computers

MDPI

Article

SENSIPLUS-LM: A Low-Cost EIS-Enabled Microchip Enhanced with an Open-Source Tiny Machine Learning Toolchain

Michele Vitelli [1,2], Gianni Cerro [3], Luca Gerevini [2], Gianfranco Miele [2], Andrea Ria [4] and Mario Molinara [2,*]

[1] Sensichips s.r.l., 04011 Aprilia, Italy
[2] Department of Electrical and Information Engineering "Maurizio Scarano", University of Cassino and Southern Lazio, 03043 Cassino, Italy
[3] Department of Medicine and Health Sciences "Vincenzo Tiberio", University of Molise, 86100 Campobasso, Italy
[4] Department of Information Engineering, University of Pisa, 56100 Pisa, Italy
* Correspondence: m.molinara@unicas.it

Abstract: The technological step towards sensors' miniaturization, low-cost platforms, and evolved communication paradigms is rapidly moving the monitoring and computation tasks to the edge, causing the joint use of the Internet of Things (IoT) and machine learning (ML) to be massively employed. Edge devices are often composed of sensors and actuators, and their behavior depends on the relative rapid inference of specific conditions. Therefore, the computation and decision-making processes become obsolete and ineffective by communicating raw data and leaving them to a centralized system. This paper responds to this need by proposing an integrated architecture, able to host both the sensing part and the learning and classifying mechanisms, empowered by ML, directly on board and thus able to overcome some of the limitations presented by off-the-shelf solutions. The presented system is based on a proprietary platform named SENSIPLUS, a multi-sensor device especially devoted to performing electrical impedance spectroscopy (EIS) on a wide frequency interval. The measurement acquisition, data processing, and embedded classification techniques are supported by a system capable of generating and compiling code automatically, which uses a toolchain to run inference routines on the edge. As a case study, the system capabilities of such a platform in this work are exploited for water quality assessment. The joint system, composed of the measurement platform and the developed toolchain, is named SENSIPLUS-LM, standing for SENSIPLUS learning machine. The introduction of the toolchain empowers the SENSIPLUS platform moving the inference phase of the machine learning algorithm to the edge, thus limiting the needs of external computing platforms. The software part, i.e., the developed toolchain, is available for free download from GitLab, as reported in this paper.

Keywords: smart platform; IoT; tiny machine learning; pollution monitoring; smart city; EIS

check for updates

Citation: Vitelli, M.; Cerro, G.; Miele, G.; Gerevini, L.; Ria, A.; Molinara, M. SENSIPLUS-LM: A Low-Cost EIS-Enabled Microchip Enhanced with an Open-Source Tiny Machine Learning Toolchain. *Computers* **2023**, *12*, 23. https://doi.org/10.3390/computers12020023

Academic Editors: Dario Bruneo and Antonio Puliafito

Received: 15 December 2022
Revised: 17 January 2023
Accepted: 17 January 2023
Published: 19 January 2023

1. Introduction

The Internet of Things (IoT) revolution has a pervasive impact on today's society since small intelligent devices are currently adopted in various sectors, spanning from medicine to industry, exerting a positive effect on people's lives and pushing the shared application of technology and communication economy [1]. At the same time, gathering huge volumes of data without using them to facilitate people's life is useless. Machine learning (ML) power can deal with that by giving devices decision capabilities and making them suitable to learn from the surrounding environment. Often, ML procedures are designed as heavy computational routines fed by enormous quantities of data and run on powerful machines. In the case of pervasive IoT devices, such a paradigm would turn into transferring a considerable number of data to cloud infrastructures for processing. Such a process is slow, has a low level of promptness, and could cause decisions to be effective much later after an unwanted phenomenon has happened. A clear example is represented by water and air monitoring in the context of smart cities [2–4], where there are many sensors, many measurement points, and a considerable number of generated data. The idea to process them locally, take suitable decisions, and eventually transfer only refined information to cloud structure is advantageous from many points of view: reduced energy costs for communication, no need to securely cipher raw data before transmission, and allowing for immediate decisions on the location. This mechanism can be referred to as edge computing, which deserves much attention from the scientific literature [5–7]. Changing the processing to edge computing has also disadvantages: a limited amount of memory, low processing capabilities, and code customization on specific platforms, to cite a few. This work presents a multi-sensor integrated platform, the SENSIPLUS-LM, consisting of hardware and software components. The hardware is based on a commercial microcontroller unit (MCU) and the SENSIPLUS microChip (SPC), a proprietary system by Sensichips S.r.l. able to perform EIS and, in general, acts as a multi-parametric sensor. The software is a toolchain able to generate suitable firmware for the MCU containing the control library for the SPC and containing ML algorithm to move intelligence on the edge. The general capability of SPC was also presented in [8,9]. The toolchain can optimize and automatize all actions needed to have ready-to-use artificial intelligence-based firmware on the edge, customized for the SENSIPLUS and the selected MCU platform. Generally speaking, it should be noted that embedded devices are not suitable for implementing any machine learning model due to their limited computational resources. For example, it is difficult to implement on low-cost MCU deep neural networks or, more generally, algorithms that require a high quantity of memory or computing power. In the current version, different algorithms were included in the toolchain and were available for C code generation. It is important to highlight that the software structure of the toolchain allowed us to easily add new ML models simply by extending a Java class, as better explained in Section 3.1.2. The algorithms implemented in the current version were designed only for classification, but the toolchain could also include regression or anomaly detection algorithms as an extension [10].

The entire toolchain was tested on a monitoring application exploiting a simple multi-layer perceptron (MLP) network to demonstrate the proposed system's capability. The capability to classify several polluting substances in a harsh environment as wastewater was revealed.

This Work is organized as follows: Section 2 reports related works in the case of machine learning for IoT. Section 3 gives a brief overview of the adopted microchip and presents the overall description of the software architecture, particularly in terms of toolchain development as well as the classes proposed to create the software modules automatically. Section 4 presents the case study, i.e., the classification of water contaminants in a wastewater context. Section 5 concludes the paper while in the Appendix A some details about the Graphical User Interface are reported.

