IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 29 June 2022, accepted 9 September 2022, date of publication 15 September 2022, date of current version 22 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3206782

TOPICAL REVIEW

Embedded Machine Learning Using Microcontrollers in Wearable and Ambulatory Systems for Health and Care Applications: A Review

MAHA S. DIAB, (Graduate Student Member, IEEE), AND ESTHER RODRIGUEZ-VILLEGAS

Wearable Technologies Laboratory, Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ London, U.K.

Corresponding author: Maha S. Diab (m.diab21@imperial.ac.uk)

This work was supported in part by the European Research Council (ERC) for the NOSUDEP Project under Grant 724334; and in part by the Engineering and Physical Sciences Research Council (EPSRC), U.K., under Grant EP/P009794/1.

ABSTRACT The use of machine learning in medical and assistive applications is receiving significant attention thanks to the unique potential it offers to solve complex healthcare problems for which no other solutions had been found. Particularly promising in this field is the combination of machine learning with novel wearable devices. Machine learning models, however, suffer from being computationally demanding, which typically has resulted on the acquired data having to be transmitted to remote cloud servers for inference. This is not ideal from the system's requirements point of view. Recently, efforts to replace the cloud servers with an alternative inference device closer to the sensing platform, has given rise to a new area of research Tiny Machine Learning (TinyML). In this work, we investigate the different challenges and specifications trade-offs associated to existing hardware options, as well as recently developed software tools, when trying to use microcontroller units (MCUs) as inference devices for health and care applications. The paper also reviews existing wearable systems incorporating MCUs for monitoring, and management, in the context of different health and care intended uses. Overall, this work addresses the gap in literature targeting the use of MCUs as edge inference devices for healthcare wearables. Thus, can be used as a kick-start for embedding machine learning models on MCUs, focusing on healthcare wearables.

INDEX TERMS Edge ML, embedded machine learning, healthcare, microcontroller, TinyML, wearable.

I. INTRODUCTION

Wearable devices have witnessed a notable increase in their use the past years. Recent statistics show the worldwide shipment of wearable units reached 533.6 million in 2021, a 20% growth from previous year [1]. These devices include smart watches, fitness trackers, hearables, and others. This users' interest in wearables trend together with advances in machine learning have led to researchers starting to look into combining the benefits of the two- i.e., usability and high ability to extract information of interest- within the context of health and care applications.

Designing wearable devices for these applications, however, is not a straightforward process. As technological advances provide advantages, they also present different challenges along the way. These challenges are summarized in Fig. 1. Although there is an evident increase in the use of wearables, this reflects the acceptance of only part of the population which voluntarily uses them. But, when wearables are intended to be used for monitoring health status and providing care and assistance, users might not show the same enthusiasm, especially the elderly or disabled. Therefore, user acceptability is a major challenge that must be taken into

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Xiang.

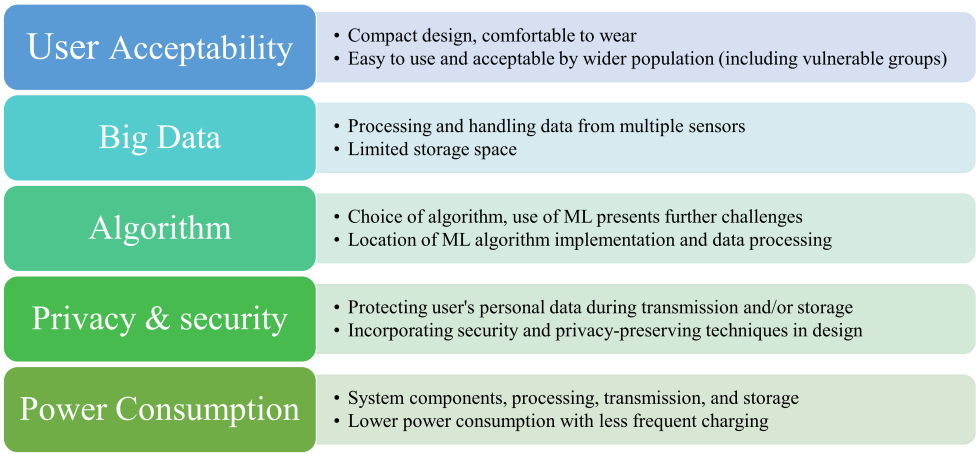

FIGURE 1. Design considerations and challenges of wearable health and care systems.

account during the system design. Devices should, for example, be compact, easy to use, comfortable to wear and have minimum maintenance [2]. Moreover, continuous long-term use of wearable devices can result in large amounts of data. This data will need to be transmitted, stored, and processed. All of this comes with its own set of constraints, in terms for example of data storage, and power consumption.

Another practical aspect to take into account in the design of a wearable device is that this will be collecting physiological data, and, in some cases, this might be linked to also personal identifiable information. This data might need to be transmitted for processing and/or storage, which can make it susceptible to security attacks during transmission and, in some cases, it might pose a threat to the user's privacy if the information extracted can somehow be linked to personal information. Hence, securing the user's private information, via for example encryption, differential privacy, and others, is one of the challenges that require careful attention in the system design [3].

Power consumption is another important challenge that needs to be taken into consideration during the design process since power will have a knock down effect on aspects that can, not only impact performance, but also acceptability. Power consumption cannot be looked up in isolation, and neither can the many different design factors that will have an effect on it. These range from the different components of the system to the chosen algorithm and its distribution in different system's parts, or the decisions on data management/transmission/storage. None of these should be optimized individually, since in the context of severely constrained wearables, optimization in one aspect is most likely going to lead to critical usability bottlenecks. When translating this to usability, the challenge is to implement a system with an overall power consumption that leads to an acceptable form factor and reducing the charging cycles to a point that this will not have a significant effect on users' adherence, where the meaning of significant is determined by the specific intended use.

Lastly, an additional challenge in the design of ML based healthcare wearables is the choice of ML algorithm and its actual physical implementation. The algorithm will handle the collected big data, dictate the system's performance (which in the context of healthcare will be linked to both safety and intended use), and its implementation location will have an effect in all the above-mentioned challenges: power consumption, privacy and security, and indirectly usability. Therefore, a closer look on the application of machine learning algorithms in healthcare, and its integration in wearable devices is a must.

A. MACHINE LEARNING IN HEALTHCARE

The application of machine learning algorithms in various medical scenarios is a relatively new, but exponentially growing, field of research. Machine learning (ML), including deep learning (DL), algorithms have the ability to handle and interpret large amount of data, identifying patterns and trends not usually noticeable by physicians. This has been demonstrated in a variety of medical areas; namely, oncology [4], [5], [6], [7], [8], pulmonology [9], [10], [11], cardiovascular diseases [12], [13], [14], [15], neurology [16], [17], [18], orthopaedics [19], [20], histopathology [21], [22], and others. Different types of medical data that have been successfully used in the context of ML for healthcare applications include [23]: (1) medical images, such as X-rays, MRI, CT scans; (2) physiological signals, such as electrocardiogram (ECG), electroencephalogram (EEG), electromyography (EMG), and Photoplethysmogram (PPG); (3) omics data such as genomics, transcriptomics, and others [24]; and (4) electronic health records. These various medical data are used in healthcare applications including diagnosis, prognosis, monitoring of disease and physical fitness, as well as in body-machine interfaces for ambient assisted living and patient rehabilitation.

The incorporation of machine learning (ML) into healthcare applications has given rise to three different architectural levels of algorithm computation: cloud, fog, and edge [25].

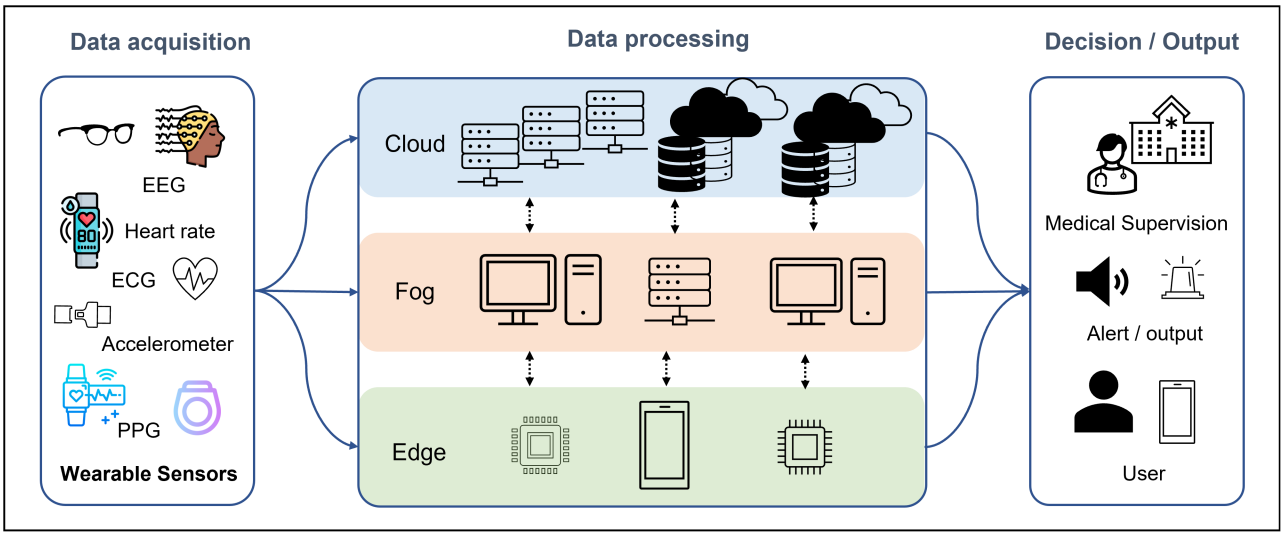

FIGURE 2. Wearable healthcare system framework demonstrating the three architectural levels for computation of machine learning algorithms: edge computation runs the ML algorithm close to sensor level; fog computation receives sensor data for processing on local network; cloud computation receives sensor data for processing at remote servers.

This is shown in Fig.2. The data collected from different wearable sensors are sent for processing at one of the three computational levels. First, computation at the cloud level depends on cloud services. This is generally needed as a result of high computational and data storage demands. Sending the data to remote servers, although advantageous in many aspects, also poses some potential drawbacks, such as the need of network connectivity, privacy and security increased risks, and, sometimes, power consumption limiting usability. Fog computation, similarly, to cloud computation, requires sending medical data for processing. However, the server is a local one, such as a PC on the local network. Edge computing, of ML algorithms, on the other hand, performs computation close to the sensor level using the collected data. This eliminates the need for transmission of raw data, increasing privacy and security, and allowing interpretability directly in the device's front end [26]. But it is restricted by the edge device computational capability. Ultimately, the choice of architecture depends on the application, device at hand, and the overall system features and specifications. In order to utilize the advantages of edge computation, several research works addressed the implementation of machine learning algorithms on edge, using different edge devices, algorithms, and deployment tools, aiming to achieve as close as possible performance to what would be obtained with the algorithms running on a PC. This has been referred to as edge inference, Edge ML, and TinyML in the case of using microcontrollers [27].

Different hardware options are available for Edge ML implementation, namely graphic processing units (GPUs), field programmable gate arrays (FPGAs), application specific integrated circuits (ASICs), and microcontroller units (MCUs) [28], [29], [30], [31], [32]. GPUs provide high computational ability by utilizing parallel processing. This enables fast inference but at the expense of high-power consumption [30]. FPGAs consume less power and can fit ML algorithms with an acceptable performance and programmability on hardware, but are less area and energy efficient than ASICs [33] (and this, in turn can affect the size of the overall healthcare device). ASICs, however, have huge development costs, which are not always affordable when taking into account the commercial route to market of the devices. MCUs, on the other hand, although also requiring higher power consumption than ASICs, can be more area and cost efficient than FPGAs. In addition, they are also easier to reprogram requiring embedded C rather than specialist knowledge in VHDL; this allowing for quick updates. The performance of ML algorithms on MCUs is dependent on multiple variables, related to both the MCUs specification, and the specific application. In general, weighting the pros and cons of the hardware options, MCUs are the most appealing option for use in wearable devices, because of their low power consumption, latency, size, flexibility, and cost. Therefore, this review explores the use of MCUs as edge device for inferences in wearable devices for healthcare applications.

An example design for a future health and care wearable device using TinyML technology is presented in Fig. 3. Data collected from wearable sensors and related public datasets are used for building the ML model. The process of building the ML model follows the conventional process requiring pre-processing, data splitting for training followed by evaluation of the chosen model. Development of the model is then achieved through error calculations on validation set and changing of hyperparameters as needed. Once a final optimum model is reached, the actual work related to TinyML begins. The model is optimized through different compression techniques, and then converted to an MCU compatible format. The process of model optimization through different

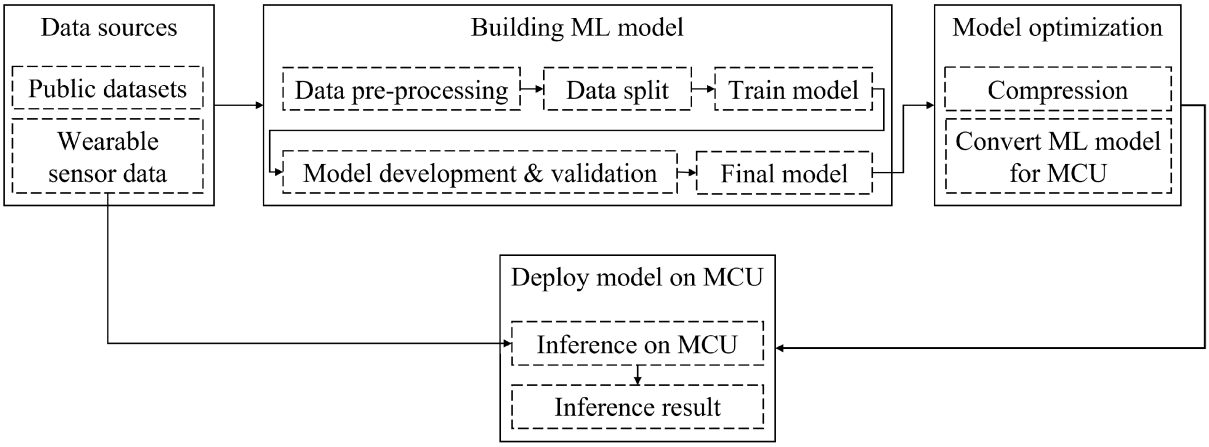

FIGURE 3. Design flow of wearable health and care device as a TinyML technology.

software tools and frameworks is later covered within the paper. Once the model is converted, it is deployed on the microcontroller for inference using direct data from the wearable sensor to output the inference result.

B. RESEARCH GAP AND PAPER CONTRIBUTION

A gap was identified within the literature in relation to the use of TinyML technology in healthcare systems. Although there have been other reviews targeting in one way or another the use of ML/DL algorithms, a number of those reviews focused on the specific application of these algorithms categorizing their use based on: type of medical data used [23], medical application [34], [35], diagnosis from medical images [36], and from the privacy and security point of view [37]. Other reviews targeted the edge implementation of ML/DL algorithms reviewing the different edge hardware [26], [28], [29], [32], [33]. Some focused on a specific platform such as FPGAs [28], [38], or ASICs [28], while others focused on the application [39], [40]. However, even within the reviews of Edge ML in biomedical applications, the use of MCUs as the edge device was hardly addressed, and the focus was instead on FPGAs, ASICs, mobile phones, or microprocessors such as the Raspberry Pi. As for reviews targeting MCU implementations, they provided a general overview of the TinyML field without a specific application [41]. To the author's knowledge, there has not been any work discussing the full pipeline for TinyML technology applied on wearable health and care devices, despite the fact that TinyML does indeed provide several advantages over cloud or fog computing which could be capitalized on, depending on the specific application. This work addresses this gap.

