applied sciences

MDPI

Article

Implementing Deep Neural Networks on ARM-Based Microcontrollers: Application for Ventricular Fibrillation Detection

Vessela Krasteva, Todor Stoyanov and Irena Jekova *

Institute of Biophysics and Biomedical Engineering, Bulgarian Academy of Sciences, Acad. G. Bonchev Str. Bl 105, 1113 Sofia, Bulgaria; vessika@biomed.bas.bg (V.K.); todor@biomed.bas.bg (T.S.)
* Correspondence: irena@biomed.bas.bg

Featured Application: This study describes the workflow for deploying deep neural networks on Raspberry Pi and ARM Cortex microcontrollers, providing inference for ventricular fibrillation detection that simulates real-time rhythm analysis in automated external defibrillators.

Abstract: GPU-based deep neural networks (DNNs) are powerful for electrocardiogram (ECG) processing and rhythm classification. Although questions often arise about their practical application in embedded systems with low computational resources, few studies have investigated the associated challenges. This study aims to show a useful workflow for deploying a pre-trained DNN model from a GPU-based development platform to two popular ARM-based microcontrollers: Raspberry Pi 4 and ARM Cortex-M7. Specifically, a five-layer convolutional neural network pre-trained in TensorFlow (TF) for the detection of ventricular fibrillation is converted to Lite Runtime (LiteRT) format and subjected to post-training quantization to reduce model size and computational complexity. Using a test dataset of 7482 10 s cardiac arrest ECGs, the inference of LiteRT DNN in Raspberry Pi 4 takes about 1 ms with a sensitivity of 98.6% and specificity of 99.5%, reproducing the TF DNN performance. An optimization study with 1300 representative datasets (RDSs), including 10 to 4000 calibration ECG signals selected by random, rhythm, or amplitude-based criteria, showed that choosing a random RDS with a relatively small size of 80 resulted in a quantized integer LiteRT DNN with minimal quantization error. The inference of both non-quantized and quantized LiteRT DNNs on a low-resource ARM Cortex-M7 microcontroller (STM32F7) shows rhythm accuracy deviation of <0.4%. Quantization reduces internal computation latency from 4.8 s to 0.6 s, flash memory usage from 40 kB to 20 kB, and energy consumption by 7.85 times. This study ensures that DNN models retain their functionality while being optimized for real-time execution on resource-constrained hardware, demonstrating application in automated external defibrillators.

Keywords: ECG processing; deep learning; CNN; VF detection; AED; edge computing; Raspberry Pi; ARM Cortex; quantization; representative dataset; TensorFlow; LiteRT for microcontrollers

Academic Editor: Keun-Chang Kwak

Received: 16 January 2025
Revised: 10 February 2025
Accepted: 11 February 2025
Published: 13 February 2025

Citation: Krasteva, V.; Stoyanov, T.; Jekova, I. Implementing Deep Neural Networks on ARM-Based Microcontrollers: Application for Ventricular Fibrillation Detection. *Appl. Sci.* **2025**, *15*, 1965. https://doi.org/10.3390/app15041965

1. Introduction

1.1. Deep Neural Networks

Deep neural networks (DNNs) are powerful computational models designed to capture complex, nonlinear, or hidden relationships between inputs and outputs through

layered architectures composed of various types, such as convolutional, normalization, pooling, dense, and long short-term memory (LSTM) layers [1,2]. DNNs require less human intervention and are more flexible than traditional machine learning methods, which often demand extensive expertise in tasks like feature extraction and engineering. Each layer refines the mapping between input and output data, but greater precision increases the model complexity.

The DNN detection efficiency comes with trade-offs, including the increased computational demand, significant energy consumption, extended execution time, and substantial memory usage. Deploying DNN models from graphical processing unit (GPU) or central processing unit (CPU) development environments to resource-constrained devices presents a significant challenge. In medical applications, a common workaround involves offloading computations to fog or cloud computing environments, either on local networks (fog) or remote servers (cloud). However, traditional cloud computing approaches, where large streams of computational tasks are processed remotely, can introduce unacceptable delays in real-time scenarios due to network congestion and latency issues [3]. Moreover, both cloud and fog computing require sending medical data for processing, which is associated with increased privacy and security risks [4].

1.2. Strategies for Edge Computing

Recently, tiny machine learning has emerged as a promising approach for implementing DNNs on microcontrollers [5–8]. Despite limited resources, this concept enables real-time, energy-efficient processing on embedded devices powered by microcontrollers, enhancing privacy and reducing reliance on external computing resources. The migration from fog or cloud computing to on-device computation is commonly referred to as edge computing [9], relying on two main strategies:

1. Optimization of DNN architectures during the design phase. This process maximizes the accuracy on a target dataset, while adhering to specific constraints, such as embedding size, kernel/stride size, hidden layer size, the number of model parameters, and the storage space required for intermediate computations—all of which significantly impact the memory requirements [10,11]. Quantization-aware optimization is performed by using specialized frameworks [12,13] that simulate the quantization (e.g., mapping weights and activations to 8-bit integers) but compute gradients in floating-point to preserve training accuracy. This approach ensures that the final model keeps its accuracy while deployed on the real device [14].
2. Application of compression techniques to pre-trained DNN models, such as post-training quantization, binarization, pruning, and clustering [8]. Post-training quantization freezes a pre-trained DNN and reduces memory usage and latency by converting floating-point parameters to lower-precision integers, often with minimal accuracy loss [15–18]. Binarization, an extreme form of quantization, reduces weights and activations to a single bit for maximum compression [19,20]. Pruning eliminates unnecessary parameters, either through structured pruning, which removes channels or filters to improve inference speed at some accuracy cost, or unstructured pruning, which zeroes out individual weights with minimal accuracy impact [21,22]. Clustering replaces similar weights with a smaller set of common centroid values, significantly reducing model size while preserving performance [23].

1.3. Deep Neural Networks in Embedded Systems

The optimized DNN models can be embedded into application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs) or integrated system-on-chip (SoC) circuits. While ASICs designed for CNN inference present high efficiency, their

development demands significant design effort and high costs, limiting their accessibility for widespread deployment [24]. In contrast, FPGAs have emerged as a flexible alternative, supporting various biomedical engineering and medical diagnostic applications [25–27].

Many embedded SoCs prefer Acorn RISC Machine (ARM) microprocessors due to their architecture being optimized for low-power applications [24]. ARM-based SoCs are becoming more powerful, thanks to the hardware floating-point unit in processors post-M4F [8] and advancements in firmware and frameworks for DNN model implementation on STM32 microcontrollers. Notable tools include the open source library Cortex Microcontroller Software Interface Standard Neural Network (CMSIS-NN) for ARM Cortex-M processors [28], LiteRT for microcontrollers [29], µTensor for ARM architectures [30], PyTorch Mobile for iOS and Android mobile devices [31], and the X-Cube-AI package for STM32 microcontrollers [32].

1.4. Deep Neural Networks for Electrocardiogram Signal Processing

Advancements in wearable devices with integrated biosensors have enabled heart rhythm diagnostics using single-lead electrocardiograms (ECGs). One-dimensional (1D) convolutional neural networks (CNNs) are widely used for classifying heartbeats [33,34], short-term arrhythmias [35–46], and noise affecting interpretation [35–38]. CNN models can distinguish various rhythms like normal sinus, atrial fibrillation [41–46], atrial flutter, and ventricular fibrillation [39,40] and show promise in detecting conditions like heart failure [47] and myocardial infarction [48,49]. Despite high accuracy, most ECG diagnostic CNN models are tested on GPU-based development platforms.

A few studies present ECG processing on FPGA platforms for wearable devices using various neural architectures, such as 1D CNNs [50,51], probabilistic neural networks [52], artificial neural networks [53], spiking neural networks [54], multilabel deep CNNs [55], and sparse CNNs [56]. Recently, CNN models for ECG analysis have been embedded in ARM-based SoCs through (i) the CMSIS-NN library on Nordic nRF52832 with an ARM Cortex-M4 processor [57] and (ii) LiteRT for microcontrollers on ARM Cortex-M4F using X-CUBE-AI [58], Raspberry Pi 3 (ARMv8) [7], and ARM Cortex-M4 (nRF52840) [59]. Studies also explored ECG classifications using optimized 2D CNN and LSTM models with STM32Cube.AI on ARM Cortex cores [60,61]. A real-time myocardial infarction detection neural network was also developed for a Cortex-M4 microcontroller [13]. A key aspect of all referenced studies involving embedded systems is their focus on providing detailed information about their specific implementations.

The focus of this study is on a critical application of 1D CNNs, detecting cardiac arrest rhythms during out-of-hospital cardiopulmonary resuscitation using ECG signals from defibrillator pads. Numerous studies demonstrate that CNNs can effectively distinguish between pulseless and pulsatile rhythms [62] and accurately detect shockable rhythms, including ventricular fibrillation, under various conditions, including the following: artefact-free ECGs [63–67], ECGs with chest compression artifacts [68–72], and ECGs during continuous cardiopulmonary resuscitation with ongoing chest compressions and insufflations, without the need to identify the artifacts [73]. Although DNN technologies have the potential for improving current resuscitation practices, they are mostly tested offline with databases in GPU-based development platforms. One recent study [74] took a further step toward embedding computationally efficient deep CNN models in the setting of a digitally connected automated external defibrillator (AED). The authors reported performance evaluated under specific embedded system constraints; however, they do not present details for the actions needed for the implementation of the CNN in the AED embedded microcontroller.

1.5. Research Objectives

This study aims to show the complete workflow for deploying a pre-trained DNN model from a GPU-based development platform to two popular ARM-based microcontrollers: Raspberry Pi 4 and ARM Cortex-M7. The focus is on DNN conversion and post-training quantization to create a model that is lightweight, faster, and less resource consuming. A key challenge addressed is the optimization of the representative dataset size and content for pre-inference calibration, ensuring minimal information loss during quantization. Performance is evaluated in terms of accuracy, latency, and memory usage, highlighting the feasibility of real-time ECG analysis on embedded systems. The workflow is applied to a high-performing CNN model for detection of ventricular fibrillation during out-of-hospital cardiac arrest [67]. This approach can be extended to numerous published DNN models optimized for rhythm analysis during cardiopulmonary resuscitation. Thus, we make the global link from GPU-based deep learning to AED deployment of critical ECG processing algorithms that can provide enhanced performance and improve resuscitation practices.

2. Materials and Methods

2.1. Methodological Concept

Efficient ECG signal processing algorithms based on diverse DNN architectures can be created using the functionality of the TensorFlow (TF) library within the Python Integrated Development Environment (IDE). The design, optimization, training, and testing of these DNN models typically benefit from the computational power of GPU-based workstations. Once trained, these models can be employed in development environments for in silico studies or deployed on edge devices to support in vivo applications. Implementing trained TF models on edge devices; however, needs a conversion process. This process ensures that the models retain their functionality while being optimized for execution on resource-constrained hardware. Figure 1 illustrates the main steps of the conversion pipeline for DNN models followed in this study:

1. Loading a trained TF DNN model into the development environment.
2. Conversion to Lite Runtime (LiteRT) format, producing lightweight models for efficient execution on devices with limited processing power and memory.
3. Post-training quantization, optimizing LiteRT DNNs to significantly reduce model size and computational complexity while keeping accuracy loss to a minimum.
4. Implementation in ARM-based microcontroller boards:
    - Raspberry Pi 4: The LiteRT Lite model is executed using the TensorFlow Lite Runtime library in Python, ensuring compatibility with the device's Linux-based operating system and Python environment.
    - ARM-based CPU systems: The model is adapted to LiteRT for Microcontrollers, a specialized library in C++ designed for resource-constrained environments, ensuring compatibility with embedded systems and standalone microcontroller platforms.