2. Related Works

The paradigm of artificial intelligence (AI) on the edge [11], often referred to as tiny machine learning (TinyML), allows researchers to solve many of the existing problems in cloud-based approaches, pointing to its valid implementation. It is a rapidly evolving sector, with many challenges to deal with [12]. Many software libraries are available for TinyML because the scientific and industrial communities extensively collaborate to make ML algorithms suitable to be run in low-cost and low-power devices [13,14]. TensorFlow Lite for microcontrollers was developed by Google [15]. TensorFlow Lite is organized into two main modules: the converter and the interpreter. The converter transforms the code by reducing the model's size and optimizing it. The interpreter runs the model on embedded devices [16]. These frameworks are used in different contexts, such as in the industrial field [17] and in the context of smart cities [18]. uTensor is a free integrated environment that helps the prototyping and rapid implementation of TinyML systems [19]. It consists of an inference engine, a data collection tool, and a graphical tool for data processing. NanoEdge AI Studio is a tool for selecting machine learning libraries: It allows one to test them before final implementation using an emulator [20]. Edge Impulse is a service for building cloud machine learning models for edge devices. Edge Impulse supports AutoML for edge platforms [21]. STM32Cube.AI is a code generation and optimization software platform that allows the use of machine learning and AI-related tasks for STM32 ARM Cortex M-based boards [22]. With the [23] toolkit, it is possible to efficiently run neural-network-based routines on low-cost hardware, such as the ones equipped with architectures such as ARM Cortex-M or RISC-V. In the literature, the possibility to deal with open-source frameworks, whose target is represented by IoT applications, is also highlighted [24]. These studies provide tools to analyze data from a high-level perspective and automatically generate codes targeted to specific hardware. There are also several examples of industrial products intended for AI on the edge, such as [25] or [26], by ST Microelectronics and Bosch Sensortec, respectively.

None of the tools presented in the previous section has adequate features to be integrated with the MCU control libraries created for SPC. In fact, the raw data acquired from SPC have to be pre-processed in a suitable way to be ready for ML, and the high level of generalization offered by other toolchains generates C code not customized and optimized for the selected combination of MCU and SPC.

At the same time, it has been demonstrated that the SENSIPLUS platform is suitable for identifying pollutants in water [27,28] or air [8,29] in an effective way. The open issue is that, in all these cases, the machine learning models have been deployed on external devices (PC, Workstation, or into the cloud).

This is the motivation behind the toolchain presented in this paper: to generate an MCU firmware customized for the SPC that starts from feature capture and includes all software aspects, such as the pre-processing and ML stages, for finally moving AI on the edge. The toolchain presented in this paper is suitable for any Arduino-compliant microcontroller unit (MCU) and is the first designed specifically for the SPC and its APIs (SPC is described in Section 3). The entire toolchain and the SPC represent the original contribution of this work:

- The SPC is unique among its kind due to being highly configurable and able to control internal and external sensors both through electrical impedance spectroscopy (EIS) and voltammetry;
- The SPC can be used to exploit any transducer that generates variations in electrical impedance;
- The C code produced by the toolchain is customized for deployment with SPC and can work on any Arduino-compatible MCU;
- The toolchain is open source and available for download on GitLab [30].

3. The SENSIPLUS MicroChip and the Toolchain Architecture

SPC is a proprietary micro-analytical sensing platform equipped with on-chip sensing capabilities. Developed by Sensichips s.r.l. and the Department of Information Engineering of the University of Pisa [31], it is equipped with several internal and external ports, and with a versatile analog front end that allows performing EIS on both internal and external sensors (see Figure 1). It is characterized by extremely low power consumption (less than 1.4 mW), very small sizes (1.5 mm $\times$ 1.5 mm), and a potentially low cost (less than 10 USD in large-scale production).

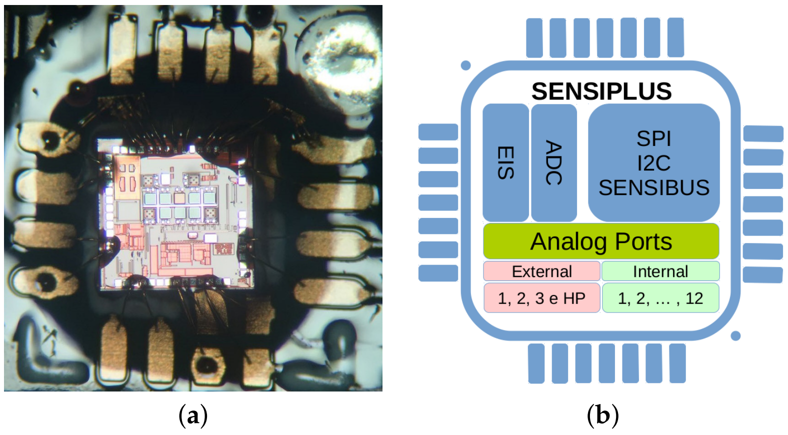


Figure 1. The SENSIPLUS microChip (SPC) schematics: (**a**) picture taken with a microscope; (**b**) schematic representation.

The SPC has no computational capabilities and, therefore, was implemented with a microcontroller unit (MCU) equipped with communication capabilities (wired and wireless). The connection between SPC and MCU was made with the SENSIBUS, a one-wire serial protocol that allows connecting many SPC slaves, owing to its 6-byte addressing mode.

The toolchain is freely available for download [30] and is open source. This section presents the modules that make up the toolchain shown in Figure 2. The main modules of the toolchain are the "code generator" and the "builder". The "code generator" works assuming the parameters of the chosen machine learning model as input; the obtained C code is ready for compilation and execution on the target MCU. Starting from the C code obtained using the "code generator", the builder compiles it preparing the firmware to be loaded on the MCU. The code generator works by exploiting the tags in the C code templates and the makefiles, thus allowing the injection of what is necessary for their specialization. For example, in the code (Listings 1 and 2) it is possible to identify some tags with a specific task. During source generation, the _INIT_CLASSIFIER_ tag refers to the code needed to prepare the model. The _INIT_SENSOR_ tag identifies the code to initialize the desired sensors. In the construction phase, however, the _OUT_DIR_ tag defines the position of the compilation result. The _HARDWARE_ tag refers to the compilers used in the host system, and the HEX tag represents the reference to a temporary file used during the process. The toolchain is written in Java, and some of the main classes are presented in the next subsections.