This work fills the gap found in literature in terms of review and analysis of the use of TinyML in wearable healthcare systems, covering also the existing tools needed for implementation of TinyML based systems, both hardware and software. This includes the different types of MCUs used thus far analyzing their different specifications and design limitations; and the software tools and frameworks used in optimization and deployment of ML model on MCU, also analyzing their use and compatibility with specific hardware. The paper also covers how these hardware and software tools have been used in the design of the reported healthcare aimed TinyML systems. Overall, this work can serve as a guide for researchers and system designers when trying to make system architecture decisions in the initial phases of the design process.

C. RESEARCH METHODOLOGY AND PAPER ORGANIZATION

Peer-reviewed articles written in English reporting TinyML wearable systems for healthcare applications were included in this review. The articles were searched for using Google Scholar and IEEExplore. Further articles and data repositories were obtained from the citations of found articles. Search terms used included: edge machine learning, embedded machine learning, TinyML, inference on microcontroller, machine learning on microcontrollers combined with health and care (healthcare) applications, wearable devices/systems. 1240 records were found in this search. Article titles and abstracts were used for initial screening to identify articles presenting health and care systems with embedded ML algorithms. This led to the exclusion of 1207 records. Selected articles (33 studies) were further screened based on eligibility criteria: articles with missing information related to on-board performance metrics, algorithm training and deployment, not using MCUs, and are non-wearable systems were excluded. This resulted in further exclusion of 16 studies. The remaining 17 studies were included in the review.

Data extraction was undertaken on eligible articles using a customized data extraction form. Information related to the system application, components, sensor placement, dataset used for training, ML algorithm and architecture, software tools for deployment, choice of MCU, and performance achieved for each TinyML system was extracted; together with performance metrics focusing on memory usage, time per inference, and power per inference.

TABLE 1. Summary of specifications for ARM and AVR microcontrollers used for edge inference in healthcare applications.

Microcontroller	Processor Core (F in the name refers to floating point)	Clock Speed	Flash	RAM	Voltage range	Current
STM32L476JG/RG [48]	Arm 32-bit Cortex-M4F	80 MHz	1 MB	128 KB	1.71 V - 3.6 V	117 μA/MHz
STM32L475VG [49]	Arm 32-bit Cortex-M4F	80 MHz	1 MB	128 KB	1.71 V - 3.6 V	117 μA/MHz
STM32F303RE [50]	Arm 32-bit Cortex-M4F	72 MHz	512 KB	64 KB	2.0 V - 3.6 V	379 μA/MHz
STM32F756ZG [51]	Arm 32-bit Cortex-M7F	216 MHz	1 MB	340 KB	1.7 V - 3.6 V	602 μA/MHz
STM32F769NI [52]	Arm 32-bit Cortex-M7F	216 MHz	2 MB	532 KB	1.7 V - 3.6 V	546 μA/MHz
nRF52832 [53]	Arm 32-bit Cortex-M4F	64 MHz	512 KB	64 KB	1.7 V - 3.6 V	58 μA/MHz
nRF52840 [54]	Arm 32-bit Cortex-M4F	64 MHz	1 MB	256 KB	1.7 V - 5.5 V	52 μA/MHz
ATmega2560 [55]	8-bit AVR	16 MHz	256 KB	8 KB	2.7 V - 5.5 V	875 μA/MHz
ATmega328p [56]	8-bit AVR	8 MHz	32 KB	2 KB	2.7 V - 5.5 V	437 μA/MHz

The rest of the paper is organized as follows. Section II will cover features and specifications of the most commonly used MCUs in healthcare devices for on-device inference. Then, software tools including frameworks and libraries used in deployment of ML algorithms onto MCU are discussed in section III. The various wearable systems using on-device inference are reviewed afterwards in section IV. Finally, a discussion of the reviewed MCU options, software tools, and their application in the implementation of TinyML healthcare wearable devices is presented in section V followed by concluding remarks in section VI.

II. MICROCONTROLLERS AS EDGE INFERENCE DEVICE

Although using MCUs for inference in the context of wearable healthcare devices has clear advantages, there are also important hardware limitations that need to be considered during design and deployment of ML algorithms. Those limitations are linked to the specification of the MCU. Hence, aspects that need to be considered when choosing the latter include the type of processor, available memory (both RAM and flash), speed, power, and size.

The type of processor used in a chosen MCU determines the performance of the board to a certain level. Its architecture and bus width relay information about how instructions for certain functions are executed and the number of cycles required. A 32-bit processor can handle more data in comparison to an 8-bit processor at any specific time, thus requiring fewer number of instruction cycles. As for the instruction set architecture (ISA) of the processor, within the reviewed papers, three main ISAs were utilized in the chosen MCU boards, namely: Advanced RISC Machine (ARM), Advanced Virtual RISC (AVR), and RISC-V. The three architectures are somewhat similar, being based on a reduced instruction set computer (RISC) architecture. The major difference to highlight is that both AVR and ARM (commonly used) are license-based architectures, requiring membership from vendors for hardware incorporation into their designed SoC. While RISC-V- a fairly new ISA- is an open-source instruction set that provides designers freedom and flexibility of use; this leading to an increasing level of interest from the research community [42]. However, due to its recent emergence it is still under development which results in few commercially available hardware implementations using it. Two RISC-V based MCUs, Mr.Wolf (research SoC) [43], and GAP8 (commercially available by GreenWaves) [44] have been used for TinyML devices to achieve low power consumption [45], [46], [47]. The choice of ISA alone does not dictate the overall performance of the processor, as the hardware integration/architecture and available peripherals play important role too. Therefore, MCUs with similar core processor ISA can still perform differently and other characteristics of the MCU should be investigated before a decision.

Another key aspect when choosing a microcontroller for wearable devices is the power consumption since this plays a vital role in usability (linked to both size and device maintenance). This on its turn is linked to minimum accuracy of inference required by the application, as it is going to be one of the factors determining the information loss that can be afforded when transferring ML algorithms from high computational servers to the edge. The latter is also going to be affected by the memory constraints of the chosen MCU. The available memory in an MCU both Flash and RAM generates further constraints on the size and architecture of the deployed ML models. There should be enough storage for both collected data and ML model parameters (weights and activation functions), as well as enough RAM to process data and run inference. Speed of the processor is also a factor to take into account, which can be crucial in the context of certain applications since it will determine how fast an inference and corresponding decision can be made. And the size is also important because wearable devices are heavily volume and area constrained. Overall, considering the limitations of MCUs, it is important in the design process to carefully establish the acceptable performance trade-offs as a function of all these different variables, taking into account the device intended use. Table 1 summarizes the specifications of the different ARM and AVR MCUs that have been used in the context of wearable healthcare devices, including type of processor, its speed, available memory Flash and RAM, as well as operating voltage range with current consumption when MCU is in run mode.

From the table, it can be seen how ARM is the most dominant provider for various Vendors, including STMicroelectronics [48], [49], [50], [51], [52], and Nordic Semiconductors [53], [54]. ARM provides low power Cortex-M processors suitable for use in embedding neural network architectures for on-device inference [57]. Within the different Cortex-M processors, Cortex-M4, and Cortex-M4F

(a version with an addition of floating-point unit (FPU)) are mostly used in the realization of edge inference systems having been the core of a number of reported wearable medical systems [45], [58], [59], [60]. Another Cortex-M processor that has also been used is the Cortex-M7(F), which provides higher performance capabilities at cost of power consumption when compared to Cortex-M4(F) processors. As for the AVR processors (ATmega) with 8-bit bus width by Microchip [61], they are mostly incorporated in Arduino boards and Tiny circuits.

In regard to the RISC-V microcontrollers, the GAP8 by GreenWaves Technology [44] and Mr. Wolf [43] were used in TinyML healthcare applications. They are both based on the parallel ultra-low power platform (PULP) [62], which is an open-source platform utilizing the open-source instructions of RISC-V. The architecture is a bit different from that of ARM and AVR. It has two set of cores to achieve fast processing with ultra-low power. The first is a fabric controller (FC) operating as a normal MCU for control, security and communication functions. The second is a compute cluster of 8 32-bit RISC-V cores for parallel operation and computationally intensive operation providing speedup of operation. GAP8 and Mr. Wolf share similar PULP design architectures with few differences. GAP8 is based on TSMC 55nm technology, while Mr.Wolf on TSMC 40nm technology. The maximum clock frequency provided by GAP8 is 250 MHz while Mr.Wolf is 450 MHz. Moreover, GAP8 does not support floating point operation as Mr. Wolf does. GAP8 integrates a CNN accelerator, the Hardware Convolution Engine (HWCE) in the cluster core for running CNN algorithms. As for the memory, the PULP architecture provides three levels: L1, L2, and L3 which is an optional external memory. L2 provides 512 KB of memory arranged in multi-banks that are accessible by all processors. While L1 has a smaller capacity of 80 KB for GAP8 and 64 KB for Mr.Wolf shared between all cores of cluster processor. The power range for GAP8 is 3.6 μW - 75 mW operating on voltage between 0.8 V - 0.2 158 V, while Mr. Wolf is 72 μW - 153 mW operating in the range of 0.8 V and 1.1 V.

III. SOFTWARE TOOLS AND FRAMEWORKS FOR EDGE INFERENCE

Choosing the right MCU for a specific application is just one part of the decision in the design process of a TinyML healthcare device. The software tools used for deployment of the model onto the MCU, their features capabilities and the compatibility of the latter with the existing MCUs can play a major role into which MCUs are a feasible choice.

In order to deploy machine learning algorithms onto MCUs, the model needs to be compressed to a size that fits the memory limitation of the chosen MCU. Two main methods for compressing a model are quantization and pruning [63], [64]. Quantization converts 32-bit floating point values of weight and activation functions into a less precise representation compatible with the MCU, an 8-bit fixed point value that will occupy less memory space. This process can be done after training the model for optimum performance (post-quantization) to provide up to 4× smaller model. Pruning, on the other hand, eliminates neurons and connections in the model's architecture that do not affect the overall model performance as much as major connections do. The use of quantization and pruning provide an MCU compatible model that reduces both power consumption and latency as a result of reduced memory usage, but at the expense of model's accuracy. The challenge is to find an acceptable trade-off between performance and power/latency for the embedded ML model. It is possible to use a single technique alone or both combined. This was demonstrated in the work of [64], where both techniques were used to compress a 7-layers convolutional neural network (CNN) and a ResNet-50 model. Both classification models were tested using the CIFAR-10 dataset [65] to classify images to one of the corresponding 10 classes. The dataset composed of 60,000 images (6000/class) divided into six batches. The sixth batch containing 1000 images from each class was kept for testing. First, the trained model's weights were extracted, and then pruned using different sparsity levels. The pruned models were retrained using the same dataset to conclude with the best performing pruned model. The chosen model was then quantized, transforming the 32-bit floating point values into 8-bit integers. The pruning of models decreased the number of parameters. In the case of the CNN model, it was reduced from 0.95376 million to 0.19118 million parameters. The combined techniques presented models with smaller number of parameters represented using lower number of bits. The resulted compressed CNN model achieved reduction in size from 15.03 MB to 0.97 MB. The accuracy of the model dropped from 84.17% to 83.73%, an acceptable 0.44% loss. While the ResNet-50 model presented a 1.32% decrease in accuracy for 20% reduction in number of parameters.

In addition to quantization and pruning, the deployment of ML algorithms onto MCU requires a certain level of optimization for embedding. To achieve this, several software tools, libraries, and frameworks are used to facilitate it, some which include quantization within their process of optimization. The ones that have been used within the context of the reviewed healthcare systems are covered in the following.

A. TensorFlow LITE

TensorFlow Lite (TFLite) is an open-source framework developed by Google that enables the deployment of machine learning models on mobile and embedded platforms [66]. The key features of this framework are: its optimized ML model for on-device deployment, its compatibility with several platforms including mobiles (iOS and Android) and microcontrollers (using TFLite for microcontrollers), and its support for multiple languages including Python, C++, Objective-C, Java, and Swift. TFLite models are represented in a FlatBuffers format, which provides reduced size and faster inference compared to the TensorFlow's Buffer format, due to its smaller code footprint and directly accessible data. The generation of the TFLite model can be done

through three different methods: (1) using existing TFLite models from available examples; (2) designing an own model through TFLite Maker; or (3) converting TF models to TFLite using the TFLite converter and applying quantization as an optimization method through the process. Multiple optimization techniques are available through the optimization toolkit, including quantization, pruning, and clustering. Once a TFLite model is finalized it is converted to a C source file for running on the MCU. Running the generated C code on MCU requires an MCU specific library version of TFLite, the ''TFLite for microcontrollers'' to run/interpret the deployed model. TFLite for microcontrollers was specifically designed for MCU deployment written in C++ 11, requiring a 32-bit platform. ARM Cortex-M-series based processors were tested for TFLite for microcontroller; and some of the supported development boards are: Arduino Nano 33 BLE Sense, SparkFun Edge, STM32F746 Discovery Kit, and others [67]. Overall, the use of TFLite generated model addresses the design constraints of embedded ML namely, power consumption, memory, latency, and privacy.

B. CMSIS-NN

CMSIS-NN is an open-source library for ARM Cortex-M processor cores, which maximizes the performance of neural networks (NNs) through a collection of optimized kernels producing minimum memory footprint [68], [69]. It allows the conversion of floating-point models into fixed point representation compatible for deployment onto Cortex-M MCUs [27]. The use of CMSIS-NN kernels improves the throughput by 4.6× and the energy efficiency by 4.9× [68]. The library kernels are divided into two functions: NNFunctions and NNSupportFunctions. The first is concerned with the implementation of NN layers, and the second with utility functions for data conversion and activation functions. The kernel APIs are simplified to allow compatibility with multiple ML frameworks, TensorFlow, PyTorch, or Caffe.

C. X-CUBE-AI

X-CUBE-AI is an artificial intelligence (AI) expansion package of STM32Cube.AI specific for STM32 microcontrollers [70]. The latter is an ecosystem which converts and then optimizes pre-trained NN models for integration on board. The package offers validation of NNs on PC and MCU, as well as evaluation of performance on STM32 MCU. The package can be simply added to the STM32CubeMX tool for use, helping optimize pre-trained models and offering help in choosing most suitable STM32 board in terms of memory and computation. For the implementation, data can be collected, cleaned and processed for model training using any of the major frameworks (TFLite, PyTorch, MATLAB, Keras, ONNX, and others), then X-CUBE-AI package can be used to automatically convert the model into a computationally optimized version with optimum memory usage ready for integration. The package provides three main features: dimensionality through assessment of model architecture and MCU needs; optimization by conversion of the pre-trained model to C-code; and support of quantization for optimum performance, and fine tuning by allocating optimum memory usage.

D. MICROSOFT EdgeML

Apart from compression of pre-trained ML models for deployment on MCU, a ready compact algorithm can be used for direct integration on resource-constrained devices. With this notion, Microsoft developed a library of algorithms (EdgeML) that can be used for direct inference on edge devices [71]. The algorithms are written in Python using TensorFlow and PyTorch and provided as C++ implementation. The four algorithms provided by EdgeML are:

- ProtoNN: an algorithm based on k-nearest neighbor (kNN) that can be used for classification and regression problems. The model occupies around 2 KB of memory.
- Bonsai: a tree-based algorithm for classification and regression problems with complex logarithmic based prediction. The model occupies around 2 KB of memory.
- EMI-RNN: recurrent neural network (RNN) based algorithm for time-series predictions with faster inference than traditional RNN.
- Fast cells: a model for smaller and faster RNN cells, including FastRNN and FastGRNN algorithms for time-series classification in place of LSTM and GRU. The model size is less than 10 KB.

SeeDot Embedded Learning Library (ELL) is a framework provided by Microsoft EdgeML for deployment of these algorithms to IoT devices [71]. SeeDot provides compilation of the model, and quantization from floating point to fixed point representation to run efficiently.