Details for each step are explained in the following methodological subsections.

2.2. Loading Trained DNN Model

This paper considers a TF DNN model trained for detection of ventricular fibrillation in a previous deep learning study [67]. To enable easy implementation in stand-alone AEDs, simplicity has been prioritized by using a minimal input of a single ECG lead, acquired at a low sampling rate (125 Hz) directly through the defibrillation pads, without pre-filtering or the use of additional sensors. Furthermore, the feature extraction and classification process

is streamlined through fully convolutional operations. The convolutional architecture of the DNN model embedded in this study (Figure 2) includes five convolutional layers with rectified linear unit (ReLU) activation, five max-pooling layers, and a fully connected output layer with sigmoid activation. The output provides the probability of ventricular fibrillation (pVF) within a range of 0 to 1, where ventricular fibrillation is detected if the probability exceeds a threshold of 0.5.

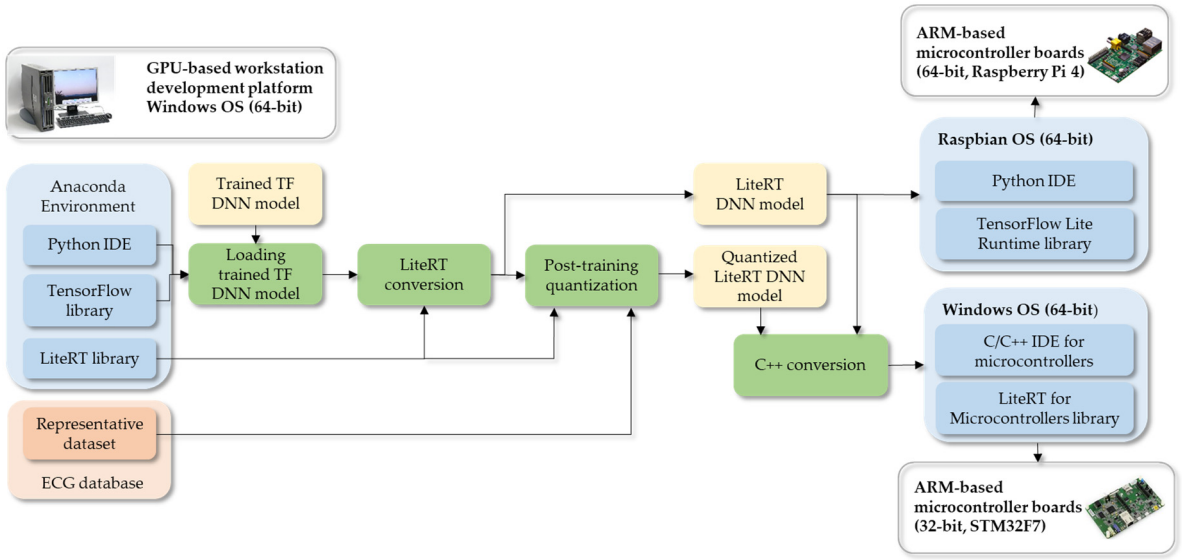

Figure 1. Flowchart of the methodological steps, showing the conversion and implementation of DNN models in microcontroller boards. OS: operating system; IDE: Integrated Development Environment; DNN: deep neural network; TF: TensorFlow; LiteRT: Lite Runtime; ARM: Acorn RISC (reduced instruction set computer) machine; GPU: graphical processing unit.

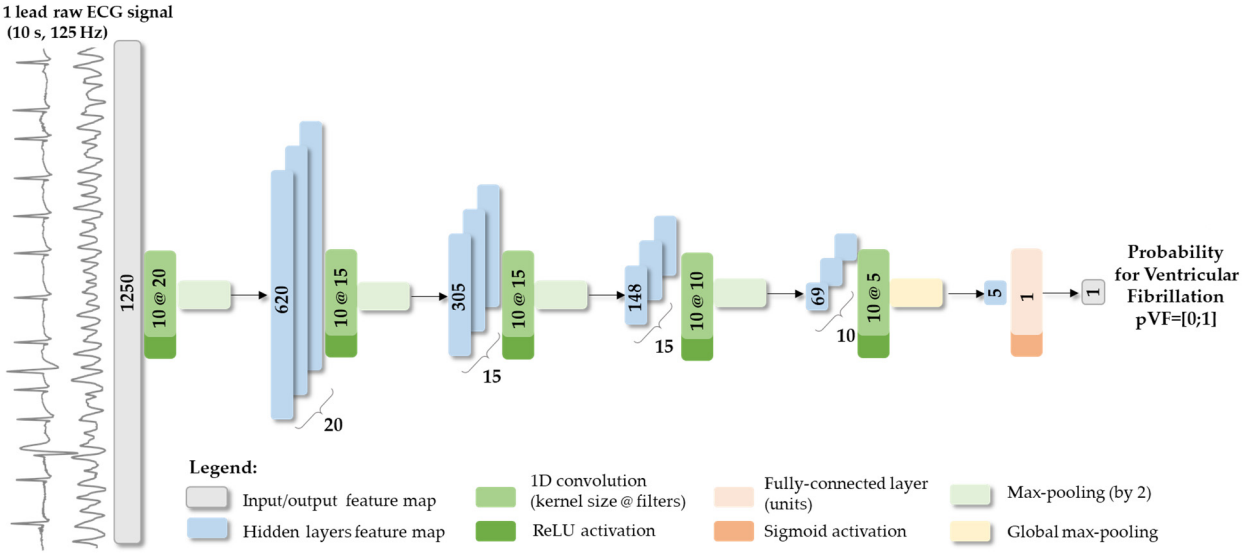

Figure 2. Architecture of the trained TF DNN model for detection of ventricular fibrillation, as selected from the optimization study [67]. The input layer processes a 10 s single-lead ECG signal sampled at 125 Hz, while the output layer consists of a single neuron presenting the probability of ventricular fibrillation (pVF). The hidden layer feature map dimensions are shaped by 1D convolution with valid padding, influenced by the kernel size. This model has a total number of 7521 parameters.

It is noteworthy that the hyperparameters of the trained DNN model presented in Figure 2 were determined through computationally intensive grid search optimization among more than 4200 architectures [67]. This process required substantial resources on

the TensorFlow-enabled development workstation, taking significant computational power and extensive input data from public Holter ECG databases and out-of-hospital cardiac arrest AED databases, spanning over 10,000 training and validation samples. We selected a trained DNN model with the largest input size, corresponding to a 10 s ECG processing interval, to evaluate the maximum resource requirements for executing the best performing models on the target test platforms.

2.3. LiteRT Conversion

The trained TF DNN model should be converted to a smaller and resource-efficient format compatible with the limited memory and computational power of microcontroller-based devices. This can be achieved by converting the TF model to a LiteRT model (short for Lite Runtime, formerly known as TensorFlow Lite) by means of the standard LiteRT runtime library, which is Google's high-performance runtime for on-device artificial intelligence (AI) [75].

The TF DNN model used in this study has a fully convolutional architecture, which is compatible with the LiteRT format and can be directly converted to an optimized flatbuffer format (identified by the .tflite file extension). In our case, we do not need any additional resources for refactoring or usage of advanced conversion techniques [76].

2.4. Post-Training Quantization

Post-training quantization is a technique used to reduce LiteRT model size, making it lighter, faster, and less resource consuming. The LiteRT converter provides several post-training quantization options, as outlined in Google AI Edge's guide for deploying AI on mobile, web, and embedded platforms [77]. We followed the decision tree in Figure 3 to identify different quantization methods applicable for our use case on 32-bit or 8-bit microcontrollers. As shown in Table 1, these methods differ in the size of the tensors used: constant tensors (weights and biases), which remain unchanged during inference, determine the LiteRT model size, while variable tensors (inputs, outputs, and activations—intermediate results during forward propagation through the network) additionally influence the processing time.

Table 1. Data types of various tensors used in LiteRT DNN models with and without post-training quantization. RDS: representative dataset for calibration.

LiteRT DNN Model	RDS Use	Inputs	Outputs	Activations	Weights	Biases
Without quantization	No	float32	float32	float32	float32	float32
Dynamic Range Quantization (DynQ)	No	float32	float32	float32	int8	float32
Integer Quantization (IntQ)	Yes	float32	float32	int8	int8	float32
Full-Integer Quantization (Full-IntQ)	Yes	int8	int8	int8	int8	int8

Figure 3 and Table 1 help to summarize the basic information on the LiteRT conversion methods used in this study:

1. LiteRT models without quantization retain full model precision, typically using 32-bit floating-point (float32) data types. The inference speed is the slowest, with the largest model size and highest resource usage.
2. Dynamic range quantization (DynQ) reduces the LiteRT model size by converting the weights from float32 to 8-bit integers (int8). Since weights are constant tensors that remain unchanged during inference, their quantization scale is dynamically computed at runtime. This quantization is partial—biases, inputs, outputs, and activation streams remain in float32, which ensures minimal accuracy loss. Although

the model size is reduced, the floating-point activation computations result in high resource usage.

3. Integer Quantization (IntQ) with float fallback is a hybrid approach where the LiteRT model uses integer quantization for internal computations (activations) but keeps the inputs/outputs as float32 streams. This is particularly useful when working with floating-point inputs or data formats that require high diagnostic precision, such as sensor inputs (e.g., ECG signals). The biases remain in float32 format because they typically involve small adjustments that require high precision to avoid error accumulation, which could otherwise degrade the model's performance. Nevertheless, the weights and internal activations are quantized to int8 to reduce memory usage and computational requirements. Since internal activations are variable tensors, their float32-to-int8 conversion requires careful pre-inference calibration, as it is likely to impact model accuracy.
4. Full-Integer Quantization (Full-IntQ) involves quantizing all tensors of the LiteRT model entirely to 8-bit integers, minimizing memory usage and computational requirements, and achieving the fastest inference speed. The float32-to-int8 conversion of all variable tensors requires adequate pre-inference calibration because the accuracy of the Full-IntQ model is most significantly impacted by quantization.

