TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities

Ramon Sanchez-Iborra and Antonio F. Skarmeta

1531-636X/20©2020IEEE

Abstract

The TinyML paradigm proposes to integrate Machine Learning (ML)-based mechanisms within small objects powered by Microcontroller Units (MCUs). This paves the way for the development of novel applications and services that do not need the omnipresent processing support from the cloud, which is power consuming and involves data security and privacy risks. In this work, a comprehensive review of the novel TinyML ecosystem is provided. The related challenges and opportunities are identified and the potential services that will be enabled by the development of truly smart frugal objects are discussed. As a main contribution of this paper, a detailed survey of the available TinyML frameworks for integrating ML algorithms within MCUs is provided. Besides, aiming at illustrating the given discussion, a real case study is presented. Concretely, we propose a multi-Radio Access Network (RAT) architecture for smart frugal objects. The issue of selecting the most adequate communication interface for sending sporadic messages considering both the status of the device and the characteristics of the data to be sent is addressed. To this end, several TinyML frameworks are evaluated and the performances of a number of ML algorithms embedded in an Arduino Uno board are analyzed. The attained results reveal the validity of the TinyML approach, which successfully enables the integration of techniques such as Neural Networks (NNs), Support Vector Machine (SVM), decision trees, or Random Forest (RF) in frugal objects with constrained hardware resources. The outcomes also show promising results in terms of algorithm's accuracy and computation performance.

I. Introduction

Machine Learning (ML) has revolutionized our way of understanding the world by enabling the inference of valuable information and knowledge from huge amount of data, so far invisible to human eye. Data are produced by our digital activity and grow at almost unlimited rates [1], hence calling for large pools of resources in terms of processing power and storage for their handling and analysis. In this regard, research and innovation advances during the last years have been notable in multiple areas of study [2]–[5]. These efforts have focused on obtaining a better response from the wide range of available ML techniques, e.g., Deep Learning (DL), Neural Networks (NNs), Reinforcement Learning (RL), clustering, etc., by exploiting the potential offered by state-of-the-art processors, data-centers, and supercomputers [6]. However, a non-negligible segment of processing entities has received scarce attention due to their inherent computing constraints: Microcontroller Units (MCUs). This tiny processing entities are usually embedded in daily-use appliances such as vehicles, medical devices, personal gadgets, etc., but they may also constitute the core processing unit of small frugal objects [7].

Digital Object Identifier 10.1109/MCAS.2020.3005467
Date of current version: 12 August 2020

Frugal objects are independent "things" specialized in perform concrete operations that can be programmed with software rather than requiring specific electronics for each task, so they can be massively produced at inexpensive cost. These devices incorporate enough processing and memory resources, sensors, and connectivity capabilities to perform the required operation, while being highly energy efficient. A concise definition of types of MCU-based devices according to their hardware resources is given in [8].

The penetration of frugal objects in our lives due to the emergence of the Internet of Things (IoT) is skyrocketing, thus provisioning them with cognitive skills opens a plethora of opportunities for developing a widespread web of collective intelligence (Fig. 1) [9]. Different paradigms such as fog or edge computing are fueling this hyper-connected distributed scheme supported by the new wave of communication technologies, which permit coordinating end-devices efforts to place intelligence close to the users [10]. Nevertheless, given the power constraints of MCUs, reducing data transmissions is crucial for their sustainability as this type of operations are more power demanding than computational tasks [11].

In this line, much data captured by sensors are wasted because of transporting costs, bandwidth limitations, or power constraints. For those reasons, performing on-device ML processing is a buzz trend today. The TinyML community [12] is boosting the integration of ML within MCU-based smart frugal objects. In fact, forecasts predict that the global edge computing market will reach 1.12 trillion dollars by 2023 [13] and some strong companies such as Ericsson are already offering TinyML-as-a-Service solutions [14]. In this paper, we provide a comprehensive overview of current challenges and opportunities

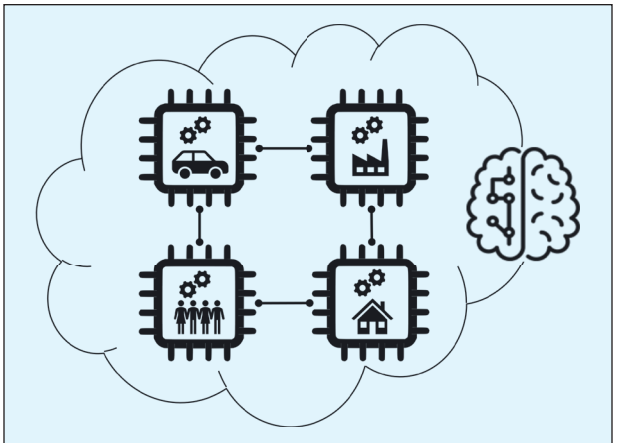

Figure 1. Interconnected smart services.

Ramon Sanchez-Iborra and Antonio F. Skarmeta are with the University of Murcia, Murcia, Spain, (ramonsanchez@um.es; skarmeta@um.es).

opened by this emerging hot-topic. A dissection of the potential services that will benefit from the introduction of ML-powered mechanisms into frugal objects is also presented. Currently available frameworks that permit to embed ML schemes into MCUs are explored and reviewed. Although some performance evaluations of certain frameworks have been reported [15], [16], to the authors' knowledge there is not any prior work providing a comprehensive survey focused on these valuable tools. Besides, we elaborate on a specific use case which is the intelligent communication interface selection in a constrained multi-Radio Access Technology (RAT) device, by considering the system status as well the characteristics of the data to be sent. To this end, an architectural design of the proposed frugal object and a real on-device implementation of a ML-based decision-making mechanism are finally showed. Therefore, the main contributions of this work are the following:

- A wide discussion of the TinyML concept as well as the range of potentially benefited applications.
- A survey of the principal available frameworks for integrating ML algorithms within MCUs.
- Insights regarding the real implementation of ML-based schemes over frugal objects to solve the problem of intelligent interface selection in a multi-RAT set up.

The rest of the paper is organized as follows. Section II provides a comprehensive overview of the TinyML landscape. Section III presents a survey of the available frameworks for integrating ML within frugal objects. Section IV focuses on the specific use case of devices with multiple communication interfaces, presenting the proposed architecture, the implementation process of ML algorithms within a MCU powered device, and the attained results. Finally, Section V summarizes the most important findings and draws future research lines.

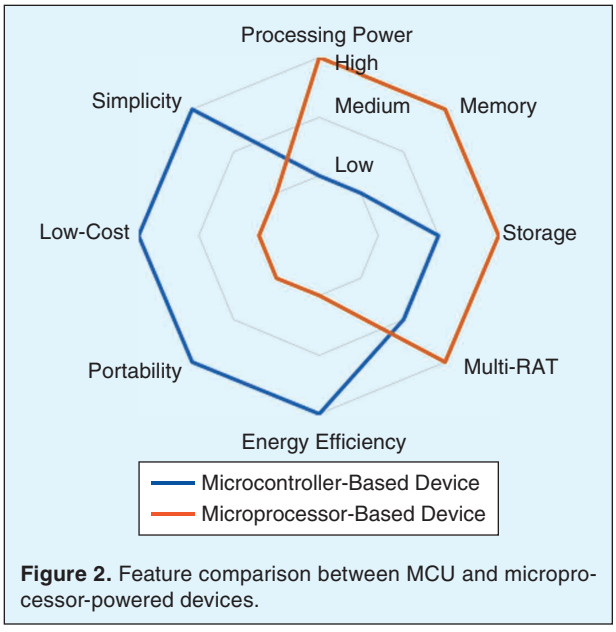

Figure 2. Feature comparison between MCU and microprocessor-powered devices.

II. Machine Learning in Frugal Objects

The society digitalization achieved during recent times has democratized the daily use of a plethora of frugal objects, e.g., wearables, environmental data collectors, actuators, etc. The integration of intelligence within these small devices brings a series of advantages that outweigh the possible cons. In the following, we discuss these aspects aiming at exploring the possibilities opened by TinyML from a realistic perspective.

A. Benefits

1) Energy Efficiency

This is the first positive point that comes to mind when adopting MCU-based solutions. Whereas powerful processors and Graphics Processing Units (GPUs) demand great amount of power, MCUs present a reduced energy consumption, even working at their maximum workload level. This permits MCUs to rely on batteries or even energy harvesting. Therefore, smart frugal objects can be placed almost everywhere without the need of being plugged to the power grid, which opens the door for novel cognitive itinerant applications [17]. Furthermore, having a scarce power consumption allows the smart unit to be coupled with larger battery-powered devices, hence converting them into connected smart entities, e.g., personal mobility devices such as scooters or segways. Fig. 2 compares the main characteristics of devices powered either by microcontroller or microprocessor.