E. GAP FLOW

Deployment of trained models onto RISC-V based MCUs of the GAP family require specific set of neural network tools provided by GAPflow [72]. GAPflow is composed of a multiple set of tools that can automatically intake a trained NN model and produce an MCU compatible algorithm for deployment and on-board inference. The tools used are, NNTool; AutoTiler; and the GCC. The NNtool translates a TFLite model (quantized or unquantized) to an ''AutoTiler Model'', which is a.c file describing the NN topology and the quantization policy of the different NN layers. The main tool in GAPflow is the AutoTiler, which is responsible for optimizing the execution of convolutional layers and minimizing access to memory, as well as use of parallel convolutional kernels that leverage the available multi-core clusters of GAP8 MCU. The optimized model files produced by the Autotiler tool defining the application code and memory allocation of constant parameters are then compiled using the GCC tool. Finally, the GAP8 executable file and flash file are used to run inference on GAP8 [72].

F. PULP-NN

PULP-NN is an open-source library similar to the CMSIS-NN library designed for use with PULP based MCUs [73]. It adopts the data flow and layout of the CMSIS-NN library [69]. The library uses a set of kernels to facilitate the deployment of deep learning models for inference on edge devices. Quantized Neural Network (QNN) models (8-bits, 4-bits, 2-bits, and 1-bit) are supported by the library using DSP extensions and multi-core architecture of the RISC-V processor to speed up it performance. For comparison, a classification QNN model trained on CIFAR-10 dataset was used for inference on GAP8 using PULP-NN library, and 2 Cortex-M MCUs, STM32H743 (Cortex-M7), and STM32L467 (Cortex-M4) using CMSIS-NN library [73]. The quantized model (8-bits) had 3 convolutional layers and a 1 fully connected layer. The PULP-NN based implementation resulted in speed up of inference, requiring less clock cycles by a factor of 19.6× (STM32H7) and 30× (STM32L4) in comparison to the CMSIS-NN implementation.

G. NEMO/DORY

NEMO (NEural Minimizer for pytOrch) is an open-source python library for compressing deep neural network (DNN) models developed in PyTorch [74]. It targets deployment of DNN models on constrained MCUs specifically PULP based MCUs with different features for model quantization. DORY (Deployment Oriented to memoRY) is a tool for automatic deployment of DNNs on memory constrained MCU [75]. It uses constraint programming to solve the tiling problem, maximizing memory usage based on constrained introduced by individual DNN layer. It provides memory management through three optimization tasks: loop tiling; memory access optimization; and memory fragmentation. Three steps are performed through DORY framework before deployment: (1) ONNX decoding of quantized DNN model in Open Neural Network Exchange (ONNX format); (2) layer analyzer, which produces an optimized code for tiling loop and calls back-end APIs for individual execution of layers; and (3) network parser, which collects information from the architecture and allocates memory buffers accordingly, generating a C file ready for MCU deployment. DORY is currently only supported by the GAP8 MCU.

H. FANN-ON-MCU

FANN-on-MCU is an open-source framework for the implementation and deployment of optimized multi-layer neural network based on fast artificial neural network (FANN) library [76]. The framework generates an optimized C code for compilation on chosen platform from a pre-trained model in FANN's format. The toolkit provides support for deployment on both ARM Cortex-M and PULP based MCUs for on-board inference of trained FANN model in either fixed or floating point representation. As for the FANN library, it is an open-source library for implementation of artificial neural networks (ANNs) [77]. It implements multi-layer neural architecture for both fully connected and sparsely connected networks in C and provides bindings to multiple languages such as MATLAB, python, and others. FANN library optimizes the number of neurons and hidden layers in an ANN architecture through cascade training [77]. A graphical interface, the FANNTool was developed to simplify the use of FANN library [78]. The tool allows changes to the model's architecture, the activation function, the weight initialization, the training method and allows monitoring of training process [78].

The choice of a software tool for optimum deployment is not limited to one library or framework. It is possible to use a combination of compatible tools and libraries. An example using both TFLite and X-CUBE-AI for the implementation of ''Hello World'' ML model on NUCLEO-F74ZG board was presented in [63]. The CNN model for the recognition of handwritten numbers (MNIST dataset [81] of 70,000 samples for the digits (0-9) was initially trained (60,000 samples for training and 10,000 for testing) in TensorFlow using Keras, having a size of 7.17 MB. It was then converted into a lighter version (2.4 MB) using TFLite and TFLite Converter tool. Further compression using the X-CUBE-AI STM32 package applied multiple actions for model reduction, including compression of weights of the fully connected layers, fusion of layers by combining two layers, and optimization of activation function leading to model size reduction by a factor of 4 (668.97 KB).

IV. TinyML HEALTH AND CARE SYSTEMS

The use of ML in the context of health and care applications has been so far mostly dependent on cloud and fog computation. The ability to perform edge inference on MCUs has only become recently feasible due to the technological advances, allowing deployment of compressed ML algorithms with minimum loss in model performance. In this work, various proof-of-concept TinyML systems are reviewed, focusing on application in health and care including medical use, ambient assistant living, and physical health for rehabilitation and fitness tracking. Related information for each system is summarized in Tables 2-5. System components and prototype placement (if available) are summarized in Table 2, while datasets used in training and testing of the ML algorithms are given in Table 3 with corresponding ML architecture in Table 4. Finally, a summary for the embedded ML implementation- including accuracy of running algorithms on board, the occupied memory, time and power consumption per single inference- are tabulated in Table 5. Most of the reviewed works reported only the accuracy of the embedded algorithm.

A proof-of-concept wearable device for Parkinson's patients was presented in [79], focusing on recovery of patients from Freeze of Gait (FoG), this is a ''brief, episodic absence or marked reduction of forward progression of the feet despite the intention to walk'' [109]. The subject-dependent device aimed to monitor the patient and provided rhythmic auditory stimulation (RAS) when a FoG

TABLE 2. Summary of TinyML healthcare systems components and device placements.

Ref	Healthcare application	System components	Placement
[79]	Detection and recovery from FoG in Parkinson's patients	3 tri-axial accelerometers, RAS module, & Arduino Mega	Ankle, leg, torso, ear
[58]	Detection of cardiac arrhythmia	nRF52832	No prototype
[45]	Detection of cardiac arrhythmia	GAP8 & STM32L475	No prototype
[59]	Prediction of blood glucose level	Continuous glucose monitor (CGM), insulin pump, sensor band, STM32F303RE	No prototype, future implementation of MCU in wearable device
[60]	Detection of epileptic seizure	2 bipolar EEG channels (F7-T7 and F8-T8), ADS1299 EEG front-end, tri-axial accelerometer, BlueNRG-MS Bluetooth low energy (BLE) network processor, STM32L476	No prototype but referred to the use of similar setup as previous work (e-Glasses [80])
[47]	Detection of epileptic seizures	Biowolf ExG wearable platform [81]	Patch-like form (Biowolf) on head
[82]	Heart rate prediction	Nucleo STM32L476RG development board	No prototype, future wrist-worn device
[46]	Stress detection	Pressure sensor, 9-axis motion sensor, microphone, ECG/EMG & bioimpedance AFE (Maxim MAX30001), galvanic skin response (GSR) front-end, 120 mAh LiPo battery, dual-source energy harvester, & two processors (Nordic nRF52832 & Mr. Wolf)	Wrist bracelet
[83]	Fall detection	TinyLily mini processor (ATMega328p),TinyLily ASL2002 module with tri-axial accelerometer (Bosch BMA250), & piezo buzzer	Hip-level belt
[84]	Fall detection	SensorTile (STM32L476JGY MCU, 2 tri-axial accelerometers, gyroscope, magnetometer, & a barometer)	Belt buckle
[85]	Fall detection	SensorTile (STM32L476JGY MCU, 2 tri-axial accelerometers, gyroscope, magnetometer, & a barometer)	No prototype
[86]	Fitness tracker	SensorTile (STM32L476JGY MCU, 2 tri-axial accelerometers, gyroscope, magnetometer, & a barometer)	Wrist band
[87]	Exercise tracker	CLOUD-JAM L4 board (STM32L476RG)	No prototype, future wrist-worn device
[88]	Motor-Imagery Brain Computer interface	STM32L475VG & STM32F756ZG	No prototype
[89]	s-EMG based hand gesture recognition	8-channel AFE (ADS1298), GAP8	Ring configuration around forearm
[90]	s-EMG based hand kinematics finger position decoding	GAP8	No prototype
[91]	Tactile sensing for prosthetic & robotics	Conductive polymer composite (CPC) based low resistive tactile sensor [92], AFE, IMU (MPU-9250), STM32F769NI discovery board	Hand glove

event was detected. It was composed of four main parts: (1) wearable sensors, which are 3 tri-axial accelerometers to be positioned at the ankle, leg, and torso providing a total of 9 readings; (2) a feature extraction model based on the time domain features for less computation; (3) a classification model for detection of a binary problem; and (4) a RAS module placed in the patient's ear for stimulating patient by metronome click-embedded music whenever a FoG event was detected. A 16 MHz ArduinoMega (ATMega 2560) MCU with 8 KB internal SRAM was used in the implementation. Several options were tested as classification models: standard ML algorithms including Decision Tree (DT), Random Forest (RF), AdaBoost (AB), k-Nearest Neighbor (k-NN), and Support Vector Machine (SVM)); and Microsoft developed Edge ML algorithms, Bonsai and ProtoNN. The standard ML models required larger memory $>$ 100 KB and were more computationally intensive compared to Bonsai and ProtoNN. Even with additional compression, 8 KB memory was required. The final system using ProtoNN with optimized feature extraction model and a window size of 2s was implemented using 1.4 KB of memory after pruning, with an average recall of 93.58% (1.3% less than standard ML).

TABLE 3. Summary of datasets used in training and evaluation of the algorithms used in the reviewed TinyML healthcare systems.

Ref	Dataset used	Description	Data sensors	Part of dataset used	Train	Test	Validation
[79]	DAPHNet [93]	Accelerometer readings from 10 patients during daily life activities (8 hrs & 20 mins)	3 tri-axial accelerometers (ankle, leg, torso) @ 64 Hz	237 FoG events. Patients 4 & 10 excluded (No FoG) (2 classes)	70 %	30 %	10-folds
[58]	Computing in Cardiology 2017 Challenge [94]	ECG readings (30s-60s long) from 8,528 subjects	ALivCor device single lead ECG @ 300 Hz	8,528 samples (4 classes)	82 %	18 %(1)	-
[45]	ECG500 [95]	20 hrs long ECG recordings from single subject (chf07) of original dataset [96], [97]	2 ECG lead @ 250 Hz	5,000 samples (5 classes) (2)	10 %	90 %	-
[59]	OhioT1DM [98]	Data collected from 12 T1D subjects over 8 weeks clinical trial	CGM sensor (Medtronic Enlite), insulin pump (Medtronic 530G or 630G), sensor band for physiological data (Basis Peak fitness band or Empatica Embrace band)	only CGM blood glucose readings every 5 mins (166461 samples)	64.8 %	19 %	16.2 %
[60]	CHB-MIT scalp EEG [99]	Data collected from 23 subjects. Total of 664 edf (most are 1 hr lonf, some are 2 & 4 hrs long) with 182 annotated seizures.	23 channels EEG electrodes, 10-20 bipolar montage @ 256 Hz	F7-T7 & F8-T8 channel readings (2 classes)	70 %	30 %(3)	10-folds
[47]	CHB-MIT scalp EEG [99]			F7-T7, T7-P7, F8-T8, T8-P8 channel readings (2 classes)	No splitting ratio given. Classes were given weight inverse to their occurrence frequency.		
[82]	PPGDalia [100]	Data collected from 15 subjects doing 8 different activities Total of 37.5 hrs of data	PPG sensor @ 64 Hz & tri-axial accelerometer @ 32 Hz (wrist worn device Empatica E4), & chest worn device RespiBAN Professional @ 700 Hz for golden HR values	PPG, accelerometer readings, & golden HR values. 64697 total samples (1 class)	Leave-one-subject-out (LOSO) cross validation		
[46]	Stress recognition in automobile drivers [97], [101]	Data collected from 17 drives lasting for (65-93 mins) driving in city & highways (number of subjects not clear)	ECG, EMG (right trapezius), GSR measured on the hand & foot, & respiration	ECG, GSR readings only. (Number of samples not given). (3 classes)	subsets of equal stress levels		
[83]	SisFall [102]	Data from 38 subjects in 2 groups (23 adults (3.5 hrs/subj.) & 15 elderly(1.5 hrs/subj.)) for 19 types of activities of daily life (ADL) & 15 types of falls	3 hip level sensors (2 accelerometers (ADXL345 & MMA8451Q)) & a gyroscope (ITG3200)	Accelerometer readings only, 4510 samples. (2 classes)	SisFall	real-time data (2 subjects)	-
[84]	SisFall Enhanced [103]	SisFall dataset [102] with temporal annotation for 3 classes (fall, alert, background)	3 hip level sensors (2 accelerometers (ADXL345 & MMA8451Q)) & a gyroscope (ITG3200)	Accelerometer readings of 38 subjects. Total samples after sub annotation not given. (3 classes)	30 subj. (12 elders)	8 subj. (3 elders)	-
[85]	SisFall Enhanced [103]			Accelerometer & gyroscope readings of 38 subjects. Total samples after sub annotation not given. (3 classes)			20% of training

TABLE 3. *(Continued.)* **Summary of datasets used in training and evaluation of the algorithms used in the reviewed TinyML healthcare systems.**

[86]	Exercise for fitness tracking [86]	15 subjects repeating 3 exercises (squat, curl, push-up). Data collected between exercises labelled as not an exercise (NAE))	Tri-axial accelerometer & tri-axial gyroscope (LSM6DSM) @ 20 Hz	Total of 700 reps (samples) for the 3 exercises. (4 classes)	80 %	20 %	-
[87]	PPG_ACC Dataset [104]	PPG & accelerometer readings from 7 subjects. Each with 5 series of 3 activities (squats, stepper, resting). Total of 17,201s recorded data.	PPG & tri-axial accelerometer (maxim integrated MAXREFDES100 health sensor) @ 400 Hz	210 recording sessions. (3 classes)	5 subj.(4)	2 subj.	-
[88]	EEG Motor Movement Imagery [97], [105]	EEG recordings for both real & imagery motor task from 109 subjects, each performing 14 experiments (2 rest, & 3 reps of 4 tasks (2 real, 2 imagery))	BCI2000 systems using 64 electrodes of 10-10 international system @ 160 Hz	Only imagery recordings from 105 subjects with 21 trials per class per subject. (4 classes)	80 %	20 %	5 folds
[89]	NinarPro DB6 [106]	sEMG readings from 10 healthy subjects. Each subject performed 12 reps of 7 grasps. Each grasp lasted for 6s with 2s rest.	14 Delsys Trigno sEMG wireless electrodes on higher half of forearm @ 2kHz	10 sessions. Total 10 sessions. (12x7 = 84 samples/session). (8 classes)	Sessions 1-5 (5)	Sessions 6-10 %	2-fold stratified cross validation
	20-sessions [89]	sEMG readings from 3 subjects, total of 20 sessions. 6 reps of 8 hand gestures. Each gesture lasted 3s with 3s rest between reps & 5s rest between gestures.	8 channel AFE (ADS1298) around middle forearm @ 4 kHz	20 sessions, each session with 8 gestures & rest. (9 classes)	Sessions 1-10 (5)	Sessions 11-20	
[90]	NinaPro DB8 [107]	12 subjects (10 able-bodied & 2 right-hand transradial amputees). Each subject recorded 3 sessions for 9 movements in both hands lasting (6s-9s) each with 3s rest. Session 1 & 2 (10 reps/movement) & session 3 (2 reps/movement)	2 rings of 8 active double differential sensors (Delsys Trigno IM Wireless EMG system) on the right forearm & 18-Degree of Freedom (DoF) (Cyberglove 2) on the left hand @ 2 kHz	3 sessions/subject; total of 198 samples/subj. (5 classes - 5 DoA (6))	Session 1 & 2	Session 3	2-folds
[91]	ETHZ-STAG Dataset [91]	Tactile & inertial data collected in 5 sessions. Each session using 16 different objects manipulated for 40 sec each. (Number of subjects not mentioned)	Conductive polymer composite (CPC) based low resistive tactile senor [92], AFE, inertial measurement unit (IMU) (MPU-9250) (@100 Hz for accelerometer & gyroscope)	Total of 340,000 tactile frames (samples). 320,000 (from the 5 sessions) + 20,000 empty hand frames. (17 classes)	4 sessions	1 session	-

(1) Data with balanced classes; (2) unbalanced dataset with 2 samples for one of the classes; (3) test data had ratio of 1:300 for ictal:no-seizure; (4) data augmentation used (oversampling) to produce balanced classes; (5) incremental training protocol; (6) the 5 DoAs are defined as linear combination of the 18 DoF.