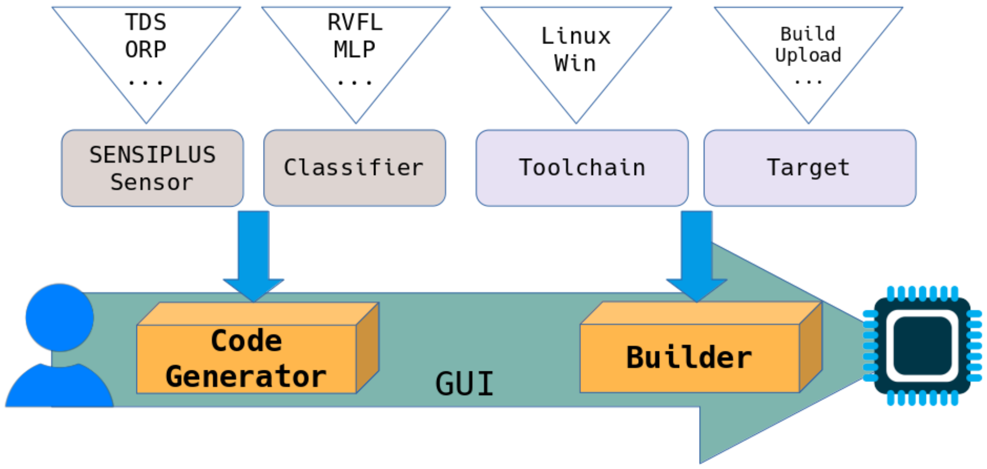


Figure 2. Toolchain architecture.

Listing 1: Extract of a C source.

```
/* Completion of the initialization phase
*  where the remaining data structures are associated and filled */
fillConfiguration();
initAllSPMeasurements();
/* SENSIPLUS initialization (involves communication with the chip) */
initChip();
_INIT_CLASSIFIER_
_INIT_SENSOR_
```

Listing 2: Extract of a makefile.

```
# Main-build Target
main-build: generic_classifier
# Tool invocations
generic_classifier: $(HEX)
        @echo 'Building␣target:␣$@'
        @echo 'Report:'
        "_HARDWARE_/xtensa-esp32-elf-size" -A "_OUT_DIR_/generic_classifier.elf"
        @echo 'Finished␣building␣target:␣$@'
```

3.1. *Module for Code Generation*

The code generator module (Figure 3) generates the C code that is ready for compilation, assuming the classifier and sensor selections as input. The toolchain's C code will automatically link to the C APIs available for SENSIPLUS by the builder module (described in the next section). The diagram below briefly describes the main classes of the code generator.

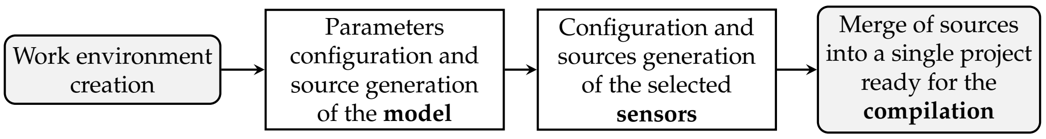

Work environment creation
Parameters configuration and source generation of the **model**
Configuration and sources generation of the selected **sensors**
Merge of sources into a single project ready for the **compilation**

Figure 3. Workflow for the code generator module.

3.1.1. The "Sensor" Class

The abstract "Sensor" java class was implemented to represent a generic sensor. The SPC can control two categories of sensors: "direct" and "derived". As direct sensors, we used temperature or voltage sensors. On the other hand, derived sensors detect the quantity of interest through impedance measurement. The toolchain in its current version already contains several extensions of the Sensor class (OffChipORP, OffChipSpecificConductivity, etc.). To add new sensors, the developer would have to extend the Sensor class by overriding the methods that generate the C code to invoke the SENSIPLUS API for initialization and subsequent measurement.

To give a practical example, in what follows, we report a brief description of the java classes of the sensors available in the smart cable water (SCW), a very compact SPC configuration with an area of 12 × 15 mm^2 that uses "derived" sensors made with interdigitated electrodes (IDEs) and connected to external analog ports of the SPC. The six IDEs on the SCW were functionalized with six metals: copper, gold, silver, nickel, palladium, and platinum (Figure 4).

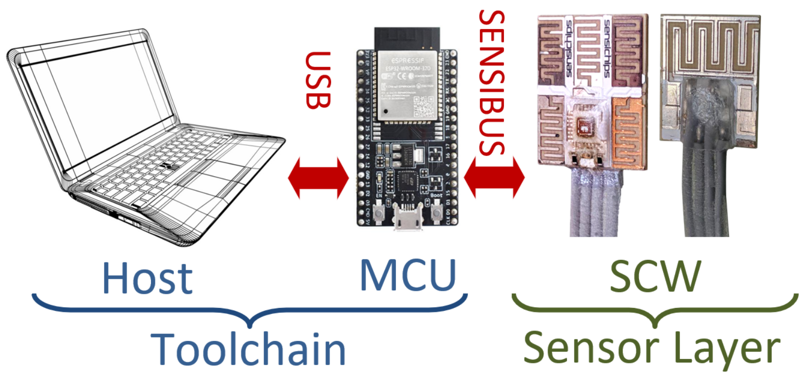


Figure 4. A smart cable water (SCW) model connected to the MCU through the SENSIBUS, a proprietary one-wire bus adopted by SPC. On the SCW the IDEs in different metals and the SPC are visible.

The following implementations of the Sensor class are available in the current version of the toolchain:

- *OffChipORP*: this class is the C code generator for the oxidation–reduction potential (ORP), which evaluates the tendency of the electrolyte to lose or acquire electrons when the solution changes by injecting a new substance;
- *OffChipSpecificConductivity*: this class is the C code generator for the conductivity measurement expressed in mS/cm, which is a measure of the ability of water to conduct electricity;
- *OffChipPracticalSalinity*: this class is the C code generator for the direct measurement of salinity, which is usually referred to as “practical salinity” and is typically derived from conductivity measurement (not measured directly).

3.1.2. The “Classifier” Class

The "Classifier" is an abstract java class and defines an abstraction of a classifier. As is known, in the life cycle of a classifier, there are two fundamental phases: training and inference. In the current implementation, the model’s training was carried out on a PC or a Workstation exploiting the Weka library [32]. After training, the selected model was input to this java class (its extension), thus generating the C code for the inference on the MCU. In the current version of the toolchain, there were different specializations of the Classifier class: multi-layer perceptron (MLP) [33], random-vector functional link (RVFL) [34], decision tree [35], etc. New ML algorithms could be added simply with further extensions of this Java class.