2) Low Cost

The simplicity and hardware constraints of frugal objects leads to a moderate cost per unit. Clearly, this has been one of the main reasons for the quick and fruitful exploitation of IoT architectures in a large range of segments: industry, agriculture, e-health, or entertainment, among others [18]. These deployments consist of a large number of frugal objects specialized on performing varied tasks. The heterogeneity of employed end-devices, as well as their reprogrammability bring endless opportunities for enriching current deployments with intelligence, hence enabling their evolution into smart and flexible systems.

3) System Reliability and Data Security

As stated above, wireless transmissions require much energy in comparison with processing tasks. This involves a real need for limiting communication activities in frugal objects. Although this could be seen as a negative aspect, it actually allows to increase system's reliability as well as data security and privacy. Big data

applications require to stream raw data over unpredictable and lossy wireless channels from end-devices to the cloud. This strategy is bandwidth consuming and prone to transmission errors and cyber-attacks, e.g., man-in-the-middle or eavesdropping, so that the transmitted data may be lost or compromised. These issues are avoided by performing on-device data processing which permits to perform fewer transmissions with aggregated or meaningless data for an attacker.

4) Latency

This is another advantage of performing local processing instead of offloading computing tasks onto the cloud. State-of-the-art IoT RATs, e.g., Low Power Wide Area Network (LPWAN) [19], have notably improved energy efficiency and transmission distances at the expense of reducing data-rate. This fact leads to very long message's times-on-air that add undesired delay to the whole operation. Besides, on-device computation efficiency of complex ML algorithms is being deeply investigated in order to achieve real-time execution and notable advances are being achieved [20].

B. Challenges

Given the discussed advantages, integrating ML within frugal objects also presents important challenges to be overcome. These are mostly related to device heterogeneity and MCU's constraints.

1) Device Heterogeneity

Regarding this aspect, the enormous range of existing devices is a notable obstacle for having a general set of TinyML tools that could be valid for most of MCU-based set-ups. These units may present different power consumption, processing capabilities, memory, storage capacity, embedded RATs, peripherals, communication ports and protocols, etc., which also hinders the design of generic benchmarking methodologies in TinyML systems [21]. Therefore, the implementation of universal development and benchmarking frameworks coping with this heterogeneity is not an easy task although it is highly necessary for increasing awareness and adoption.

2) MCU Constraints

So far computation and memory constraints have not been a limitation given the quantity and quality of available resources in modern workstations and data centers. However, up-scaling computing system's resources according to the processing needs of ML mechanisms is not a valid approach for MCUs [22]. For that reason, TinyML ecosystem players should coordinate effective efforts for achieving a functional convergence between ML and frugal objects. In this line, MCU manufacturers should ensure that their chip-sets are suitable for straight-through ML integration, which will permit an easy application development with the support of device-level software. In this regard, it is important that future firmwares could reach high reliability levels and tackle key aspects such as in-device data security and optimized transmission operations. In addition, cloud players should develop enhanced platforms and APIs to allow efficient data exchanges between objects and cloud. Finally, the adaptation of mainstream data-science frameworks to MCU requirements is crucial for a wide expansion of TinyML. Not only programming libraries but also lifecycle management tools are needed to simplify solution development, data handling, etc. As discussed in Section III, a first wave of TinyML frameworks and developing tools is already available, which will foster the arrival of novel services and applications to the market by attracting new players.

C. Applications

Many IoT applications are already integrated in our daily personal and professional activities, although many of them are prone to be strengthened with local smart processing capabilities (Fig. 3). Most of current IoT devices make an extensive use of network-related operations, which is not always the best option as discussed previously. For instance, wearables such as fitness monitors or "smart" watches are not that smart, as they rely on a Bluetooth-coupled smartphone which assumes the majority of processing tasks. Integrating ML within frugal objects permits their independence from a master device, which will notably enrich current applications landscape.

1) eHealth

This is one of the segments with greater growth potential. Health monitoring, visual assistance, hearing aids, personal sensing, activity recognition, etc., are just few examples of applications that will exploit truly-smart end-devices. Beyond the health-care perspective, augmented reality, real-time voice recognition, language translation, context-aware support systems, or precise indoor/outdoor positioning are other wearable-based services to come in a near future [23].

2) Smart Spaces

ML-enabled frugal objects will also contribute to the collective intelligence already present in scenarios such as smart cities or cognitive buildings, among many others [24]. Current IoT-based monitoring and surveillance systems, e.g., traffic, pollution, crowdsensing, etc., will evolve to smart and autonomous entities capable of making decentralized and quick decisions. The deployment simplicity and independence from the power grid allow things to be placed in remote and rural areas, hence creating smart spaces with seamless roaming among them

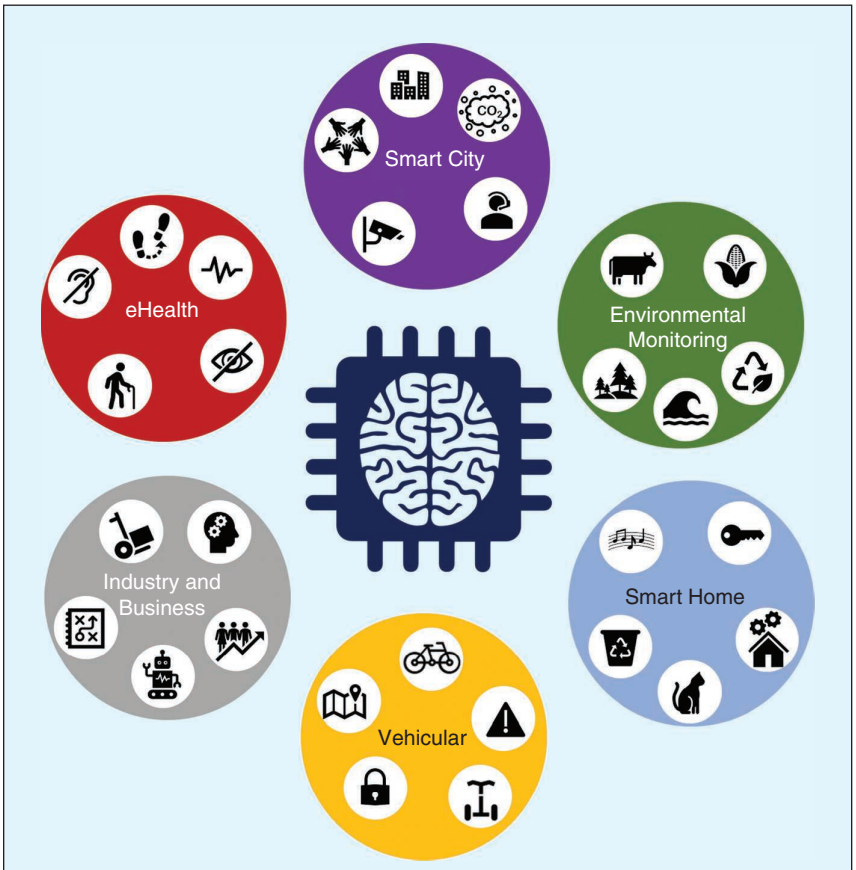

Figure 3. TinyML ecosystem.

[25]. Besides, the reduced cost of end-devices will also foster their adoption in disadvantaged areas, which may help to revamp local economy and business activities [26].

3) Vehicular Services

Thanks to the alliance between ML and MCUs, the vehicular market and, concretely, the Cooperative—Intelligent Transportation Systems (C-ITS) will be enriched with a segment of vehicles that has been ignored so far, namely, personal mobility devices. Urban mobility trends are changing during the last times due to the emergence of sustainable and healthy vehicles such as shared bikes, electric mopeds, scooters, etc. However, current vehicular networks just consider traditional road transportation means, i.e., cars, buses, trucks, etc., as their equipped On-Board Units (OBUs) require great amount of power to operate. Although some efforts have been done for connecting motorcycles to C-ITS infrastructures [27], personal vehicles have not been considered due to their novelty and inherent constraints. However, coupling these devices with MCU-based OBUs paves the way for their integration within the C-ITS and smart-city ecosystems. Therefore, light vehicles will have access to basic vehicular services such as driving safety, device status monitoring, route planning, etc.