TABLE 4. Tabulated summary of the architecture of used ML algorithms in the reviewed TinyML healthcare systems.

Ref	ML algorithm	Input	ML Architecture
[79]	ProtoNN	(128 x 45) with w = 2s @ 64 Hz; 5 TD feature for each of the 9 ACC inputs	Projection dimension d = 5. Tuned to binary implementation of ProtoNN by Gupta et al. [108]
[58]	CNN + GRU	(256 x 1) with w = 256 samples with 50 % overlap	Input (256, 1) – Conv1(128, 8) – Conv2(64, 16) – Conv3(32, 32) – Conv4(16, 64) – Conv5(8, 64) – Conv6(4, 128) – Conv7(2, 128) – Global Average Pooling (128) – GRU (64) – Dense + Softmax (4). Each Conv layer followed by average pooling (size = 2, stride =2); all Conv layers have filter size = 5.
[45]	TCN	(140 x 1) with 140 samples input (0.56 s)	Input (1 x 140) – Conv(1 x 1) – TCN1 – TCN2 – TCN3 – FC (5). Conv layer (2 filters); TCN 1–3 (filters = 11, kernel length = 11, and d = 1, 2, 4).
[59]	RNN-LSTM	5 mins blood glucose readings	LSTM (1, 32) – Dense (1, 64) – ReLU (1, 64) – Dense (1, 32) – ReLu (1, 32) – Dense (1, 1)
[60]	Random Forest(RF)	54 features from each 4s EEG epoch with data fusion of 2 EEG epochs (time separation 1/4 of average seizure)	Bagging method using bootstrap samples of training data for each tree. No further information provided.
[47]	SVM	4 level DWT for w = 8s of EEG epoch	General description of algorithms given with no details about parameters. Classification output was post processed (smoothed) using moving average of 4s window over 3 successive classifications
	RF		
	Extra Trees		
	AdaBoost		
[82]	TEMPONet	(256 x 4) with w = 8s @ 32 Hz with 75 % overlap	3 convolution blocks (2 dilated Conv, 1 strided Conv, 1 pooling layer) output channel of each block (32, 64, 128) – FC (1). All layers use ReLU activation & Batch Normalization.
[46]	MLP	5 features from overlapping windows (3 ECG and 2 GSR)	[5 50 50 3]
[83]	Bonsai	(240 x 1) with w = 3s sliding window @ 20 Hz of 4 calculated values from accelerometer readings	Single decision tree using the standard hyperparameters values.
[84]	RNN-LSTM	(100 x 3) with w = 1s @ 100 Hz	Input (n x 3) – FC (n x 32) – Batch Normalization (n x 32) – dropout – LSTM1 (n x 32) – dropout – LSTM2 (n x 32) – dropout – FC (1 x 3) – softmax. n = 100
[85]	RNN-LSTM	(256 x 6) with w = 1.28s @ 200 Hz	Input (n x 6) – FC (n x 16) – Batch Normalization (n x 16) – dropout – LSTM (n x 16) – dropout – FC (1 x 3) – softwax. n = 256. (Used weighted cross entropy loss function during training to compensate for class imbalance)
[86]	CNN-1D	(68 x 6) with w = 3.4s @ 20 Hz	Input (68 x 6) – Conv1 – ReLu – Conv2 – sigmoid – MaxPooling - output (1 x 4). Conv1 and Conv2 have kernel sizes = 3.
[87]	RNN	(30 x 4) with w = 3s @ 10 Hz with 50 % overlap	Input (n x 4) – Dense 1 (n x 32) – Batch Normalization (n x 32) – LSTM1 (n x 32)- dropout 1 (n x 32)– LSTM 2 (n x 32) – dropout 2 (n x 32) – LSTM 3 (1 x 32) – dropout 3 (1 x 32) – Dense 2 (1 x 3). n = 30
[88]	EEGNet	(n x 38) with w = 1s and 2s @ 160 Hz but down sampled by factor 3 (160/3) for 38 EEG channels	Input (n x 38) – Conv (n x 38 x 8) – Batch Normalization – DepthConv (1 x n//Nf x 16) – Batch Normalization – Exponential linear unit (ELU) – Average Pooling – Separable Conv (1 x n//Np//8 x 16) – Batch Normalization – ELU – Average Pooling – FC (4) - Softmax. Nf = 128/3; Np = 8/3; n = w * freq
[89]	TEMPONet	(300 x 14) for NinaPro DB6 and (300 x 8) for 20-sessions; with w = 150ms @ 2 kHz	3 Convolutional blocks – 2 FC blocks – FC – Softmax. Each Convolution block: (2 x (Dilated Conv – Batch Normalization – ReLu) – 1 x (Strided Conv – Average Pooling – Batch Normalization - ReLu)). FC block: (FC- Batch Normalization – ReLu - Dropout).
[90]	TEMPONet	(256 x 16) with w = 128ms @ 2 kHz	3 Convolutional blocks – 2 FC blocks (256 & 32)– FC (5). Convolution block: (2 x (Dilated Conv – Batch Normalization – ReLu) – 1 x (Strided Conv – Average Pooling – Batch Normalization - ReLu)). FC block: (FC - Batch Normalization – ReLu - Dropout).

TABLE 4. *(Continued.)* **Tabulated summary of the architecture of used ML algorithms in the reviewed TinyML healthcare systems.**

[91]	CNN	(32 x 32) Single tactile frame	Modified ResNet-18 Conv (32 x 32 x 16) -Batch Normalization – ReLU – MaxPool (16 x 16 x 16) – ResNet – Dropout – ResNet (8 x 8 x 32) – Conv (8 x 8 x 32) – Average Pooling (32) – FC (17)

w: input window size; TD: time-domain; ACC: accelerometer.

The system required 747 ms for feature extraction before optimization and 49.3 ms for the optimized feature set, with 20 ms for classification time.

Other works reported in literature focused on the detection of cardiac arrhythmia from a single lead ECG [45], [58]. In [58], a convolutional-recurrent neural network (C-RNN) with gated recurrent unit (GRU) instead of long short-term memory (LSTM) was trained using dataset from the computing in cardiology 2017 competition [94]. The model was trained to classify the input data into 4 output classes (normal rhythm, atrial fibrillation, noises, and other rhythms) using Keras with TensorFlow. The trained model's weights and activation were quantized to 8-bit fixed point representation, using the CMSIS-NN library, for deployment on Nordic's nRF52832 MCU. The model performance was tested before and after quantization. The latter caused a drop in accuracy of 0.66%, as well as 2% drop in F_1 score. The final system occupied 195.6 KB of flash memory, and 6.8 KB of RAM, whilst being able to generate an inference in 94.8 388 ms with an accuracy of 85.44%, and F_1 score of 78%.

The reported work, also within the context of detection of cardiac arrhythmia [45] used temporal convolutional network (TCN) embedded on edge. The model was trained using TensorFlow and Pytorch with the dataset ECG5000 [95] which is based on the BIDMC Congestive Heart Failure Database (chfdb) [96] considering 5 output classes. The work compared the deployment of the classification model onto two platforms, an ARM Cortex M4F (STM32L475), and a RISC-V PULP (GAP8) using different deployment/quantization tools. The GAP8 was tested using 2 deployment methods, the GAPflow and NEMO/DORY (NEMO for quantization and DORY for deployment using PULP-NN backend). The STM32L475 was tested under three deployment methods: (1) using TFLite alone; (2) using X-CUBE-AI with TFLite; and (3) using X-CUBE-AI with Keras. Among the 5 implementations, the use of GAP8 outperformed STM32L475 in terms of memory footprint, time for inference, and power per inference. The accuracy of the deployed quantized model was the same across both platforms (94%) with a negligible drop of 0.2% in accuracy for GAP8 with NEMO/DORY which had the best performance overall. For the ARM Cortex implementations, the use of X-CUBE-AI with TFLite provided the best performance out of the three. Comparing the best performance of both platforms, the GAP8 and STM32L475, the GAP8 with NEMO/DORY demonstrated energy efficiency (9.91 GMAC/s/W), 23x higher than STM32 using X-CUBE-AI with TFLite, as well as 46.8x faster inference (2.7 ms).

Edge inference was also proposed in [59] in the context of a wearable artificial pancreas systems for patients with Type 1 Diabetes (T1D). The system input readings, acquired from a continuous glucose monitoring (CGM) sensor, were used to predict blood glucose through the use of an RNN model, based on LSTM layers. The LSTM-RNN regression model was trained using TensorFlow and Keras using the ''OhioT1DM dataset'' [98], having a single LSTM layer and two dense layers with ReLU activation before the output layer. The final trained model was then deployed on an ARM Cortex-M4F STM32F303RE MCU for edge inference, occupying 34.69 KB of flash memory and 1 KB of RAM. The deployment of the model onto the STM32 MCU made use of the available X-CUBE-AI library for an 8-bit fixed point representation. As a regression problem, the performance was evaluated by calculating root mean square error (RMSE) and mean absolute error (MAE) at two prediction horizon (PH), 30-min PH (MAE $= 13.59 \pm 1.47$ mg/dL and RMSE $=$ 19.10 ± 2.04 mg/dL) and 60-min PH (MAE $= 24.25 \pm$ 2.8 mg/dL and RMSE $= 32.61 \pm 3.45$ mg/dL). The reported error between inference on edge and local server was reported to be 0.0029 mg/dL and 0.0025 mg/dL for RMSE and MSE respectively. In relation to power consumption, individual blood glucose level predictions were made within 22.2 ms for every new CGM reading (every 5 minutes), with the system being in sleep mode otherwise. This led to an ultra-low average power of 8 μW for running the algorithm on MCU. The authors did not present a final device prototype. However, they demonstrated the feasibility of using edge inference in predicting blood glucose level for future incorporation with CGM and insulin pumps for diabetes care device.

An epileptic seizure detection wearable system was proposed in [60]. Using only two EEG channels for signal acquisition (F7-T7, F8-T8), 54 features were extracted and data fusion was then used to enhance variability of data. The publicly available dataset ''CHB-MIT'' [99] was used for the training and testing of a subject-based RF ML model. The implemented system used an ADS1299 EEG AFE for data conditioning, and the trained ML model was deployed on STM32L476 MCU for on-board inference. An inference was made every 4 seconds (length of EEG epoch) and the algorithm for processing and classifying the readings required 27.9 ms per EEG epoch for both channels, consuming 7.34 mA to run the algorithm, the power consumption was not reported nor the energy or operating voltage. Operating

TABLE 5. Performance summary of reviewed TinyML healthcare systems.

Ref	Model	SW tool	Accuracy	MCU	Flash (KB)	RAM (KB)	Time/inf. (ms)	Power/inf. (mW)
[79]	ProtoNN	Edge ML	93.58 %	Arduino Mega (ATMega 2560)	1.4	NG	20	NG
[58]	CNN+GRU	CMSIS-NN	85.4 %	nRF52832	195.6	6.8	94.8	20.65
[45]	TCN	NEMO/DORY	93.8 %	GAP8 (2)	26.63	NG	2.7	38.52
		TFLite+GAPflow	94.0 %		25.03	NG	20.2	41.67
		TFLite	94.0 %	STM32L475	35.86	NG	188	45.54
		CUBE.AI+TFLite	94.0 %		35.86	NG	126.5	42.75
		CUBE.AI+Keras(1)	94.2 %		113.4	NG	374.3	47.68
[59]	RNN	X-CUBE-AI	13.59 ± 1.47 mg/dL (3)	STM32F303RE	34.69	1.0	22.22	0.008
[60]	RF	NA	94.5 % (4)	STM32L476	NG	NG	27.9	7.34 mA (7)
[47]	SVM	NA	99.6 % (4)	Mr.Wolf (2)	NG	NG	0.2	25.41
	RF	NA	100 % (4)		NG	NG	0.052	21.89
	Extra Trees	NA	100 % (4)		NG	NG	0.052	21.89
	Adaboost	NA	100 % (4)		NG	NG	0.057	22.44
[82]	TEMPONet (Best MCU)	X-CUBE-AI	5.64 BPM(1)	STM32L476RG	160	84.0	427	12.10 (5)
			7.06 BPM		94.7	24.9	376	12.10 (5)
	TEMPONet (Best Size)		6.29 BPM(1)		16.5	11.5	17.1	12.28 (5)
			7.55 BPM		8.07	7.21	19.0	12.11 (5)
[46]	MLP	FANN-on-MCU	NG	Mr.Wolf (2)	14	NG	0.061 (5)	19.58 (5)
			NG	nRF52832	14	NG	0.472 (5)	10.8 (5)
[83]	Bonsai	EdgeML	94.2 %	TinyLily (ATMega328p)	15	NG	18	NG
[84]	RNN(1)	CMSIS-NN	94.13 % (6)	STM32L476JGY	NG	82	300	5 mA (7)
[85]	RNN(1)	CMSIS-NN	93.5 % (6)	STM32L476JGY	NG	18.5	51	NG
[86]	CNN	X-CUBE-AI	100 %	STM32F756	66.53	16.25	80	NG
[87]	RNN(1)	X-CUBE-AI	95.54 %	STM32L476RG	94.7	12.03	150.1	NG
[88]	EEGNet(1) (w = 1s)	X-CUBE-AI	62.51 %	STM32L475VG (@ 80 MHz)	6.61	70.27	100.84	42.44
				STM32F756ZG (@ 80 MHz)			54.99	131.41
				STM32F756ZG (@ 216 MHz)			20.4	412.76
	EEGNet(1) (w = 2s)		64.76 %	STM32F756ZG (@ 216 MHz)	7.12	139.2	43.81	413.06
[89]	TEMPONet	PULP-NN	93.3 % (HG) 61.0 % (G)	GAP8 (@ 170 MHz)	460	NG	12.84	70 (5)
[90]	TEMPONet	NEMO/DORY	6.89°	GAP8 (2)	70.9	NG	4.76	51
[91]	CNN	X-CUBE-AI	77.84 % (8)	STM32F769NI	177	52	100	NG

NG: not given; NA: not applicable HG: hand gesture; G: grasp; (1) floating point representation (un-quantized deployment); (2) clock frequency set to 100 MHz; (3) MAE for PH = 30 minutes; (4) geometric mean of sensitivity & specificity for subject-specific classification; (5) calculated values; (6) calculated average of accuracy for the 3 classes; (7) operating voltage not reported; (8) top-3 inter-session accuracy.

the system on a 300 mAh battery, it was reported the system could continuously monitor epilepsy for 40.87 hours. The embedded ML model resulted in an average sensitivity of 96.6%, and specificity of 92.5% across all subjects. The presented wearable system referred to the use of a previously designed e-Glasss wearable [80].