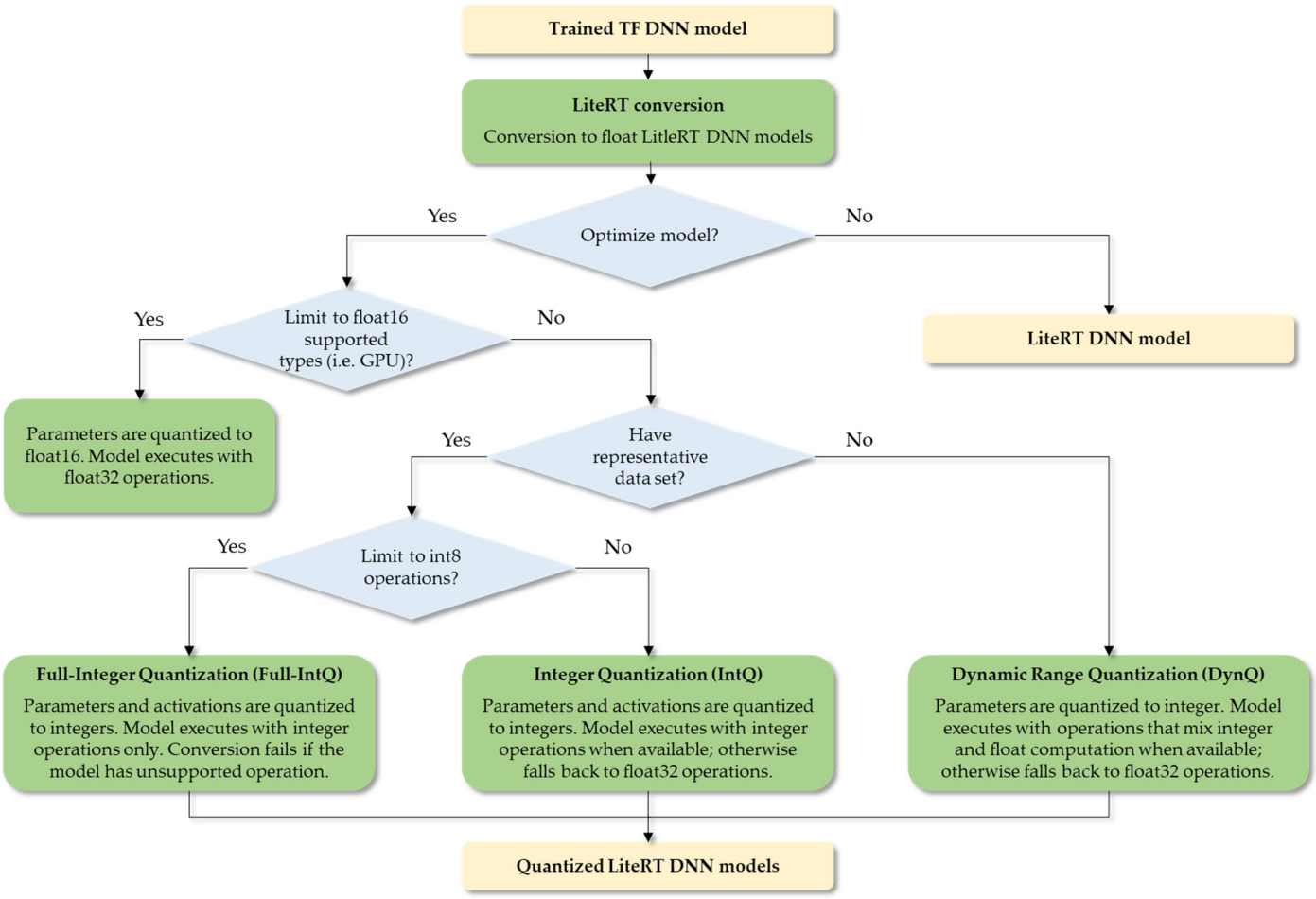

Figure 3. Flowchart of the conversion of LiteRT DNN models with and without post-training quantization. The description of the quantization methods is according to [77].

Notably, none of the above conversion methods require retraining, fine-tuning, or the use of training datasets or pipelines from the source framework. However, two of the integer quantizations (IntQ and Full-IntQ) require calibration of the variable tensors before the

inference with a representative dataset (RDS). RDS is used to collect statistical information on the maximum and minimum values of each layer's activations, ensuring minimal information loss during quantization. Two key parameters are estimated during the calibration process—scale factor and zero point. They are used in the following conversions:

- Conversion float32-to-int8 values:

$$\text{value}_{\text{int8}} = \frac{\text{value}_{\text{float32}}}{\text{scale\_factor}_{\text{float32}}} + \text{zero\_point}_{\text{int8}} \quad (1)$$

- Conversion int8-to-float32 values:

$$\text{value}_{\text{float32}} = \left(\text{value}_{\text{int8}} - \text{zero\_point}_{\text{int8}}\right).\text{scale\_factor}_{\text{float32}} \quad (2)$$

Notably, the standard library function for LiteRT conversion automatically calculates and applies the calibration parameters to the hidden activations in both IntQ and Full-IntQ models. Scarce details for this conversion are available on LiteRT 8-bit quantization specification [78]. Note that the users of Full-IntQ models must manually apply conversions to the input and output data streams if they are used as floating-point values in the diagnostic framework —specifically, (1) converting the ECG signal input and (2) the probability output for ventricular fibrillation detection (pVF), as defined in the example model in Figure 2. In this manual conversion, the scale factors and zero points of both the input and output streams can be retrieved from the Full-IntQ model's header.

2.5. Target Test Platforms

2.5.1. ARM-Based Microcontroller Board (Raspberry Pi 4)

Raspberry Pi 4 is a single board computer based on CPU ARM quad core cortex-A72 (ARM v8), 64-bit SoC @ 1.8 GHz, with 8 GB of Synchronous Dynamic Random-Access Memory (SDRAM) and installed Raspbian OS (64-bit) [79]. It is able to run LiteRT DNN models using the Tensor Flow Lite Runtime library in Python IDE—Figure 1.

The test interface to Raspberry Pi 4 embedded LiteRT DNN models is shown in Figure 4. It uses a GPU-based workstation platform like a master and Raspberry Pi like a slave. The master application run under Python IDE in Anaconda environment reads the raw ECG data (signed 16-bit) and sends them to the Raspberry Pi 4 board via standard serial communication (USB port). The slave Python application stores the received data frames into the memory (SDRAM) and feeds them to the input of the preloaded instance of the LiteRT DNN model. Then, the model is run and returns its result after a certain processing time. The result is sent back to the master application for comparison with the ground truth.

2.5.2. ARM-Based Microcontroller Board (STM32F7)

Our STM32F7 board is based on the STM32F769IDISCOVERY kit, which is a complete development platform for the STMicroelectronics Arm® Cortex®-M7 core-based STM32F769NI microcontroller with 2 MB of flash memory, 512 kB of Synchronous Random-Access Memory (SRAM), and 16 MB of SDRAM. The microcontroller has a Floating-Point Unit module and a Flexible Memory Controller for SDRAM with a data bus width of 32-bits and maximum CPU Clock of 216 MHz [80]. STM32F7 is able to run C++ LiteRT DNN models using the Lite RT library for microcontrollers [29] embedded in the firmware. The firmware is developed, compiled, and stored into the flash memory by means of C/C++ IDE for microcontrollers (STM32CubeIDE) [81]—Figure 1.

The test interface to STM32F7 embedded C++ LiteRT DNN models is shown in Figure 4. It uses the same communication with the master test application as described

for the Raspberry Pi 4 above. The data are received in the slave STM32F7 via Universal Synchronous Asynchronous Receiver Transmitter (USART) port and are stored in the SDRAM by Direct Memory Access (DMA). The firmware feeds the data to the input of the preloaded instance of the C++ DNN model. Then, the model is run and returns its result after a certain processing time. The result is sent back to the master application for comparison with the ground truth.

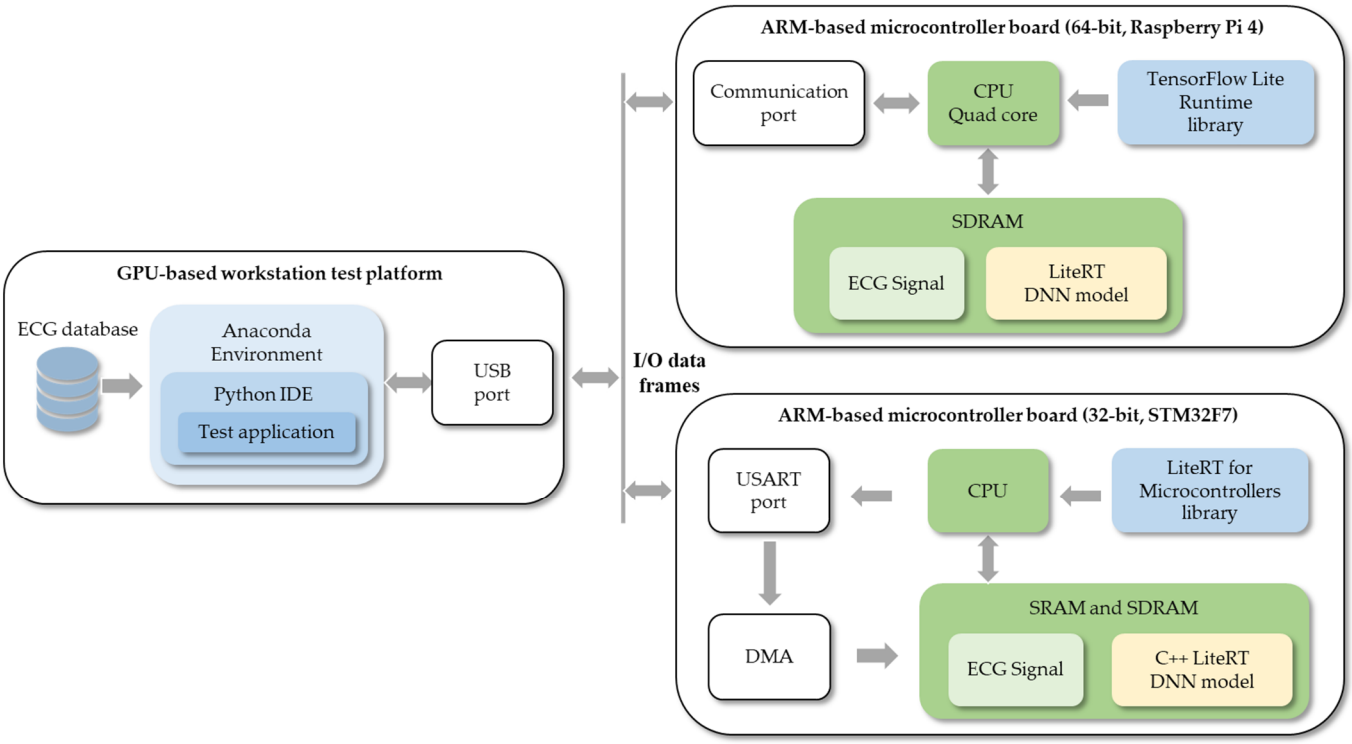

Figure 4. Test interface of ARM-based microcontroller boards (slave) connected to GPU-based workstation test platform (master). ECG: Electrocardiogram; I/O: Input/Output; IDE: Integrated Development Environment; USB: Universal Serial Bus; USART: Universal Synchronous Asynchronous Receiver Transmitter; CPU: Central Processing Unit; DMA: Direct Memory Access; SDRAM: Synchronous Dynamic Random-Access Memory; SRAM: Synchronous Random-Access Memory; LiteRT DNN: Lite Runtime Deep Neural Network.

By specification, SRAM is faster than SDRAM; therefore, our firmware prioritizes SRAM for storing critical components, such as the instance of the class "MicroInterpreter" from the LiteRT library for microcontrollers, as well as the C++ LiteRT DNN model parameters. However, as SRAM has limited capacity and is also shared with other modules (e.g., USART, flexible memory controller, etc.), the efficient use of SDRAM is essential. In this implementation, a significant portion of SDRAM is allocated to support the "MicroInterpreter" operations with the DNN model parameters, as well as to store the input, output, and temporal data. An additional section of SDRAM is reserved for the operation of the DMA module, ensuring efficient data handling across the system.

2.6. Optimization and Test Strategy

Figure 5 illustrates the deployment of each DNN model in this study for testing on its respective target platform: the TF DNN on a GPU-based workstation, the LiteRT DNN on both the Raspberry Pi 4 and STM32F7 microcontroller, and the quantized LiteRT models (DynQ, IntQ and Ful-IntQ) on the STM32F7 microcontroller. Before testing, two of the quantized LiteRT models (IntQ and Ful-IntQ) undergo an optimization process to calibrate

their variable tensors for accurate quantization. The materials and methods used in each process of Figure 5 are detailed in the following subsections.