4) Industry 4.0

The potential impact of TinyML is enormous in industrial and manufacturing sectors as they are experiencing a digitalization process under the umbrella of the Industry 4.0 revolution [28]. The processing tasks in these fields are highly heterogeneous in terms of computing or memory requirements, which causes that MCUs are not always capable of running certain tasks. However, the integration of ML-based Decision Support Systems (DSSs) within MCUs may permit them to determine whether to assume a certain computation task or offloading it onto upper processing layers such as edge or cloud [29]. Besides, the refinement of production processes, decision making, asset tracking, etc. will be driven by the integration of intelligence along the production chain as well.

5) Smart Agriculture and Farming

These sectors are also under a deep digital transformation [30]. By adopting ML-based smart monitoring and control, both of them will increment the efficiency and health of crops and animals, which will lead to greater revenues due to the improvement of production quality. Finally, common daily activities and home appliances will also reach the next level with the support of ML-enabled devices. For example, biometric e-locks will be generalized as well as context-aware domotic systems and wellbeing applications. Intelligent pet trackers and feeders, smart garden-irrigation control, or advanced safety & security home systems, are other use cases that will be enabled thanks to the upcoming TinyML revolution [31].

III. Tinyml Frameworks

Given the great potential of TinyML to revolutionize multiple sectors, a number of libraries and tools for easing the implementation of ML algorithms on constrained platforms are available in the market. TinyML frameworks can be categorized into three different families. The most common approach relates to the conversion of already trained models for adapting them to the restrictions of MCUs. Thus, these tools usually take inference engines generated by well-known ML libraries such as TensorFlow [32], Scikit-Learn [33], or PyTorch [34], among others,

and port their code in order to be executed by scarce resource devices. The second approach bets on integrating ML libraries within MCUs, hence providing them with local training and analysis capabilities. This permits to generate models from data retrieved by the own device, which may improve the accuracy of the model and also enables the implementation of unsupervised learning algorithms. Finally, the last approach relies on the availability of a dedicated co-processor that supports the main computing unit on ML-specific tasks. This strategy permits to enhance the computation performance, although it is the less common approach as the price and complexity of the processing platforms are notably increased.

Big technological companies are promoting the expansion of the TinyML ecosystem by offering open-source development libraries. Google has launched TensorFlow Lite [35], which includes a set of tools to adapt TensorFlow models aiming at making them to be runnable in mobile and embedded devices. This framework is composed of two main elements, namely, the converter, which ports TensorFlow models to optimized code that can be executed in constrained platforms, and the interpreter, which actually runs the code generated by the converter. The optimized models, which cover a number of algorithms from the NN family, can be executed in a range of platforms such as smart-phones, embedded Linux and MCUs. Focusing on the latter, the optimized code is provided in C++ 11 and requires 32-bit processors to be run. It has been successfully tested in devices with ARM Cortex-M series processors, e.g., Arduino nano, and other architectures such as ESP32. Given the notoriety of Arduino, a specific library for this platform is available through its IDE. Besides the development of TensorFlow Lite, another related Google's initiative has been the adaptation of the GO programming language to microcontrollers [36]. In turn, Microsoft has also contributed to the TinyML scene by releasing its open source Embedded Learning Library (ELL) [37]. This framework permits to design and deploy pre-trained ML models onto constrained platforms, e.g., ARM Cortex-A and Cortex-M architectures such as Arduino, Raspberry Pi, and micro:bit. ELL can be understood as an optimizing cross-compiler that runs on a regular desktop computer and outputs C++ code ready to be executed on the targeted single-board computer. ELL's API can be employed either from C++ or Python and makes use of pre-trained NN models produced by the Microsoft Cognitive Toolkit (CNTK) [38], Darknet [39], or in Open Neural Network Exchange (ONNX) format [40].

ARM is also pushing for the integration of ML in their devices. The ARM-NN toolkit [41] is part of Linaro's ML Initiative [56] and allows to integrate NN workloads within ARM Cortex-A CPUs, ARM Mali GPUs, and Ethos Neural Processing Units (NPUs). ARM-NN is an open-source project compatible with existing ML frameworks such as TensorFlow and Caffe [57], and the ONNX format. As ARM-NN does not provide support to the Cortex-M CPUs, ARM has also released the Cortex Microcontroller Software Interface Standard-NN (CMSIS-NN) [42], which is a collection of NN kernels optimized to be executed on this processor series. CMSIS-NN is able to convert floating point-based NNs generated by popular libraries, e.g., TensorFlow, PyTorch, and Caffe, into fixed-point format that can run on Cortex-M devices. Another electronics manufacturer that has developed specific libraries for its devices is STMicroelectronics. Concretely, the STM32 Cube.AI toolkit [43] permits to integrate pre-trained NNs within STM32 ARM Cortex-M-based microcontrollers. It generates STM32-compatible C code from NN models generated by Tensorflow and Keras [58], or in the standard ONNX format. As an interesting feature, STM32Cube.AI permits to run large NNs by storing weights and activation buffers in external flash memory and RAM, respectively.

Besides open-source initiatives, some institutions and companies have also launched licensed products. The Fraunhofer Institute for Microelectronic Circuits and Systems (IMS) has developed the Artificial Intelligence for Embedded Systems (AIfES) library [44], which is a C-based and platform-independent tool for generating NNs compatible with a range of MCUs, e.g., Arduino UNO, ATMega32U4, or STM32 F4 Series. It can also produce efficient code for Windows in the form of Dynamic Link Library (DLL) and Linux platforms such as Raspberry Pi. In its current version, it is able to extracts all parameters from models produced by TensorFlow or Keras. Different from the frameworks reviewed above, AIfES permits to implement the NN training process on the embedded device, i.e., it belongs to the second family of frameworks identified above. Cartesiam's NanoEdge AI Studio [45] is another commercial product which permits to create ML static libraries to embed them in Cortex-M-based MCUs. Similar to AIfES, it allows to integrate the training process within the constrained device but, in addition, it also permits to run unsupervised learning algorithms on the MCU. As a part of this toolkit, a microcontroller emulator is provided in order to test and debug the generated model prior to its deployment on the device.

Surprisingly, all the frameworks presented so far only focus on a single type of ML algorithm, namely, NN. However, different initiatives released by makers, developers, and researchers also consider other ML techniques such as Naive Bayes classifier, decision trees, k-Nearest Neighbors (k-NN), among others. MicroML [46] is a project that permits to port Support Vector Machine (SVM) and Relevance Vector Machine (RVM) algorithms to C code that can run in a number of MCUs, e.g., Arduino, ESP8266, ESP32, and others with C support. It interoperates with the

widely-used Scikit-learn toolkit and transforms the models generated by this library in order to be executed on 8-bit microcontrollers with 2 kb of RAM. Also compatible with Scikit-learn, sklearn-porter [47] permits to transpile trained estimators to C, Java, JavaScript, GO, Ruby, and PHP. Given this variety of supported languages, it is a very complete framework as it is also compatible with a range of ML algorithms. Specifically for C language, sklearn-porter is able to port Scikit-learn models of SVM, decision trees, Random Forest (RF), and AdaBoost classifier. However, the versatility of this library hinders it to generate microcontroller-optimized code in terms of required RAM. A similar tool is m2cgen [48], which is also able to transpile Scikit-learn-trained models into a native code, e.g., Python, C, Java, Go, JavaScript, Visual Basic, etc. In this case, both the number of compatible algorithms as well as the target programming languages are even greater than those supported by sklearn-porter. A more limited tool is Weka-porter [49], which coverts decision tree classifiers generated by WEKA [59] into C, Java and JavaScript codes. EmbML [50] is a library that permits to convert models trained by WEKA or Scikit-learn into C++ source-code files that can be compiled and executed in constrained platforms. This library has been tested in Arduino and Teensy [60] boards and it supports different ML algorithms such as decision trees, SVM, and NN.

emlearn [51] is a framework that produces portable C99 code from Python libraries such as Scikit-learn or Keras. This library is compatible with generated models of a range of algorithms, e.g., RF, decision trees, Naive Bayes, or linear models, and has been tested in different platforms, namely, AVR Atmega, ESP8266 and Linux. Focused on extending the compatibility of TensorFlow models, uTensor [52] and TinyMLgen [53] are two libraries that produce C code ready to be integrated within different families of MCUs. While the former is focused on Mbed boards [61], the latter exports the trained model to a plain C array, hence ready to be used in a wider range of microcontrollers.