Each of these sub-classes was conceptualized for generating the C code of an ML model, assuming the related parameters obtained via training as input.

As a case study, the MLP C code generated using the toolchain was tested to evaluate both the inference times and the accuracy achieved on the MCU and investigate whether it was comparable to the implementation on the PC of standard libraries. The details of these assessments are given in Section 4.

3.1.3. The “CodeGenerator” Class

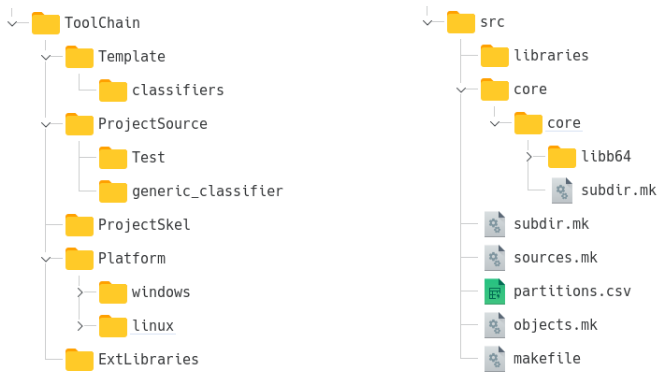
The “CodeGenerator” java class calls java classes for managing sensors and classifiers and returns the C code optimized for the problem under consideration. It also generates the invocation of the appropriate library functions of the SENSIPLUS C API for the MCU. The related C source code was structured as follows:

- A code that is independent of the classification problem at hand; this code initializes the SPC and appropriately manages the resources of the MCU;
- A code that implements the inferential engine of the classifier;
- A code with calls to sensor control C functions, integrated with code generated by the Sensor’s derived classes.

3.2. Module to Create "Builder"

By toolchain, we mean a set of utilities, compilers, and libraries devoted to generating a firmware suitable for the MCU of the C code that implements an ML algorithm (Figure 5a). The first version of the toolchain presented in this paper targets a family of processors based on the Xtensa architecture produced by the Espressif company. One of the reasons for this choice was certainly the good ratio quality/performance of these devices [36]. The Arduino was chosen as a programming model, and the toolchain was cross-platform (can be used on Windows, Linux, or macOS). Below are the principal components of the toolchain:

- OS-dependent executables that allow compiling, linking, and debugging;
- Scripts for communicating with cards hosting a specific MCU, such as the components for loading the firmware in the persistent memory of the devices;
- Other executables that perform additional tasks but are not strictly necessary for compilation, including, for example, executables to partition the device memory and load data into it; the specific cases in which they were used are described below.



(**a**) (**b**)

Figure 5. Directory structure: (**a**) typical directory structure managed by the toolchain; (**b**) typical makefile structure of a project sample.

3.2.1. The Class "MakeConfigGenerator"

The "MakeConfigGenerator" generates the makefiles related to the project. The routines of this module allow one to create the files necessary for compilation with the proper directives in the makefiles. An example of the generated file structure is reported in Figure 5b.

3.2.2. Classes for MCU Utility

The ESP32BoardConfig and BoardPartitionScheme java classes contain the different characteristics of the boards on which the MCU is mounted to correctly generate the firmware. The ESP32BoardConfig defines the MCU family in terms of available memory, communication interfaces, etc.). The BoardPartitionScheme describes memory partitioning.

3.2.3. Target Class

The Target class implements a sequence of operations into three methods:

- *The preExecution step*: this step makes all the preliminary activity for the following steps;
- *The targetBuildExecution step*: this step generates the firmware;
- *The postExecution step*: this step loads the firmware on the MCU, completing all the toolchain operations.

Several implementations were derived from these classes that represent different use cases. These were the different operations performed on the devices.

As mentioned previously, a partitioning system is needed for loading persistent data (firmware). Device configurations can be efficiently changed by acting on this memory area without regenerating the entire firmware. In fact, the UloadSpiffTarget java class has the task of correctly formatting the available memory. This target supports wear smoothing, file system consistency checks, etc., and is based on the SPI Flash File System (SPIFFS). The UploadFirmwareTarget java class includes only the phase of uploading the firmware into the device memory. The GenericTarget class implements and manages the compilation from sources through makefiles (Figure 6). Since the build process can be time-consuming, the system uses makefiles to avoid unnecessary builds.

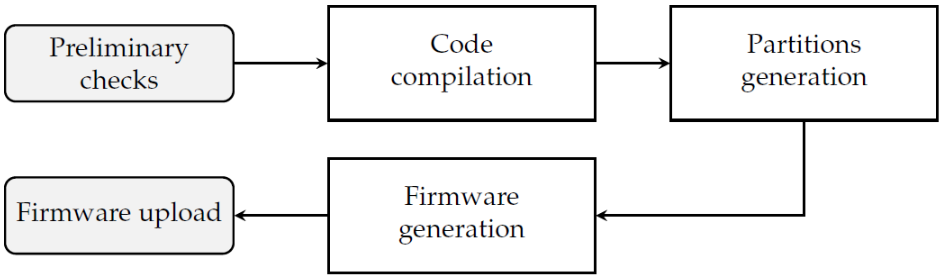


Figure 6. The execution model of GenericTarget class.

4. Inference Performance Evaluation

As a case study, an MLP model was transformed into C language and tested directly on the target device. Two kinds of tests were performed on the generated firmware: evaluating the numerical precision and the inference time and comparing the accuracy on MCU with the accuracy on the PC/Workstation. All the tests were carried out on a board equipped with the ESP32 microcontroller, endowed with a floating-point unit: the Wemos D32 pro [37].

4.1. Numerical Precision and Inference Time Analysis

To evaluate the numerical precision and inference time, the MLP was tested in various configurations regarding the number of neurons in the hidden layer, different activation functions, and different precisions of parameter representation.

The precision and response times were tested on MLP configurations, as shown in Table 1.

Table 1. MLP tested configurations.

# of Neurons	**# of Neurons**	**# of Neurons**
Input Layer	**Single Hidden Layer**	**Output Layer**
3	16	6
3	32	6
3	64	6
8	16	6
8	32	6
8	64	6

The first test is related to the numerical precision adopted for model representation, using 32- or 64-bit floating points.

