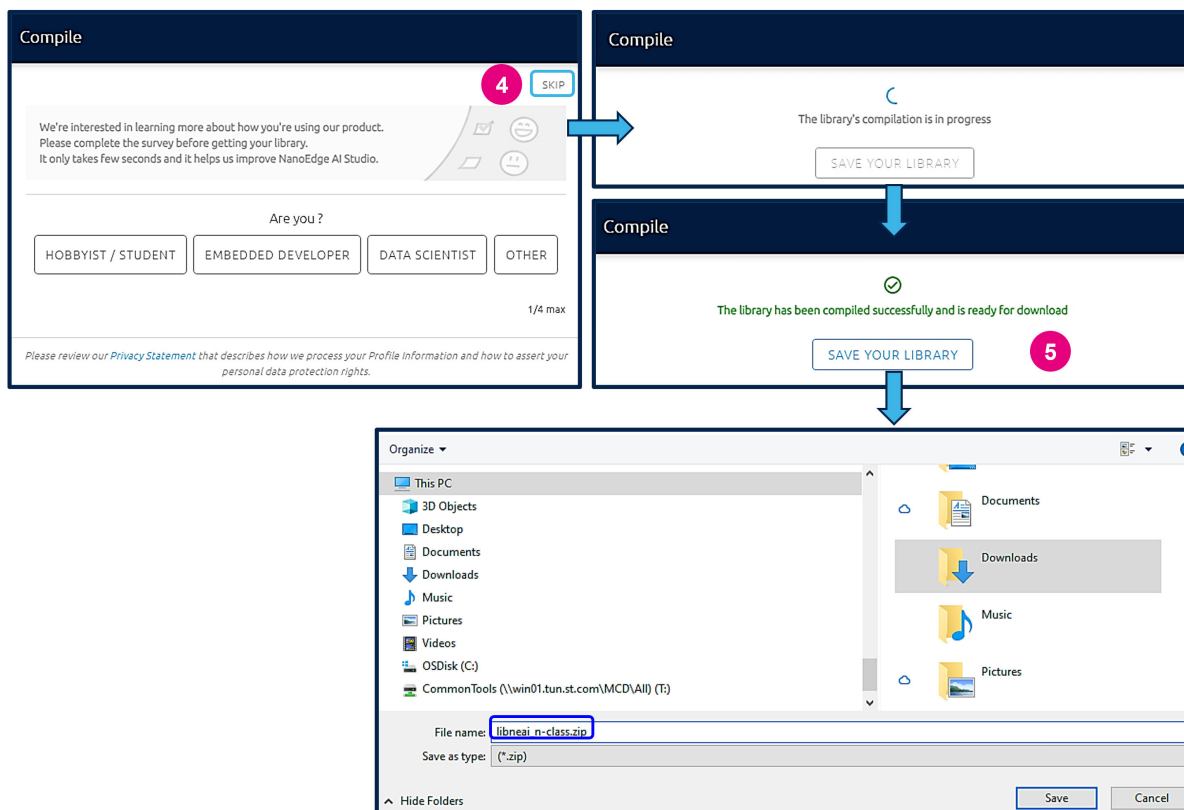
3. Push the **Get Library .a** button.

Figure 65. Deployment setting (NCC)



4. Complete the survey or select **SKIP** option in the window to start the compilation and ZIP file creation.
5. Push on **SAVE YOUR LIBRARY** to store the ZIP file.

Figure 66. Library compilation and ZIP file saving (NCC)



For further details about NanoEdgeAI Studio, see [NanoEdge AI Studio Documentation](#).

4.3 NEAI library integration

The **FP-AI-PDMWBSOC** firmware released contains “stub” demonstration libraries that are useful for observing functionality and behavior. These libraries must be properly replaced by those generated from NanoEdgeAIStudio.

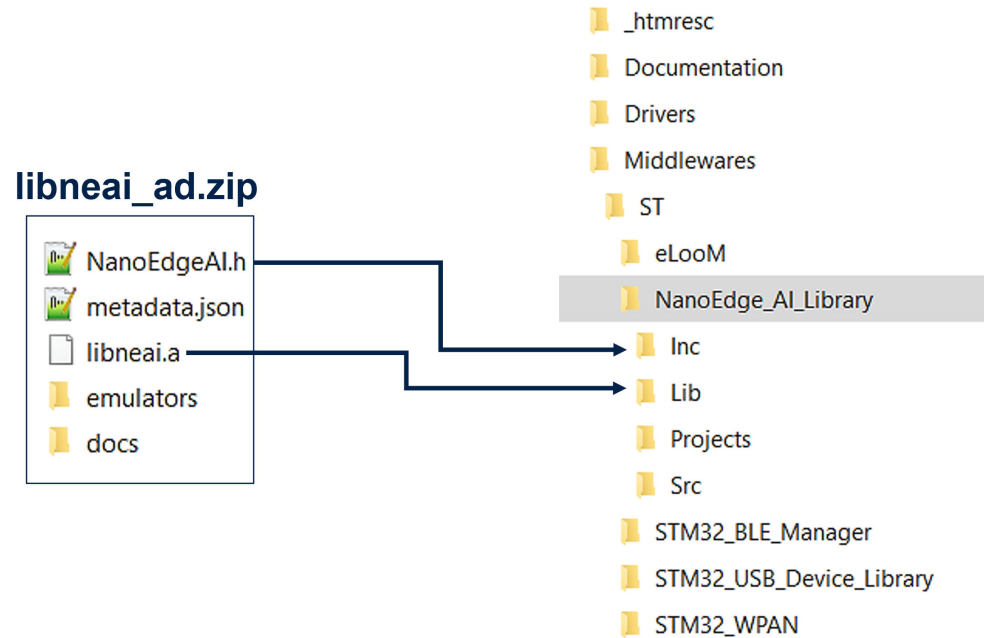
4.3.1 NEAI AD library integration

Open the ZIP file generated in [Section 4.2.1.5: NEAI AD library deployment](#) and proceed as follow:

- Open the following path in the project folder:
- **FP-AI-PDMWBSOC\Firmware\Middlewares\STNanoEdge_AI_Library\Lib**
- Replace the existing stub libraries with the generated libraries:
 - **libneai.a** (for IAR and STM32CubeIDE projects)
 - **libneai.lib** (for KEIL® projects)
- Open the path: **FP-AI-PDMWBSOC\Firmware\Middlewares\STNanoEdge_AI_Library\Inc**

- Replace header files with new deployed:
 - **NanoEdgeAI.h**

Figure 67. NEAI AD lib update in firmware architecture



- Compile again the FW application to update the application binary with the new library.
- Update the STEVAL-PROTEUS directly as explained in the [Section 3.3.3](#) and [Section 3.3.4](#).

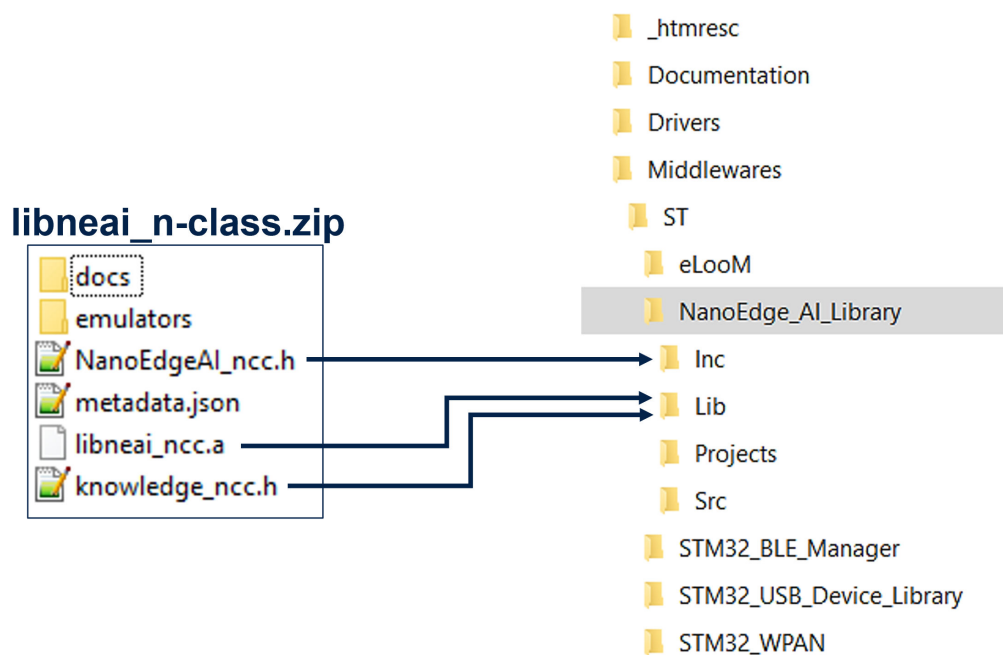
4.3.2 NEAI N-Class library integration

Open the ZIP file generated in [Section 4.2.2.5: NEAI N-Class library deployment](#) and proceed as follow:

- Open the following path in the project folder: **FP-AI-PDMWBSOC\Firmware\Middlewares\ST\NanoEdge_AI_Library\Lib**
- Replace the existing stub library with the generated library:
 - **libneai_ncc.a** (for IAR and STM32CubeIDE projects)
 - **libneai_ncc.lib** (for KEIL® projects)
- Open the path: **FP-AI-PDMWBSOC\Firmware\Middlewares\ST\NanoEdge_AI_Library\Inc**

- Replace header files with new deployed:
 - **NanoEdgeAI_ncc.h**
 - **knowledge_ncc.h**

Figure 68. NEAI NCC lib update in firmware architecture



- Compile again the FW application to update the application binary with the new library.
- Update the STEVAL-PROTEUS directly as explained in the [Section 3.3.3](#) and [Section 3.3.4](#).

5 Running FP-AI-PDMWBSOC

5.1 Using the STEVAL-PROTEUS with the STBLESensor app

5.1.1 Bluetooth® Low Energy connectivity features

A typical Bluetooth® Low Energy application involves two parts: a Bluetooth® Low Energy master and a Bluetooth® Low Energy slave.

The STEVAL-PROTEUS plays the role of the slave (or peripheral), acting as the Bluetooth® Low Energy GATT server. It advertises and waits for connection.

A mobile device that runs the [STBLESensor](#) app is the master (or central), acting as the Bluetooth® Low Energy GATT client. It scans for devices and establishes the connection.

The STEVAL-PROTEUS sends data to the Bluetooth® Low Energy GATT client via Bluetooth® Low Energy and to a laptop via UART.

5.1.2 Bluetooth® Low Energy workflow overview

In the Bluetooth® Low Energy workflow, the STEVAL-PROTEUS performs the following actions:

1. initializes the hardware and software resources, making them ready for Bluetooth® Low Energy connection to a smartphone
2. sends the libraries status, battery level and, if a detection is running, also the asset status, inside the Bluetooth® Low Energy advertising message
3. accepts the Bluetooth® Low Energy connection request by a mobile device running the [STBLESensor](#) app
4. sends information about the AI libraries status and allows setting some parameters related to the libraries and sensors
5. sends information about the battery status

The mobile device that runs the [STBLESensor](#) can start the training and inference phases.