Finally, two proposals from the Academia are focused on optimizing the implementation of different types of NNs on ARM Cortex-M processors. CMix-NN [54] is an open-source mixed-precision library for quantized NNs deployment on MCUs. This tool supports convolutional kernels with precision of 2, 4, or 8 bits, for any of the operands. In their work, authors made use of the convolutional NN models produced by MobileNets [62]. In turn, FANN-on-MCU [55] is another open-source framework which is built upon the Fast Artificial Neural Network (FANN) library [63] aiming at running lightweight NNs on microcontrollers. Concretely, this toolkit is able to process Multi-Layer Perceptrons (MLPs) trained with FANN and produces code that, besides the mentioned ARM Cortex-M processor, can run on Parallel Ultra-Low Power Platforms (PULP) [64].

Table I summarizes the main characteristics of the frameworks reviewed in this section.

IV. Use Case: Smart Multi-Rat Device

Most of the latest efforts related to the evolution of the IoT ecosystem have focused on the development of novel communication technologies and protocols to provide end-devices with ubiquitous connectivity. Thus, it is common to find MCU-based objects with a variety of integrated RATs, which present different characteristics in terms of data-rate, latency, energy consumption, etc. As discussed previously, a convenient management of communication activities is crucial for ensuring the sustainability of the end-device; besides, the Quality of Service (QoS) requirements of data traffic should be taken into account as well. For those reasons, we propose the application of ML techniques to smartly decide which is the most adequate communication interface for transmitting a given message by considering both the status of the device and the characteristics of the message to be sent. Adopting a ML approach provides flexibility to the interface selection mechanism in the case of adding new communication interfaces or removing some of the existing ones. Thus, the multi-RAT frugal object is enriched with awareness capabilities unavailable for current IoT devices. Aiming at providing insights regarding the design and implementation processes, firstly we propose and describe a multi-RAT smart object architecture and, then, we explore the integration process of TinyML-based algorithms on real hardware in order to tackle the mentioned decision problem.

A. Architecture

The proposed architecture is shown in Fig. 4. The core element of the device is the MCU which gathers the processing unit as well as program and data memories. In this regard, we consider constrained processing platforms as those described in [8]. Focusing on storage, it is usual to find commercial devices with SD-card modules to expand their capacity, so this is not a big limitation in current development boards. As observed in the diagram, the device is also equipped with a series of embedded sensors as well as digital/analog pins for enabling the connection of external elements. To this end, a range of communication protocols are frequently employed in MCUs, e.g., Universal Asynchronous Receiver/Transmitter (UART), Serial Peripheral Interface (SPI), or Inter-Integrated Circuit (I2C). Although displays are highly consuming, they are useful to show important events or status messages. Besides, recently small e-Paper displays have arrived to the MCU market, which present insignificant energy consumption levels [65].

The availability of multiple RATs embedded in the device is also one of its key characteristics. WiFi or Bluetooth are the most usual technologies as they are very

Table I.
TinyML frameworks.

Framework	Algorithms	Compatible Platforms	Output Languages	Interoperable External Libraries	Publicly Available	Main Developer
TensorFlow Lite [35]	Neural networks	ARM Cortex-M	C++ 11	TensorFlow	Yes	Google
ELL [37]	Neural networks	ARM Cortex-M ARM Cortex-A Arduino micro:bit	C C++	CNTK Darknet ONNX	Yes	Microsoft
ARM-NN [41]	Neural networks	ARM Cortex-A ARM Mali Graphics Processors ARM Ethos Processor	C	TensorFlow Caffe ONNX	Yes	ARM
CMSIS-NN [42]	Neural networks	ARM Cortex-M	C99	TensorFlow Caffe PyTorch	Yes	ARM
STM 32Cube. AI [43]	Neural networks	STM32	C	Keras TensorFlow Lite Caffe ConvNetJs Lasagne	Yes	STMicroelectronics
AIfES [44]	Neural networks	Windows (DLL) Raspberry Pi Arduino ATMega32U4 STM32 F4 Series ARM Cortex-M4	C	TensorFlow Keras	No	Fraunhofer IMS
NanoEdge AI Studio [45]	Unsupervised learning	ARM Cortex-M	C	—	No	Cartesiam
MicroMLGen [46]	SVM RVM	Arduino ESP32 ESP8266	C	Scikit-learn	Yes	Particular developer
sklearnporter [47]	SVM Decision tree Random Forest Ada Boost Classifier k-NN Naive Bayes Neural networks	Multiple constrained & non-constrained platforms	C GO Java JavaScript PHP Ruby	Scikit-learn	Yes	Particular developer
m2cgen [48]	Linear regression Logistic regression Neural networks SVM Decision tree Random Forest LGBM Classifier	Multiple constrained & non-constrained platforms	C C# Dart Go Java JavaScript PHP PowerShell Python R Visual Basic	Scikit-learn	Yes	Particular developer

(*Continued*)

Table I.
TinyML frameworks. (*Cotinued*)

Framework	Algorithms	Compatible Platforms	Output Languages	Interoperable External Libraries	Publicly Available	Main Developer
weka-porter [49]	Decision trees	Multiple constrained & non-constrained platforms	C Java JavaScript	Weka	Yes	Particular developer
EmbML [50]	Decision trees Neural networks SVM	Arduino Teensy	C++	Scikit-learn Weka	No	Research group
emlearn [51]	Decision trees Neural networks Naive Gaussian Bayes Random forest	AVR Atmega ESP8266 Linux	C	Keras Scikit-learn	Yes	Particular developer
uTensor [52]	Neural networks	mBed boards	C++11	TensorFlow	Yes	Particular developer
TinyMLgen [53]	Neural networks	ARM Cortex-M ESP32	C	TensorFlow Lite	Yes	Particular developer
CMix-NN [54]	Neural networks	ARM Cortex-M	C	Mobilenet	Yes	Research group
FANN-on-MCU [55]	Neural networks	ARM Cortex-M PULP	C	FANN	Yes	Research group

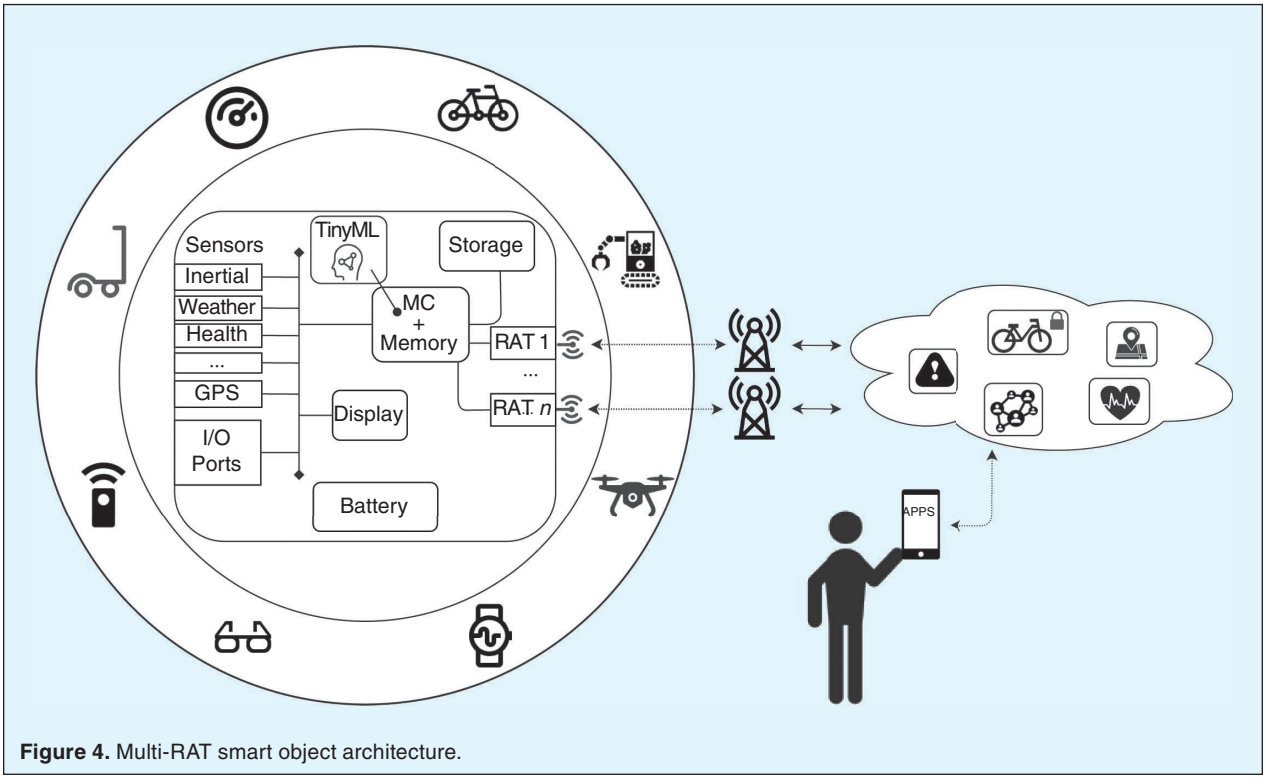

Figure 4. Multi-RAT smart object architecture.