The tests demonstrate that the result remained the same up to the eighth decimal digit, varying with the number of bits used. At the same time, the comparison with the reference value obtained on the PC remained unchanged up to the fourth decimal digit.

The inference time was evaluated with different tests with both floating-point representations. Furthermore, the impact on the performance of the Softmax activation function applied to the output neurons was evaluated. For the other layers, the ReLu activation function was always used. The time spent between input and output, expressed in microseconds (µs), was calculated 10,000 times for each execution, feeding the input with randomly generated values.

The evaluations that are shown in Figure 7a–c highlight the benefit of using a 32-bit floating-point representation. In general, for the tested configuration, the inference time was always less than a millisecond. Finally, the configuration was composed of 8 input neurons, 64 in the hidden layer, and 6 in outputs, and Softmax was used as an output activation function to evaluate the current consumption. The Agilent 34401A Digital Multimeter was utilized, highlighting a current consumption equal to 66.4 mA, resulting higher than the idle case (41.8 mA).

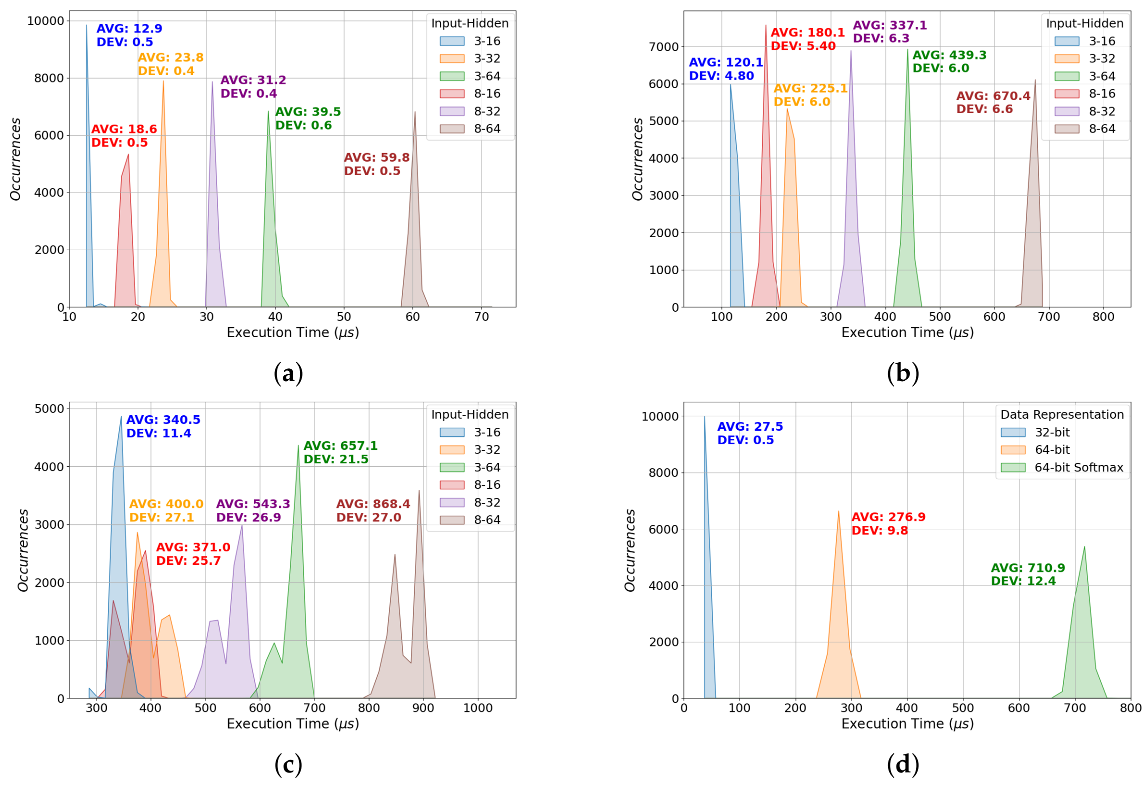


Figure 7. Evaluation of inference time for different MLP configurations. In the first three subplots, the number of neurons in the input and hidden levels varies, while the output is always fixed at six. In the last subplot, the performance of the model used in the real use case is highlighted: (**a**) inference time evaluation with 32-bit floating-point representation; (**b**) inference time evaluation with 64-bit floating-point representation; (**c**) inference time evaluation with 64-bit floating-point representation and Softmax activation function; (**d**) analysis of inference times of MLP configurations in the proposed real use case. The model was composed of 10 neurons in the input layer, a single hidden layer with a size of 16 neurons, and 13 neurons in the output layer.

4.2. Analysis of Inference Accuracy with a Real-Use Case

A model related to a real-use case for detecting pollutants in water was used as a final test. In this case, the SCW was used to obtain the measurements. Ten features were collected from sensors based on EIS built on the six IDE available on the SCW (Table 2).

During a previous experimental campaign, MLP with a single hidden neuron layer was found to have the best performance/consumption trade-off [27]. The structure of the model was composed of 10 neurons in the input layer, a single hidden layer with a size of 16 neurons, 13 neurons in the output layer, and a Softmax as an activation function on the last layer. The dataset for training the model contained the measures obtained on 12 substances plus the background (the wastewater—WW) and has been made publicly available in [38]. The measurement protocol for dataset acquisition was divided into two steps:

- The first 600 samples were acquired in warm-up mode; during this time, sensors were exposed to WW only;
- The next 1000 samples were acquired after analyte injection.

Table 2. Selected features (sensors).

	IDE	Frequency of Acquisition	Physical Values
1	PLATINUM	78 kHz	RESISTANCE
2	GOLD	78 kHz	RESISTANCE
3	PLATINUM	200 Hz	RESISTANCE
4	PLATINUM	200 Hz	CAPACITANCE
5	GOLD	200 Hz	RESISTANCE
6	GOLD	200 Hz	CAPACITANCE
7	SILVER	200 Hz	RESISTANCE
8	SILVER	200 Hz	CAPACITANCE
9	f NICKEL	200 Hz	RESISTANCE
10	NICKEL	200 Hz	CAPACITANCE

In this way, each acquisition contained 1600 samples; the first 600 samples were measured in WW, and the remaining 1000 samples were measured with the analyte mixed with WW. For each substance, 10 acquisitions or experiments were carried out. More details on the substances can be found in Table 3. The inference times are shown in Figure 7d. The offline validation of the models was carried out with the K-fold cross-validation method with K equal to 10, obtaining an overall accuracy of 92.75% with the confusion matrix shown in Figure 8. The obtained results show that the system is particularly good at identifying some substances, such as INT_NELSEN and SODIUM_CHLORIDE, which have an average accuracy of 99% with a standard deviation of 0.2%. On the other hand, they show that some substances are confused with others, such as the case of ACETONE and ETHANOL. With K-fold cross-validation, we obtained ten models: the one with the best accuracy was moved to the microcontroller with the toolchain system presented in this work.