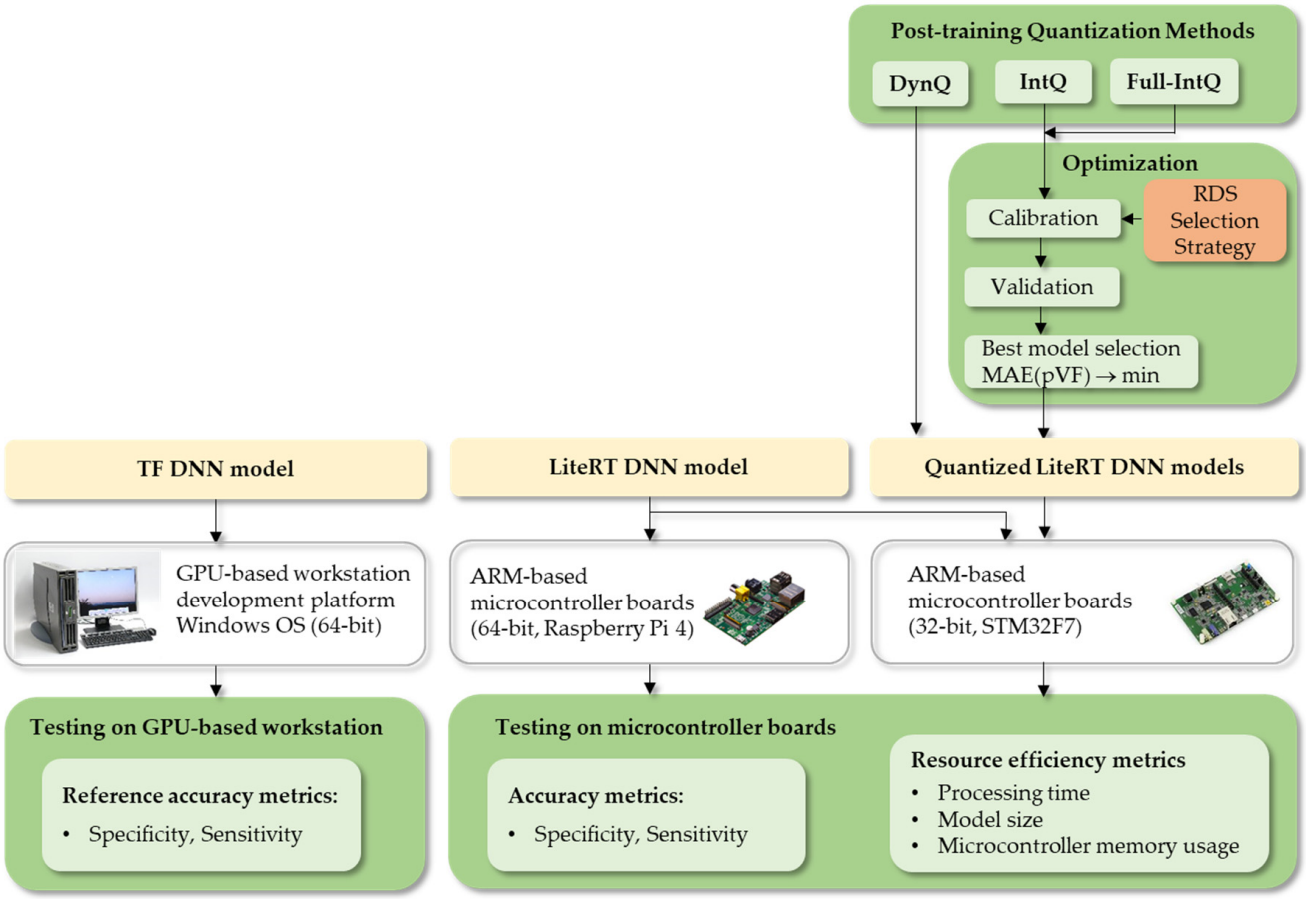

Figure 5. Optimization and test workflow of the DNN models in the target platforms. DynQ: dynamic range quantization; IntQ: integer quantization; Full-IntQ: full-integer quantization; RDS: representative dataset; MAE(pVF): mean absolute error of the probability for detection of ventricular fibrillation.

2.6.1. Materials

This study uses a proprietary clinical ECG database (Schiller Médical, Wissembourg, France) provided for research purposes and for the retrospective investigation of cardiac arrest rhythms. The database was collected by Schiller AEDs (FRED EASY in 2011 and DEFIGARD TOUCH 7 in 2017) used during OHCA interventions by the Paris Fire Brigade (BSPP, Brigade des Sapeurs-Pompiers de Paris). The reanimation protocol applied CPR with a 30:2 compression/ventilation ratio and chest compression rate of 100–120 min^{-1}, paused every 2 min for a standard AED rhythm analysis, following the European Research Council Adult Basic Life Support guidelines [82,83]. The database was anonymized before the study to ensure the medical confidentiality, without information about the patient identity, epidemiological data, diagnosis, drug therapy, or outcome.

Single-lead ECG signals were acquired via defibrillation pads placed in antero-apical position on the patient thorax. These signals were filtered by the AED hardware within a 1–30 Hz bandwidth to suppress baseline drift and high-frequency noise, then sampled at 500 Hz. To reduce computational load, the ECG signals were downsampled to 125 Hz to match the DNN input format shown in Figure 2. For each OHCA intervention, 10 s ECG episodes collected during standard AED rhythm analysis were extracted and annotated by consensus among three experts, following the American Heart Association (AHA) rhythm annotation guidelines [84]:

- Coarse ventricular fibrillation (VF) with peak-to-peak ECG amplitude > 200 μV;
- Normal sinus rhythm (NSR) with visible P-QRS-T waves and heart rate 40–100 bpm;
- Other non-shockable rhythm (ONR), including atrial fibrillation/flutter, sinus bradycardia, supra-ventricular tachycardia, premature ventricular contractions, heart blocks, etc.;
- Asystole (ASYS) with low-amplitude ECG, having peak-to-peak signal deflection $\leq$ 100 μV for more than 4 s.

Following the AHA statement [84], the annotations were categorized into shockable (VF) and non-shockable (NSR, ONR, ASYS) rhythms. The data were divided patient-wise into independent datasets for calibration, validation, and testing, with the number of 10 s ECG episodes defined in Table 2. Rhythm distribution within and between datasets was not controlled but reflected the content of the OHCA interventions recorded during non-overlapping periods and predefined before the study—calibration dataset (733 patients in 2011), validation dataset (1604 patients in 2017), and test dataset (1312 patients in 2017). This approach assures generalizing out-of-distribution [85], which is critical and challenging in real applications, especially in the case of critical medicine.

Table 2. Number of 10 s ECG episodes, included in the calibration, validation, and test datasets, collected from out-of-hospital cardiac arrest databases with respective arrhythmia annotations. The calibration dataset includes both noise-free and noise-corrupted ECG signals, while validation and test datasets include only noise-free cases as required by the AHA guidelines [84].

	Out-of-Hospital Cardiac Arrest ECG Databases		
	Calibration	**Validation**	**Test**
Shockable rhythms			
Ventricular fibrillation (VF)	255	433	414
Non-shockable rhythms			
Normal sinus rhythm (NSR)	162	254	247
Other non-shockable rhythm (ONR)	1251	2514	2267
Asystole (ASYS)	2638	5331	4554

2.6.2. Optimization of Quantized LiteRT DNN Models Using Representative Datasets

As illustrated in Figure 5, the optimization process is applied to two quantized LiteRT models (IntQ and Ful-IntQ), both requiring calibration to convert the variable tensors to the int8 format, according to Table 1 and Equations (1) and (2). In theory, the representative dataset (RDSs) used for the calibration are the only input influencing the quantization process of TFLite models. Therefore, selecting an optimal number and content of RDS samples from the calibration dataset is essential to achieve the best possible quantization performance.

We quantify the error introduced by quantization as the difference between the output of the quantized LiteRT DNN model (denoted as pVF^{Q}) and the output of the reference TF DNN model (pVF^{REF}) across all ECG recordings in the validation database (size N), using the mean absolute error (MAE) as the statistical metric. Further MAE of pVF is interpreted as a percentage relative to the maximum probability range of 1:

$$\mathrm{MAE(pVF)} = 100\frac{\sum_{i=1}^{N}\left|pVF^{Q} - pVF^{REF}\right|}{N}, \% \quad (3)$$

Using the validation result (3) to compare models generated with different RDS selections, the optimal model is the one that provides $\mathrm{MAE(pVF)} \rightarrow \min$. Notably, we ensure the independence between MAE(pVF) estimation and the RDS selection by using

non-overlapping calibration and validation datasets, according to Table 2. The content of the ECG recordings included in RDS might be different. We study four RDS selection strategies:

- RDS random selection: The RDS includes S ECG recordings randomly chosen from the calibration database.
- RDS rhythm-based selection: The RDS includes S ECG recordings randomly selected from the calibration database in a balanced proportion of annotated rhythms (VF, NSR, ONR, and ASYS).
- RDS amplitude-based selection: The RDS includes S ECG recordings randomly selected from the calibration database, evenly distributed across five root mean square (RMS) amplitude ranges of the ECG signal: <100 μV, 100–200 μV, 200–500 μV, 500–1000 μV, and >1000 μV.
- Total calibration database: RDS contains the entire calibration dataset (4306 samples).

The first three RDS selection strategies evaluate 13 different RDS sizes, i.e., S = 10, 20, 40, 80, 120, 160, 200, 250, 300, 350, 400, 500, or 1000 ECG recordings from the calibration dataset. For each RDS size, 100 random sets of ECG recordings are generated, resulting in a total of 1300 RDS configurations analyzed to optimize the quantization process for both IntQ and Full-IntQ LiteRT models independently.

2.6.3. Testing

As illustrated in Figure 5, the testing of different DNN models is performed directly on their respective target platforms:

- GPU-based workstation: Runs the original TF DNN model within its development environment.
- Raspberry Pi 4 microcontroller: Executes the embedded LiteRT DNN model without quantization.
- STM32F7 microcontroller: Runs the C++ LiteRT DNN model tested both without quantization and with three quantization types (DynQ, IntQ, and Full-IntQ).

Two types of performance metrics are evaluated for the execution of each model on the test dataset:

1. Accuracy metrics:

The detection accuracy for Sh rhythms (VF) and NSh rhythms (NSR, ONR, and ASYS) is evaluated with the standard metrics for reporting the shock advisory performance in AEDs [84] in terms of sensitivity (Se) and specificity (Sp):

$$\mathrm{Se} = 100.\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}},\% \quad \mathrm{Sp} = 100.\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}},\% \tag{4}$$

where true positives (TPs) and true negatives (TNs) are the correctly detected Sh and NSh cases, respectively, while false positives (FPs) count the NSh classified as Sh and false negatives (FNs) accumulate the Sh cases that were recognized as NSh.

2. Resource efficiency metrics:

- Execution time (Inference latency): Represents the total time taken by the microcontroller to execute all DNN operations in the end-to-end ECG diagnostic workflow. This includes processing the input—a 10 s ECG segment—and generating the corresponding output, which is the VF detection probability.
- Model size (flash memory usage): Represents the storage space required to deploy the DNN model on the microcontroller's flash memory impacting how efficiently the model fits within the available memory constraints.
- Microcontroller memory usage: Represents the static RAM (SRAM) used for temporary storage during inference (e.g., input/output tensors, intermediate activations), or the

Synchronous Dynamic RAM (SDRAM) used as external memory for larger datasets or computations when internal SRAM capacity is insufficient.