The work in [47] presented another implementation of a seizure detection algorithm on MCU for use in future wearable system. The authors investigated the seizure detection algorithm considering different number of factors, such as the number of EEG channels used, the pre-processing window size for discrete wavelet transform(DWT), post-processing of classification results to minimize the reported false negatives, and finally the training/testing of ML models on global or subject specific use cases. The investigation was applied using four different supervised models: SVM, RF, Extra Trees, and AdaBoost, for final implementation on embedde physiological platform the Biowolf [81]. BioWolf is an ExG wearable device, composed of 8-channel AFE ((ADS1298) for signal acquisition, Mr. Wolf for processing and embedding ML algorithms, and an nRF52832 for communication. The CHB-MIT public dataset [99] was used for evaluation of the different algorithms in different use cases. The pre-processing of the data used a 4 level DWT comparing window sizes 2s, 4s, and 8s. The dataset had lower number of seizure epochs compared to non-seizure epochs. Therefore, in order to ''penalize the false alarm rate'', each class was given a weight equal to the inverse of its occurrence frequency. The four seizure classification models were then trained/tested under four scenarios based on the global/subject-specific, and 23 channels/4 temporal channels (F7-T7, T7-P7, F8-T8, T8-P8). Moreover, in order to smooth-out the detection of a seizure, a post processing technique, the moving average with 4s window was used. The four models using the four temporal EEG channels with an 8s window size were finally deployed onto the MCU (Mr. Wolf) for performance comparison. For subject-specific inference, RF, ET, and AB achieved 100% sensitivity and specificity with zero false positives, while SVM reported a 99.4% specificity with 2.7 false positives per hour. The RF, ET, and AB models had better performance than SVM in terms of time and energy per inference. The reported results are summarized in the given tables.

Heart rate monitoring system based on PPG readings was presented in [82]. The work used a publicly available dataset, the PPG-DaLiA: ''a PPG dataset for motion compensation and heart rate estimation in Daily Life Activities'' [100] to train and analyse regression models based on temporal convolutional network (TCN), the TEMPONet. Raw data from PPG sensor and tri-axial accelerometer (to mitigate for motion artifacts) collected from wrist, were fed to the regression model for heart rate estimation. Models were trained using Python 3.6, TensorFlow 1.14. The authors used an automatic Neural Architecture Search (NAS) tool starting with a single ''seed'' (TEMPONet) to produce a family of models for analysis. The MorphNet [110] was chosen as the NAS algorithm to help explore the search space for models with reduced complexity. The MorphNet reduces either model memory footprint, its number of Multiply-and-Accumulate (MAC) operations, or both together through optimizing the number of channels in each layer. Three models were highlighted for analysis within the work namely, BestMAE, BestSize, BestMCU. BestMAE model achieved lowest MAE of 5.30 beats per minutes (BPM) with 232k trainable parameters which exceeded available memory. BestSize reduced the number of parameters down to 5k with acceptable increase of MAE to 6.29 BPM. As for the BestMCU, the model presented a compromise between the first two to fit on the resource constrained MCU, achieving MAE of 5.64 BPM using 41.7k parameters. Both BestSize and BestMCU models were deployed on STM32L476RG development board for performance evaluation in both 32-bit floating point and 8-bit quantized format. Performance results are reported in Table 5 achieving an average power consumption of 12.1 mW per inference.

In addition to the above mentioned works focusing on systems in the context of physical health, work has also been reported in the context of mental health. A wearable smart bracelet, named InfiniWolf, for stress detection was presented in [46]. InfiniWolf (size of a coin) was assembled to contain two MCUs, the Nordic nRF52832 and Mr. Wolf, a pressure sensor, a 9-axis motion sensor, a microphone, an ECG/EMG and bioimpedance analog front-end (AFE) (Maxim MAX30001), a low power galvanic skin response (GSR) front-end, and 120 mAh LiPo battery. The bracelet also provided two energy harvesting sources based on thermal and solar energy, which eliminated the need for recharging of the LiPo battery. The system was tested in an application for stress detection based on collected readings from ECG and GSR, implementing a multi-layer perceptron (MLP) using fast artificial neural network (FANN) library [76]. The model was given five features as input to classify into one of the three output classes (stress, medium stress, no stress). The estimated size of the model was reported to be 14 KB. The trained model was deployed on both MCUs to demonstrate the energy efficiency of Mr. Wolf over nRF52832. It was shown that Mr. Wolf performs better. Hence final inference was done on Mr. Wolf, while the nRF52832 was used for the purpose of communication, monitoring of battery level, and support of data processing when needed. Mr. Wolf used in parallel computation using its 8 RI5CY cores provided a 4.9x speedup compared to a Cortex-M4F MCU during an inference, which required 6126 cycles at frequency of 100 MHz (equivalent to 61.26 μs) consuming 1.2 μJ of energy (around 20 mW).

Wearable systems for ambient assisted living (AAL) have also been reported employing edge inference in place of cloud inference. Specific intended uses included fall detection in elderly [83], [84], [85]. The fall detection system developed in [83] provided an offline real-time wearable hip-level belt with embedded ML for inference. It used the Bonsai algorithm trained in TensorFlow using the SisFall dataset [102],

and the SeeDot library for model quantization and deployment on MCU. The SisFall dataset is not labeled, therefore authors had to manually label events into fall or activities of daily living (ADL). The original data was recorded for 10-15 seconds long, which can not be evaluated on MCU. Therefore, the authors focused on training the model using the actual event at center of the input window. Choosing the right frame of the event (fall or ADL) was done based on maximum total variance of acceleration and was validated by recorded videos. The model was deployed on a TinyLily processor incorporating the ATMega328p MCU chosen for its sewable compact design (coin size). A compatible TinyLily ASL2002 module with tri-axial Bosch BMA250 accelerometer provided input for inference and a piezo buzzer was used for acoustic output when fall is detected. The prototype sewed the three components into the wearable belt. The inference model on MCU was tested on real-time acquired data from two people doing 9 different activities, 3 of which were fall. The Bonsai model was tested using different sampling frequencies, window sizes, and number of features taking into account memory constraint and model performance. The best performing model reported achieved an accuracy of 94.2% for window size of 3s using 4 features extracted from absolute acceleration and variance of the tri-axial accelerometer at a sampling frequency of 20 Hz.

The two other systems reported for fall detection [84], [85] chose a deep learning model for inference. Training using RNN was carried out with an enhanced version of the SisFall dataset [103], which was manually labeled to provide 3 output classes (fall, alert, and background). The work in [84] used an LSTM based RNN deployed on a STMicroelectonic device, the SensorTile. The latter has a STM32L476JGY MCU operating at 80 MHz. Beside the MCU, the tile includes two tri-axial accelerometers, gyroscope, magnetometer, and a barometer. Initially, the LSTM model was trained using TensorFlow, achieving an accuracy of 98% for fall detection, 90.93% for Alert, and 93.12% for background. The model was then deployed on the MCU without quantization, making use of the CMSIS-library of the ARM Cortex-M4 MCUs occupying 82 KB of the total 128 KB memory, and providing an inference within 300 ms for 1 second input window sampled at 100 Hz. Although the use of floating point representation of the LSTM model provided the same classification performance as on workstation (in terms of accuracy, sensitivity, and specificity), it was at the cost of higher memory occupancy and longer time for inference. The calculated consumed current for operating the 32-bit floating point representation of the trained LSTM model reached 5 mA. This allows 20 hours of operation using 100 mAh battery as reported. This number of hours would unavoidably be reduced as other indispensable electronic blocks were accounted in the system architecture.

As for the fall detection presented in [85], the work investigated the implementation of the best classification model on three levels: (1) the inputs (3D accelerometer, 3D gyroscope, or both); (2) the number of LSTM layers (1 or 2); and (3) the size of inner cells (4, 8, 16, 32). The use of accelerometer readings alone showed better performance than the gyroscope alone, while the use of both provided a noticeable increase in accuracy. The number of LSTM layers (1, 2) had very little influence on the results. Therefore, a single layer was chosen taking into consideration memory needs. As for the cell size, both 16 and 32 presented nearly similar results for best performance. Therefore, both inputs were used and the RNN with single LSTM layer and 16 cells was trained in TensorFlow. To minimize the effect of imbalanced classes, the authors used a weighted cross-entropy loss function. Each window contributing to the gradient is weighted by the inverse of its class size in training set. For the embedded design, the SensorTile was also chosen with an ARM Cortex M4F MCU, and the RNN model was deployed using CMSIS library without quantization keeping its 32 floating point format. Hence, the classification performance of the model was very close to that of the workstation with a reported mean squared numerical error in the order of 10^{-7}. The embedded model achieved an average accuracy of 93.52% across the three classes, occupying less than 18.5 KB of memory and requiring a processing time of 51 ms/window. It was also reported that a wearable device with minimal architecture is estimated to run for 132 hours using a 100 mAh battery. Moreover, an expected decrease in memory usage by 2.5 KB and of 1 ms in inference time was reported to be possible if only accelerometer readings were used, translating into 4 hours increase in battery life.

Other application of embedded ML in healthcare was demonstrated in fitness tracking. In [86], a wrist worn fitness tracker was designed to monitor the type of exercise and number of its repetitions. Three exercises were targeted (squat, curl, and push-ups). The wearable was designed using the STMicroelectronics SensorTile module, which incorporates the STM32L476 MCU, a Bluetooth low energy (BLE), LSM6DSM 581 (tri-axial accelerometer and tri-axial gyroscope) and other unused sensors. The device was used to collect real-time data for use in training and testing of an ML model. As summarized in Table 3, data was collected from 15 subjects at a sampling frequency of 20 Hz. The collected data was classified into four classes, having the fourth as not an exercise (NAE) for resting or the time between exercise. The NAE was used as an indication for start/end of an exercise, which helped in counting the number of repetitions. A window size of 3.4s (68 samples) was used as input with 6 readings (3 accelerometer, 3 gyroscope). A total of 700 repetitions were collected and split into 80% and 20% for training and testing. The trained CNN was tested in Keras at first (50 cases) reporting 100% accuracy. Then using X-CUBE-AI, the model was translated into MCU compatible format for another testing. Testing on the MCU was done using a total of 70 repetitions which were all correctly classified. However, the testing set was imbalanced having 6 curls, 15 push-ups, 13 squats, and 36 NAE. Having more NAE events was expected as resting between each exercise was classified as NAE. The model on MCU was also compressed

by factor of 8 reporting no effect on model accuracy. Results of embedding are reported in tables.

Another embedded system for exercise recognition was presented in [87]. Data collected from PPG and tri-axial accelerometer (to compensate for motion artifacts) were fed to a RNN model for classification. The RNN model was trained and tested using the publicly available PPG-ACC dataset [104]. Data records of 5 out of the total 7 subjects were used for training, while data for the remaining 2 subjects were used for testing of the model. Data pre-processing including cleaning and normalization, data down-sampling using simple decimation, and data augmentation, were all applied on the training data. The training data had an imbalanced record for the three classes (squats, stepper, and resting). Hence, authors used data augmentation, specifically oversampling method (duplicating records of less occurring classes) to balance the classes for improved training process. The performance of the classification model after down sampling using different decimation factors was reported. The best performing model was achieved using decimation factor of 40 i.e., sampling frequency of 10 Hz (original sampling frequency was 400 Hz) having 30 samples per window size of 3s. Information related to implementation and performance of final model on STM32L476RG are summarized in the given tables.

In the context of assistive devices for rehabilitation, multiple works have been proposed. The authors in [88] presented a wearable device to monitor EEG signals for motor imagery commands. A compact CNN algorithm, for use in EEG based brain computer interface (BCI) application, the EEGNet [111] was used. The EEGNet model was trained and validated on the Physionet EEG Motor Movement/Imagery dataset [97], [105]. From the original dataset, only motor imagery (MI) recordings of 105 subjects were used, each subject with 21 trials per class for each of the 4 motor imagery classes: left fist (L), right fist (R), both feet (F), and rest (0). The model was trained and tested using 5-fold cross validation for a global model and 4-fold cross validation for a subject specific transfer learning (SS-TL) model with an input window of 3s. The subject specific model made use of transfer learning from the global model. Initially, the global model is trained, then each subject within the test set has its trials split for a 4-fold cross validation and the model is then trained. Both presented models were compared to the baseline state of the art CNN model presented in [112] using same datasets, validation methods, and input size window. Performance under three cases was checked: 2 classes (L/R), 3 classes (L/R/0), and 4 classes (L/R/F/0). The global EEGNet model performed better than global CNN model in all three cases, with an increase in accuracy of 2.05%. 5.25%, and 6.49% for the 2, 3, 4 classes cases respectively. As for the SS-TL model, compared to the CNN SS-TL model, the 2-classes model presented a decrease in accuracy of 2.17%, while an increase by 0.82% and 2.32% for 3, and 4-class models. While comparison between global and SS-TL, the improvement provided by the SS-TL over global for the EEGNet model was not as significant as the improvement noticed in the CNN model. Therefore, the authors chose to proceed implementation using their global model. The model trained on Keras and TensorFlow was deployed in its 32-bit floating point representation onto 2 STM32 MCU boards using X-CUBE-AI package. These MCUs have an ARM Cortex-M processor; the first with Cortex-M4F (STM32L475VG) for lower power operation, while the other Cortex-M7 core (STM32F756 Nucleo 144) for higher processing capability. Due to memory constraints, the classification model was reduced in size by investigating the effect of temporal down sampling, reduced input window, and reduced channel numbers on model accuracy. Down sampling factors (ds) of 2 and 3 reported maximum decrease in accuracy of 0.32% and 1.25% respectively. Channel reduction from original 64 channels to 32, 19, and 8 channels were examined reporting a decrease in accuracy of 0.95%, 2.66% and 6.52% respectively. As for window size, 2s and 1s window resulted in accuracy reduction of 1.62% and 3.6% respectively. Two final models were deployed on the two MCUs. Down sampling factor of 3 and input of 32 EEG channels were used in both models. However, the input window for the first model was 1s (deployed on both MCUs), while the second model used 2s window and deployed only on the Cortex-M7 MCU due to limited memory of the Cortex-M4F MCU. The trade-off between speed and power consumption was clearly demonstrated in this work, where the choice of MCU depends on the system's intended use which would result in different design criteria prioritizing either speed or power.

Another proposed system focused on human-machine interaction for application in prosthetic control was presented in [89]. The wearable system was composed of eight-electrode surface-EMG (s-EMG) AFE (ADS1298) placed around the forearm and GAP8 MCU for hand gesture recognition. A novel deep learning model based on TCN topology, Temporal Embedded Muscular Processing Online Network (TEMPONet) was trained and tested on two datasets. The first was a publicly available dataset, the Non-Invasive Adaptive hand Prosthetics Database 6 (NinaPro DB6) [106]. The second was their own 20-sessions dataset collected using the wearable prototype. Relevant information of both datasets are summarized in Table 3. Incremental training protocol was used, splitting the first half of the datasets for training, and the second for testing, with 2-fold stratified cross validation. The resulting average accuracy of testing sessions are reported in Table 5. It was noticed the performance of TEMPONet on the NinaPro DB6 was much lower than the 20-sessions dataset. This was due to the fact the NinaPro dataset differentiates between different grasps, while the 20-session datasets between different gestures. Nevertheless, it was reported that the performance of TEMPONet on NinaPro DB6 presented an increase in accuracy of 7.8% compared to the state-of-the-art models. The trained TEMPONet was quantized to 8-bit representation after training using PULP-NN library for deployment on GAP8. An accuracy loss of 0.4% and 4.2% was reported due to quantization in comparison to the

floating-point model for the 20-session dataset and NinaPro DB6 dataset respectively. While an improvement in memory footprint resulted from quantization, resulting in its decrease by a factor of 4.

Unlike the conventional approach of gesture recognition through classification of sEMG signals into limited set of static predefined gestures, the work in [90] approached the problem with a regression model. The target of using regression model was to decode sEMG data into hand kinematic (joint angle), allowing a more natural control which can be applied in controlling prosthetic. The purpose of the work was to produce a regression model to fit embedded systems and maintain close performance to the state-of-the-art (SoA) hand kinematic regression models which cannot fit embedded systems. A TCN based model, the TEMPONet was trained and tested using the publicly available Non-Invasive Adaptive hand Prosthetics Database 8 (NinaPro DB8) dataset [107]. The dataset provided sEMG signals for decoding of finger position to be used in the estimation of hand kinematics rather than classification of gestures. The TCN model was implemented using PyTorch 1.6, using raw sEMG signals as input to the TEMPONet and using exponential moving average method for post processing of output. The final trained model deployed on GAP8 provided a MAE of 6.89° (0.15° better than SoA), while occupying 70.9 KB of memory and providing an inference within 4.76 ms consuming 0.243 mJ per inference.