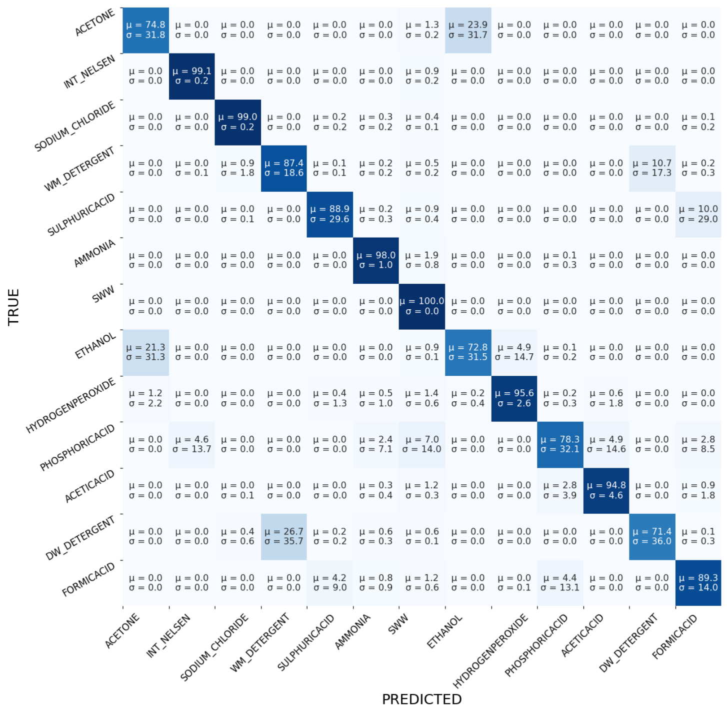


Figure 8. Summary confusion matrix, obtained from 10 folds, with mean value and standard deviation.

Table 3. Details of the chemical composition of substances. In particular, for each analyte, 10 mL was inserted in 100 mL of wastewater; the composition of this 10 mL amount is described in the analyte column.

	Substances	Brief Description	Analyte Composition	Concentration in 100 mL
1	ACETONE	C_3H_6O	10 mL	71,727.27 mg/L
2	INT_NELSEN	Off-The-Shelf product	9.2 mL H_2O + 0.8 mL Dish Detergent	7476.36 mg/L
3	SODIUM_CHLORIDE	$NaCl$	9.98 mL H_2O + 0.02 g Sodium Chloride	181.81 mg/L
4	WM_DETERGENT	Off-The-Shelf product	9.2 mL H_2O + 0.8 mL Washing Machine Detergent	7476.36 mg/L
5	SULPHURICACID	H_2SO_4	9.9 mL H_2O + 0.1 mL Sulphuric Acid	1798.20 mg/L
6	AMMONIA	NH_3	9.7 mL H_2O + 0.3 mL Ammonia 30/33%	2454.54 mg/L
7	WASTEWATER	Real domestic sewage $pH = 7.4$, conductivity = 1.341 mS		
8	ETHANOL	C_2H_5OH	10 mL	71,727.27 mg/L
9	HYDROGENPEROXIDE	H_2O_2	8 mL H_2O_2 35%	83,703.70 mg/L
10	PHOSPHORICACID	H_3PO_4	9.8 mL H_2O + 0.2 mL Phosphoric Acid 75%	2327.27 mg/L
11	ACETICACID	CH_3COOH	9.5 mL H_2O + 0.5 mL Acetic Acid 80%	4772.72 mg/L
12	DW_DETERGENT	Off-The-Shelf product	9.2 mL H_2O + 0.8 mL Dishwasher Detergent	7476.36 mg/L
13	FORMICACID	CH_2O_2	9.8 ml H_2O + 0.2 mL Formic Acid 85%	2181.81 mg/L

5. Conclusions

This paper presented a joint system composed of a microcontroller, a sensing device, and edge intelligence development named SENSIPLUS-LM. It consists of a multi-sensor microchip endowed with EIS capabilities and empowered by an open-source, end-to-end toolchain.

The presented toolchain involves the firmware generation for the MCU necessary to drive the SPC, starting from a suitable trained ML model. Although the toolchain is strictly related to the SPC, it has no other constraints: The target MCU could be any Arduino-compliant architecture. The toolchain code is released in open-source mode and available online. The toolchain is also described in sufficient detail to allow its upgrade with the insertion of new sensors and/or new ML algorithms. The presented toolchain empowers the SENSIPLUS platform moving the inference phase of the ML algorithm to the edge, improving the local data processing and its suitability to be adopted in IoT distributed monitoring systems.

Author Contributions: Conceptualization, M.M. and M.V.; investigation, M.M., M.V. and G.C.; methodology, M.M.; project administration, M.M.; software, M.V.; supervision, M.M.; validation, G.C.; writing—original draft preparation, M.M., M.V. and G.C.; writing—review and editing, L.G., G.M. and A.R. All authors have read and agreed to the published version of the manuscript.

Funding: The research leading to these results received funding from the European Unions Horizon 2020 research and innovation program under grant agreement SYSTEM No. 787128. The authors are solely responsible for it, and it does not represent the opinion of the Community, and the Community is not responsible for any use that might be made of the information contained therein.

Data Availability Statement: A subset of the data used in this paper is available on this website: https://aida.unicas.it/data/JKSU_2022.zip (accessed on 18 January 2023).