- Energy consumption: Represents the power consumed by the microcontroller board during the execution of a DNN model over time (t), calculated as follows:

$$E = U.I.t, \quad (5)$$

where the current (I) and voltage (V) are measured with a commercial USB current voltage tester, providing a precision of 0.01 A and 0.01 V, respectively. The tester is connected between the USB power supply and the microcontroller board to monitor power consumption accurately.

3. Results

3.1. Optimization of Quantized LiteRT DNN Models Using Representative Datasets

The platform for quantization of LiteRT DNN Models is a GPU-based workstation with Intel i5-2300 CPU @ 2.80 GHz, 24 GB RAM, NVIDIA GeForce RTX 3080-12 GB GPU, running Windows 10 OS (64-bit) and Python 3.8.18 with installed libraries including TensorFlow 2.10.0.

One of the main issues in the development phase is the time required for post-training quantization to produce the final product—a quantized LiteRT DNN model ready for deployment on the microcontroller. Figure 6 summarizes all measurements for the post-training quantization times of both IntQ and Full-IntQ models, which were observed to be linearly dependent on the RDS size. For RDS sizes of up to 1000 ECG recordings, the quantization process is relatively fast, requiring between 17 s and 20 s. However, when the LiteRT conversion uses the total calibration dataset (over 4300 samples), the post-training quantization time increases to 34 s for Full-IntQ and 37 s for IntQ. Notably, IntQ models are slower because they require additional time for internal conversion of input/output data from float32 to int8 for inner activations, whereas Full-IntQ models handle input/output conversion externally as int8, avoiding internal conversions.

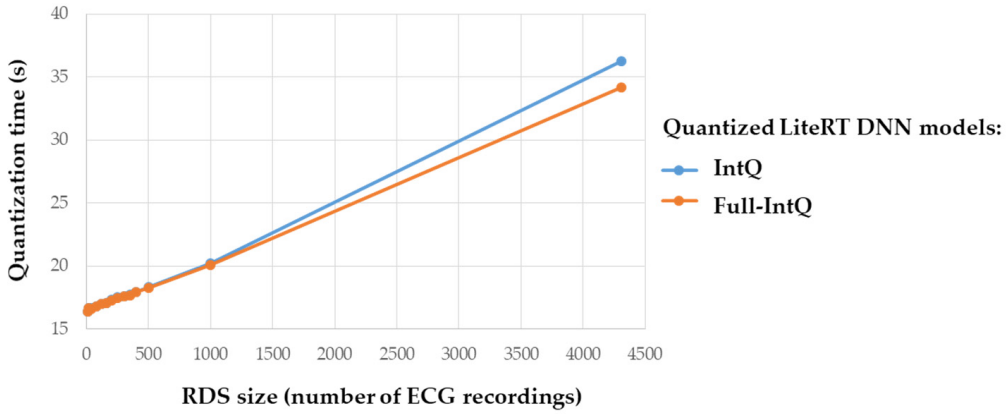

Figure 6. Measurements of post-training quantization times for the Integer (IntQ) and Full-Integer (Full-IntQ) quantized LiteRT models as a function of the representative dataset size (RDS = 10, 20, 40, 80, 120, 160, 200, 250, 300, 350, 400, 500, 1000, and 4306 ECG recordings) selected from the calibration dataset.

Given that the RDS is limited and includes only specific cases from the calibration dataset, a critical challenge in post-training quantization is to ensure the optimal RDS content. Figure 7 illustrates three distinct RDS configurations achieved through different selection strategies. The random selection strategy (1st column) creates an RDS dominated by the predominant rhythm in the calibration dataset (70% ASYS). The rhythm-based selec-

tion strategy (2nd column) ensures RDS with a balanced rhythm composition (30% NSR, 30% ONR, 20% ASYS, 20% VF). Finally, the amplitude-based selection strategy (3rd column) enables the collection of low-amplitude ASYS with RMS < 100 μV (20%), extreme-amplitude noises with RMS > 1000 μV (20%), and normal-amplitude ECGs within the RMS range of 100–1000 μV (60%).

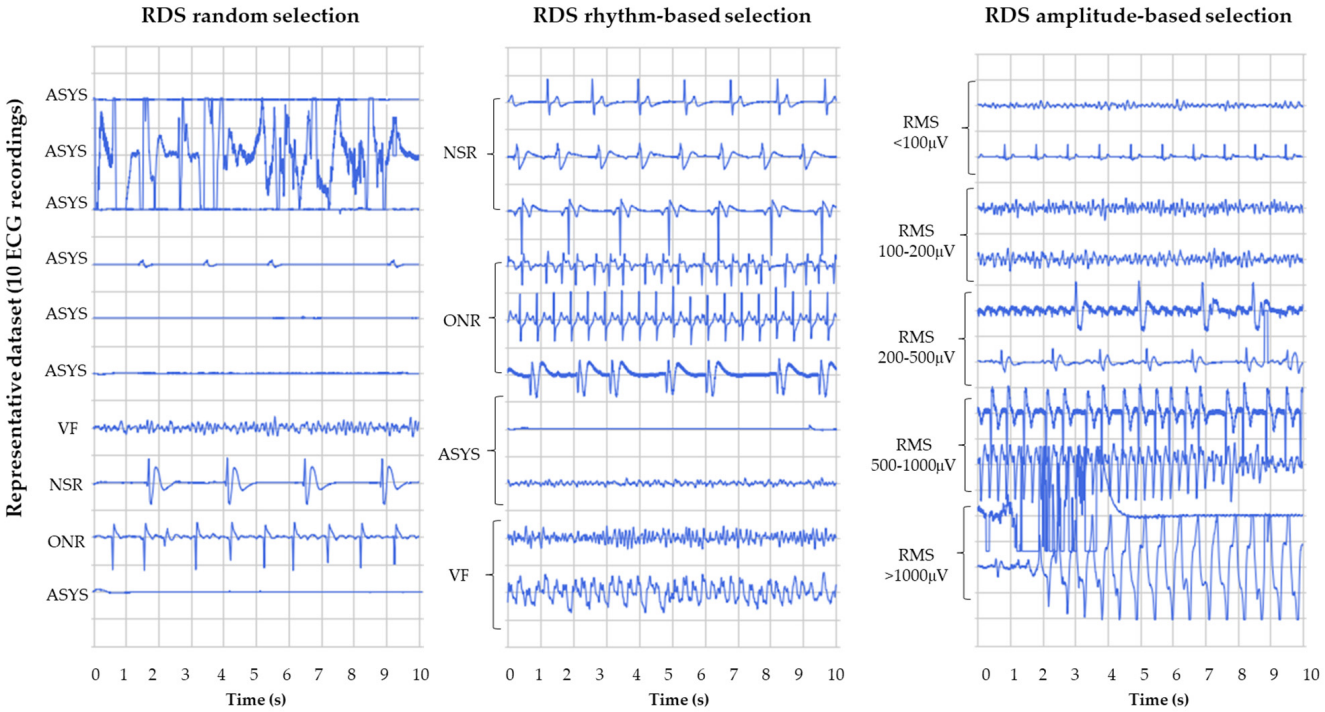

Figure 7. Examples of representative datasets (RDSs), including 10 ECG recordings from the calibration dataset and selected by the defined 3 RDS selection strategies: random selection (1st column), rhythm-based selection (2nd column), and amplitude-based selection (3rd column). VF: ventricular fibrillation; NSR: normal sinus rhythm; ONR: other non-shockable rhythm; ASYS: asystole; RMS: root mean square value of the ECG amplitude.

There are apparent differences between the example RDS configurations shown in Figure 7, which are hypothesized to have varying impacts on the quantization error, defined as MAE(pVF) in Equation (3). The error is measured using the validation database, with statistical conclusions drawn from generalizations across 100 random RDS configurations for each size, as shown in Figure 8. Distributions with a wider min–max range indicate a greater influence of the RDS content on the quantization error. Notably, wide distributions are observed in the quantization of IntQ LiteRT models with limited RDS size $\leq$40 and most Full-IntQ LiteRT models, presenting quantization error span from 0.6% to over 4%. These findings highlight the importance of carefully optimizing RDS content.

From the 100 quantized models per RDS size in Figure 8, RDS optimization identifies the best representative with the lowest quantization error for each size. MAE(pVF) of all best models is compared in Figure 9. Both quantized LiteRT versions are identified as optimal at an RDS size of 80 using the random RDS generator, achieving MAE(pVF) values of 0.6% for IntQ and 0.7% for Full-IntQ. Other RDS generation strategies, such as rhythm-based strategies, appear equally effective, while amplitude-based strategies are less effective. A consistent trend of increasing quantization error is observed as RDS size grows, with the maximal error (about 2%) occurring when the entire calibration dataset is used for RDS.

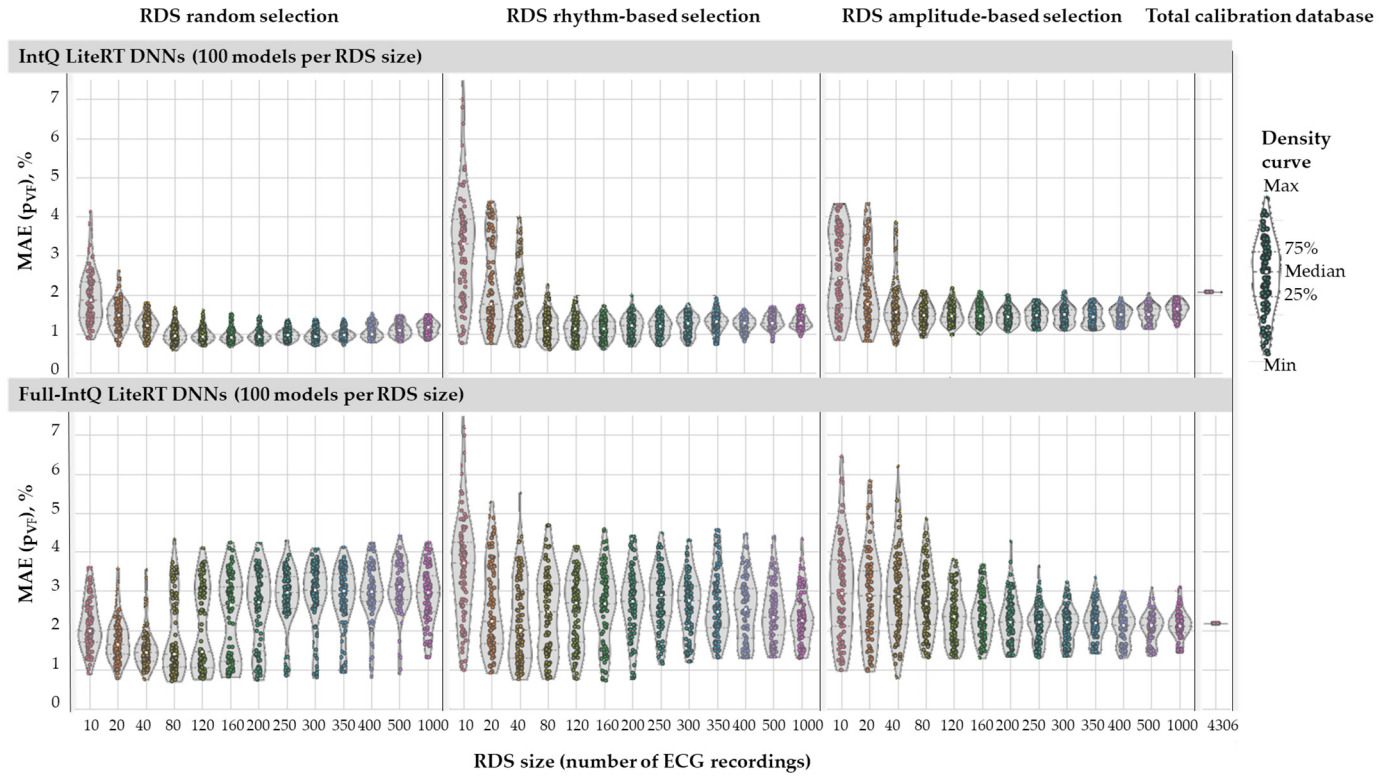

Figure 8. Statistical analysis of MAE (pVF) in function of the representative dataset (RDS) for two types of quantized LiteRT DNN models: IntQ (**top**) and Full-IntQ (**bottom**). MAE(pVF) density distributions (violin plots) are calculated on the validation database for 100 LiteRT DNN models quantized with each RDS size (13 sizes, including from 10 to 1000 training ECG recordings) and three RDS selection methods (random, rhythm-based, and amplitude-based). The validation MAE(pVF) for a LiteRT DNN model quantized on the total calibration database (4306 ECG recordings) is provided as a reference (on right).