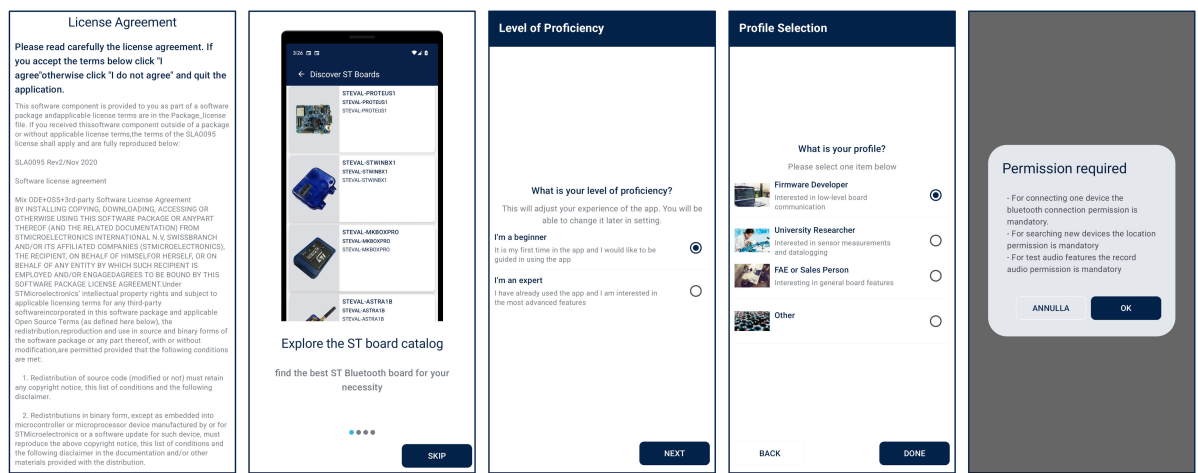
The application also sends information via UART to the PC, through the STDC14+ST-LINK+USB cable.

5.1.3 Bluetooth® Low Energy advertising

Since the evaluation board has been successfully programmed as described in [Section 3.3.3](#) and [Section 3.3.4](#) has been switched on, the STEVAL-PROTEUS, as the Bluetooth® Low Energy peripheral device, sends Bluetooth® Low Energy advertising packets to establish a connection with a Bluetooth® Low Energy central device.

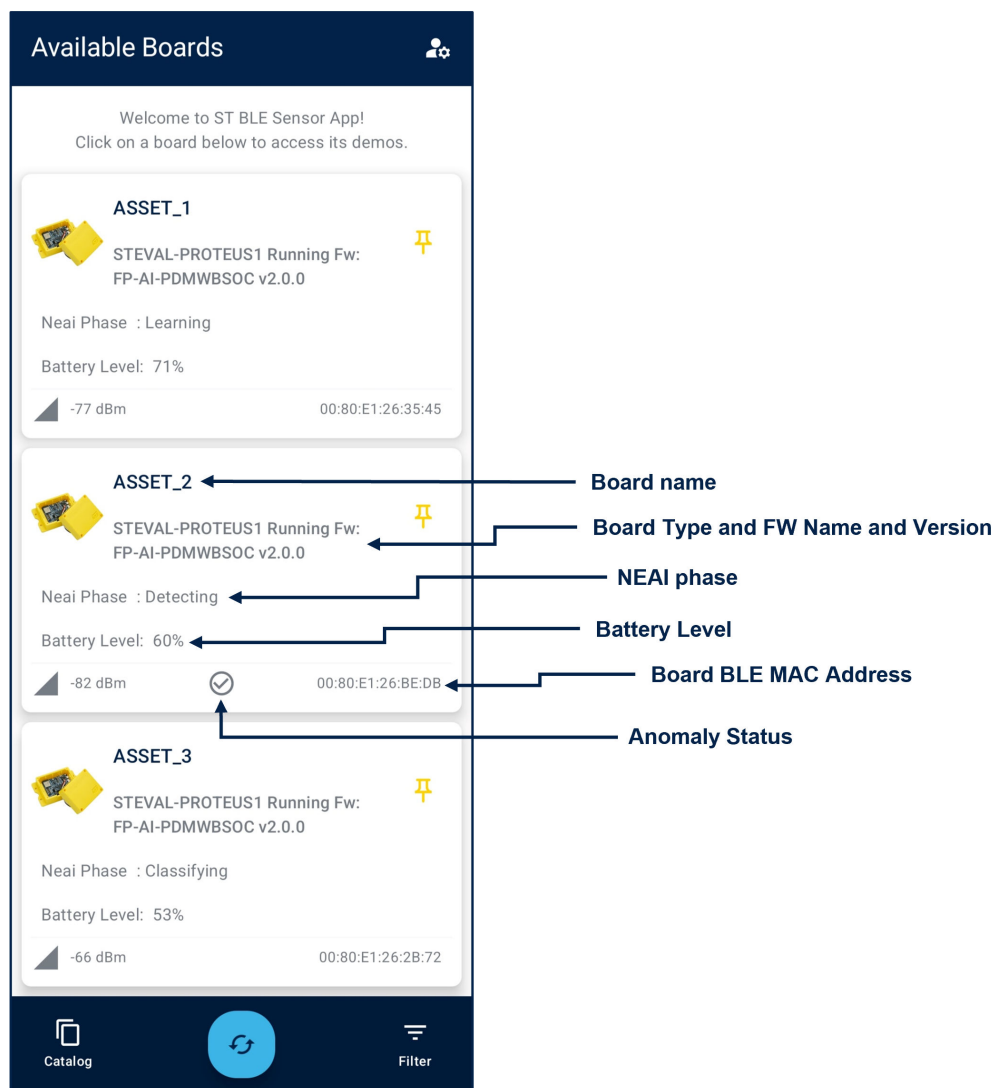
Launching the [STBLESensor](#), you can connect a device.

Figure 69. Main page of the STBLESensor app for Android



When you open the **STBLESensor** App for the first time, you will need to follow few steps before you can start using it. After granting all the required permissions, a new page displaying all available boards for connection will be shown.

Figure 70. Device list page (for Android STBLESensor)



Active phase and battery level aside, if a detection is running, the STEVAL-PROTEUS sends the *Asset Status* (normal or anomaly behavior) to the Bluetooth® Low Energy advertising packet.

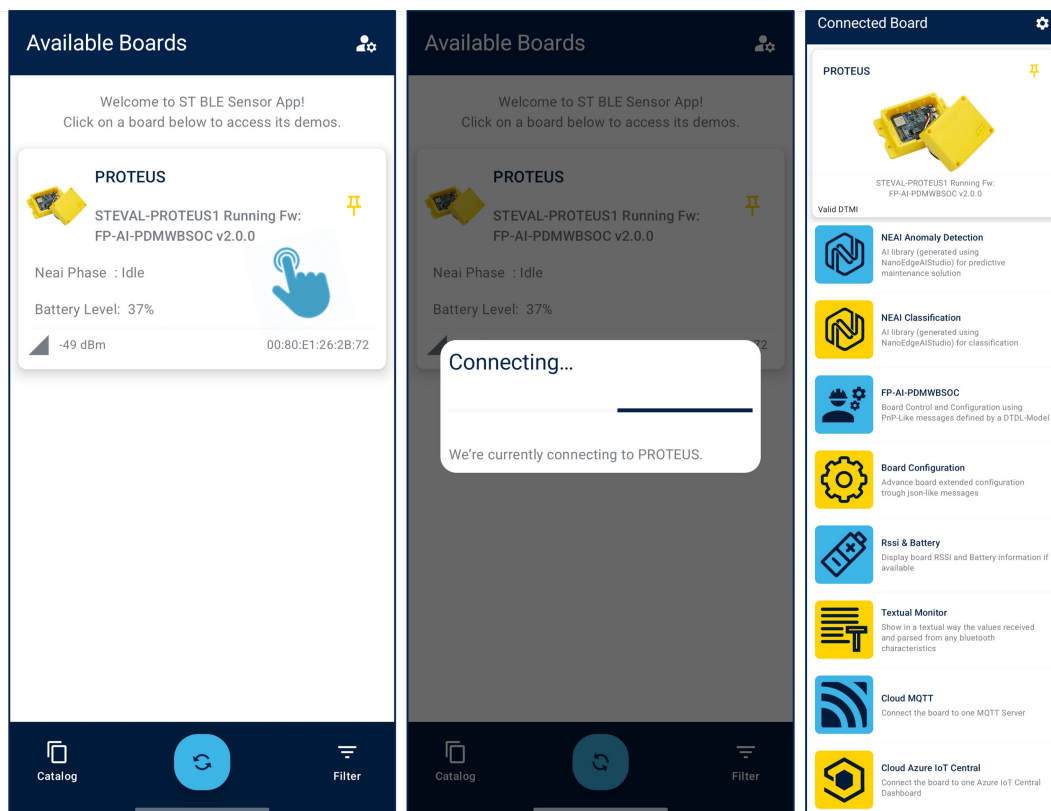
Table 4. Asset status icons

Normal behavior	Anomaly behavior

5.1.4 Connect a Bluetooth® Low Energy device and choose the application demo

To connect a device, just touch it. After that, you can retrieve all the information that the **STBLESensor** can manage for the selected board.

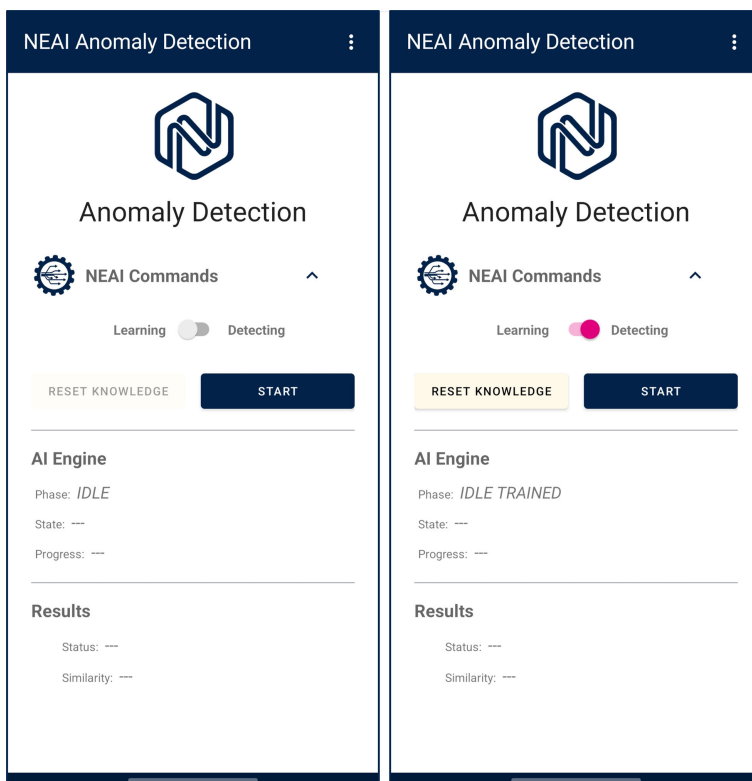
Figure 71. Device connection and sensor demo summary



5.1.5 NEAI anomaly detection demo

As soon as the STEVAL-PROTEUS has been connected, user can open the NEAI Anomaly Detection page selecting it from the menu.