Further application of embedded ML in prosthetic was presented in the work of [91]. The proposed TinyML system utilized embedded machine learning in the acquisition and processing of tactile information from sensor arrays arranged in hand shape. The proposed embedded system named SmartHand was designed for application in prosthetic and robotics that lack tactile feedback, for addition of smart sense of touch. The work reproduced a hand-shaped multi-array tactile sensor presented in [92] using a scalable tactile glove (STAG) for acquisition of tactile information. The presented system in [91] provides three modes of operation: (i) data collection; (2) real-time visualization through a graphic user interface (GUI); and (3) SmartHand embedded system. The data collection mode was used to collect 340,000 tactile frames (32×32 matrix) for 16 objects and empty hand during 5 sessions at 100 Hz with 40 seconds of data collection per object. The visualization mode provided an interface showing the collected frame and real-time classification result if required. The SmartHand system embedded a convolutional neural network based on modified ResNet-18 model for the on-board classification of acquired data. The model was trained in PyTorch before deployment. Results are reported in the given tables. The power consumption per inference was not reported, however the full system measured 505 mW including the IMU and MCU supplied at 3.3 V and a readout circuit at 5V.

The use of MCUs as the choice for edge inference as mentioned earlier is relatively new. Hence, most systems for health and care applications are based on academic research so far. However, one commercially available system that has successfully implemented its edge computation using Cortex-M4 MCU is the Amiko Respiro [113]. Although the device is not wearable, it is worth mentioning. Amiko Respiro uses sensors embedded with ML algorithms for smart monitoring of use of inhalers. It is an add-on sensor which fits to multiple commercially available inhalers. The smart sensor collects data related to vibrations from inhaler during use, information related to patient's inhalation frequency and time, and breathing pattern providing real time feedback. Furthermore, information related to lung capacity and inhalation techniques and other important parameters are calculated on device using the embedded ML algorithm. Moreover, the sensor comes with a smartphone app that patients can use to monitor their use of inhalers, and a professional dashboard for clinicians is also available if required for remote monitoring of patient's use.

V. DISCUSSION

The TinyML systems covered in this review had been designed for a variety of health and care applications. As shown in Fig. 4, some targeted the detection and management of medical conditions, while others focused on elderly fall detection, as well as fitness tracking and prosthetics for rehabilitation. This demonstrated the capability of utilizing TinyML for a wide range of intended uses. However, due to this difference in target application, it is difficult and unfair to compare systems against each other. Each application targeted a different problem, be it classification or regression, requiring the use of different algorithms, architectures, hyperparameters, and datasets which resulted in application-dependent algorithms. This defined the performance of the base model which then went through post-processing before deployment on MCU. Hence, the model's performance was initially influenced by the original floating point base model before deployment.

In the context of software tools and frameworks, the reviewed tools can be categorized based on their MCU compatibility as shown in Fig.5. TFLite, Microsoft EdgeML, and FANN-on-MCU support a wide range of MCUs. TFLite limits the MCU options to 32-bit platforms, while EdgeML provides compressed algorithms for use on any MCU. FANN-on-MCU provides current support for ARM and PULP based MCUs [46]. The other tools are more specific, targeting a certain family of MCU; such as CMSIS-NN for ARM based MCUs [58], [84], [85], while PULP-NN and NEMO for PULP based MCUs [45], [89], [90]. More specific targeted MCU software tools are X-CUBE-AI for use with only STM32 MCUs [45], [59], [82], [86], [87], [88], [91], GAPflow for GAP MCUs, and DORY with current support for only GAP8. Due to these compatibility concerns, the choice of MCU and software tools was co-dependent.

It was also noticed the target of most of these tools/frameworks is compression of neural network based architectures, while classical machine learning algorithms where not supported, this resulting on them not being used when classical ML algorithms were deployed on MCU

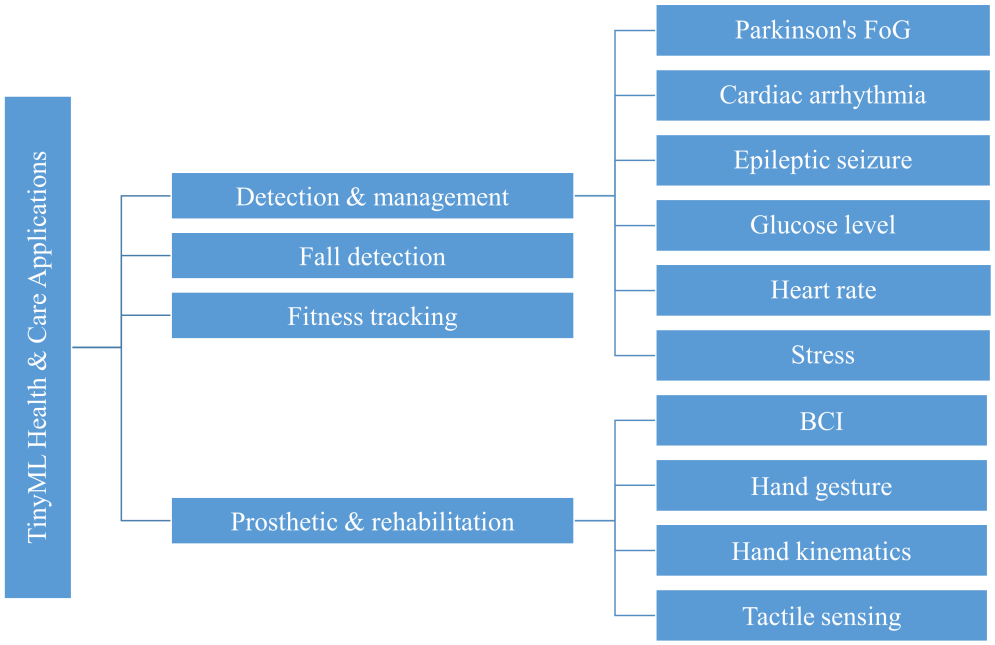

FIGURE 4. The different application areas of TinyML systems for health and care.

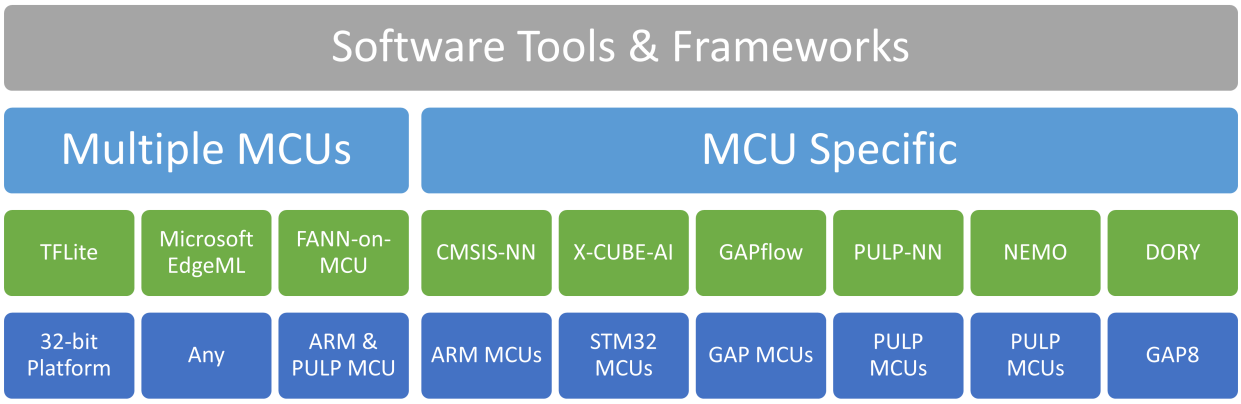

FIGURE 5. Categorization of software tools and frameworks based on their hardware MCU compatibility.

[47], [60]. The reason behind this exclusion could be related to the architecture complexity and intensive computational requirements of the neural networks in comparison to classical algorithms. Nevertheless, some works still used the classical ML algorithms through deployment of the compact version provided by Microsoft EdgeML ProtoNN [79] and Bonsai [83], in place of the conventional one. The use of these compacted algorithms achieved low memory footprint down to 1.4 KB [79]. Concurrently, it was noticed that AVR microcontrollers were only used with EdgeML algorithms due to incompatibility with other tools and frameworks. Other systems relying on ARM or PULP MCUs used other compatible tools. The effect of using different tools can be observed in the work of [45], where the use of NEMO/DORY against TFLite and GAPflow for deployment on GAP8 provided a speed up in inference time by a factor of 7.48, and 3.15 mW decrease in power/inference at the negligible cost of 1.6 KB increase in Flash occupancy and a 0.2% drop in accuracy. As for Cortex-M4 deployment, the use of X-CUBE-AI for STM32 MCU with TFLite versus the Keras floating-point model provided memory savings by a factor of $\times$3.16 and speed up in inference $\times$2.95 with a drop of 4.93 mW/inference at the cost of only 0.2% decrease in accuracy.

In terms of targeted MCUs, their percentage usage in Tiny ML health and care systems based on the core processor is shown in Fig. 6. More than half of the reviewed systems used ARM based MCUs, followed by RISC-V MCUs, and only tenth used AVR MCUs. 8-bit MCUs are more suitable for use in simpler operations as opposed to 32-bit MCUs which can

handle more complex computations. This is demonstrated by the lack of their use when deeper networks were deployed, as well as the lack of targeted software tools. As for the 32-bit MCUs, both ARM and RISC-V MCUs were used for deployment of complex networks for on-board inference. Although the RISC-V is fairly new, it was able to present itself as a competitor to the well-established ARM processors. One of the main reasons for this is likely to be the fact it is open source, allowing developers and vendors to use and develop it without royalty charges or license as opposed to the propriety ARM. The drawback of this, though, is the limited support that is offered in software and environment development when compared to the larger support community of ARM. On that note, 25% of MCUs in the reviewed work which used RISC-V was noticed to be authored by a research group which is a developer of the RISC-V MCU PULP architecture. Therefore, the distribution in Fig. 6 can be regarded as biased to user's familiarity and preference rather than a representation of processor capabilities.

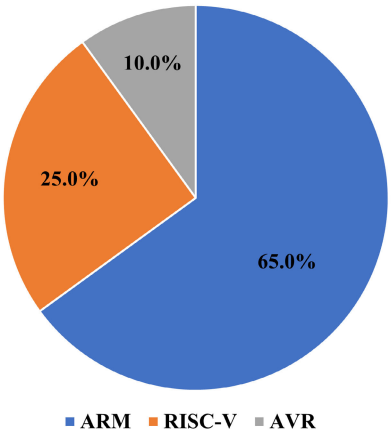

FIGURE 6. Percentage distribution of MCUs used in TinyML healthcare systems based on core processor type.

In terms of ARM processors, it was observed that only Cortex-M4 and Cortex-M7 were utilized in the TinyML healthcare applications. In some cases, this was also likely to be the result of the factors described above, since a wider range of ARM processors are available which in certain situations could also have been used. More specifically, ARM provides three main processor families, Cortex-A, Cortex-R, and Cortex-M (which is dedicated for microcontrollers). Within the Cortex-M processors, other than the M4 and M7 processors, there is also the M0, M0+, M1, M3, M23, M33, M35 and M55. A comparison of the different processors and their specifications is summarized in Table 6. Based on the ARM ISA, the processors can be divided into 3 groups: (a) Armv6-M for M0, M0+, M1; (b) Armv7-M for M3, M4 and M7; and (c) Armv8-M (Baseline and Mainline) for M23, M33, M35P, and M55. The Armv6-M processors provide less computational capability compared to M4 and M7, and among the three processor, only M0+ might be considered for use in TinyML applications. This is because, M1 processors were optimized especially for FPGA use, while M0+ is an optimized version of M0 providing better performance, and lower energy consumption, as well as an optional Memory Protection Unit (MPU) for task isolation. The M7, similar to the M4, has optional floating-point unit (FPU) and a digital signal processing (DSP) unit, but its performance is superior with the use of 6-stages instruction pipeline and the optional addition of instruction and data cache/TCM [114]. The M3 has a very close structure to the M4 but without the FPU and DSP extensions. In the case where neither the FPU nor the DSP units are needed the M3 processor could be used instead of the M4, as it occupies less area and consumes lower dynamic power.

The M23 with Armv8-M Mainline ISA, and M33, M35P, and M55 with Armv8-M Baseline ISA, advanced Cortex-M processors provide an optional security extension for software isolation, the ARM TrustZone technology, which can be beneficial in certain TinyML based healthcare applications where security is key. In addition to the software protection layer, the M35 further provides built-in physical protection against invasive and non-invasive physical attacks. Amongst the four processors, the M23 does not provide optional FPU nor DSP extension, making it the smallest and lowest power one with TrustZone security. Similar to the M7, the M55 further provides optional instruction and data cache/TCM, making it "Arm's most AI-capable Cortex-M processor" [114], [115]. With this wide range of options, it is clear that depending on the specific intended use, as well as performance and safety constraints different processors might be the most suitable design choice.

Apart from the MCU core processor, the memory limitations and clock speed of the MCU also played important roles in the overall system performance. One common goal pursued by all researchers was to maintain a balance between on-board performance, in terms of time/inference and power/inference, and memory constraints. The memory footprint was decided by the algorithm and its architecture, as well as the post-processing techniques used through the software tools. The effect of doubling the input window size was reflected in an approximate doubling of the RAM occupancy and the time/inference, with slight increase of Flash (0.51 KB) and an accuracy improvement of 2.25% [88]. The purpose of the algorithm also affected the memory footprint and the overall performance. This was observed in the implementation of the same algorithm (TEMPONet) for a 9 classes classification problem versus a regression problem, where the classification problem was more complex requiring $\times 6.5$ of Flash memory and $\times 2.7$ more time/inference although running at $\times 1.7$ higher clock speed than the regression problem [89], [90]. The effect of the algorithm's architecture and input size is further observed in the deployment of an RNN model on STM32L476 [84], [85], [87]. Although the input size was increased in [85], the removal of one of the LSTM layers and the decrease in hyperparameter values achieved a drop of factor 4.43 in RAM footprint with negligible loss in accuracy of 0.63% and speeded up inference time by

TABLE 6. Comparison table for the ARM Cortex-M processors [114], [116].

Feature	Cortex-M0	Cortex-M0+	Cortex-M1	Cortex-M3	Cortex-M4	Cortex-M7	Cortex-M23	Cortex-M33	Cortex-M35P	Cortex-M55
ISA	Armv6-M	Armv6-M	Armv6-M	Armv7-M	Armv7-M	Armv7-M	Armv8-M Baseline	Armv8-M Mainline	Armv8-M Mainline	Armv8-M Mainline
TrustZone for Armv8-M	✗	✗	✗	✗	✗	✗	✓(option)	✓(option)	✓(option)	✓(option)
FPU	✗	✗	✗	✗	SP (option)	SP, DP (option)	✗	SP (option)	SP (option)	HP, SP, DP (option)
DSP	✗	✗	✗	✗	✓	✓	✗	✓(option)	✓(option)	✓
Pipeline	3-stage	2-stage	3-stage	3-stage	3-stage	6-stage	2-stage	3-stage	3-stage	4-stage
CoreMark®/MHz	2.33	2.46	1.83	3.45	3.54	5.29	2.64	4.10	4.10	4.40
Maximum MPU regions	0	8	0	8	8	16	16	16	16	16
Instruction cache	✗	✗	✗	✗	✗	0-64 kB	✗	✗	2-16 kB	0-64 kB
Data cache	✗	✗	✗	✗	✗	0-64 kB	✗	✗	✗	0-64 kB
Instruction TCM	✗	✗	0-1 MB	✗	✗	0-16 MB	✗	✗	✗	0-16 MB
Data TCM	✗	✗	0-1 MB	✗	✗	0-16 MB	✗	✗	✗	0-16 MB
Dynamic power (µW/MHz)	66*	47.4*	-	141*	151*	58.5**	3.86**	12.0**	14.7**	-
Floor plan area (mm^2)	0.11*	0.098*	-	0.35*	0.44*	0.105**	0.0088**	0.028**	0.091**	-

SP: single precision; DP: double precision; HP: half precision.* 180ULL (7-track, typical 1.8V, 25°C). ** 40LP (9-track, typical 0.99V, -40°C).

almost a factor of 6. An intermediate compromise for the implementation of the RNN decreased the input size but with the addition of an extra LSTM layer to achieve ×6.8 drop in memory requirement and two times increase in inference speed [87] compared to [84]. Other than the algorithm, the clock speed directly affected the performance, where a factor of 2.7 increase in clock speed achieved a speedup of ×2.69 in inference, with a cost of ×3.14 increase in power [88]. The role of the processor in the overall performance was noted in the use of Cortex-M4 versus M7 with higher performance capabilities, where a speed up in time/inference of ×1.83 was achieved but at a cost of tripled power consumption [88]. The use of RISC-V MCU with ×1.56 clock speed against a Cortex-M4 MCU, provided a faster inference by a factor of 7.7 at a power cost increase of ×1.8.