well-known standards that can reach high data-rates. Other options based on cellular communications such as 4 G/5G may be also considered. However, all these RATs require great amount of energy to work. For that reason, LPWAN-based solutions such as LoRaWAN, Sigfox, or Narrow Band—Internet of Things (NB-IoT) [19] are currently being adopted by the IoT market as they provide very long transmission distances with highly reduced energy consumption, although with the limitation of severely limiting the transmission rate. Given the heterogeneous characteristics of the available RATs, it is crucial to precisely select the most adequate technology to guarantee both the energy efficiency of the device as well as the QoS of the transmitted packets. At this point, TinyML comes into play with the aim of smartly supporting this decision. As discussed in next Section, a number of ML techniques may be employed to this end, thanks to the algorithm adaptations provided by the afore-reviewed TinyML frameworks.

This multi-RAT smart board can be integrated within a plethora of devices and gadgets, e.g., vehicles, wearables, sensors, robots, etc., which will be enriched with intelligence and communication capabilities. These elements are connected to the cloud or other devices through the available communication interfaces in order to gain access to a range of next-generation services, which are finally accessed by means of end-user's personal devices such as smartphones or tablets.

B. Problem Statement

We consider the problem of intelligently selecting the most adequate communication interface in a multi-RAT set up when certain data is generated to be transmitted in a non-connection-oriented fashion. Concretely, given the architecture presented above, we consider an end-device producing sporadic messages and that is equipped with two communication interfaces, namely, LoRaWAN and NB-IoT. As both of them are LPWAN-based solutions, they provide long range communications with reduced power consumption, which are highly valued characteristics for moving devices. However, given these common general attributes, as shown in Table II, both present unique characteristics in terms of bandwidth, uplink/downlink latency, duty-cycle, etc. Therefore, by considering these RAT features as well as the device status and the message characteristics, it should be decided whether to transmit a message or not, and if so, the most adequate communication interface should be selected.

The set of all possible actions (**A**) is $\{a_0, a_1, a_2\}$, where a_0 represents the action of dropping the message and $\{a_1, a_2\}$ indicates sending the data using LoRaWAN or NB-IoT interfaces, respectively. Regarding the device status, several aspects are considered. The percentage of remaining battery (**B**) is one of the key factors to evaluate as the power consumption of both interfaces clearly differs. The coverage level (**C**) of each RAT $\{c_1, c_2\}$, is also taken into account in order to avoid transmissions over a lossy link. Besides, LoRaWAN makes use of license-free bands, e.g., Industrial, Scientific and Medical (ISM), so the access to this wireless medium is normally limited by international regulations. Thus, we model the availability of the LoRaWAN interface with the boolean feature **L**. We consider that NB-IoT does not present this restriction as it makes use of licensed spectrum. We also assume that there are not dedicated queues for each interface; instead, we consider a *One in—One out* message-forwarding system. This assumption is made given the memory constraints of MCUs, which prevent them of storing great amount of data in RAM. Two message features are also evaluated in the decision process. The data size (**S**), in bytes, is considered as LoRaWAN has a maximum payload length of 240 B. Finally, the urgency level of the message (**U**) is also contemplated with the aim of supporting different classes of services.

With the features identified above, the current system status is defined by the following vector $(B, \{c_1, c_2\}, L, S, U)$, which is employed for determining the most adequate action $A \in \{a_0, a_1, a_3\}$. We tackle this issue as a supervised multi-class classification problem. To this end, a dataset has been generated to train a number of algorithms that have been embedded in a real constrained device as explained in the following.

C. Dataset

A synthetic training dataset of 10,000 samples, i.e., features vectors, has been generated and each of the features composing the input vectors has received a value randomly calculated. The values for the message urgency (**U**) and the coverage of each RAT (c_1, c_2) have been generated by assigning them uniformly distributed values in the range [0%, 100%]. The values for the device's remaining battery (**B**) and the packet size (**S**), have been randomly calculated in the ranges [1%, 100%] and [50 bytes, 500 bytes], respectively, following a uniform distribution as well. Finally, the LoRaWAN interface availability (**L**), which was modeled as a boolean input, has received true/false values uniformly distributed.

Each of the 10,000 samples has been labelled with the most adequate action considering its input features. To

Table II.
Main characteristics of RATs under consideration.

RAT	Throughput	Latency	RX Sensitivity	Power Consumption
LoRaWAN	Low	High	High	Low
NB-IoT	High	Low	Medium	Medium

this end, some hard restrictions have been established in order to select a communication interface for sending a given message (Table III). With these restrictions and considering the characteristics of each RAT (Table II), we have developed an automatic labelling scheme as follows. LoRaWAN interface is preferred for small packets with low urgency, given its highly reduced power consumption. Nevertheless, it is not the best option for transmitting urgent packets because of its limited throughput and the long latency introduced. In turn, NB-IoT interface is more suitable for transporting urgent messages, given its lower latency and higher data-rate. However, its more limited sensitivity causes that its usage is not recommended with a coverage level below 10%. The same applies to the battery level, when the remaining energy is below 10%, using the NB-IoT interface is discarded due to its higher power consumption. When one of these situations happen, short urgent packets are sent via the LoRaWAN interface if it is available ($L = true$). Of course, considering the mentioned restrictions, some combinations of the input features may lead to the decision of dropping messages.

D. TinyML integration and evaluation

Given the notoriety and mass adoption of Arduino Uno, this board has been chosen as the testbench for evaluating the performance of TinyML frameworks. We consider it as a good candidate for conducting this benchmarking due to its processing and memory constraints, as it is equipped with an 8 bit processor at 16 MHz, flash memory of 32 KB, and 2 kB of Static RAM (SRAM). According to the device classification defined in [8], this platform belongs to the most constrained type of MCUs (Class 0).

Taking a look at Table I, the only publicly available frameworks that can produce code for Arduino Uno are MicroMLGen [46], sklearnporter [47], m2cgen [48], and emlearn [51]. The workflow followed by these tools is similar in all of them: firstly, a selected algorithm is trained using a Python-based ML library; thereafter, the model generated by this library is ported to C code by the corresponding TinyML translation tool; finally, the produced code is integrated within an Arduino sketch by importing it as an external library.

Regarding m2cgen, as discussed above, although this framework is highly versatile as it supports many algorithms and output languages, it does not generate optimized code in terms of memory allocation. We proved this fact by porting to C code a number of models produced by Scikit-learn, namely, RF, decision trees, and SVM. In every case, m2cgen generated non-compilable code for Arduino Uno due to unallowed memory and object manipulations.

MicroMLGen is specialized in translating Scikit-learn's SVM models to Arduino-like platforms. It is able to handle different SVM kernels such as linear, polynomial, and Radial Basis Function (rbf). SVMs are highly sensitive to the number of training samples as they determine the number of generated support vectors which have a direct impact on the required memory. Table IV summarizes the attained results in terms of accuracy and F1-score provided by Scikit-learn as well as program memory footprint and classification time in Arduino Uno. Regarding the latter, we present the average running time for 100 random input vectors as well as the confidence interval ($\alpha = 0.05$). The alpha SVM parameter has been fixed to 0.0001. It can be seen how the classifier accuracy is improved by increasing the number of training samples, no matter the kernel employed. The linear kernel is the fastest one and shows better accuracy than the rbf kernel, which presents the greatest memory footprint. Observe that both of them surpass the maximum MCU's program memory when employing 1000 and 1250 training examples due to the large number of generated support vectors. Finally, the polynomial kernel is the most accurate one and also presents the most moderate memory footprint. However, it is the slowest kernel although always under the threshold of 200 ms.

sklearnporter has been evaluated when dealing with SVMs as well. However, the produced C code from the Scikit-learn's models is not optimized for constrained devices as mentioned previously, hence it provoked an unstable behaviour of the Arduino unit. Given the inconsistent performance obtained, we could not collect reliable outcomes to present here.