Figure 72. NEAI anomaly detection demo (IDLE and IDLE TRAINED)



The STEVAL-PROTEUS remains in the idle state until it gets a command to start a processing phase: learning or detecting.

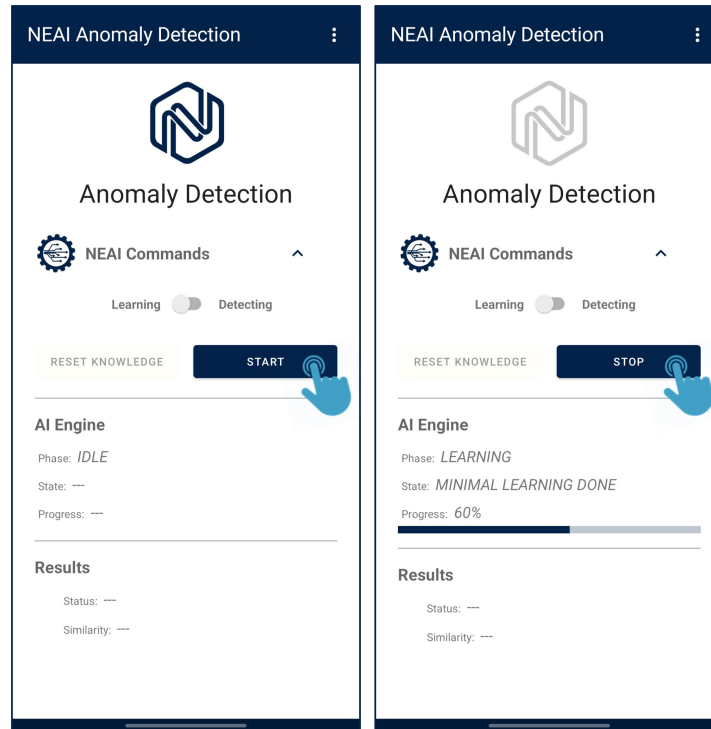
If you launch a learning phase, the library is trained with a number of signals that can be configured through the demo settings. The minimum amount of signals to ensure the proper operation is a predefined value in / *STM32CubeFunctionPack_AI_PDMWBSOC_v1.0//Middlewares/ST/NanoEdge_AI_Library/Inc/NanoEdgeAI.h*.

If a number of signals higher than the minimum value is used, the phase becomes IDLE TRAINED and the RESET KNOWLEDGE button is activated: by pressing it, the original hyperparameters of the library are restored, which restarts from the IDLE state.

Using the switch and start button, you can launch the learning or detecting phase:

- By moving the switch on learning side, the learning phase starts; during this phase, the vibration data are generated and sent to the NEAI library to train the ML model.

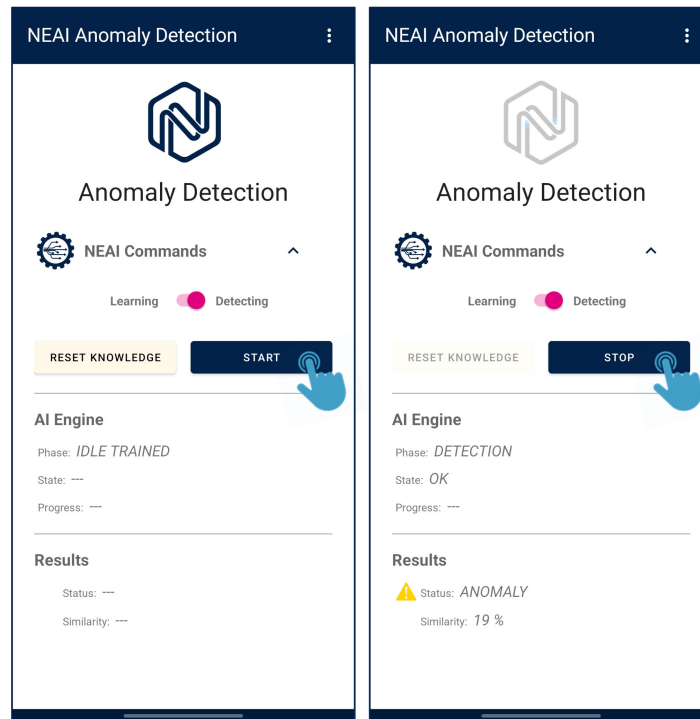
Figure 73. Learning on the NEAI demo



- By moving the switch on detecting side, the detecting phase starts; during this phase, the vibration data are generated and sent to the NEAI library to recognize if the asset behavior is normal or abnormal by an indicator that is named *similarity*, this last assumes (value from 0 to 100%).

If the similarity is greater than the set threshold value, an anomaly occurred. You can also set the threshold value by demo settings.

Figure 74. Detecting on the NEAI demo



In both the learning and detecting phase, the start button is replaced by the stop button to end the processing phase.

Note: The learning phase can be automatically stopped via timer or signal settings (see [Section 5.1.5.1: NEAI anomaly detection demo settings](#)) but, by default, both are initialized to zero. Therefore you need to press the stop button to end the learning.

Note: You cannot start both phases at the same time.

The NEAI demo consists of two sections: *AI Engine* and *Signal*. These sections provide several info related to the NEAI library status and the detected signal.

Within *AI Engine* you can find:

- the active phase of the NanoEdgeAI library:
 - IDLE**, if the library is not running.
 - LEARNING**, if the library is learning from the raw data generated by a sensor to train the ML model.
 - DETECTING**, if the library is monitoring the raw data generated by a sensor to detect anomaly behaviors.
 - IDLE TRAINED**, if the library is not running but a learning phase has been done and the minimum signal quantity has been learned.
- The NEAI state, which describes the library operation:
 - NOT ENOUGH LEARNING**, when you launch a learning or a detecting phase without having learnt the minimum quantity of signals.
 - MINIMAL LEARNING DONE**, when you have launched a learning phase and you have achieved the minimum quantity of signals to learn.
 - OK**, when you have launched a detecting phase and the library works properly.
- The progress bar, which is active only if you have set the learning time or signals to learn; in this case, the progress bar is filled according to the time or signal settings.

The *Signal* section is active during the detecting phase. Within *Signal* you can find:

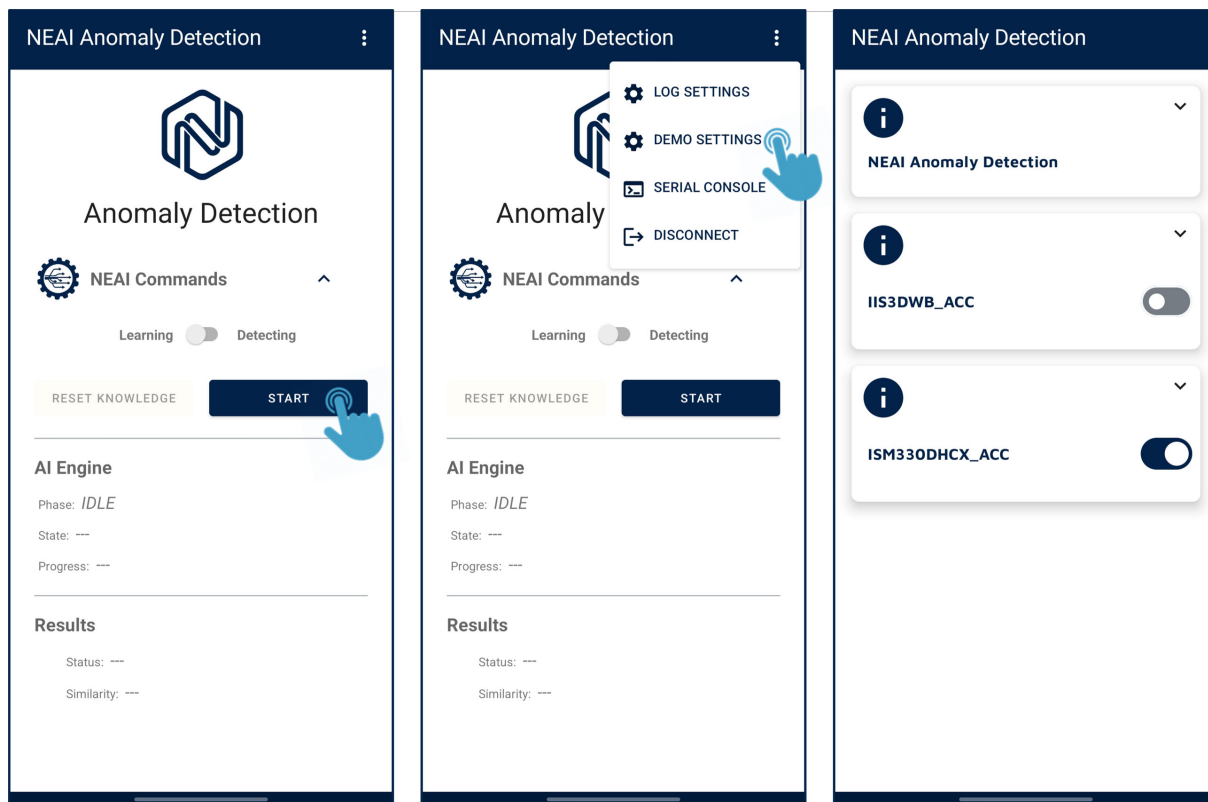
- The status of your asset, which can be normal or anomaly according to the similarity indicator provided by the NEAI library.
- The similarity indicator, which can assume values in the range of 0-100%; if this indicator is less than the threshold value, the asset status is anomaly, otherwise the status is normal.

5.1.5.1

NEAI anomaly detection demo settings

Using demo setting button, user can access to several setting option where library parameters and sensors configuration can be changed according to user needs.