The overall performance of the TinyML system in the context of health and care applications is affected by different factors. A few points can be summarized and deducted from the above observations. First, the algorithm's architecture and application (classification/regression) will directly affect the memory footprint and accuracy of the model. The input size also affects the model's performance, where enough samples are needed for an inference, but too many will affect the RAM occupancy and the time/inference. Hence, the first challenge is to train and develop a well performing model, finding a balance between the input size and architecture's hyperparameters. This, in its turn, brings with it challenges related to the use of ML in healthcare such as the availability and reliability of datasets to train the base model. The following challenge would be the compression of the model whilst keeping a balance between accuracy loss and required memory footprint. This is conditioned to the post processing techniques and software tools used which are also constrained by the choice of MCU. Hence, the choice of MCU and software tools is a co-dependent process. Choosing the most suitable MCU for an application is on its own a non-trivial decision, considering the number of choices available. In order to find a balance between performance, memory, speed, and power a list of prioritized specifications needs to be set at first.

As TinyML is still a growing field, constant developments and advances are surfacing each day to address these challenges and provide an easier path for the implementation of TinyML systems. One of these advances are the TinyML development platforms which provide a bridge between the development and training of the ML algorithm and its deployment on embedded systems. These platforms incorporate some of the libraries and tools previously reviewed such as TFLite, and CMSIS-NN, and provide compatibility with multiple commercially available boards; allowing designers to collect data, train, test, optimize, and deploy their model on different MCUs using a single stop. Some of these newly developed platforms are Edge Impulse, imagimob, Qeexo, NanoEdge AI, OctoML Apache TVM, and others. These platforms are worth exploring as they provide a smooth starting point for beginners and can be used to

compare different implementation scenarios and assess their trade-offs.

VI. CONCLUSION

The emerging field of TinyML technology, deploying machine learning algorithms on resource-constraints embedded devices, can provide several advantages over cloud computing related to latency, power, and privacy. In this work, we explored the application of TinyML in wearable health and care systems. Several health and care applications including detection and management of medical conditions, fitness tracking, elderly fall detection, and rehabilitation using prosthetics were found in this review, which demonstrate the feasibility of TinyML applications. The TinyML pipeline, covering the chosen MCUs and software tools and frameworks were also covered and discussed within the review. This included the specifications and limitations of available hardware options, related to MCU core processor, clock speed, and available memory, as well as the features and hardware compatibility constraints of the software tools and frameworks. A comprehensive review of the TinyML wearable health and care systems was later presented and summarized for further analysis and discussion, in terms of system implementation, design constraints, and performance metrics related to memory occupancy, accuracy, speed, and power. This showed that the overall performance of TinyML systems depends on the software tools and choice of MCU, which was on its turn also conditioned by the target application and ML problem at hand. Overall, this review covers a gap in literature in terms of the application of TinyML in wearable health and care devices and can help system designers by guiding them through their initial system design decisions and trade-offs.

DATA ACCESS STATEMENT

Not applicable. There is no data associated with the article.

REFERENCES

[1] Statista. *Wearables | Statista*. Accessed: Jun. 8, 2022. [Online]. Available: https://www.statista.com/study/15607/wearables-statista-dossier/

[2] M. Wu and J. Luo, "Wearable technology applications in healthcare: A literature review," *Online J. Nursing Informat.*, vol. 23, no. 3, 2019. [Online]. Available: https://www.himss.org/resources/wearable-technology-applications-healthcare-literature-review

[3] F. Sabry, T. Eltaras, W. Labda, K. Alzoubi, and Q. Malluhi, "Machine learning for healthcare wearable devices: The big picture," *J. Healthcare Eng.*, vol. 2022, pp. 1–25, Apr. 2022.

[4] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, Jan. 2017.

[5] Z. Jiang, F.-F. Yin, Y. Ge, and L. Ren, "A multi-scale framework with unsupervised joint training of convolutional neural networks for pulmonary deformable image registration," *Phys. Med. Biol.*, vol. 65, no. 1, Jan. 2020, Art. no. 015011.

[6] T. Apiparakoon, N. Rakratchatakul, M. Chantadisai, U. Vutrapongwatana, K. Kingpetch, S. Sirisalipoch, Y. Rakvongthai, T. Chaiwatanarat, and E. Chuangsuwanich, "MaligNet: Semisupervised learning for bone lesion instance segmentation using bone scintigraphy," *IEEE Access*, vol. 8, pp. 27047–27066, 2020.

[7] Z. Zhao, J. Zhao, K. Song, A. Hussain, Q. Du, Y. Dong, J. Liu, and X. Yang, "Joint DBN and Fuzzy C-means unsupervised deep clustering for lung cancer patient stratification," *Eng. Appl. Artif. Intell.*, vol. 91, Jun. 2020, Art. no. 103571.

[8] D. Arefan, A. A. Mohamed, W. A. Berg, M. L. Zuley, J. H. Sumkin, and S. Wu, "Deep learning modeling using normal mammograms for predicting breast cancer risk," *Med. Phys.*, vol. 47, no. 1, pp. 110–118, Jan. 2020.

[9] M. Anthimopoulos, S. Christodoulidis, L. Ebner, A. Christe, and S. Mougiakakou, "Lung pattern classification for interstitial lung diseases using a deep convolutional neural network," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1207–1216, Jun. 2016.

[10] L. Y. W. Tang, H. O. Coxson, S. Lam, J. Leipsic, R. C. Tam, and D. D. Sin, "Towards large-scale case-finding: Training and validation of residual networks for detection of chronic obstructive pulmonary disease using low-dose CT," *Lancet Digit. Health*, vol. 2, no. 5, pp. e259–e267, May 2020.

[11] J. Park, J. Yun, N. Kim, B. Park, Y. Cho, H. J. Park, M. Song, M. Lee, and J. B. Seo, "Fully automated lung lobe segmentation in volumetric chest CT with 3D U-Net: Validation with intra- and extra-datasets," *J. Digit. Imag.*, vol. 33, no. 1, pp. 221–230, Feb. 2020.

[12] H.-H. Chen, C.-M. Liu, S.-L. Chang, P. Y.-C. Chang, W.-S. Chen, Y.-M. Pan, S.-T. Fang, S.-Q. Zhan, C.-M. Chuang, Y.-J. Lin, L. Kuo, M.-H. Wu, C.-K. Chen, Y.-Y. Chang, Y.-C. Shiu, S.-A. Chen, and H. H.-S. Lu, "Automated extraction of left atrial volumes from two-dimensional computer tomography images using a deep learning technique," *Int. J. Cardiol.*, vol. 316, pp. 272–278, Oct. 2020.

[13] A. Ghorbani, D. Ouyang, A. Abid, B. He, J. H. Chen, R. A. Harrington, D. H. Liang, E. A. Ashley, and J. Y. Zou, "Deep learning interpretation of echocardiograms," *NPJ Digit. Med.*, vol. 3, no. 1, pp. 1–10, Dec. 2020.

[14] S. W. Baalman, F. E. Schroevers, A. J. Oakley, T. F. Brouwer, W. van der Stuijt, H. Bleijendaal, and L. A. Ramos, "A morphology based deep learning model for atrial fibrillation detection using single cycle electrocardiographic samples," *Int. J. Cardiol.*, vol. 316, pp. 130–136, Oct. 2020.

[15] U. R. Acharya, H. Fujita, S. L. Oh, Y. Hagiwara, J. H. Tan, and M. Adam, "Application of deep convolutional neural network for automated detection of myocardial infarction using ECG signals," *Inf. Sci.*, vol. 415, pp. 190–198, Nov. 2017.

[16] K. T. Oh, S. Lee, H. Lee, M. Yun, and S. K. Yoo, "Semantic segmentation of white matter in FDG-PET using generative adversarial network," *J. Digit. Imag.*, vol. 33, no. 4, pp. 1–10, 2020.

[17] S. Saha, A. Pagnozzi, P. Bourgeat, J. M. George, D. Bradford, P. B. Colditz, R. N. Boyd, S. E. Rose, J. Fripp, and K. Pannek, "Predicting motor outcome in preterm infants from very early brain diffusion MRI using a deep learning convolutional neural network (CNN) model," *NeuroImage*, vol. 215, Jul. 2020, Art. no. 116807.

[18] Q. Dou, H. Chen, L. Yu, L. Zhao, J. Qin, D. Wang, V. C. Mok, L. Shi, and P.-A. Heng, "Automatic detection of cerebral microbleeds from MR images via 3D convolutional neural networks," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1182–1195, May 2016.

[19] J. Xu, D. Xu, Q. Wei, and Y. Zhou, "Automatic classification of male and female skeletal muscles using ultrasound imaging," *Biomed. Signal Process. Control*, vol. 57, Mar. 2020, Art. no. 101731.

[20] A. Jodeiri, R. A. Zoroofi, Y. Hiasa, M. Takao, N. Sugano, Y. Sato, and Y. Otake, "Fully automatic estimation of pelvic sagittal inclination from anterior-posterior radiography image using deep learning framework," *Comput. Methods Programs Biomed.*, vol. 184, Feb. 2020, Art. no. 105282.

[21] Y. Song, L. Zhang, S. Chen, D. Ni, B. Lei, and T. Wang, "Accurate segmentation of cervical cytoplasm and nuclei based on multiscale convolutional network and graph partitioning," *IEEE Trans. Biomed. Eng.*, vol. 62, no. 10, pp. 2421–2433, Oct. 2015.

[22] K. Sirinukunwattana, S. E. A. Raza, Y.-W. Tsang, D. R. J. Snead, I. A. Cree, and N. M. Rajpoot, "Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1196–1206, May 2016.

[23] F. Piccialli, V. D. Somma, F. Giampaolo, S. Cuomo, and G. Fortino, "A survey on deep learning in medicine: Why, how and when?" *Inf. Fusion*, vol. 66, pp. 111–137, Feb. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253520303651

[24] C. M. Micheel, S. J. Nass, and G. S. Omenn, "Omics-based clinical discovery: Science, technology, and applications," in *Evolution of Translational Omics: Lessons Learned and the Path Forward*. Washington, DC, USA: National Academies Press, 2012, pp. 34–37.

[25] L. Greco, G. Percannella, P. Ritrovato, F. Tortorella, and M. Vento, "Trends in IoT based solutions for health care: Moving AI to the edge," *Pattern Recognit. Lett.*, vol. 135, pp. 346–353, Jun. 2020.
[26] M. P. Véstias, R. P. Duarte, J. T. de Sousa, and H. C. Neto, "Moving deep learning to the edge," *Algorithms*, vol. 13, no. 5, p. 125, 2020.
[27] R. Sanchez-Iborra and A. F. Skarmeta, "TinyML-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits Syst. Mag.*, vol. 20, no. 3, pp. 4–18, 3rd Quart., 2020.
[28] M. Rahimiazghadi, C. Lammie, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, and G. Indiveri, "Hardware implementation of deep network accelerators towards healthcare and biomedical applications," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 6, pp. 1138–1159, Nov. 2020.
[29] P. Jawandhiya, "Hardware design for machine learning," *Int. J. Artif. Intell. Appl.*, vol. 9, no. 1, pp. 63–84, Jan. 2018.
[30] L. Du and Y. Du, "Hardware accelerator design for machine learning," in *Machine Learning-Advanced Techniques and Emerging Applications*. Rijeka, Croatia: IntechOpen, 2017.
[31] M. P. Véstias, "Processing systems for deep learning inference on edge devices," in *Convergence of Artificial Intelligence and the Internet of Things*. Cham, Switzerland: Springer, 2020.
[32] T. S. Ajani, A. L. Imoize, and A. A. Atayero, "An overview of machine learning within embedded and mobile devices–optimizations and applications," *Sensors*, vol. 21, no. 13, p. 4412, Jun. 2021.
[33] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 22180–225134, 2020.
[34] G. Rong, A. Mendez, E. B. Assi, B. Zhao, and M. Sawan, "Artificial intelligence in healthcare: Review and prediction case studies," *Engineering*, vol. 6, no. 3, pp. 291–301, 2020.
[35] R. Zemouri, N. Zerhouni, and D. Racoceanu, "Deep learning in the biomedical applications: Recent and future status," *Appl. Sci.*, vol. 9, no. 8, p. 1526, Apr. 2019.
[36] A. M. Hafiz and G. M. Bhat, "A survey of deep learning techniques for medical diagnosis," in *Information and Communication Technology for Sustainable Development*. Cham, Switzerland: Springer, 2020, pp. 161–170.
[37] A. Qayyum, J. Qadir, M. Bilal, and A. Al-Fuqaha, "Secure and robust machine learning for healthcare: A survey," *IEEE Rev. Biomed. Eng.*, vol. 14, pp. 156–180, 2020.
[38] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2018.
[39] M. Hartmann, U. S. Hashmi, and A. Imran, "Edge computing in smart health care systems: Review, challenges, and research directions," *Trans. Emerg. Telecommun. Technol.*, vol. 33, no. 3, p. e3710, Mar. 2022.
[40] Y. Wei, J. Zhou, Y. Wang, Y. Liu, Q. Liu, J. Luo, C. Wang, F. Ren, and L. Huang, "A review of algorithm & hardware design for AI-based biomedical applications," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 2, pp. 145–163, Apr. 2020.
[41] D. L. Dutta and S. Bharali, "TinyML meets IoT: A comprehensive survey," *Internet Things*, vol. 16, Dec. 2021, Art. no. 100461.
[42] *RISC-V*. Accessed: Nov. 20, 2021. [Online]. Available: https://riscv.org/
[43] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr.Wolf: An energy-precision scalable parallel ultra low power SoC for IoT edge processing," *IEEE J. Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, Jul. 2019.
[44] *Greenwaves Technologies. Gap8 Next Generation Processor for Smart Sensors*. Accessed: Jul. 8, 2021.[Online]. Available: https://greenwaves-technologies.com/gap8_smart_sensors/
[45] T. M. Ingolfsson, X. Wang, M. Hersche, A. Burrello, L. Cavigelli, and L. Benini, "ECG-TCN: Wearable cardiac arrhythmia detection with a temporal convolutional network," in *Proc. IEEE 3rd Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2021, pp. 1–4.
[46] M. Magno, X. Wang, M. Eggimann, L. Cavigelli, and L. Benini, "InfiniWolf: Energy efficient smart bracelet for edge computing with dual source energy harvesting," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 342–345.
[47] T. M. Ingolfsson, A. Cossettini, X. Wang, E. Tabanelli, G. Tagliavini, P. Ryvlin, L. Benini, and S. Benatti, "Towards long-term non-invasive monitoring for epilepsy via wearable EEG devices," in *Proc. IEEE Biomed. Circuits Syst. Conf. (BioCAS)*, Oct. 2021, pp. 01–04.
[48] *STM32l476xx*, DS10198 Datasheet Rev 8, STMicroelectronics, Geneva, Switzerland, 2019.
[49] *STM32l475xx*, DS10969 Datasheet Rev 5, STMicroelectronics, Geneva, Switzerland, 2019.
[50] *STM32F303xD STM32F303xE*, DocID026415 Datasheet Rev 5, STMicroelectronics, Geneva, Switzerland, 2016.
[51] *STM32F756xx*, DocID027589 Datasheet Rev 4, STMicroelectronics, Geneva, Switzerland, 2016.
[52] *STM32F765xx STM32F767xx STM32F768xx STM32F769xx*, DS11532 Datasheet Rev 7, STMicroelectronics, Geneva, Switzerland, 2021.
[53] *nRF52832*, Product Specification V1.8, Nordic Semiconductor, Trondheim, Norway, 2021.
[54] *nRF52840*, Product Specification V1.7, Nordic Semiconductor, Trondheim, Norway, 2021.
[55] *ATmega 640/V-1280/V-1281/V-2560/V-2561/V*, DS40002211A Datasheet, Microchip Technology Incorporated, Chandler, AZ, USA, 2020.
[56] *ATmega48A/PA/88A/PA/168A/PA/328/P*, DS40002061B Datasheet, Microchip Technology Incorporated, Chandler, AZ, USA, 2020.
[57] R. Chéour, S. Khriji, and O. Kanoun, "Microcontrollers for IoT: Optimizations, computing paradigms, and future directions," in *Proc. IEEE 6th World Forum Internet of Things (WF-IoT)*, Nov. 2020, pp. 1–7.
[58] A. Faraone and R. Delgado-Gonzalo, "Convolutional-recurrent neural networks on low-power wearable platforms for cardiac arrhythmia detection," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 153–157.
[59] T. Zhu, L. Kuang, K. Li, J. Zeng, P. Herrero, and P. Georgiou, "Blood glucose prediction in type 1 diabetes using deep learning on the edge," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2021, pp. 1–5.
[60] R. Zanetti, A. Aminifar, and D. Atienza, "Robust epileptic seizure detection on wearable systems with reduced false-alarm rate," in *Proc. 42nd Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Jul. 2020, pp. 4248–4251.
[61] Microchip. *8-Bit AVR MCUS*. Accessed: Nov. 20, 2021. [Online]. Available: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus
[62] PULP. (2019). *Pulp Hardware Reference Manual, Version 1.6*. Accessed: Jul. 8, 2021. [Online]. Available: https://github.com/pulp-platform/pulp/blob/master/doc/datasheet.pdf
[63] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for AI-enabled IoT devices: A review," *Sensors*, vol. 20, no. 9, p. 2533, 2020.
[64] U. Kulkarni, S. M. Meena, S. V. Gurlahosur, P. Benagi, A. Kashyap, A. Ansari, and V. Karnam, "AI model compression for edge devices using optimization techniques," in *Modern Approaches in Machine Learning and Cognitive Science: A Walkthrough: Latest Trends in AI*, vol. 2, V. K. Gunjan and J. M. Zurada, Eds. Cham, Switzerland: Springer, 2021, pp. 227–240, doi: 10.1007/978-3-030-68291-0_17.
[65] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Tech. Rep., 2009. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.222.9220
[66] TensorFlow. *TensorFlow Lite*. Accessed: Jun. 22, 2021. [Online]. Available: https://www.tensorflow.org/lite/guide
[67] *TensorFlow Lite for Microcontrollers*. Accessed: Jun. 22, 2021. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers
[68] N. Suda, *Machine Learning on Arm Cortex-M Microcontrollers*, ARM, Cambridge, U.K., 2019, pp. 1–10.
[69] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm Cortex-M CPUs," 2018, *arXiv:1801.06601*.
[70] STMicroelectronics. *X-Cube-AI, AI Expansion Pack for STM32cubemx*. Accessed: Jun. 22, 2021. [Online]. Available: https://www.st.com/en/embedded-software/x-cube-ai.html
[71] D. K. Dennis, S. Gopinath, C. Gupta, A. Kumar, A. Kusupati, S. G. Patil, and H. V. Simhadri. *EdgeML: Machine Learning for Resource-Constrained Edge Devices*. [Online]. Available: https://github.com/Microsoft/EdgeML
[72] GreenWavesTechnologies. *NN Quick Start Guide*. Accessed: Nov. 30, 2021. [Online]. Available: https://greenwaves-technologies.com/sdk-manuals/nn_quick_start_guide/
[73] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: A computing library for quantized neural network inference at the edge on RISC-V based parallel ultra low power clusters," in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 33–36.