As discussed above, the family of NN-based algorithms is the one that has received greater attention from the TinyML community. In this case, we have evaluate the performance of the MLP implementation provided by the emlearn framework. We have employed Scikit-learn for producing trained networks with different number of hidden layers and neurons per layer, using the sigmoid activation function. As the number of

Table III.
Feature restrictions for generating the dataset.

RAT	Battery Level (B) (%)	Coverage (C) (%)	LoRaWAN Availability (L)	Packet Size (S) (bytes)	Urgency (U) (%)
LoRaWAN	>5	>5	True	<240	<50
NB-IoT	>10	>10	—	—	—

training samples does not impact on the weight of the produced model, the 80% of the dataset has been employed for training and the other 20% for test. The obtained outcomes are shown in Table V. It can be seen that in the MLP case the limiting factor is the Arduino Uno's SRAM, as the network's weights and biases are stored there. Thus, when the number of neurons per layer is 10, the maximum number of hidden layers for not consuming all the unit's SRAM is 3. Regarding the algorithm's performance, a trade-off between accuracy and network complexity is evidenced. However, employing one hidden layer with 4 neurons permits an accuracy of 0.97, which is the best figure for this simple configuration. It is clear the direct relationship between the overall number of neurons in the network and the processing time for each classification operation, which should be also considered when selecting the most adequate system configuration. However, observe that the times employed by the MLP classifier are notably shorter than those needed by the MicroMLGen's SVMs. The achieved accuracy is also better for the MLP algorithm.

The adaptation of other well-known ML mechanisms such as decision trees and RF has been evaluated as well. To this end, we have employed the sklearn-porter and emlearn frameworks to port the models generated by Scikit-learn. In this case, sklearnporter produced suitable code for Arduino Uno, given that decision trees and RF are lighter than SVMs. A similar performance analysis to those presented previously is shown in Table VI. The dataset has been split as in the case of MLP evaluation, i.e., 80% for training and 20% for evaluating the produced classifier. Observe how these algorithms capture the logic employed for labelling our synthetic dataset, which is evidenced by the good accuracy obtained. By increasing the depth of the decision tree, we improve the classifier performance up to a perfect accuracy of 1 with decision trees of 6 levels. Regarding the RF classifier, we have evaluated one model consisting of 10 estimators (trees) and we also obtain a perfect accuracy although the memory footprint and processing time are notably greater for this scheme. Comparing both frameworks, sklearnporter provides faster and lighter decision trees in comparison with emlearn. However, the opposite behaviour is obtained for the RF algorithm.

In the light of the obtained results, the performance of decision trees and RF are notably better than that of SVM and MLP for solving the intelligent communication interface selection issue under consideration. They achieve greater accuracy with reduced memory footprint and computation delay. Therefore, for the specific case of smart multi-RAT configuration management, this family of algorithms would be the most adequate to be integrated within the MCU.

To sum up, it can be concluded that great advances have been made so far for integrating ML algorithms within frugal devices. These achievements were unimaginable few time ago and they pave the way for the arrival of smart applications and services that will not require a continuous processing support from the cloud. However, we are just in the inception of this movement and more efforts are needed for the integral development of the TinyML ecosystem.

Table IV.
SVM generated by MicroMLGen.

Samples/Kernel	Linear	Polynomial	rbf
250	Accuracy = 0.7816 F1 = 0.7119 Memory = 10678 B Time = 11.1±0.2 ms	Accuracy = 0.7992 F1 = 0.7643 Memory = 10264 B Time = 73.7±0.3 ms	Accuracy = 0.7355 F1 = 0.6298 Memory = 12172 B Time = 34.7±0.03 ms
500	Accuracy = 0.7871 F1 = 0.7161 Memory = 17782 B Time = 22.3±0.1 ms	Accuracy = 0.8552 F1 = 0.8074 Memory = 15636 B Time = 73.2±0.7 ms	Accuracy = 0.7472 F1 = 0.6661 Memory = 20574 B Time = 72±0.047 ms
750	Accuracy = 0.7922 F1 = 0.723 Memory = 25298 B Time = 34.1±0.2 ms	Accuracy = 0.8561 F1 = 0.8347 Memory = 20642 B Time = 107.2±0.3 ms	Accuracy = 0.75 F1 = 0.6815 Memory = 26644 B Time = 98.9±0.07 ms
1000	Accuracy = 0.7921 F1 = 0.7232 Memory = 30156 B Time = 41.8±0.2 ms	Accuracy = 0.8688 F1 = 0.8443 Memory = 25530 B Time = 140.1±0.2 ms	Accuracy = 0.7757 F1 = 0.7402 Memory = 35184 B (out-of-range)
1250	Accuracy = 0.7939 F1 = 0.725 Memory = 37930 B (out-of-range)	Accuracy = 0.8757 F1 = 0.845 Memory = 31968 B Time = 183.6±0.4 ms	Accuracy = 0.7818 F1 = 0.7484 Memory = 44058 B (out-of-range)

Table V.
MLP generated by emlearn.

Hidden layers/ Neurons	2	4	6	8	10
1	Accuracy = 0.8835 F1 = 0.8393 Memory = 6750 B SRAM = 392 B Time = 1.9+0.006 ms	Accuracy = 0.9735 F1 = 0.9737 Memory = 6830 B SRAM = 472 B Time = 2.7+0.009 ms	Accuracy = 0.961 F1 = 0.9605 Memory = 6910 B SRAM = 552 B Time = 3.5+0.01 ms	Accuracy = 0.9705 F1 = 0.9701 Memory = 6990 B SRAM = 648 B Time = 4.3+0.01 ms	Accuracy = 0.968 F1 = 0.9679 Memory = 7070 B SRAM = 744 B Time = 4.9+0.03 ms
2	Accuracy = 0.8045 F1 = 0.7319 Memory = 6790 B SRAM = 430 B Time = 2.4+0.005 ms	Accuracy = 0.907 F1 = 0.878 Memory = 6926 B SRAM = 566 B Time = 3.8+0.009 ms	Accuracy = 0.9544 F1 = 0.9541 Memory = 7094 B SRAM = 734 B Time = 5.5+0.01 ms	Accuracy = 0.9742 F1 = 0.9742 Memory = 7294 B SRAM = 950 B Time = 7.3+0.02 ms	Accuracy = 0.9691 F1 = 0.9691 Memory = 7526 B SRAM = 1198 B Time = 9.2+0.02 ms
3	Accuracy = 0.8903 F1 = 0.8431 Memory = 6826 B SRAM = 468 B Time = 2.9+0.01 ms	Accuracy = 0.9746 F1 = 0.9745 Memory = 7018 B SRAM = 660 B Time = 5+0.01 ms	Accuracy = 0.9543 F1 = 0.9532 Memory = 7274 B SRAM = 916 B Time = 7.4+0.02 ms	Accuracy = 0.9705 F1 = 0.9709 Memory = 7594 B SRAM = 1252 B Time = 10.1+0.02 ms	Accuracy = 0.9795 F1 = 0.9797 Memory = 7978 B SRAM = 1652 B Time = 13.2+0.03 ms
4	Accuracy = 0.888 F1 = 0.8392 Memory = 6864 B SRAM = 506 B Time = 3.6+0.009 ms	Accuracy = 0.922 F1 = 0.9242 Memory = 7112 B SRAM = 754 B Time = 6.2+0.03 ms	Accuracy = 0.924 F1 = 0.9238 Memory = 7456 B SRAM = 1098 B Time = 9.3+0.03 ms	Accuracy = 0.945 F1 = 0.9453 Memory = 7896 B SRAM = 1154 B Time = 13+0.06 ms	Accuracy = 0.944 F1 = 0.9422 Memory = 8432 B SRAM = 2106 B (out-of-range)
5	Accuracy = 0.715 F1 = 0.6108 Memory = 6902 B SRAM = 544 B Time = 3.9+0.03 ms	Accuracy = 0.882 F1 = 0.8385 Memory = 7206 B SRAM = 848 B Time = 7.3+0.03 ms	Accuracy = 0.9285 F1 = 0.9269 Memory = 7638 B SRAM = 1280 B Time = 11.1+0.07 ms	Accuracy = 0.931 F1 = 0.9297 Memory = 8198 B SRAM = 1856 B Time = 15.8+0.08 ms	Accuracy = 0.9245 F1 = 0.9183 Memory = 8886 B SRAM = 2560 B (out-of-range)

Table VI.
Decision trees and RF generated by sklearnporter and emlearn.