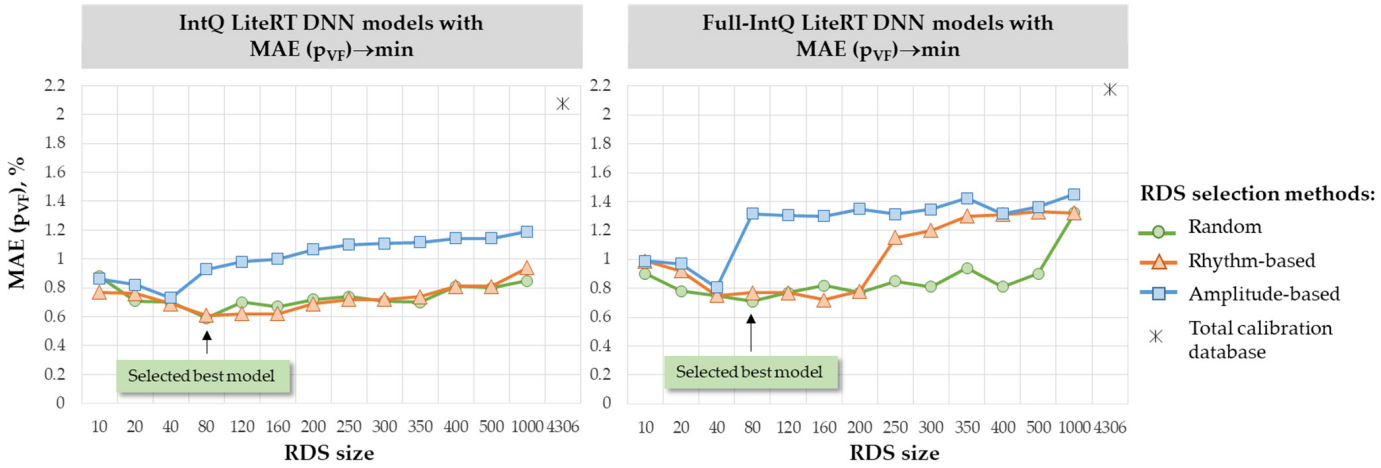

Figure 9. Comparison of the quantized LiteRT DNN models (IntQ on left and Full-IntQ on right), presenting the lowest mean absolute error MAE (pVF) for the validation database in each violin plot of Figure 7. The selected best quantized models (RDS = 80 ECG recordings by random selection) are highlighted.

3.2. Resource Efficiency Metrics

Table 3 provides a comprehensive summary of the restrictions associated with deploying DNN models on resource-constrained hardware. It presents data on model sizes, microcontroller memory usage, execution times, and energy consumption for both non-quantized and quantized LiteRT models, evaluated on their respective target microcontroller platforms—Raspberry Pi 4 (Python 3.7.3, Tensor Flow Lite Runtime 2.8.0) and STM32F7 (STM32CubeIDE 1.17.0, Lite RT library for microcontrollers [86]). For comparison, the resource metrics for executing the original TF DNN model on the GPU-based design platform (as described in Section 3.1) are also included. All models are evaluated using the ECG recordings from the test dataset.

Table 3. Evaluation of resource efficiency for executing DNN models across three platforms: a GPU-based workstation, a Raspberry Pi 4, and STM32F7 microcontrollers. Execution time is reported as mean value ± standard deviation for all ECG recordings in the test database. Energy consumption is measured based on power usage during the mean execution time. NA: not available or not applicable for GB-sized RAM memories in PC workstation and Raspberry Pi 4. ERR: error message by the debugger "Hybrid models are not supported on Lite RT for microcontrollers [86]".

	Model Size (Flash)	SRAM Usage	SDRAM Usage	Execution Time	Energy Consumption
TensorFlow DNN model in GPU-based workstation					
TF DNN model	149,452 Bytes	NA	NA	1.007 ± 0.147 ms	NA
LiteRT DNN model in Arm-based microcontroller (Raspberry Pi 4)					
Non-Quantized	40,372 Bytes	NA	NA	1.064 ± 0.025 ms	3.91 mJ
C++ LiteRT DNN models in Arm-based microcontroller (STM32F7)					
Non-quantized	40,372 Bytes	234,476 Bytes (44.72% *)	202,860 Bytes (2.44% #)	4824 ± 0.612 ms	4592 mJ
DynQ	20,712 Bytes	ERR	ERR	ERR	ERR
IntQ	20,424 Bytes	214,528 Bytes (40.92% *)	54,956 Bytes (0.66% #)	615.7 ± 0.468 ms	586 mJ
Full-IntQ	20,120 Bytes	214,224 Bytes (40.86% *)	54,804 Bytes (0.66% #)	614.6 ± 0.483 ms	585 mJ

*: % of available SRAM memory of 512 kB; #: % of available SDRAM memory of 16 MB.

We note that dynamic range quantization of the LiteRT DNN model was successfully performed via LiteRT converter on the development platform. However, attempting to deploy the C++ LiteRT DynQ model on the STM32F7 microcontroller resulted in an error caused by the LiteRT library for microcontrollers [86], as it does not currently support hybrid models. As a result, data on SRAM/SDRAM usage and execution time for this model could not be reported in Table 3. It is worth noting that future versions of the LiteRT library for microcontrollers may potentially include support for hybrid models in STM32F7 microcontrollers.

3.2.1. Model Size (Flash Memory Usage)

According to Table 3, the original TF DNN model has a size of approximately 150 kB. After LiteRT conversion, the model size is reduced by 3.75 times to about 40 kB, requiring minimal free memory on the Flash card. This reduction enables the execution of non-quantized models on both the Raspberry Pi 4 and STM32F7 microcontroller boards. Further quantization, achieved by using 8-bit integer weights instead of 32-bit floating-point weights (Table 1), doubles the size reduction, shrinking the model to approximately 20 kB. This optimization facilitates deployment on memory-constrained microcontrollers.

3.2.2. Microcontroller Memory Usage

In Table 3, SRAM/SDRAM memory usage is evaluated exclusively for LiteRT DNN models deployed on the STM32F7 microcontroller, as the Raspberry Pi 4 utilizes only a negligible portion of its ample 8 GB SDRAM capacity. The non-quantized LiteRT DNN model occupies 235 kB of SRAM (45% of the 512 kB available) and 203 kB of SDRAM (2.5% of the 16 MB available) on the STM32F7 microcontroller. In comparison, the quantized IntQ and Full-IntQ LiteRT models require almost the same amount of SRAM—214 kB (41% of 512 kB)—but considerably less SDRAM, consuming only 55 kB (0.66% of 16 MB) due to the use of int8 data.

3.2.3. Execution Time (Inference Latency)

The Raspberry Pi 4 board executes the non-quantized LiteRT DNN with a mean latency of less than 1.1 ms, comparable to the performance of the GPU-based development platform running the TensorFlow DNN. While the standard deviation of latency is negligible on both platforms, it is approximately six times higher on the GPU-based platform, likely due to task overhead from the Windows OS.

When the non-quantized LiteRT DNN is deployed on the STM32F7 microcontroller, the process for analysis of 10 s ECG rhythm takes over 4.8 s. This inference latency exceeds the data collection period by nearly 50%, limiting real-time applications. However, after LiteRT DNN quantization, this latency is significantly reduced to approximately 615 ms for both IntQ and Full-IntQ models. The small standard deviation of less than 0.5 ms indicates that the analysis process inside CNN is stable and unaffected by variations in the input signal or the rhythm being analyzed. This stability is crucial for ensuring a reliable real-time decision-making process without unexpected delays, under real-life conditions involving diverse rhythm variations and artifacts.

3.2.4. Energy Consumption

On the Raspberry Pi 4 board, current and voltage consumption are measured as I = 0.43 A, U = 5.24 V in idle mode and I = 0.7 A, U = 5.24 V during the execution of the LiteRT DNN model. This corresponds to an increase in power consumption of 1.42 W (2.25 W in idle mode vs. 3.67 W during execution), which can be attributed to the simultaneous activation of up to four processor cores during DNN analysis. Considering that the mean execution time is 1.064 ms, the total energy consumption of the Raspberry Pi 4 board for model execution is estimated at 3.91 mJ (Table 3).

For the STM32F769IDISCOVERY kit (including additional consumption from indicator LEDs and microcontroller dedicated for ST-Link connection), current and voltage consumption remains unchanged between idle mode and DNN execution, measured at I = 0.2 A, U = 4.76 V, resulting in a power consumption of 0.952 W. This is due to the single-core architecture, where the processor operates at a fixed clock frequency, maintaining relatively equal power consumption regardless of the executed instructions. Considering the model-specific mean execution times, the energy consumption is 4592 mJ for the non-quantized model and about 585 mJ for both quantized models (Table 3). This indicates that quantization reduces energy consumption for DNN execution by about 7.85 times.

3.3. Accuracy Metrics

Table 4 presents the shock advisory performance of all DNN models for the detection of four rhythm categories, evaluated in terms of Sp (NSR, OR, Asystole) and Se (VF) on the test dataset. The accuracies presented in the first column correspond to the original TF DNN model tested on the GPU-based platform, which we found to be identical to those of the non-quantized LiteRT DNN models running on both the Raspberry Pi 4 and STM32F7

microcontroller boards. This is due to the fact that non-quantized LiteRT models retain full model precision. After quantization, a slight increase in Sp (NSR) is observed for the IntQ model and in Sp (ASYS, ONR) for the Full-IntQ models, while Se (VF) remains consistent across all tested models.

Table 4. Test accuracy performance of all DNN models investigated in this study, according to the test scheme in Figure 5. The accuracy of the TF DNN model (reference) and non-quantized LiteRT DNN models running on Raspberry Pi 4 and STM32F7 microcontroller boards are shown in the first column, yielding identical results. The results for the quantized LiteRT DNN models (IntQ and Full-IntQ) running on the STM32F7 microcontroller are presented in the second and third columns.