Acknowledgments: This work was also supported by MIUR (Minister for Education, University and Research, Law 232/216, Department of Excellence).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
ML	Machine Learning
API	Application Programming Interface
ADC	Analog to Digital Converter
AFE	Analog Front End

ANN	Artificial Neural Network
EIS	Electrical Impedance Spectroscopy
I2C	Inter-Integrated Circuit
IoT	Internet of Things
MCU	Microcontroller Unit
MLP	Multi-Layer Perceptron
ORP	Oxidation–Reduction Potential
RVFL	Random-Vector Functional Link
SCW	Smart Cable Water
SPC	SENSIPLUS Chip
SPI	Serial Peripheral Interface
TinyML	Tiny Machine Learning
TDS	Total Dissolved Solids

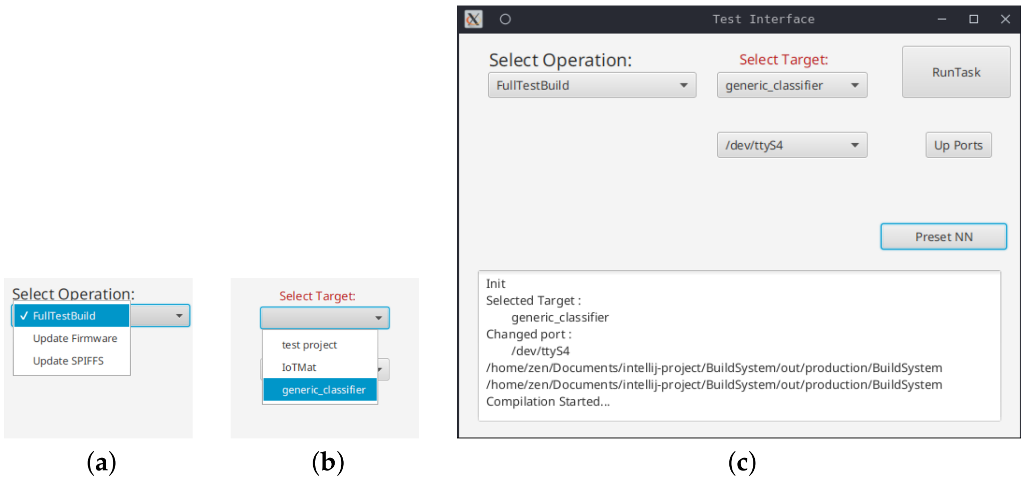
Appendix A. Graphical User Interface

A simple GUI in JavaFX was implemented and integrated into the downloadable software to allow the easy use of the toolchain. Through the GUI, it was possible to configure the execution of the toolchain by specifying, for example, which serial port to use for the firmware upload and how to partition the SPIFFS of the MCU. In addition, by selecting the starting folder of the project, all files could be formatted to be ready for loading on the MCU.

Example of Usage

Here are the steps necessary for use:

1. In the first step, it is possible to select between three options: FullTestBuild to use all the toolchain components, "Update Firmware" to only upload the firmware, and "Update SPIFFS" to make only the upload of the configuration files (Figure A1a);
2. In the second step, it is possible to select the target. By choosing the "generic_classifier" item, the generation of a complete ANN classifier is completed (Figure A1b);
3. The "Preset NN" button generates the machine learning model taking a CSV file containing the necessary information as input;
4. The last step consists of selecting the serial port on which the MCU is connected, allowing the whole process to start (Figure A1c).



Select Operation: FullTestBuild
Select Target: generic_classifier
/dev/ttyS4

```
Init
Selected Target :
    generic_classifier
Changed port :
    /dev/ttyS4
/home/zen/Documents/intellij-project/BuildSystem/out/production/BuildSystem
/home/zen/Documents/intellij-project/BuildSystem/out/production/BuildSystem
Compilation Started...
```

Figure A1. Operation and target configuration. Step 1: operation Selection; Step 2: target selection; Step 3: firmware generation and upload: (**a**) Step 1; (**b**) Step 2; (**c**) Step 3.