[74] F. Conti, "Technical report: NEMO DNN quantization for deployment model," 2020, *arXiv:2004.05930*.
[75] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "DORY: Automatic end-to-end deployment of real-world DNNs on low-cost IoT MCUs," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1253–1268, Aug. 2021.
[76] 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.*, vol. 7, no. 5, pp. 4403–4417, Feb. 2020.
[77] S. Nissen, "Implementation of a fast artificial neural network library (FANN)," Dept. Comput. Sci., Univ. Copenhagen (DIKU), Copenhagen, Denmark, Tech. Rep. 31, 2003, vol. 31, p. 29.
[78] B. Kuyumcu. *Fann Tool*. Accessed: Nov. 30, 2021. [Online]. Available: https://code.google.com/archive/p/fanntool/
[79] H. Gokul, P. Suresh, B. H. Vignesh, R. P. Kumaar, and V. Vijayaraghavan, "Gait recovery system for Parkinson's disease using machine learning on embedded platforms," in *Proc. IEEE Int. Syst. Conf. (SysCon)*, Aug. 2020, pp. 1–8.
[80] D. Sopic, A. Aminifar, and D. Atienza, "e-Glass: A wearable system for real-time detection of epileptic seizures," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2018, pp. 1–5.
[81] V. Kartsch, G. Tagliavini, M. Guermandi, S. Benatti, D. Rossi, and L. Benini, "BioWolf: A sub-10-mW 8-channel advanced brain–computer interface platform with a nine-core processor and BLE connectivity," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 5, pp. 893–906, Oct. 2019.
[82] M. Risso, A. Burrello, D. J. Pagliari, S. Benatti, E. Macii, L. Benini, and M. Pontino, "Robust and energy-efficient PPG-based heart-rate monitoring," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.
[83] L. Oden and T. Witt, "Fall-detection on a wearable micro controller using machine learning algorithms," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, Sep. 2020, pp. 296–301.
[84] E. Torti, A. Fontanella, M. Musci, N. Blago, D. Pau, F. Leporati, and M. Piastra, "Embedded real-time fall detection with deep learning on wearable devices," in *Proc. 21st Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2018, pp. 405–412.
[85] M. Musci, D. De Martini, N. Blago, T. Facchinetti, and M. Piastra, "Online fall detection using recurrent neural networks on smart wearable devices," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1276–1289, Jul. 2021.
[86] M. Merenda, M. Astrologo, D. Laurendi, V. Romeo, and F. G. Della Corte, "A novel fitness tracker using edge machine learning," in *Proc. IEEE 20th Medit. Electrotechnical Conf. (MELECON)*, Jun. 2020, pp. 212–215.
[87] M. Alessandrini, G. Biagetti, P. Crippa, L. Falaschetti, and C. Turchetti, "Recurrent neural network for human activity recognition in embedded systems using ppg and accelerometer data," *Electronics*, vol. 10, no. 14, p. 1715, 2021.
[88] X. Wang, M. Hersche, B. Tömekce, B. Kaya, M. Magno, and L. Benini, "An accurate EEGNet-based motor-imagery brain–computer interface for low-power edge computing," in *Proc. IEEE Int. Symp. Med. Meas. Appl. (MeMeA)*, Jun. 2020, pp. 1–6.
[89] M. Zanghieri, S. Benatti, A. Burrello, V. Kartsch, F. Conti, and L. Benini, "Robust real-time embedded EMG recognition framework using temporal convolutional networks on a multicore IoT processor," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 2, pp. 244–256, Apr. 2020.
[90] M. Zanghieri, S. Benatti, A. Burrello, V. J. K. Morinigo, R. Meattini, G. Palli, C. Melchiorri, and L. Benini, "SEMG-based regression of hand kinematics with temporal convolutional networks on a low-power edge microcontroller," in *Proc. IEEE Int. Conf. Omni-Layer Intell. Syst. (COINS)*, Aug. 2021, pp. 1–6.
[91] X. Wang, F. Geiger, V. Niculescu, M. Magno, and L. Benini, "SmartHand: Towards embedded smart hands for prosthetic and robotic applications," in *Proc. IEEE Sensors Appl. Symp. (SAS)*, Aug. 2021, pp. 1–6.
[92] S. Sundaram, P. Kellnhofer, Y. Li, J.-Y. Zhu, A. Torralba, and W. Matusik, "Learning the signatures of the human grasp using a scalable tactile glove," *Nature*, vol. 569, no. 7758, pp. 698–702, 2019.
[93] M. Bachlin, M. Plotnik, D. Roggen, I. Maidan, J. M. Hausdorff, N. Giladi, and G. Troster, "Wearable assistant for Parkinson's disease patients with the freezing of gait symptom," *IEEE Trans. Inf. Technol. Biomed.*, vol. 14, no. 2, pp. 436–446, Mar. 2010.
[94] G. Clifford, C. Liu, B. Moody, L.-W. Lehman, I. Silva, Q. Li, A. Johnson, and R. Mark, "AF classification from a short single lead ECG recording: The PhysioNet computing in cardiology challenge 2017," in *Proc. Comput. Cardiol. Conf. (CinC)*, Sep. 2017, pp. 1–4.
[95] Y. Chen, Y. Hao, T. Rakthanmanon, J. Zakaria, B. Hu, and E. Keogh, "A general framework for never-ending learning from time series streams," *Data Mining Knowl. Discovery*, vol. 29, no. 6, pp. 1622–1664, Nov. 2015.
[96] D. S. Baim, W. S. Colucci, E. S. Monrad, H. S. Smith, R. F. Wright, A. Lanoue, D. F. Gauthier, B. J. Ransil, W. Grossman, and E. Braunwald, "Survival of patients with severe congestive heart failure treated with oral milrinone," *J. Amer. College Cardiol.*, vol. 7, no. 3, pp. 661–670, 1986.
[97] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, Jun. 2000.
[98] C. Marling and R. Bunescu, "The OhioT1DM dataset for blood glucose level prediction: Update 2020," in *Proc. CEUR Workshop*, vol. 2675, 2020, p. 71.
[99] A. H. Shoeb, "Application of machine learning to epileptic seizure onset detection and treatment," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, USA, 2009.
[100] A. Reiss, I. Indlekofer, P. Schmidt, and K. Van Laerhoven, "Deep PPG: Large-scale heart rate estimation with convolutional neural networks," *Sensors*, vol. 19, no. 14, p. 3079, Jul. 2019.
[101] J. A. Healey and R. W. Picard, "Detecting stress during real-world driving tasks using physiological sensors," *IEEE Trans. Intell. Transp. Syst.*, vol. 6, no. 2, pp. 156–166, Jun. 2005.
[102] A. Sucerquia, J. D. López, and J. F. Vargas-Bonilla, "SisFall: A fall and movement dataset," *Sensors*, vol. 17, no. 1, p. 198, Jan. 2017.
[103] *Sisfall Temporally Annotated*. Accessed: Oct. 18, 2021. [Online]. Available: https://bitbucket.org/unipv_cvmlab/sisfalltemporallyannotated/src/master/
[104] G. Biagetti, P. Crippa, L. Falaschetti, L. Saraceni, A. Tiranti, and C. Turchetti, "Dataset from PPG wireless sensor for activity monitoring," *Data Brief*, vol. 29, Apr. 2020, Art. no. 105044.
[105] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "BCI2000: A general-purpose brain-computer interface (BCI) system," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 1034–1043, Jun. 2004.
[106] F. Palermo, M. Cognolato, A. Gijsberts, H. Müller, B. Caputo, and M. Atzori, "Repeatability of grasp recognition for robotic hand prosthesis control based on sEMG data," in *Proc. Int. Conf. Rehabil. Robot. (ICORR)*, Jul. 2017, pp. 1154–1159.
[107] A. Krasoulis, S. Vijayakumar, and K. Nazarpour, "Effect of user practice on prosthetic finger control with an intuitive myoelectric decoder," *Frontiers Neurosci.*, vol. 13, p. 891, Sep. 2019.
[108] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma, and P. Jain, "ProtoNN: Compressed and accurate kNN for resource-scarce devices," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1331–1340.
[109] J. G. Nutt, B. R. Bloem, N. Giladi, M. Hallett, F. B. Horak, and A. Nieuwboer, "Freezing of gait: Moving forward on a mysterious clinical phenomenon," *Lancet Neurol.*, vol. 10, no. 8, pp. 734–744, Aug. 2011.
[110] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "MorphNet: Fast & simple resource-constrained structure learning of deep networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1586–1595.
[111] V. Lawhern, A. Solon, N. Waytowich, S. M. Gordon, C. Hung, and B. J. Lance, "EEGNet: A compact convolutional neural network for EEG-based brain–computer interfaces," *J. Neural Eng.*, vol. 15, no. 5, 2018, Art. no. 056013.
[112] H. Dose, J. S. Møller, H. K. Iversen, and S. Puthusserypady, "An end-to-end deep learning approach to MI-EEG signal classification for BCIs," *Expert Syst. Appl.*, vol. 114, pp. 532–542, Dec. 2018.
[113] P. Ferguson. (2018). *This AI-Powered Asthma Inhaler Keeps Tabs on Your Dose so You Don't Have to*. [Online]. Available: https://www.arm.com/blogs/blueprint/ai-asthma-inhaler-keeps-tabs-dose
[114] *Arm Cortex-M Processor Comparison Table*, ARM, Cambridge, U.K., 2022.

[115] Arm Developer. *Cortex-m55*. Accessed: Jun. 8, 2022. [Online]. Available: https://developer.arm.com/Processors/Cortex-M55
[116] Arm Developer. *Processors-Microcontrollers*. Accessed: Jun. 8, 2022. [Online]. Available: https://developer.arm.com/Processors#aq=%40navigationhierarchiescategories%3D%3D%22Processor%20products%22%20AND%20%40navigationhierarchiescontenttype%3D%3D%22Product%20Information%22&numberOfResults=48&f[navigationhierarchiesprocessortype]=Microcontrollers

MAHA S. DIAB (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees (Hons.) in electrical and electronics engineering from the University of Sharjah, Sharjah, United Arab Emirates, in 2016 and 2019, respectively. She is currently pursuing the Ph.D. degree with the Wearable Technologies Laboratory, Department of Electrical and Electronic Engineering, Imperial College London, supported by the Department's Ph.D. Scholarship. She was awarded the Graduate Research Assistantship for her master's degree. She worked with the Mixed Analog-Digital Smart Electronics Circuits and Systems (MADSECS) Research Group, University of Sharjah, as a Graduate Research Assistant (2016–2019). She continued working as a Research Assistant after graduation with the MADSECS Research Group, University of Sharjah (2019–2020). Her current research interests include biomedical circuits and systems and machine learning focused towards low-power wearable medical technology.

ESTHER RODRIGUEZ-VILLEGAS is currently a Professor (Chair) of low-power electronics at Imperial College London, originally known for her engineering techniques to drastically reduce power in integrated circuits. She subsequently focused her research on life-science applications, founding the Wearable Technologies Laboratory. This laboratory specializes on both: creating innovative wearable medical technologies to improve management and diagnosis of chronic diseases; and neural interfaces to facilitate brain research whilst improving animals' welfare. She is also a Founder and a Co-CEO/CSO, of two active life-sciences companies, such as Acurable and TainiTec. She was elected as a fellow of the U.K. Royal Academy of Engineering, in 2020. She has received many international recognitions and awards, including an IET Innovation Award in 2009, a Global XPRIZE-Award in 2014, an AAALAC 3Rs Award in 2018, and a Silver Medal of the U.K. Royal Academy of Engineering in 2020. She was also named the Top Scientist/Engineer in Spain under the age of 36 in 2009 (Complutense Award). She has served in many prestigious international technical committees, including, but not limited, to the Administrative Committee of the IEEE Solid State Circuit Society, IEEE ISSCC, IEEE ISCAS, and IEEE ESSCIRC.