	TF DNN (Reference) LiteRT DNN (Non-quantized)	**LiteRT DNN (Quantized)**	
		IntQ	**Full-IntQ**
Normal sinus rhythm (Specificity)	99.60% (246/247)	100% (247/247)	99.60% (246/247)
Other non-shockable rhythms (Specificity)	98.85% (2241/2267)	98.85% (2241/2267)	98.94% (2243/2267)
Asystole (Specificity)	99.54% (4533/4554)	99.54% (4533/4554)	99.74% (4542/4554)
Ventricular fibrillation (Sensitivity)	98.55% (408/414)	98.55% (408/414)	98.55% (408/414)

The performance presented in Table 4 can be further analyzed in detail in Figure 10, which shows the DNN output observations, specifically pVF, for the test dataset. As the regression scatterplots represent overlapping data points, the histograms are important to show the real data distributions (number of observations). The histograms for ventricular fibrillation reveal that the quantization limits the maximal pVF range to 0.95 (IntQ model) and 0.9 (Full-IntQ model). Nevertheless, this has no effect on performance, given that a rhythm is classified as ventricular fibrillation at a threshold of pVF ≥ 0.5. Additionally, the scatterplots in Figure 10 highlight all cases where the decisions of the original TF DNN model differ from those of the quantized LiteRT DNN models (IntQ and Full-IntQ).

The analysis of the inverse decisions of the IntQ model (Figure 10, top) is as follows:

- NSR—A single TN decision differs from the TF DNN model's decision, resulting in a 0.4% increase in Sp for NSR.
- ONR—One TN and one FP decision differ from the TF DNN model's output. These inverse decisions balance each other, leaving the Sp for ONR unchanged.
- ASYS—Two TN and two FP decisions diverge from the TF DNN model's output. As with ONR, these inverse decisions balance each other, keeping the Sp for ASYS unchanged.
- VF—No inverse decisions are observed, and Se remains unchanged.

The analysis of the inverse decisions of the Full-IntQ model (Figure 10, bottom) is as follows:

- NSR—No inverse decisions are observed, and Sp remains unchanged;
- ONR—Two TN decisions deviate from the TF DNN model's output, resulting in a 0.1% increase in Sp for ONR;
- ASYS—Nine TN decisions deviate from the TF DNN model's output, contributing to a 0.2% increase in Sp for ONR;
- VF—No inverse decisions are observed, and Se remains unchanged.

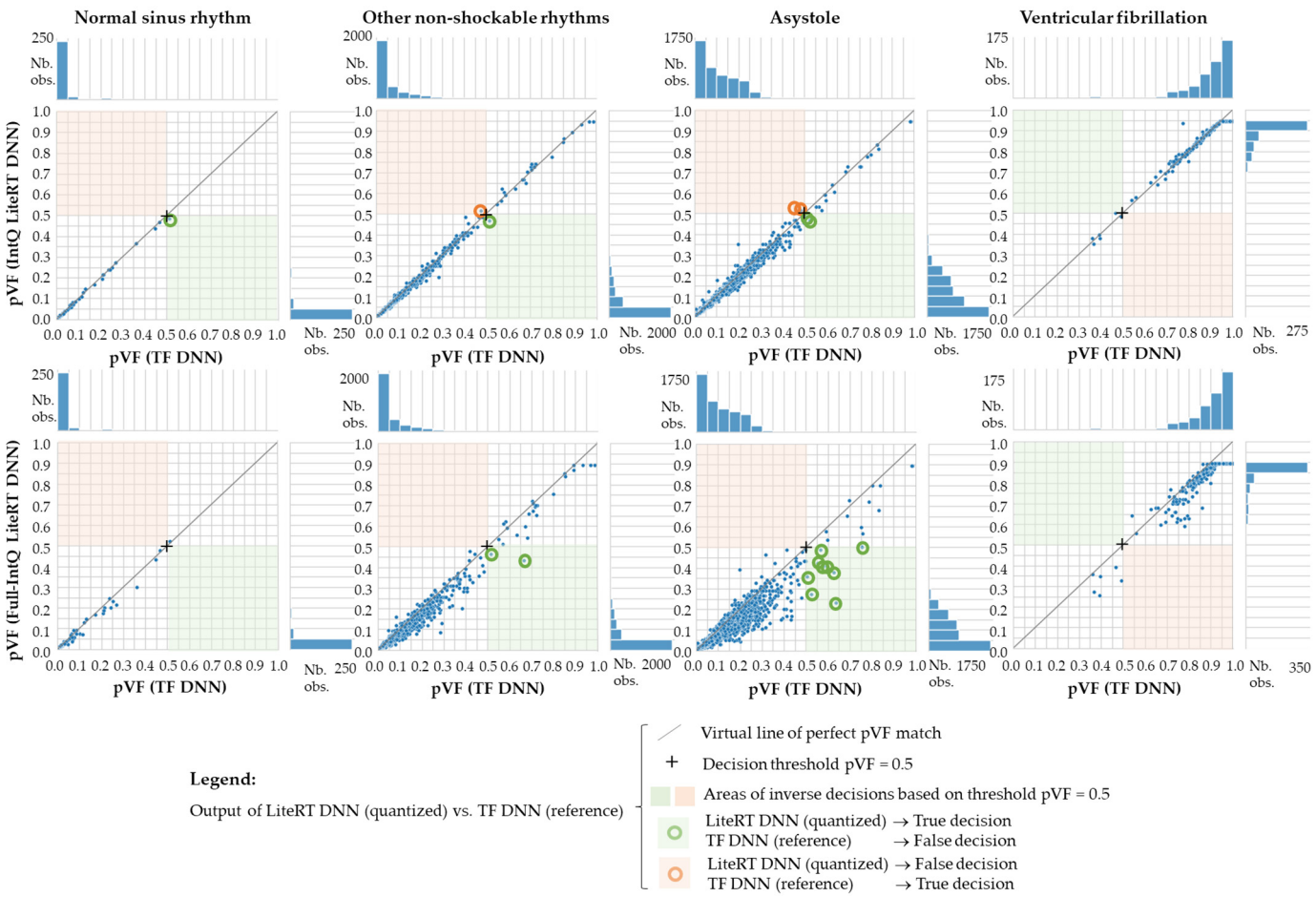

Figure 10. Observations of pVF output of TF DNN (reference) vs. quantized LiteRT DNN (IntQ on top and Full-IntQ on bottom) in the test dataset, illustrating the performance results in Table 4 for pVF threshold = 0.5. Observation points are presented as scatter plots and histograms for four rhythms: normal sinus rhythm, other non-shockable rhythms, asystole, and ventricular fibrillation.

Figure 11 presents an overview of representative signals that result in inversed classification outcomes for the quantized LiteRT DNN (IntQ and Full-IntQ) compared to the reference TF DNN model. These signals often exhibit complex, non-trivial rhythms, sometimes accompanied by artifacts, which pose challenges for any VF detection algorithm. Above each example, the pVF estimations are displayed (TF DNN → quantized LiteRT DNN), revealing that the IntQ model's pVF output shows differences within the range of 0.03–0.05, corresponding to up to 10% of the pVF threshold. In contrast, the Full-IntQ model demonstrates larger deviations, ranging from 0.06 to 0.41, reaching up to 80% of the threshold. Despite these deviations, the cases with inversed classification outcomes are negligible for the test dataset—0.1% (7/7482 for IntQ and 11/7482 for Full-IntQ).

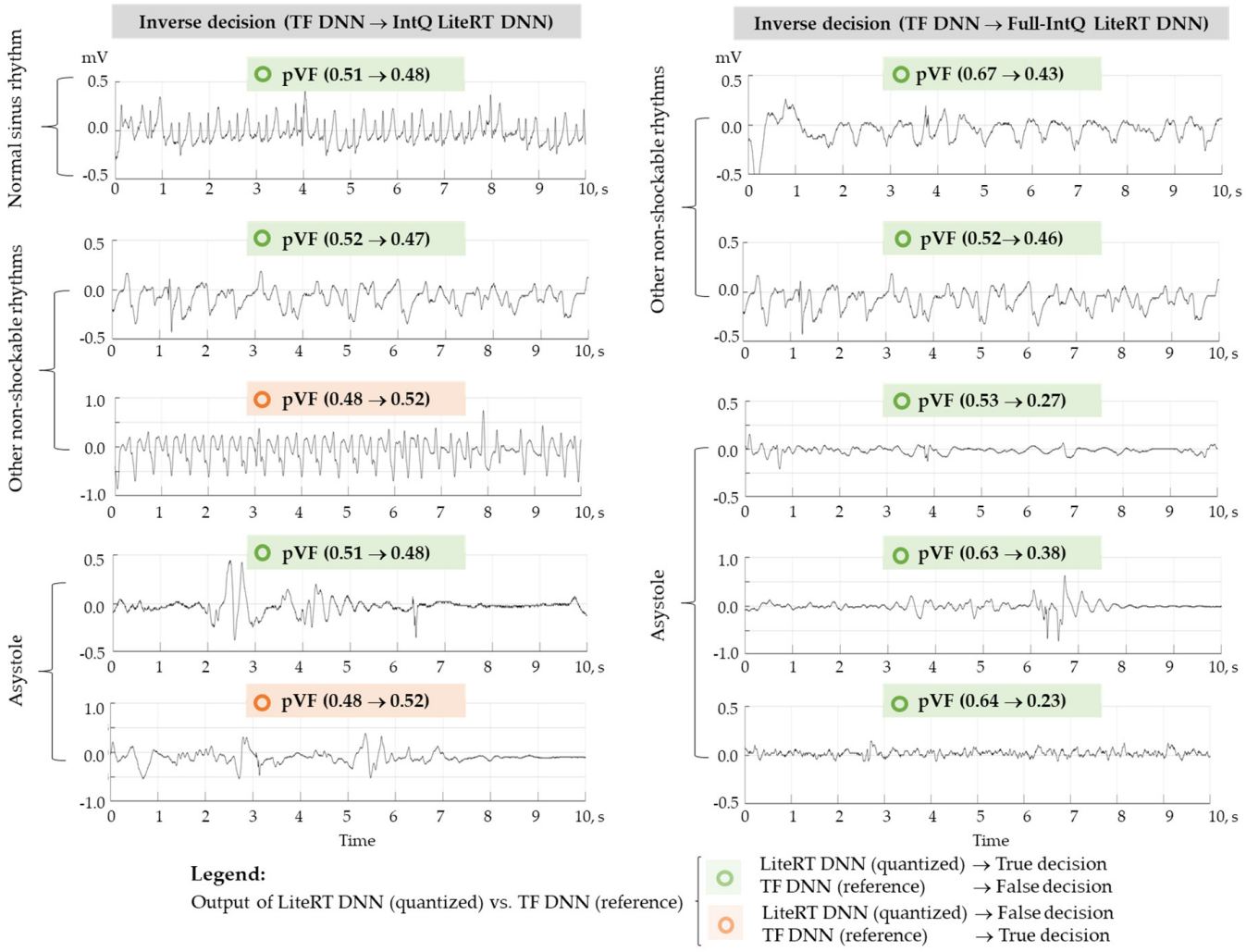

Figure 11. Examples of 10 s ECG strips with inverse classification outcomes of TF DNN (reference) vs. quantized LiteRT DNN (IntQ on left and Full-IntQ on right) based on a decision threshold pVF = 0.5. The cases are highlighted in Figure 10.

4. Discussion

This study provides positive evidence for deploying deep neural networks on conventional microcontrollers, considering that DNNs have the potential to improve rhythm analysis for ventricular fibrillation detection in stand-alone AED devices. By implementing a quantization strategy, we enable low-resource microcontrollers to execute deep CNN models, achieving accurate shock-advisory results on cardiac arrest ECG data (Se = 98.6%, Sp = 99.5%) with internal computation latencies that can be reduced from 4.8 s to 0.6 s, thereby ensuring real-time operational capability.