Figure 75. NEAI anomaly detection demo settings



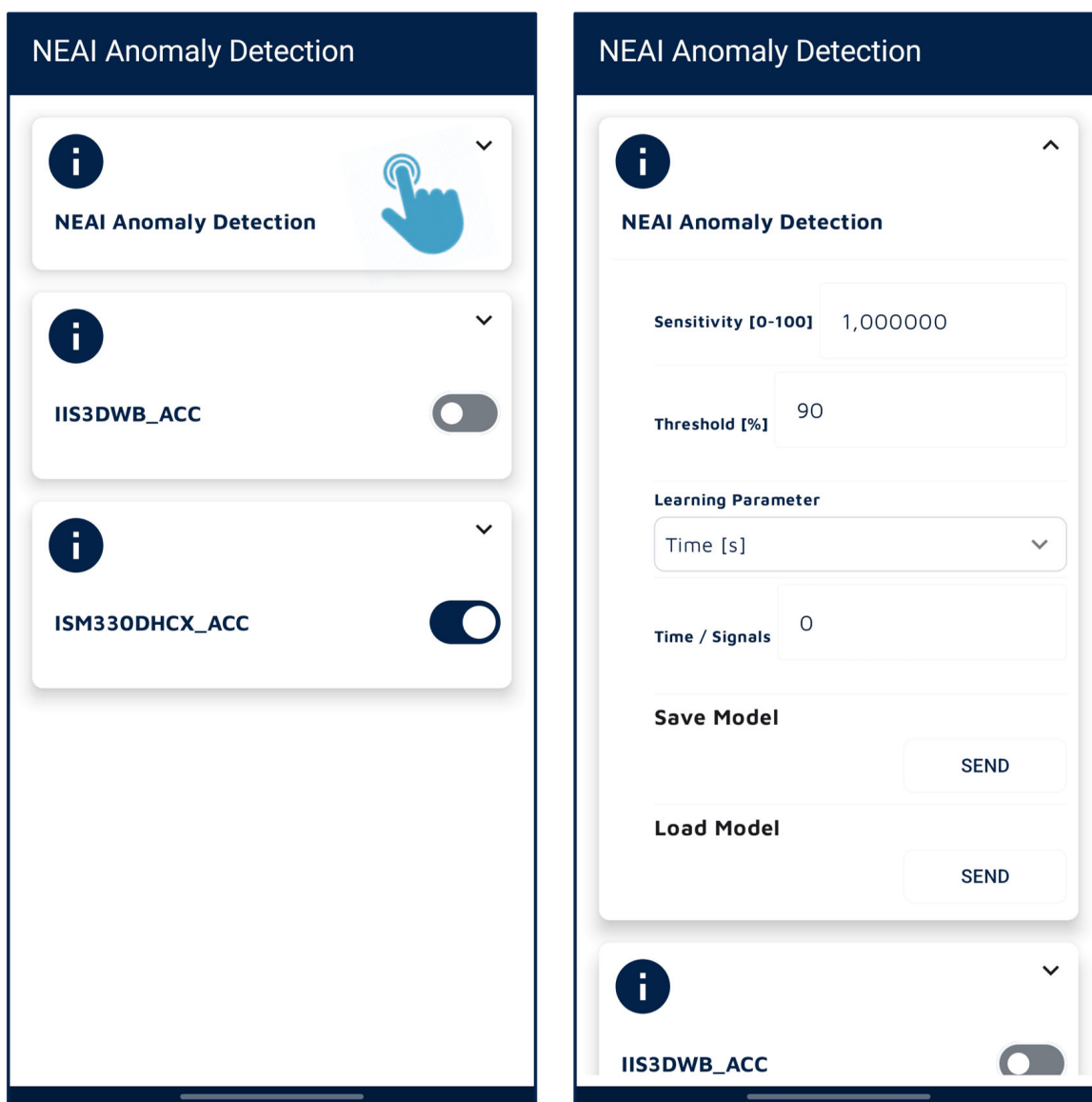
Anomaly detection section allow to view the current values and set the following parameters:

- The sensitivity of the ML model. It can be in the range of 0 to 100. A sensitivity value between 0 and 1 (excluded) decreases the sensitivity of the model, while a value between 1 and 100 increases it.
- The similarity threshold value under which a signal is considered normal or abnormal.
- The learning parameter, it can be equals to *Time* or *Signals*, if *Time* has been selected, after the specified number of seconds the phase will be automatically stopped, else *Signals* has been selected, when the specified number of signals have been learned the phase will be automatically stopped.
- Time/Signals field, where user can specify the leaning duration, if *Time* has been chosen, or number of signals, if *Signals* has been chosen.

Furthermore, there are two buttons which allow to save and load in or from the NOR external memory.

Note: *It is not allowed to set the Time and Signals as learning parameter for the same learning session.*

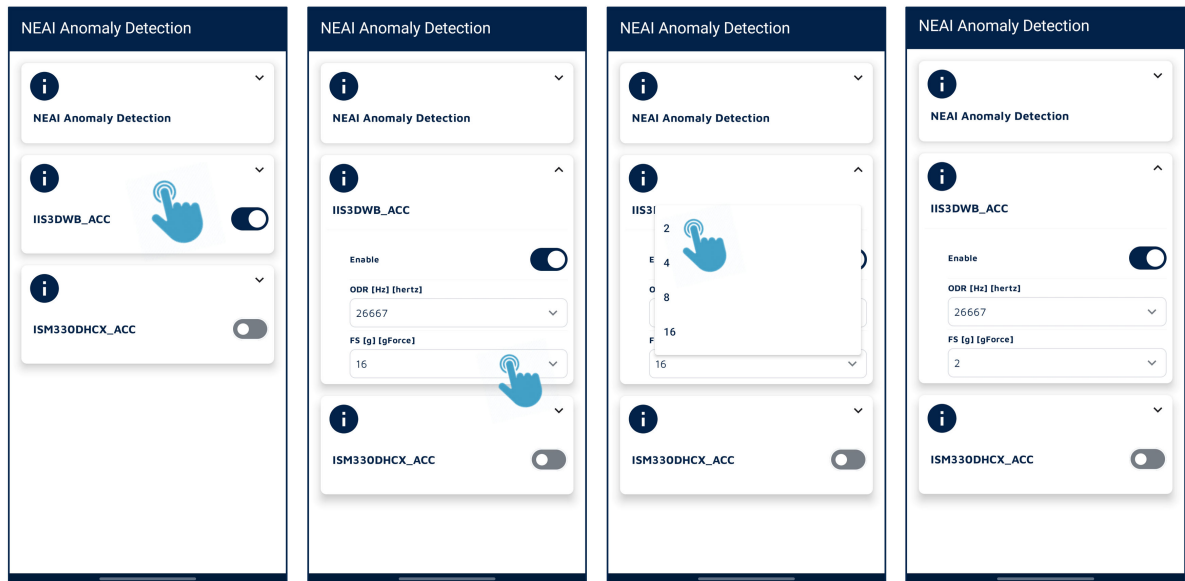
Figure 76. NEAI anomaly detection demo settings – Library parameters



Sensors sections allow to enable or disable, set the nominal output data rate and the full scale for every available sensor (IIS3DWB_ACC and ISM330DHCX_ACC).

Note: Only one sensor at a time can be enabled: the best sensor setup to use is the one chosen during the data acquisition phase.

Figure 77. NEAI anomaly detection demo settings – Sensors parameters

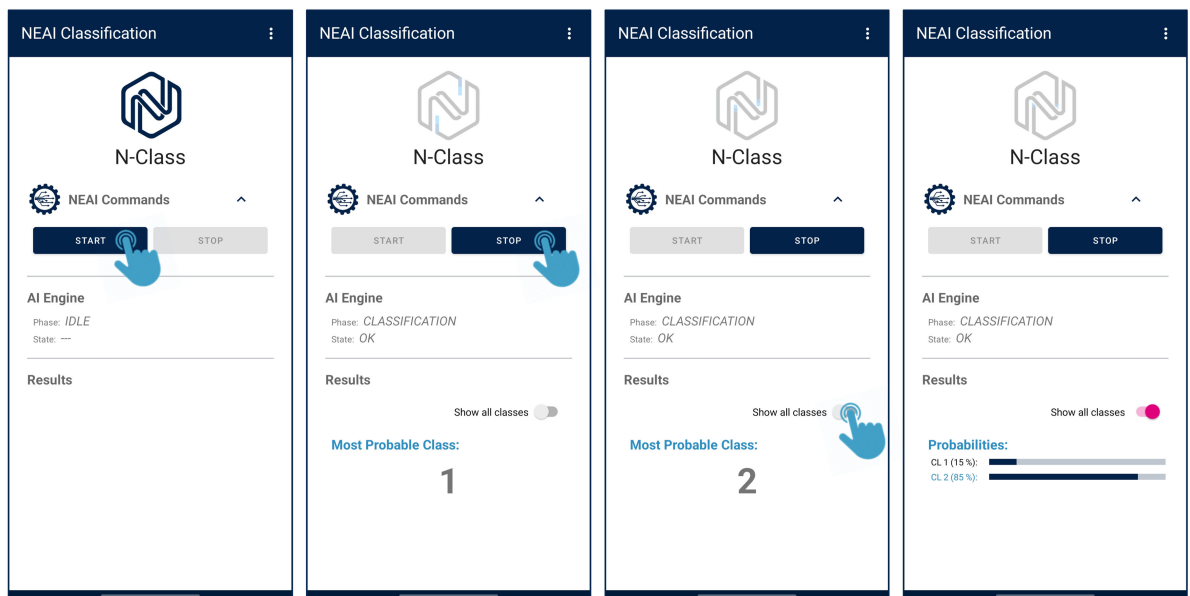


5.1.6 NEAI classification demo

As soon as the STEVAL-PROTEUS has been connected, user can open the *NEAI Classification* page selecting it from the menu.

The STEVAL-PROTEUS remains in the idle state until it gets a command to start the classification phase.

Figure 78. Classifying on the NEAI demo



Using the start button, you can launch the classification phase, the vibration data are generated and sent to the NEAI library to classify it.

The NEAI demo consists of two sections: AI Engine and Signal. These sections provide several info related to the NEAI library status and the classified signals.

Within AI Engine you can find:

- The active phase of the NanoEdgeAI library:
 - IDLE, if the library is not running.
 - CLASSIFICATION, if the library is classifying the raw data generated by a sensor.
- The NEAI state, which describes the library operation:
 - OK, when you have launched a classification phase and the library works properly.

The Signal section is active during the classification phase. Within Signal you can find:

- The *Most Probable Class*, namely the recognized class name, it is the class to which has been attributed the highest probability.
- Using the *Show all classes* switch, user can view the probabilities associated to every classes.

5.1.6.1

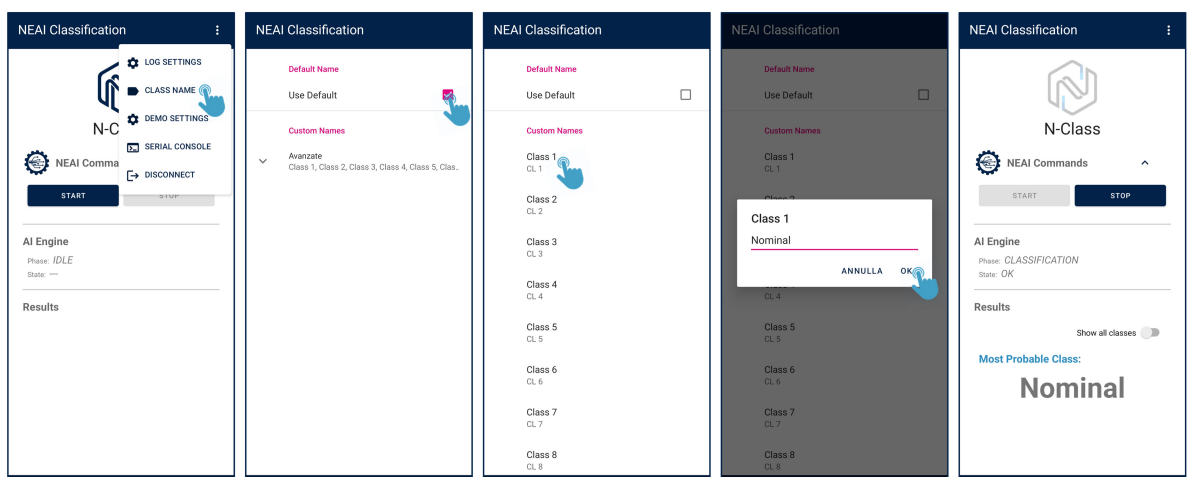
Classes names customization

Since classes names is strongly dependent from user use case, it is possible to change the default names through the *class name* setting.