Library/ Algorithm	Decision Tree Depth = 3	Decision Tree Depth = 4	Decision Tree Depth = 5	Decision Tree Depth = 6	RF Estimators = 10
sklearnporter	Accuracy = 0.9155 F1 = 0.8886 Memory = 3610 B Time = 40.56±0.63 μs	Accuracy = 0.9155 F1 = 0.8886 Memory = 3670 B Time = 45.28±1.55 μs	Accuracy = 0.9935 F1 = 0.9935 Memory = 3730 B Time = 48.32±9.48 μs	Accuracy = 1 F1 = 1 Memory = 3770 B Time = 48.75±5.6 μs	Accuracy = 1 F1 = 1 Memory = 27814 B Time = 252.24±7 μs
emlearn	Accuracy = 0.9155 F1 = 0.8886 Memory = 3840 B Time = 103.28±3.87 μs	Accuracy = 0.9155 F1 = 0.8886 Memory = 3904 B Time = 103.58±7.18 μs	Accuracy = 0.9935 F1 = 0.9935 Memory = 3952 B Time = 103.96±83.82 μs	Accuracy = 1 F1 = 1 Memory = 40016 B Time = 104.08±3.87 μs	Accuracy = 1 F1 = 1 Memory = 13160 B Time = 193.6±6.06 μs

V. Conclusions

The integration of ML within IoT devices will provide them with a flexibility and processing capabilities never seen before. The TinyML paradigm proposes to adapt advanced ML techniques to the constraints of MCUs in order to convert them in real smart objects. In this work, a comprehensive review and discussion of this novel and vibrating ecosystem have been presented. The related challenges and opportunities have been identified as well as the potential services that will be enabled by these intelligent small elements. The TinyML frameworks available in the market that currently permit the integration of complex ML mechanisms within MCUs have been also reviewed and their applicability has been discussed. Finally, a realistic use case has been explored. Concretely, a multi-RAT smart object architecture has been proposed and the ML integration process has been explained. A number of TinyML frameworks have been employed for solving the problem of selecting the most adequate communication interface considering both the status of the device and the characteristics of the data to be sent. We have demonstrated that several classification algorithms can be integrated within an Arduino Uno board for tackling this issue, namely, SVM, MLP, decision trees, and RF. From the attained outcomes, we can conclude that decisions trees and RF present the best performance in terms of accuracy, memory footprint, and classification speed. Other algorithms also showed good results in terms of classification accuracy but their implementation should be carefully addressed given the memory restrictions of the considered MCU. As future work, we plan to consider other constrained devices in order to benchmark additional TinyML frameworks. Besides, other relevant aspects such as decision safety and model correctness in sensitive fields of application, e.g., eHealth, will be considered.

Acknowledgment

This work has been supported by the European Commission, under the projects IoTCrawler (Grant No. 779852), CyberSec4Europe (Grant No. 830929) and Fed4IoT (Grant No. 814918), and by the Spanish Ministry of Science, Innovation and Universities, under the project PERSEIDES (Grant No. TIN2017-86885-R with ERDF funds).

Ramon Sanchez-Iborra received the B.Sc. degree in telecommunication engineering and the M.Sc. and Ph.D. degrees in information and communication technologies from the Technical University of Cartagena, in 2007, 2013, and 2016, respectively. He is currently an Assistant Professor and a Researcher with the Information and Communications Engineering Department, University of Murcia. His main research interests are evaluation of QoE in multimedia services, management of wireless mobile networks, green networking techniques, and IoT/M2M architectures.

Antonio Skarmeta received the B.S. degree (Hons.) from the University of Murcia, Spain, the M.S. degree from the University of Granada, and the Ph.D. degree from the University of Murcia, all in computer science. He has been a Full Professor with the University of Murcia, since 2009. He has been part of many EU FP projects and even coordinated some of them. He has published more than 200 international articles. His main interests include the integration of security services, identity, the IoT, 5G, and smart cities.

References

[1] L. Columbus, "10 Charts that will change your perspective of big data's growth," Tech. Rep., 2018. [Online]. Available: https://softwarestrategiesblog.com/2018/06/02/10-charts-that-will-change-your-perspective-of-big-datas-growth/

[2] Y. Sun, M. Peng, Y. Zhou, Y. Huang, and S. Mao, "Application of machine learning in wireless networks: Key techniques and open issues," *IEEE Commun. Surveys Tut.*, vol. 21, no. 4, pp. 3072–3108, 2019. doi: 10.1109/COMST.2019.2924243. [Online]. Available: https://ieeexplore.ieee.org/document/8743390/

[3] T. Baltrusaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 423–443, Feb. 2019. doi: 10.1109/TPAMI.2018.2798607. [Online]. Available: https://ieeexplore.ieee.org/document/8269806/

[4] Q. Chen et al., "A survey on an emerging area: Deep learning for smart city data," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 3, no. 5, pp. 392–410, Oct. 2019. doi: 10.1109/TETCI.2019.2907718. [Online]. Available: https://ieeexplore.ieee.org/document/8704334/

[5] G. Li, R. Gomez, K. Nakamura, and B. He, "Human-centered reinforcement learning: A survey," *IEEE Trans. Human-Mach. Syst.*, vol. 49, no. 4, pp. 337–349, Aug. 2019. doi: 10.1109/THMS.2019.2912447. [Online]. Available: https://ieeexplore.ieee.org/document/8708686/

[6] T. Luo et al., "DaDianNao: A neural network supercomputer," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 73–88, Jan. 2017. doi: 10.1109/TC.2016.2574353. [Online]. Available: http://ieeexplore.ieee.org/document/7480791/

[7] B. Garbinato, R. Guerraoui, J. Hulaas, M. Monod, and J. H. Spring, "Pervasive computing with frugal objects," in *Proc. IEEE 21st Int. Conf. Advanced Information Networking and Applications Workshops (AINAW'07)*, 2007, pp. 13–18. [Online]. Available: http://ieeexplore.ieee.org/document/4224076/

[8] C. Bormann, M. Ersue, and A. Keranen, *Terminology for Constrained-Node Networks*, IETF RFC 7228, 2014.

[9] M. S. Hadj Sassi, F. G. Jedidi, and L. C. Fourati, "A new architecture for cognitive internet of things and big data," *Proc. Comput. Sci.*, vol. 159, pp. 534–543, 2019. doi: 10.1016/j.procs.2019.09.208. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050919313924

[10] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges," *Future Gener. Comput. Syst.*, vol. 78, pp. 680–698, Jan. 2018. doi: 10.1016/j.future.2016.11.009. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X16305635

[11] B. Martinez, M. Monton, I. Vilajosana, and J. D. Prades, "The power of models: modeling power consumption for IoT devices," *IEEE Sensors J.*, vol. 15, no. 10, pp. 5777–5789, Oct. 2015. doi: 10.1109/JSEN.2015.2445094. [Online]. Available: http://ieeexplore.ieee.org/document/7122861/

[12] TinyML. 2020. [Online]. Available: https://www.tinyml.org/

[13] C. MacGillivray and M. Torchia, "Internet of Things: Spending trends and outlook," Tech. Rep., 2019. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US45161419

[14] H. Doyu, "TinyML as a Service and the challenges of machine learning at the edge," Ericsson, Tech. Rep., 2019. [Online]. Available: https://www.ericsson.com/en/blog/2019/12/tinyml-as-a-service