References

1. Datta, P.; Sharma, B. A survey on IoT architectures, protocols, security and smart city based applications. In Proceedings of the 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Delhi, India, 3–5 July 2017; pp. 1–5. [CrossRef]
2. Cennamo, N.; Arcadio, F.; Perri, C.; Zeni, L.; Sequeira, F.; Bilro, L.; Nogueira, R.; D'Agostino, G.; Porto, G.; Biasiolo, A. Water monitoring in smart cities exploiting plastic optical fibers and molecularly imprinted polymers. The case of PFBS detection. In Proceedings of the 2019 IEEE International Symposium on Measurements & Networking (M&N), Catania, Italy, 8–10 July 2019; pp. 1–6. [CrossRef]
3. Hancke, G.P.; Silva, B.D.C.e.; Hancke, G.P., Jr. The Role of Advanced Sensing in Smart Cities. *Sensors* **2013**, *13*, 393–425. [CrossRef] [PubMed]
4. Carminati, M.; Sinha, G.R.; Mohdiwale, S.; Ullo, S.L. Miniaturized Pervasive Sensors for Indoor Health Monitoring in Smart Cities. *Smart Cities* **2021**, *4*, 146–155. [CrossRef]
5. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* **2018**, *6*, 6900–6919. [CrossRef]
6. Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; Zhang, J. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proc. IEEE* **2019**, *107*, 1738–1762. [CrossRef]
7. Kaur, K.; Garg, S.; Aujla, G.S.; Kumar, N.; Rodrigues, J.J.P.C.; Guizani, M. Edge Computing in the Industrial Internet of Things Environment: Software-Defined-Networks-Based Edge-Cloud Interplay. *IEEE Commun. Mag.* **2018**, *56*, 44–51. [CrossRef]
8. Molinara, M.; Ferdinandi, M.; Cerro, G.; Ferrigno, L.; Massera, E. An End to End Indoor Air Monitoring System Based on Machine Learning and SENSIPLUS Platform. *IEEE Access* **2020**, *8*, 72204–72215. [CrossRef]
9. Betta, G.; Cerro, G.; Ferdinandi, M.; Ferrigno, L.; Molinara, M. Contaminants detection and classification through a customized IoT-based platform: A case study. *IEEE Instrum. Meas. Mag.* **2019**, *22*, 35–44. [CrossRef]
10. Gerevini, L.; Cerro, G.; Bria, A.; Marrocco, C.; Ferrigno, L.; Vitelli, M.; Ria, A.; Molinara, M. An end-to-end real-time pollutants spilling recognition in wastewater based on the IoT-ready SENSIPLUS platform. *J. King Saud Univ.-Comput. Inf. Sci.* **2023, *35*, 505–519.** . [CrossRef]
11. Deng, S.; Zhao, H.; Fang, W.; Yin, J.; Dustdar, S.; Zomaya, A.Y. Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence. *IEEE Internet Things J.* **2020**, *7*, 7457–7469. [CrossRef]
12. Ray, P.P. A review on TinyML: State-of-the-art and prospects. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 1595–1623. [CrossRef]
13. Dutta, D.L.; Bharali, S. TinyML Meets IoT: A Comprehensive Survey. *Internet Things* **2021**, *16*, 100461. . t.2021.100461. [CrossRef]
14. Saha, S.S.; Sandha, S.S.; Srivastava, M. Machine Learning for Microcontroller-Class Hardware: A Review. *IEEE Sens. J.* **2022**, *22*, 21362–21390. [CrossRef] [PubMed]
15. TensorFlow Lite for Microcontrollers—Tensorflow.org. Available online: https://www.tensorflow.org/lite/microcontrollers (accessed on 15 September 2022).
16. Soro, S. TinyML for Ubiquitous Edge AI. *arXiv* **2021**, arXiv:2102.01255.
17. Karimipour, H.; Derakhshan, F. (Eds.) Artificial Intelligence for Threat Detection and Analysis in Industrial IoT: Applications and Challenges. In *AI-Enabled Threat Detection and Security Analysis for Industrial IoT*; Springer International Publishing: Cham, Switzerland, 2021; pp. 1–6. [CrossRef]
18. Bilal, M.; Usmani, R.S.A.; Tayyab, M.; Mahmoud, A.A.; Abdalla, R.M.; Marjani, M.; Pillai, T.R.; Targio Hashem, I.A., Smart Cities Data: Framework, Applications, and Challenges. In *Handbook of Smart Cities*; Augusto, J.C., Ed.; Springer International Publishing: Cham, Switzerland, 2020; pp. 1–29. [CrossRef]
19. uTensor. Available online: http://utensor.ai (accessed on 15 September 2022).
20. NanoEdge AI Studio. Available online: https://cartesiam-neai-docs.readthedocs-hosted.com/ (accessed on 15 September 2022).
21. Edge Impulse. Available online: https://www.edgeimpulse.com/ (accessed on 15 September 2022).
22. STM32Cube.AI. Available online: https://www.st.com/content/st_com/en/ecosystems/stm32-ann.html (accessed on 15 September 2022).
23. Wang, X.; Magno, M.; Cavigelli, L.; Benini, L. FANN-on-MCU: An Open-Source Toolkit for Energy-Efficient Neural Network Inference at the Edge of the Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 4403–4417. [CrossRef]
24. Sliwa, B.; Piatkowski, N.; Wietfeld, C. LIMITS: Lightweight Machine Learning for IoT Systems with Resource Limitations. In Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC), Online, 7–11 June 2020; pp. 1–7. [CrossRef]
25. SPC5-STUDIO-STMicroelectronics—st.com. Available online: https://www.st.com/en/development-tools/spc5-studio.html (accessed on 15 September 2022).
26. BME688 Software—Bosch-Sensortec.com. Available online: https://www.bosch-sensortec.com/software-tools/software/bme688-software (accessed on 15 September 2022).
27. Bria, A.; Cerro, G.; Ferdinandi, M.; Marrocco, C.; Molinara, M. An IoT-ready solution for automated recognition of water contaminants. *Pattern Recognit. Lett.* **2020**, *135*, 188–195. [CrossRef]
28. Ferdinandi, M.; Molinara, M.; Cerro, G.; Ferrigno, L.; Marrocco, C.; Bria, A.; Di Meo, P.; Bourelly, C.; Simmarano, R. A Novel Smart System for Contaminants Detection and Recognition in Water. In Proceedings of the 2019 IEEE International Conference on Smart Computing (SMARTCOMP), Washington, DC, USA, 12–15 June 2019; pp. 186–191. [CrossRef]

29. Cerro, G.; Ferdinandi, M.; Ferrigno, L.; Molinara, M. Preliminary realization of a monitoring system of activated carbon filter RLI based on the SENSIPLUS® microsensor platform. In Proceedings of the 2017 IEEE International Workshop on Measurement and Networking (M&N), Naples, Italy, 27–29 September 2017; pp. 1–5. [CrossRef]
30. SENSIPLUSToolchain · GitLab—gitlab.com. Available online: https://gitlab.com/t8664 (accessed on 15 September 2022).
31. Ria, A.; Cicalini, M.; Manfredini, G.; Catania, A.; Piotto, M.; Bruschi, P. The SENSIPLUS: A Single-Chip Fully Programmable Sensor Interface. In Proceedings of the International Conference on Applications in Electronics Pervading Industry, Environment and Society, Genova, Italy, 26-27 September 2022; Saponara, S., De Gloria, A., Eds.; Springer International Publishing: Cham, Switzerland 2022; pp. 256–261.
32. Frank, E.; Hall, M.A.; Holmes, G.; Kirkby, R.; Pfahringer, B.; Witten, I.H. Weka: A machine learning workbench for data mining. In *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers*; Maimon, O., Rokach, L., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1305–1314.
33. Hastie, T.; Tibshirani, R.; Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2009.
34. Rosato, A.; Altilio, R.; Panella, M. On-line Learning of RVFL Neural Networks on Finite Precision Hardware. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5. [CrossRef]
35. Quinlan, J.R. *C4.5: Programs for Machine Learning*; Elsevier: Amsterdam, The Netherlands, 2014.
36. Sudharsan, B.; Salerno, S.; Nguyen, D.D.; Yahya, M.; Wahid, A.; Yadav, P.; Breslin, J.G.; Ali, M.I. TinyML Benchmark: Executing Fully Connected Neural Networks on Commodity Microcontrollers. In Proceedings of the 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), New Orleans, LA, USA, 14 June–31 July 2021; pp. 883–884. [CrossRef]
37. D32 Pro; WEMOS Documentation—docs.wemos.cc. Available online: https://docs.wemos.cc/en/latest/d32/d32_pro.html (accessed on 15 September 2022).
38. Public Link for Downloading the Acquired Dataset. 2022. Available online: https://aida.unicas.it/data/JKSU_2022.zip (accessed on 11 November2022).