To customize the classes names, user needs to follow the steps below:

- Open setting window and tap on *Class Name* option.
- Uncheck *Use Default* flag.
- Tap on the classes which you want to name and enter the new names.
- Return to the main page and start classification with your customized classes names

Figure 79. Classes names customization



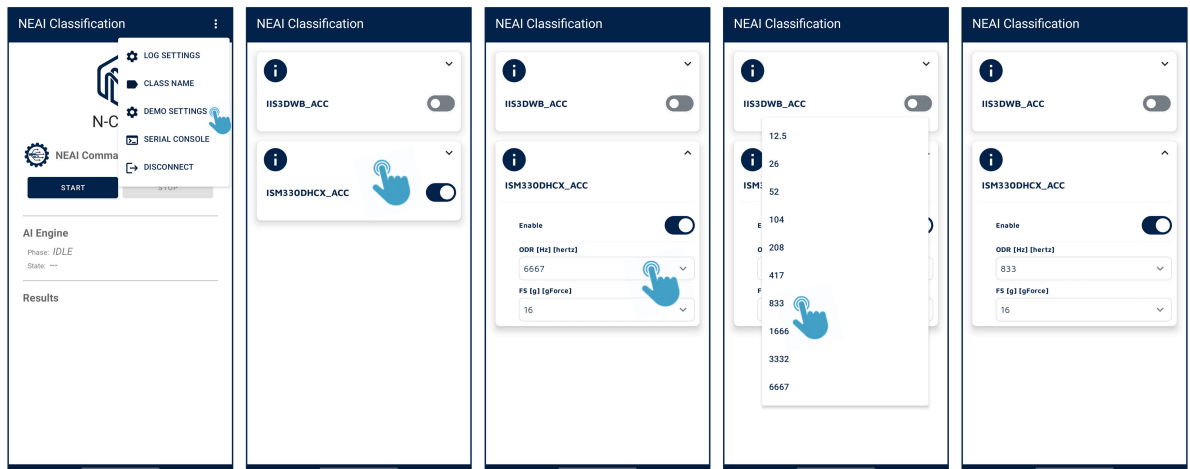
5.1.6.2

NEAI classification demo settings

Using demo setting button, user can access to several setting option where sensors configuration can be changed according to user needs. As explained in [Section 5.1.5.1](#), it is easy to enable or disable sensors and set the nominal output data rate and full scale.

Note: Only one sensor at a time can be enabled: the best sensor setup to use is the one chosen during the data acquisition phase.

Figure 80. NEAI classification demo settings – Sensors Parameters



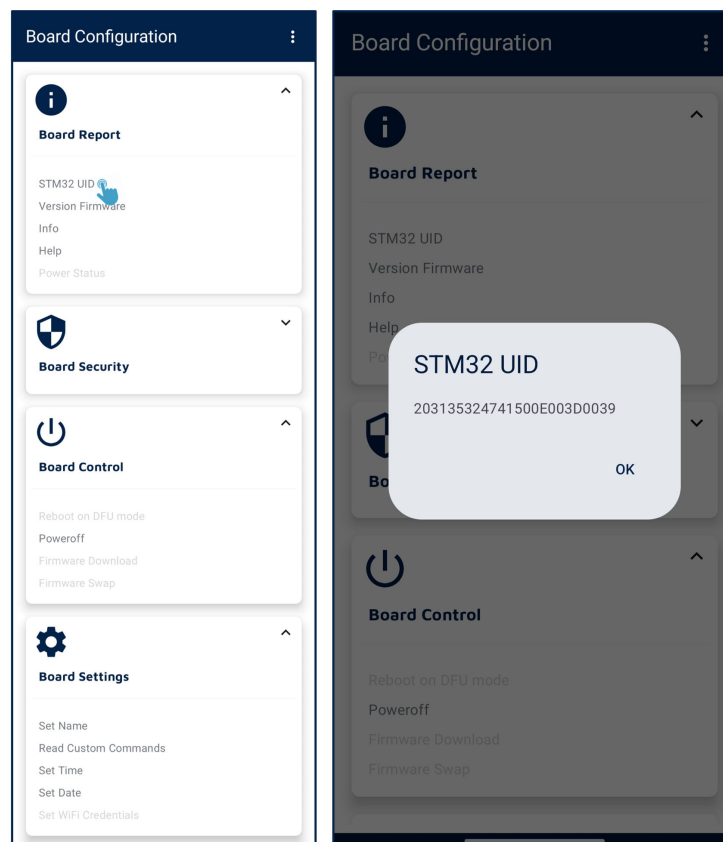
5.1.7 Board configuration

Touching *Board Configuration*, the mobile app shows information about the board.

In the *Board Report* section, you can retrieve the following information.

- **STM32 UID:** the unique 96bit ID of STM32WB mounted on the STEVAL-PROTEUS.

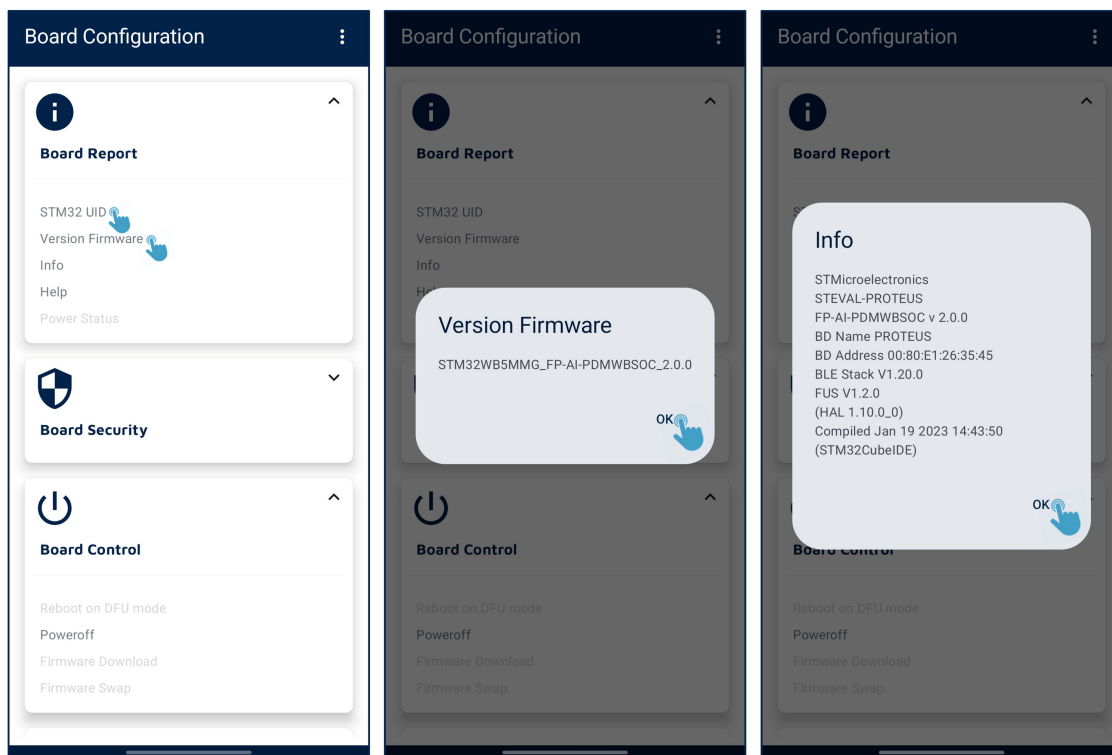
Figure 81. Board configuration summary and STM32 unique ID



- the **Version firmware:** the firmware release loaded on the STEVAL-PROTEUS.

- **Info:** a set of more detailed information about the firmware loaded into the STEVAL-PROTEUS.

Figure 82. Firmware version and info

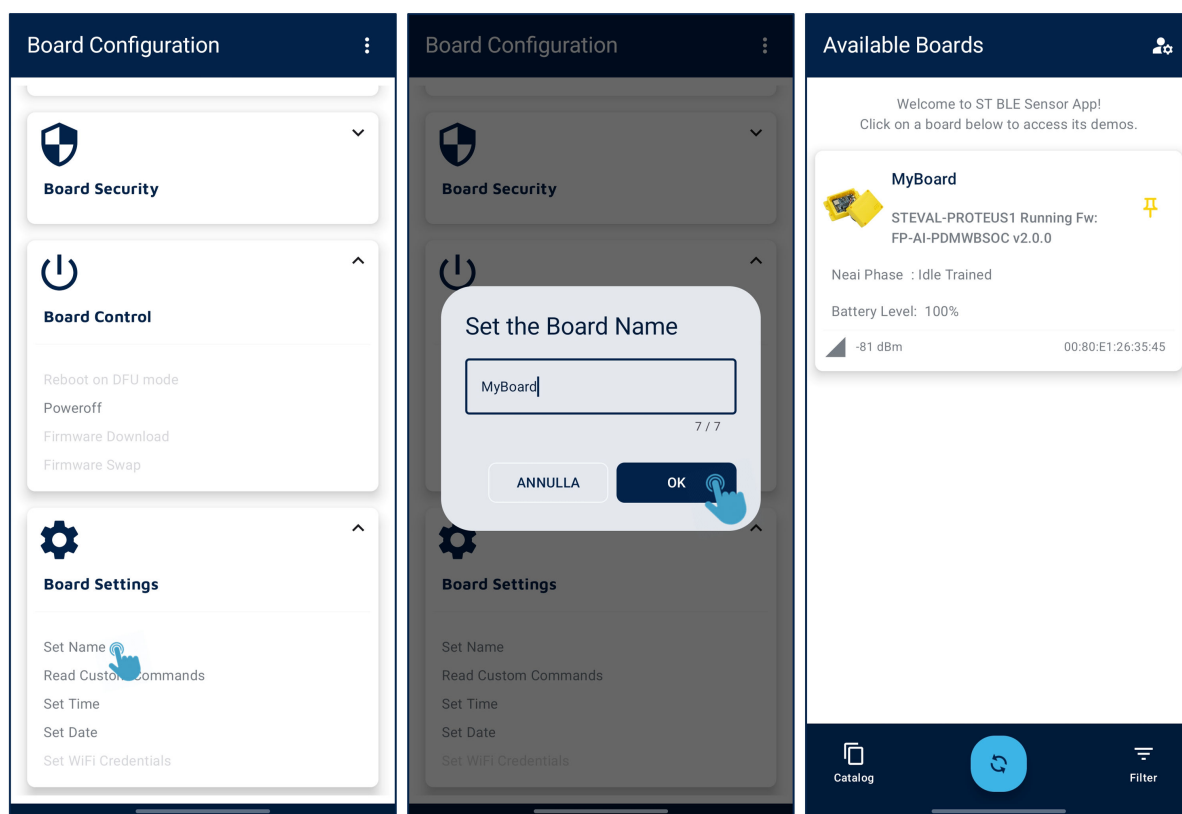


In the *Board Control* section, you can power off the board.