[15] H.-T. Pham, M.-A. Nguyen, and C.-C. Sun, "AIoT solution survey and comparison in machine learning on low-cost microcontroller," in *Proc. IEEE Int. Symp. Intelligent Signal Processing and Communication Systems (ISPACS)*, Dec. 2019, pp. 1–2. [Online]. Available: https://ieeexplore.ieee.org/document/8986357/
[16] V. Falbo, T. Apicella, D. Aurioso, L. Danese, F. Bellotti, R. Berta, and A. De Gloria, "Analyzing machine learning on mainstream microcontrollers," in *Proc. Int. Conf. Applications Electronics Pervading Industry, Environment and Society*, 2020, pp. 103–108. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-37277-4_12
[17] R. Costa, "The internet of moving things [industry view]," *IEEE Technol. Soc. Mag.*, vol. 37, no. 1, pp. 13–14, Mar. 2018. doi: 10.1109/MTS.2018.2795092. [Online]. Available: http://ieeexplore.ieee.org/document/8307131/
[18] M. Quigley and M. Burke, "Low-cost Internet of Things digital technology adoption in SMEs," *Int. J. Manage. Pract.*, vol. 6, no. 2, p. 153, 2013. doi: 10.1504/IJMP.2013.055828. [Online]. Available: http://www.inderscience.com/link.php?id=55828
[19] R. Sanchez-Iborra and M.-D. Cano, "State of the art in LP-WAN solutions for industrial IoT services," *Sensors*, vol. 16, no. 5, p. 708, 2016. doi: 10.3390/s16050708.
[20] W. Niu et al., "PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning," in *Proc. 25th Int. Conf. Architectural Support Programming Languages and Operating Systems*, New York, Mar. 2020, pp. 907–922. doi: 10.1145/3373376.3378534. [Online]. Available: https://dl.acm.org/doi/10.1145/3373376.3378534
[21] C. R. Banbury et al., "Benchmarking TinyML systems: Challenges and direction," in *Proc. 1st Int. Workshop Benchmarking Machine Learning Workloads Emerging Hardware - 3rd Conf. Machine Learning and Systems (MLSys)*, 2020. [Online]. Available: https://arxiv.org/pdf/2003.04821.pdf
[22] E. Liberis and N. D. Lane, Neural networks on microcontrollers: saving memory at inference via operator reordering. 2019.
[23] P. Randhawa, V. Shanthagiri, and A. Kumar, "A review on applied machine learning in wearable technology and its applications," in *Proc. IEEE Int. Conf. Intelligent Sustainable Systems (ICISS)*, Dec. 2017, pp. 347–354. [Online]. Available: https://ieeexplore.ieee.org/document/8389428/
[24] E. Ahmed, I. Yaqoob, A. Gani, M. Imran, and M. Guizani, "Internet-of-things-based smart environments: state of the art, taxonomy, and open research challenges," *IEEE Wireless Commun.*, vol. 23, no. 5, pp. 10–16, Oct. 2016. doi: 10.1109/MWC.2016.7721736. [Online]. Available: http://ieeexplore.ieee.org/document/7721736/
[25] A. Rizzardi, D. Miorandi, S. Sicari, C. Cappiello, and A. Coen-Porisini, "Networked smart objects: Moving data processing closer to the source," in *(Lecture Notes Institute Computer Sciences, Social Informatics and Telecommunications Engineering)*, 2016, pp. 28–35. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-47075-7_4
[26] A. Roy, A. M. Zalzala, and A. Kumar, "Disruption of things: A model to facilitate adoption of IoT-based innovations by the urban poor," *Proc. Eng.*, vol. 159, pp. 199–209, 2016. doi: 10.1016/j.proeng.2016.08.159. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877705816323074
[27] J. Santa and P. J. Fernández, "Seamless IPv6 connectivity for two-wheelers," *Pervas. Mobile Comput.*, vol. 42, pp. 526–541, Dec. 2017. doi: 10.1016/j.pmcj.2017.09.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574119217300652
[28] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the Internet of Things and industry 4.0," *IEEE Ind. Electron. Mag.*, vol. 11, no. 1, pp. 17–27, Mar. 2017. doi: 10.1109/MIE.2017.2649104. [Online]. Available: http://ieeexplore.ieee.org/document/7883994/
[29] S. Deshmukh and R. Shah, "Computation offloading frameworks in mobile cloud computing: A survey," in *Proc. IEEE Int. Conf. Current Trends Advanced Computing (ICCTAC)*, Mar. 2016, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/7567332/
[30] P. Jayaraman, A. Yavari, D. Georgakopoulos, A. Morshed, and A. Zaslavsky, "Internet of Things platform for smart farming: Experiences and lessons learnt," *Sensors*, vol. 16, no. 11, p. 1884, Nov. 2016. doi: 10.3390/s16111884. [Online]. Available: http://www.mdpi.com/1424-8220/16/11/1884
[31] P. Warden and D. Situnayake, *Tinyml: Machine Learning with tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly UK, 2019.
[32] TensorFlow. 2020. [Online]. Available: https://www.tensorflow.org/
[33] Scikit-learn. 2020. [Online]. Available: https://scikit-learn.org/
[34] PyTorch. 2020. [Online]. Available: https://pytorch.org/
[35] TensorFlow Lite. 2020. [Online]. Available: https://www.tensorflow.org/lite/
[36] TinyGo. 2020. [Online]. Available: https://tinygo.org/
[37] Embedded Learning Library. 2020. [Online]. Available: https://microsoft.github.io/ELL/
[38] The Microsoft Cognitive Toolkit. 2020. [Online]. Available: https://docs.microsoft.com/en-us/cognitive-toolkit/
[39] Darknet: Convolutional Neural Networks. 2020. [Online]. Available: https://github.com/pjreddie/darknet
[40] Open Neural Network Exchange. 2020. [Online]. Available: https://onnx.ai/
[41] ARM-NN. 2020. [Online]. Available: https://github.com/ARM-software/armnn
[42] CMSIS-NN. 2020. [Online]. Available: https://arm-software.github.io/CMSIS_5/NN/html/
[43] STM32Cube.AI 2020, [Online]. Available: https://www.st.com/en/embedded-software/x-cube-ai.html
[44] AIfES. 2020. [Online]. Available: https://www.ims.fraunhofer.de/en/Business_Units_and_Core_Competencies/Electronic-Assistance-Systems/Technologies/Artificial-Intelligence-for-Embedded-Systems-AIfES.html
[45] NanoEdge AI Studio. 2020. [Online]. Available: https://cartesiam.ai/
[46] MicroML. 2020. [Online]. Available: https://github.com/eloquentarduino/micromlgen
[47] D. Morawiec, sklearn-porter, 2020. [Online]. Available: https://github.com/nok/sklearn-porter
[48] m2cgen. 2020. [Online]. Available: https://github.com/BayesWitnesses/m2cgen
[49] Weka-porter. 2020. [Online]. Available: https://github.com/nok/weka-porter
[50] L. Tsutsui da Silva, V. M. A. Souza, and G. E. A. P. A. Batista, "EmbML Tool: Supporting the use of supervised learning algorithms in low-cost embedded systems," in *Proc. IEEE 31st Int. Conf. Tools Artificial Intelligence (ICTAI)*, Nov. 2019, pp. 1633–1637. [Online]. Available: https://ieeexplore.ieee.org/document/8995408/
[51] J. Nordby, "emlearn: Machine Learning inference engine for microcontrollers and embedded devices," Mar. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2589394
[52] uTensor. 2020. [Online]. Available: https://github.com/uTensor/uTensor
[53] TinyMLgen. 2020. [Online]. Available: https://github.com/eloquentarduino/tinymlgen
[54] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "CMix-NN: Mixed low-precision CNN library for memory-constrained edge devices," *IEEE Trans. Circuits Syst. II, Express Briefs*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/9049084/
[55] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "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.*, to be published. doi: 10.1109/JIOT.2020.2976702. [Online]. Available: https://ieeexplore.ieee.org/document/9016202/
[56] Linaro. 2020. [Online]. Available: https://www.linaro.org/
[57] Caffe. 2020. [Online]. Available: https://caffe.berkeleyvision.org/
[58] Keras. 2020. [Online]. Available: https://keras.io/
[59] G. Holmes, A. Donkin, and I. Witten, "WEKA: a machine learning workbench," in *Proc. IEEE ANZIIS '94 - Australian New Zealnd Intelligent Information Systems Conf.*, 1994, pp. 357–361. [Online]. Available: http://ieeexplore.ieee.org/document/396988/
[60] Teensy. 2020. [Online]. Available: https://www.pjrc.com/teensy/
[61] Mbed platforms. 2020. [Online]. Available: https://os.mbed.com/platforms/
[62] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," Apr. 2017. [Online]. Available: http://arxiv.org/abs/1704.04861
[63] Fast Artificial Neural Network Library (FANN). 2020. [Online]. Available: http://leenissen.dk/fann/wp/
[64] D. Rossi et al., "PULP: A parallel ultra low power platform for next generation IoT applications," in *Proc. IEEE Hot Chips 27 Symp. (HCS)*, Aug. 2015, pp. 1–39. [Online]. Available: http://ieeexplore.ieee.org/document/7477325/
[65] Y Wang, Z Luo, G Qiu, and G Zhou, "Design of Smart Watch system based on E-paper," in *Proc. 10th IEEE Int. Conf. Nano/Micro Engineered and Molecular Systems*, Apr. 2015, pp. 327–330. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7147437