In the *Board Settings* section, you can handle the following information:

- **Set name:** to change the board connected name, keeping it internally, to show it during Bluetooth® Low Energy advertising.

Figure 83. Board name settings



- **Read custom commands:** to open a customized panel containing the commands setting directly built by the firmware, as described in the next section.

5.1.7.1

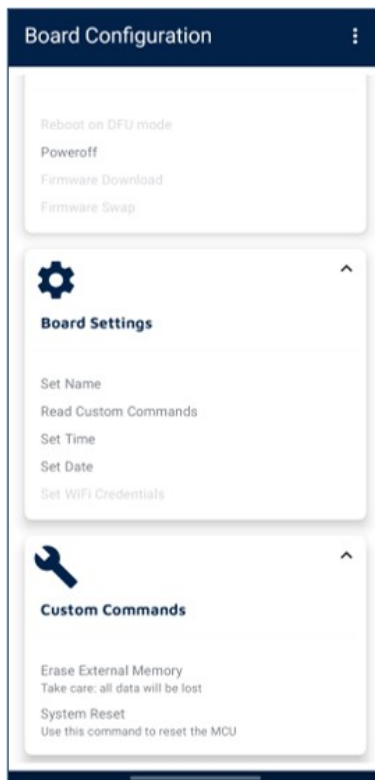
Custom commands

Tap on *Read Custom Commands* button to open the *Custom Commands* section.

This customized menu provides commands to:

- Erase external NOR flash memory content
- Reset the MCU

Figure 84. Read custom commands

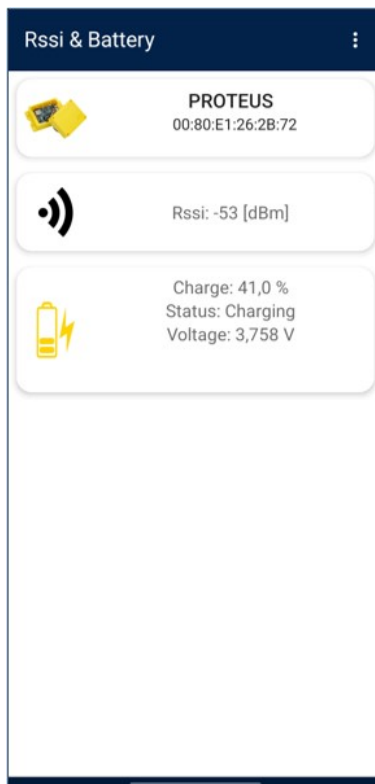


5.1.8 RSSI & Battery

Touching *RSSI & Battery* in the *STBLESensor* menu, the user gets information about:

- the board name and MAC address
- the RSSI
- the battery information such as the level and status of the charge and voltage

Figure 85. RSSI and battery status



5.1.9 Use FUOTA to test different libraries

You can easily switch between different application binaries that contain libraries with different performance in terms of score, accuracy, RAM, or flash memory, or generated for different use cases.

The best practice could be to:

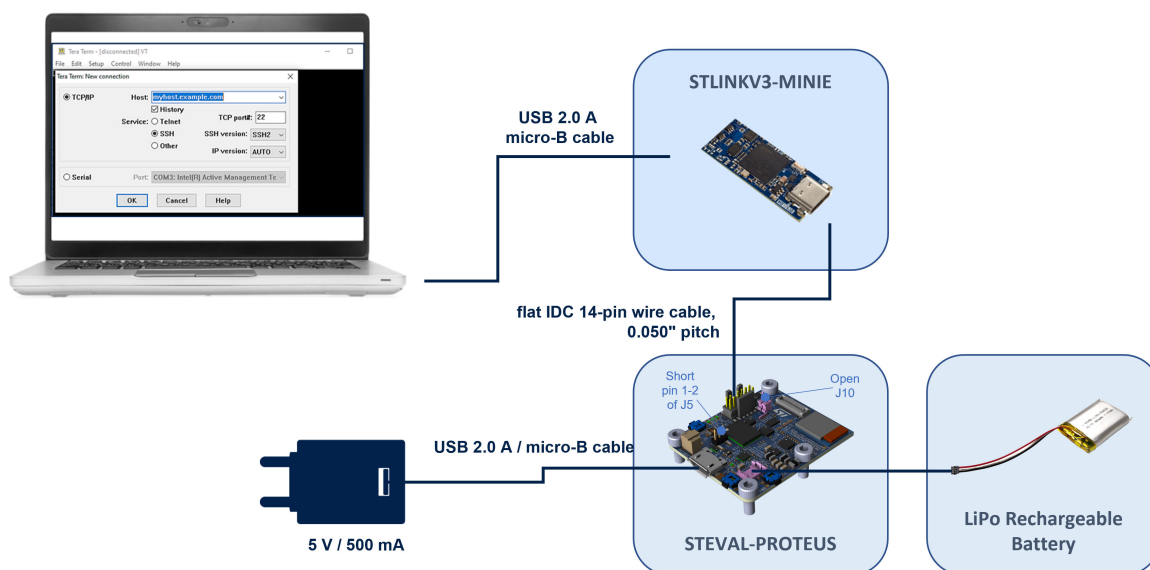
- Generate several NanoEdge AI libraries for different anomaly detection or/and classification use cases
- Embed these compiled binaries into the [FP-AI-PDMWBSOC](#), creating several application binaries
- Use the FUOTA to switch through the application, to cover and test different application use cases

5.1.10 Debug using a terminal emulator

The [STLINK-V3MINIE](#) also provides a virtual COM port interface, which allows the host PC to communicate with the target microcontroller through a UART. It also allows the STEVAL-PROTEUS that runs the [FP-AI-PDMWBSOC](#) to send information to a terminal console when connected to a PC.

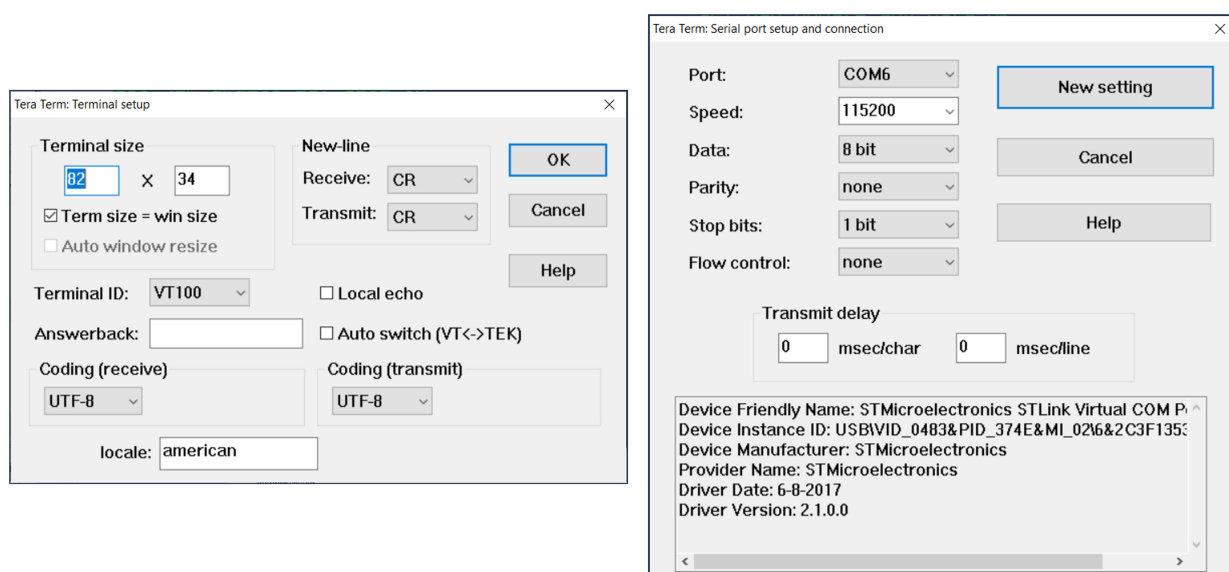
The figure below shows how to connect the STEVAL-PROTEUS to a PC able to run a terminal emulator such as Tera Term.

Figure 86. STEVAL-PROTEUS connection with a terminal emulator



Configure the terminal emulator as shown below.

Figure 87. PC terminal setting for the debug console



This debug terminal allows you to monitor:

- the system initialization: to check if all the application tasks and application services are properly initialized and started
- the Bluetooth® Low Energy initialization: to check if all the Bluetooth® Low Energy services and characteristics are properly initialized; it also shows the information about the Bluetooth® Low Energy stack and the FUS version, the Bluetooth® Low Energy address, etc.
- the application state machine: to check if state transactions occurred according to the state machine design

Note: You can find more details about application state machine in `\Documentation\FP-AI-PDMWBSOC_Package.chm`.

The following codeblock shows a possible output on the debug terminal.

```
System Initialization
INIT: reset flags: 0x4000000
SysTS: System timestamp service ready.
INIT: added 7 managed tasks.
INIT: task hardware initialization.
...
DONE.
NMP: initialization done.
BCP: initialization done.
SPIBUS: connected device: 1
SPIBUS: connected device: 2
INIT: system initialized.
INIT: free heap = 76416.
INIT: SystemCoreClock = 64000000Hz.
IIS3DWB: start.
ISM330DHCX: start.
SPI: start.
SPIBUS: start the driver.
CTRL: start.
NMP: load NEAI AD data from NOR Flash memory ... no stored data were found!
NEAI: start.
DPT1:0 DPU_MESSAGE_ID_ADD_LISTENER
UTIL: start.
BLE: start.

SHCI_SUB_EVT_CODE_READY - WIRELESS_FW_RUNNING

*****
STMicroelectronics
STM32STM32WB5MMG - Anomaly Detection and Classification by NEAI
FP-AI-PDMWBSOC V2.0.0 Compiled Jan 19 2023 14:43:50 (STM32CubeIDE)
MCU Unique device ID is 0x203135324741500E0022004B
MCU Flash Size is 1024 KB

BD Address 00:80:E1:26:2B:72
BLE Stack V1.13.0
BLE Stack Branch 0 Type 2
FUS V1.2.0
*****

BlueST-SDK V2
Config Service added successfully
Console Service added successfully
BLE PnPLike features ok
BLE NEAI Anomaly Detection char is ok
BLE NEAI Classification char is ok
BLE Battery features ok
BLE Ota features ok
Features Service added successfully (Status= 0x0)
aci_gap_set_discoverable OK Filter=0
aci_gap_update_adv_data OK
```

5.2 Using the STEVAL-PROTEUS with the command line interface

The CLI allows you to control the application by sending the same commands, described in the [STBLESensor](#) app, via command-line input to be processed on the device.

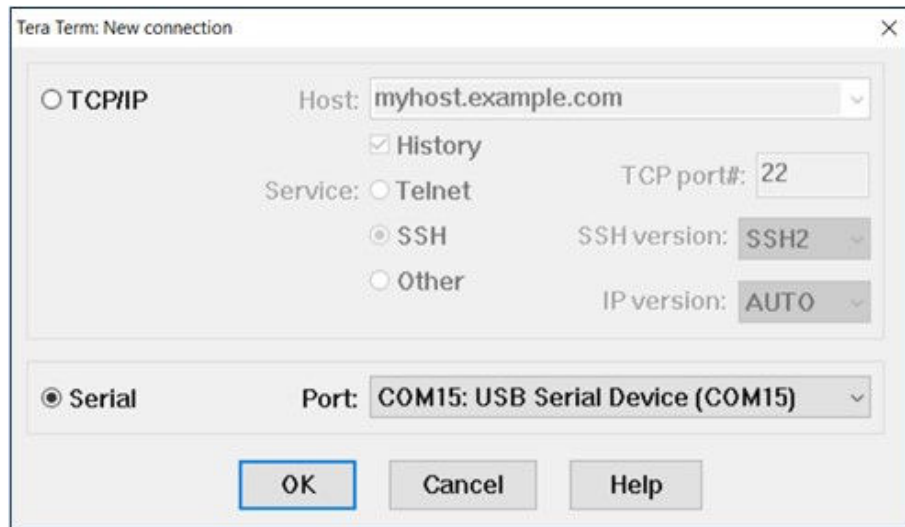
To use this interface, you need:

- a laptop with Tera Term software
- a micro-B to A USB cable

5.2.1 Set the serial terminal configuration

Start Tera Term and select the proper connection, that is the serial, which indicates the right port connected (USB serial device, COM15 in the example).

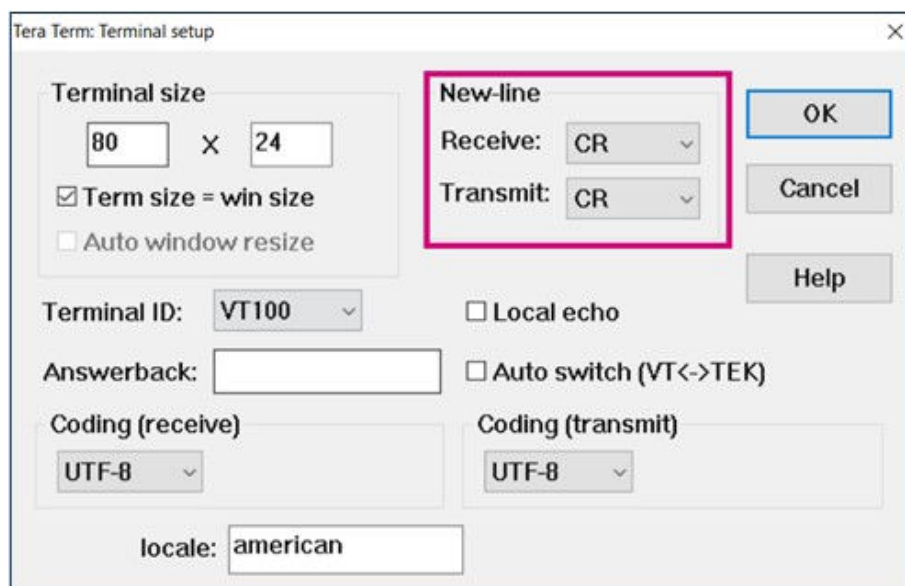
Figure 88. Open the CLI via PC terminal console



Use the following parameters for the terminal setup:

- New line
- Receive: CR
- Transmit: CR

Figure 89. Tera Term connection settings

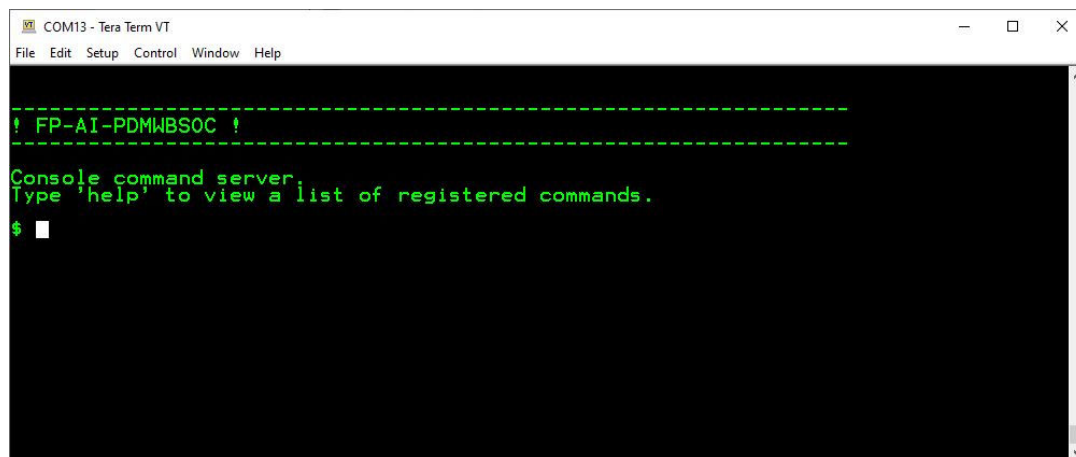


The interactive console can be used with the default values.

5.2.2 Start the FP-AI-PDMWBSOC firmware

Restart the board by pressing the reset button. The following welcome screen is displayed on the terminal.

Figure 90. Start the CLI program execution



From this point, start entering the commands directly or type *help* to get the list of the available commands along with their usage guidelines.

5.2.3 Execution phases and command summary

The two system execution phases are:

- **NanoEdge AI learning:** sensor data are transmitted to the NanoEdge AI library to train the model.
- **NanoEdge AI detecting:** sensor data are transmitted to the NanoEdge AI library to detect anomalies.
- **NanoEdge AI classifying:** sensor data are transmitted to the NanoEdge AI library to classify asset behaviours.

Each execution phase can be started and stopped through a user command, `$ start <execution phase>`, issued through the CLI, where the valid values for the `execution phase` are:

- **neai_learn**
- **neai_detect**
- **neai_class**

An execution context, which is a set of parameters that control the execution, is associated with each execution phase. One single parameter can belong to more than one execution context. The CLI provides commands to set and get the execution context parameters. The execution context cannot be changed while an execution phase is active.

If you try to set a parameter that belongs to any active execution context, the requested parameter is not modified.

Important: Do not plug/unplug the USB connector during the learning/detecting phase.

Table 5. CLI commands list

Command name	Command string	Note
CS1 - Generic commands		
help	help	Lists all registered commands with brief usage guidelines, including the list of the applicable parameters
info	info	Shows the firmware details and version
uid	uid	Shows the STM32 UID
reset	reset	Resets the MCU system
CS2 - AI specific commands		
start	start <phase>	Starts an execution phase according to its execution context. The available execution phases are: <ul style="list-style-type: none"> • neai_learn to learn the normal conditions using the NanoEdge AI library for anomaly detection • neai_detect to start the detection of the anomalies using the NanoEdge AI library for anomaly detection • neai_class to start the classification of asset behaviours using the NanoEdge AI library for N-Class classificatio
neai_init	neai_init	(Re)initializes the NanoEdge AI model and forgets any learning made on the device. Used at the beginning or to reinitialize the NanoEdge AI anomaly detection to start from scratch at any time
neai_set	neai_set <parameter> <value>	Sets a NanoEdge AI specific parameter. The valid parameters and values are: <ul style="list-style-type: none"> • sensitivity: float [0, 100] • threshold: integer [0, 100] • signals: integer [0, MAX_SIGNALS] • timer: integer [0, MAX_TIME_MS] • sensor: integer [0, 1]
neai_get	neai_get <parameter>	Displays the value of the NanoEdge AI parameter. The valid parameters are <ul style="list-style-type: none"> • sensitivity • threshold • signals • timer • sensor • all
CS3 - Sensor configuration commands		
sensor_info	sensor_info	Gets a list of all supported sensors and their ID
sensor_set	sensor_set <id> <parameter> <value>	Sets the value of a parameter for a sensor with the sensor ID provided in id . The settable parameters are: <ul style="list-style-type: none"> • FS: full scale • ODR: output data rate • enable: active [1 for true (active), 0 for false (disactive)]
sensor_get	sensor_get <id> <parameter>	Gets the value of a parameter for a sensor with the sensor ID provided in id . The valid parameters are: <ul style="list-style-type: none"> • enable: active [true (active), false (inactive)] • ODR: output data rate • ODR_List: list of supported ODRs • FS: full scale • FS_List: list of supported FSs • all: all sensor configurations

Note: The learning phase can be stopped automatically through a timer or signals setting. However, by default, both are initialized to zero. Therefore, you need to press the escape button to stop the learning.

5.2.4 Configuring the sensors

Through the CLI interface, you can configure the supported sensors for sensing and condition monitoring applications. The list of all the supported sensors can be displayed on the CLI console by entering the command `sensor_info`. This command prints the list of the supported sensors along with their IDs. The user can configure these sensors using these IDs. The configurable options for these sensors include:

- **enable**: to activate or deactivate the sensor
- **ODR**: to set the output data rate of the sensor from the list of available options
- **FS**: to set the full-scale range from the list of available options

Note: The best sensor setup is the one used during the data acquisition phase.

The current value of any of the parameters for a given sensor can be printed using the `$ sensor_get <sensor_id> <param>` command.

The information about the sensor can be printed using the `$ sensor_get <sensor_id> all` command.

The values for any of the available configurable parameters can be set through the `$ sensor_set <sensor_id> <param> <val>` command.

5.3 Using the STEVAL-PROTEUS with hardware buttons and LEDs

5.3.1 Button-operated mode

The button-operated mode for the FP-AI-PDMWBSOC allows running the application on the STEVAL-PROTEUS even in the absence of the CLI console and Bluetooth® Low Energy connection.

In this mode, you can control the sensor node through the user button instead of the interactive CLI console or STBLESensor. The default values for the node parameters and settings for the operations during the auto mode are provided in the firmware. Based on these configurations the learning and detecting phases can be started and stopped through the user button on the node.

Note: The auto-mode setup provides the IIS3DWB active with the last FS selected (by default, it is 16 G).

5.3.2 Buttons description

The button-operated mode can work with or without the CLI and STBLESensor. It is fully compatible and consistent with the others UI.

The supporting hardware for this function pack version has three buttons:

1. the user button, the only button usable by the software
2. the reset button, connected to the STM32 MCU reset pin
3. the power button, connected to the power management

There are also four LEDs:

1. the red LED, controlled by software
2. the green LED, controlled by software
3. the blue LED, controlled by software
4. the yellow LED, controlled by hardware, indicates the charging status when powered through a USB cable

Therefore, the basic user interactions for button-operated operations can be done through one button (user) and three LEDs (red, green, and blue).

5.3.3 User button operation and purpose

In the extended autonomous mode, you can trigger any of the four execution phases. The available modes are:

- *idle*: the system is waiting for a command (the library might be already trained or not)
- *neai_learn*: sensor data are transmitted to the NanoEdge AI library to train the model
- *neai_detect*: sensor data are transmitted to the NanoEdge AI library to detect anomalies
- *neai_class*: sensor data are transmitted to the NanoEdge AI library to classify asset behaviours

To control the execution phases, there are three different button press modes for the user button.

Table 6. User button operation

Press type	Description	Action
Short press	The button is pressed for less than 200 ms and released	Stops the running execution phase if any, otherwise it starts the N-Class Classification library execution
Long press	The button is pressed for more than 200 ms and released	Starts the learning phase of Anomaly Detection library
Double press	A succession of two short presses	Starts the detecting phase of Anomaly Detection library

5.3.4

LEDs operation and purpose

The onboard LEDs indicate the status of the current execution phase, assuming a specific configuration.

Table 7. Red and green LEDs for NEAI phases

NEAI lib phase	Red LED	Green LED
Idle	Off	Off
Idle trained	Off	Off
Learning	Off	Blink
Detecting	On, if an anomaly occurs	On, if no anomaly occurs
Classifying	Off	Off

Table 8. Blue LED for Bluetooth® Low Energy connectivity

Bluetooth® Low Energy connection to a smartphone	Blue LED
Connected	Slow blink
Not connected	Fast blink

The LED status allows you to know the state of the sensor node even when the USB is not plugged and/or the smartphone is not connected to the board.

Revision history

Table 9. Document revision history

Date	Revision	Changes
20-Sep-2022	1	Initial release.
09-Sep-2024	2	<p>Updated Section Introduction, Section 1.2: Architecture, Section 1.3: Folders, Section 2: Application scenario, Table 1. Hardware platform details, Section 3.2: Hardware setup, Section 3.3: Software setup, Section 3.3.3.1: Application firmware update, Section 3.3.3.2: Wireless stack firmware update, Figure 15. STEVAL-PROTEUS flash programming scenario, Section 4: Making and embedding the NEAI libraries, Section 4.1.1: Uploading HS data log application firmware, Section 4.2: Library generation using NanoEdgeAI Studio, Section 5.1.3: Bluetooth® Low Energy advertising, Section 5.1.4: Connect a Bluetooth® Low Energy device and choose the application demo, Section 5.1.7: Board configuration, Section 5.2.3: Execution phases and command summary, Section 5.3.3: User button operation and purpose and Section 5.3.4: LEDs operation and purpose.</p> <p>Added Section 4.2.1: NEAI anomaly detection library , Section 4.1.2.6: Convert the dataset in NanoEdgeAI format, Section 4.2.2: NEAI N-Class Classification library, Section 4.3: NEAI library integration, Section 5.1.5.1: NEAI anomaly detection demo settings and Section 5.1.6: NEAI classification demo .</p>

Contents

1	FP-AI-PDMWBSOC software expansion for STM32Cube	2
1.1	Overview	2
1.2	Architecture	2
1.3	Folders	3
1.4	APIs	4
2	Application scenario	5
3	System setup guide	7
3.1	STEVAL-PROTEUS1 industrial sensor evaluation kit	7
3.2	Hardware setup	10
3.3	Software setup	11
3.3.1	STEVAL-PROTEUS flash memory mapping	11
3.3.2	Toolchains settings for STM32WB memory mapping	12
3.3.3	STEVAL-PROTEUS flash programming via Bluetooth® Low Energy OTA	12
3.3.4	STEVAL-PROTEUS flash memory programming via ST-LINK	17
4	Making and embedding the NEAI libraries	22
4.1	Data acquisition by data log	22
4.1.1	Uploading HS data log application firmware	22
4.1.2	Creating HS data log by CLI	24
4.2	Library generation using NanoEdgeAI Studio	38
4.2.1	NEAI anomaly detection library	38
4.2.2	NEAI N-Class Classification library	47
4.3	NEAI library integration	56
4.3.1	NEAI AD library integration	56
4.3.2	NEAI N-Class library integration	57
5	Running FP-AI-PDMWBSOC	59
5.1	Using the STEVAL-PROTEUS with the STBLESensor app	59
5.1.1	Bluetooth® Low Energy connectivity features	59
5.1.2	Bluetooth® Low Energy workflow overview	59
5.1.3	Bluetooth® Low Energy advertising	59
5.1.4	Connect a Bluetooth® Low Energy device and choose the application demo	61
5.1.5	NEAI anomaly detection demo	61
5.1.6	NEAI classification demo	67
5.1.7	Board configuration	69
5.1.8	RSSI & Battery	72
5.1.9	Use FUOTA to test different libraries	73

5.1.10	Debug using a terminal emulator.	73
5.2	Using the STEVAL-PROTEUS with the command line interface.	75
5.2.1	Set the serial terminal configuration	75
5.2.2	Start the FP-AI-PDMWBSOC firmware	77
5.2.3	Execution phases and command summary.	77
5.2.4	Configuring the sensors	79
5.3	Using the STEVAL-PROTEUS with hardware buttons and LEDs.	79
5.3.1	Button-operated mode	79
5.3.2	Buttons description	79
5.3.3	User button operation and purpose.	79
5.3.4	LEDs operation and purpose.	80
Revision history		81
List of tables		84
List of figures.		85

List of tables

Table 1.	Hardware platform details	7
Table 2.	STM32WB Application Binaries to use	19
Table 3.	STM32WB coprocessor wireless binaries to use	21
Table 4.	Asset status icons	60
Table 5.	CLI commands list	78
Table 6.	User button operation	80
Table 7.	Red and green LEDs for NEAI phases	80
Table 8.	Blue LED for Bluetooth® Low Energy connectivity	80
Table 9.	Document revision history	81

List of figures

Figure 1.	FP-AI-PDMWBSOC software architecture	2
Figure 2.	FP-AI-PDMWBSOC package folder structure	3
Figure 3.	FP-AI-PDMWBSOC application overview	5
Figure 4.	FP-AI-PDMWBSOC working flow	6
Figure 5.	STEVAL-PROTEUS1 evaluation kit	7
Figure 6.	STEVAL-PROTEUS main devices	8
Figure 7.	STEVAL-PROTEUS power connector, SWD, user button, and LED	9
Figure 8.	STEVAL-PROTEUS mounting and power-on	10
Figure 9.	QR codes for STBLESensor	11
Figure 10.	STEVAL-PROTEUS memory map for Bluetooth® Low Energy OTA applications	12
Figure 11.	Start FUOTA and select FP-AI-PDMWBSOC binary application	13
Figure 12.	FUOTA progress and completion	14
Figure 13.	Start FUOTA and select wireless stack binary	16
Figure 14.	Searching the WBFUOTA in the device list	17
Figure 15.	STEVAL-PROTEUS flash programming scenario	18
Figure 16.	Connecting the board	18
Figure 17.	Erasing & Programming selection	19
Figure 18.	Application firmware installation overview	20
Figure 19.	Firmware upgrade services	20
Figure 20.	Firmware upgrade overview	21
Figure 21.	Dataset acquisition, NEAI library creation, deployment, and integration	22
Figure 22.	DTLG100 discovered by STBLESensor app	23
Figure 23.	DTLG100 Bluetooth® Low Energy available demos for STBLESensor app	23
Figure 24.	DTLG100 info showed by STBLESensor app	24
Figure 25.	STEVAL-PROTEUS connected to a PC	24
Figure 26.	High-speed data log utility folder	25
Figure 27.	USB HS-Datalog CLI ready to start	26
Figure 28.	USB HS-Datalog CLI acquisition running	26
Figure 29.	USB HS-Datalog CLI acquisition completed	27
Figure 30.	DeviceConfig.json - device attributes	27
Figure 31.	DeviceConfig.json - device info	28
Figure 32.	DeviceConfig.json - sensor	29
Figure 33.	DeviceConfig.json - sensor descriptor	30
Figure 34.	DeviceConfig.json - sensor status	31
Figure 35.	DeviceConfig.json - tag config	32
Figure 36.	AcquisitionInfo.json	33
Figure 37.	<i>filename.dat</i> file - data stream structure	33
Figure 38.	HS-DL_Plot.bat shell	35
Figure 39.	IIS3DWB data plot example	36
Figure 40.	NanoEdgeAI dataset format	36
Figure 41.	Datasets converter batch example	37
Figure 42.	Datasets converter batch example	37
Figure 43.	Start a project (AD)	38
Figure 44.	Six-step workflow	38
Figure 45.	Project settings (AD)	39
Figure 46.	Classified data import	40
Figure 47.	Start benchmark (AD)	41
Figure 48.	Benchmark AD execution (1 of 2)	41
Figure 49.	Benchmark AD execution (2 of 2)	42
Figure 50.	Benchmark results and models summary (AD)	43
Figure 51.	Models validation (AD)	44
Figure 52.	Validation results (AD)	45
Figure 53.	Deployment setting (AD)	46

Figure 54.	Library compilation and ZIP file saving (AD)	47
Figure 55.	Start an N-Class Classification project	47
Figure 56.	Five-step workflow	48
Figure 57.	Project settings (NCC)	48
Figure 58.	NCC classified data import	49
Figure 59.	Start benchmark (NCC)	50
Figure 60.	Benchmark NCC execution (1 of 2)	50
Figure 61.	Benchmark NCC execution (2 of 2)	51
Figure 62.	Benchmark results and models summary (NCC)	52
Figure 63.	Models validation (NCC)	53
Figure 64.	Validation results (NCC)	54
Figure 65.	Deployment setting (NCC)	55
Figure 66.	Library compilation and ZIP file saving (NCC)	56
Figure 67.	NEAI AD lib update in firmware architecture	57
Figure 68.	NEAI NCC lib update in firmware architecture	58
Figure 69.	Main page of the STBLESensor app for Android	59
Figure 70.	Device list page (for Android STBLESensor)	60
Figure 71.	Device connection and sensor demo summary	61
Figure 72.	NEAI anomaly detection demo (IDLE and IDLE TRAINED)	62
Figure 73.	Learning on the NEAI demo	63
Figure 74.	Detecting on the NEAI demo	64
Figure 75.	NEAI anomaly detection demo settings	65
Figure 76.	NEAI anomaly detection demo settings – Library parameters	66
Figure 77.	NEAI anomaly detection demo settings – Sensors parameters	67
Figure 78.	Classifying on the NEAI demo	67
Figure 79.	Classes names customization.	68
Figure 80.	NEAI classification demo settings – Sensors Parameters	69
Figure 81.	Board configuration summary and STM32 unique ID	69
Figure 82.	Firmware version and info	70
Figure 83.	Board name settings	71
Figure 84.	Read custom commands	72
Figure 85.	RSSI and battery status	73
Figure 86.	STEVAL-PROTEUS connection with a terminal emulator	74
Figure 87.	PC terminal setting for the debug console	74
Figure 88.	Open the CLI via PC terminal console	76
Figure 89.	Tera Term connection settings	76
Figure 90.	Start the CLI program execution	77

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2024 STMicroelectronics – All rights reserved