While numerous advanced deep CNN models have shown excellent performance in rhythm classification, particularly when prospectively tested on OHCA databases using powerful GPU-based development platforms [65–73], such analyses are often restricted to offline operating modes. Although effective in controlled environments, these DNN models are typically regarded as impractical for real-life clinical applications due to their reliance on high computational resources and limited consideration of the constraints imposed by embedded systems. This study shows that deep learning models can run on limited-resource hardware, opening the door for their integration into real-world AED devices. Particularly, we investigate two popular types of ARM-based microcontrollers (Raspberry

Pi 4 and STM32F7), which feature high and low levels of computational capabilities, respectively, that can be of certain interest to different developers.

The powerful Raspberry Pi 4 microcontroller is fully supported by embedded software libraries in Python and TensorFlow Lite, allowing it to directly execute converted LiteRT models without the need for additional design efforts, such as post-training quantization or addressing performance degradation concerns. In our tests, a directly converted LiteRT DNN running on the Raspberry Pi 4 replicates the accuracy of the reference TF DNN model (Table 4) and achieves a fast and highly reproducible inference time of just 1.064 ± 0.025 ms, comparable to the execution time on the GPU-based development platform (Table 3). While the Raspberry Pi 4 demonstrates significant advantages for running DNNs, several limitations might be considered when evaluating its suitability for integration into AEDs—compact, battery-operated devices that demand high reliability during emergencies and extended standby periods. These limitations include high power consumption (2.25 W in idle mode, 3.67 W during DNN model execution); large physical size; need for cooling; a Linux operating system not optimized for responsiveness during critical situations; additional protective measures required to withstand extreme temperatures, humidity, and vibrations; and increased costs for peripheral components and adaptations to meet medical device compliance and stringent regulatory standards for AEDs. For these reasons, simpler, low-power, real-time microcontrollers, such as STM32 or similar ARM-based chips are typically preferred in AED designs.

The low-power STM32F7 microcontroller running the same LiteRT DNN model as Raspberry Pi 4 in this study presented considerable inference latency of over 4.8 s (Table 3), which adds almost 50% to the 10 s data collection period, limiting real-time applications. This underscores that running DNNs on the STM32F7 microcontroller without reducing computational complexity is not viable for real-time processing. We found that simple dynamic range quantization, which does not require external resources such as a calibration dataset, is not applicable to the STM32F7 microcontroller. This limitation arises because the current version of the LiteRT library for microcontrollers does not support hybrid models. Therefore, integer post-training quantization with the use of a calibration dataset is the next solution level. The quantization process itself is efficient, requiring approximately 37 s on a conventional GPU-based platform to process a calibration dataset of around 4300 samples, with negligible quantization time differences between integer and full-integer quantization methods (Figure 6). However, our findings reveal that using the entire calibration dataset results in a quantization error of approximately 2%, which is not optimal (Figure 8). Consequently, additional design time is necessary to identify the optimal selection of representative data from the calibration dataset, specifically determining both the RDS size and content that can minimize quantization error. In this study, we evaluate three practical strategies for RDS selection based on randomness, rhythm type, and ECG amplitude (Figure 7), using an extended set of 1300 RDS configurations for each strategy. As shown in Figure 8, no significant differences are observed between the RDS selection strategies in terms of the provided min–max ranges for the quantization error MAE(pVF). Nevertheless, Figure 9 reveals that the random RDS selection consistently achieves minimal error. For integer quantization, a single randomly selected RDS with a size ranging from 80 to 1000 cases produces errors between 0.6% and 1.8% (Figure 8, top). Interestingly, smaller RDS sizes closer to 80 samples consistently result in lower minimal errors (Figure 9, left). In contrast, full-integer quantization presents a broader error range, spanning from 0.7% to 4.5% (Figure 8, bottom), although the minimal error is similarly achieved with an RDS size of 80 samples (Figure 9, right).

Table 4 reveals that both non-quantized and quantized LiteRT DNNs reproduce the shock advisory accuracy of the reference TF DNN model, generalized for a large test set

with 7482 ECGs from OHCA interventions. All DNN models definitively fulfil the AHA performance goals for arrhythmia analysis algorithms in AEDs [84], presenting Se = 98.6% for VF (goal >90%), Sp = 99.6–100% for NSR (>99%), Sp = 98.9% for ONR (>95%), and Sp = 99.5–99.7% for asystole (>95%). As illustrated in Figure 10, the IntQ and Full-IntQ models present only 7 and 11 cases (0.1% of total test dataset) with inversed shock-advisory decisions, respectively. Reviewing these cases in Figure 11 reveals ECG signals with various morphologies, including tall T-waves, or rapid heart rate, or low and wide QRS complexes, or asystoles with low- and high-amplitude noises, all of which can present a challenge to any VF detection algorithm.

To the best of our knowledge, the only study that has validated a CNN model for VF detection on the hardware of an AED for which it was specifically designed is by Shen et al. (2023) [74] (see Table 5 for details). This study reported comparable shock advisory accuracy evaluated under specific embedded system constraints (Se = 98%, Sp = 100%), giving a result within 383 $\pm$ 29 ms after processing a 7 s raw ECG signal sampled at 300 Hz. Notably, this is about 1.6 times faster than the execution time of our low-resource ARM Cortex-M7 microcontroller (615 $\pm$ 0.5 ms for processing of 10 s ECG input), although both CNN implementations support almost real-time shock advisory feedback within 5–6% of the data collection time. Execution time variations are likely attributable to differences in the hyperparameters of the pre-trained CNN models. Although both models include five convolutional layers with a comparable number of filters, the kernel sizes differ (10 in this study vs. 3 in [74]), resulting in approximately 2.3 times more parameters in our model (7521 vs. 3200 as inferred from the network description in [74]). A similar impact of the total amount of trainable parameters, combined with the influence of different activation functions in DNNs, has been reported by Huber et al. [87]. Furthermore, [74] does not provide details for the steps of implementing the CNN model in the AED, making it unclear whether the original model was quantized or how quantization might have influenced both accuracy and resource efficiency metrics.

Table 5. Comparison of microcontroller-embedded CNN models for the detection of ventricular fibrillation. The information from Shen et al. [74] is presented as interpreted from the original article and its supplementary material.

	This Study (Full-IntQ)	**Shen et al. (2023) [74]**
Test platform:		
Microprocessor	32-bit @ 216 MHz	32-bit @ 200 MHz
On-chip flash	2 MB	2 MB
SRAM	512 kB	512 kB
DNN model:		
Input: one-lead ECG	1250 samples (10 s @ 125 Hz)	2100 samples (7 s @ 300 Hz)
CNN layers	5	5
Filters	{20, 15, 15, 10, 5}	16 per layer (average)
Kernel size	10	3
Activation	ReLU	Leaky ReLU
Max-pooling layers	5	5
Output: one unit	sigmoid activation	SoftMax activation
Performance metrics:		
Sensitivity	98.6% (408/414)	98.0% (350/357)
Specificity	99.5% (7031/7068)	100% (648/648)
Execution time	614.6 $\pm$ 0.483 ms	383 $\pm$ 29 ms

The resource efficiency metrics for all models optimized in this study (LiteRT DNN, IntQ, Full-IntQ) are compared with those reported in other published studies addressing

ECG processing models implemented on ARM-based platforms (Table 6). It is important to note that these comparisons are made under varying conditions in terms of model complexity and test platforms. The referenced studies primarily utilize CNN architectures with different number of convolutional blocks (comprising convolution and max or average pooling layers): one block in [7], four blocks in [58], five blocks in this study, and seven blocks in [57]. Additionally, [59] employs a combination of a convolutional layer and four residual blocks. Variations also exist in input size, which depends on the specific application, e.g., ranging from 64 samples for a single heartbeat classification in [7] up to 1250 samples for short-term arrhythmia detection in this study. Therefore, the data presented in Table 6 should be interpreted as providing a general impression of the ranges within which the evaluated metrics vary. These differences in architecture, input size, and application context preclude direct comparisons of the models' efficiency when deployed on different edge devices.

Table 6. Comparison of model size, memory usage, and execution time for various DNNs with convolutional architectures implemented on ARM-based platforms for ECG processing. NA: not available or not applicable; QAT: quantization-aware training.

Model	Input	TEST PLATFORM	Model Size	SRAM Usage	Execution Time
CNN for VF detection (this study) LiteRT DNN model	1250 samples (10 s @ 125 Hz)	Raspberry Pi 4 board based on ARM Cortex-A72	40 kB	NA	1 ms
CNN for heartbeat classification [7]	64 samples (0.5 s @ 128 Hz)	Raspberry Pi 3 board based on ARM Cortex-A53			
QAT model			27 kB	12–24 MB *#	1 ms
IntQ model			14 kB	21 MB *	0.65 ms
Full-IntQ model			15 kB	12–24 MB *#	0.67 ms
CNN for VF detection (this study)	1250 samples (10 s @ 125 Hz)	STM32F7 board based on ARM Cortex-M7			
IntQ model			20 kB	214 kB	616 ms
Full-IntQ model			20 kB	214 kB	615 ms
CNN for heartbeat classification [58]	140 samples (0.56 s @ 250 Hz)	STM32L475 board based on ARM Cortex-M4	36 kB	NA	127 ms
CNN for arrhythmia detection [57]	640 samples (6 s @ 107 Hz)	nRF52832 SoC based on ARM Cortex-M4	196 kB	9 kB	379 ms
MobileNetV2 for arrhythmia detection, optimized for size and speed [59]	1000 samples (10 s @ 100 Hz)	nRF52840 SoC based on ARM Cortex-M4	149–177 kB	157 kB	298–480 ms

* The total memory usage in [7] is reported for the model and test database loaded together. # Range of values obtained for different model settings.

In Table 6, the published model sizes range from 14 kB to 177 kB, with our IntQ and Full-IntQ models (20 kB) being comparable to the smallest sizes reported in [7]. These sizes

are well-suited for deployment on resource-constrained STM boards with 1–2 MB of flash memory [80,88] and nRF52 SoCs with 0.5–1 MB of flash memory [89,90].

The SRAM usage for DNN models with a raw single-lead ECG input ranges from 9 to 214 kB, as reported in [57,59] and this study. The 214 kB SRAM usage observed in this study is comparable to the 157 kB reported in [59], especially when considering the difference in input sizes (1250 samples in this study vs. 1000 samples in [59]). The SRAM usage in [57] is remarkably low, however, reported for a few specific allocations: 0.8 kB for GRU gates and hidden states, 4 kB for convolution buffers, and two 2 kB arrays for convolution activations. In contrast, the significantly higher SRAM usage of 12–24 MB reported in [7] can be attributed the additional storage of the entire test dataset, which includes approximately 50,000 heartbeats, alongside the model itself.

In Table 6, the execution time varies significantly depending on the test platform, with models implemented on Raspberry Pi single-board computers achieving execution times of no more than 1 ms. In contrast, execution times on STM32 boards and nRF52 SoCs range from 298 to 616 ms. The execution times reported in [57,59] and this study (480 ms, 379 ms, and 615 ms, respectively) are comparable, taking into account the differences in input sizes (1000, 640, and 1250 samples, respectively).

The presented workflow and experimental results for TF to LiteRT conversion and quantization demonstrate the feasibility of deploying a pre-trained DNN with a fully convolutional architecture on ARM-based microcontrollers. Expanding this approach to support other neural architectures, such as those incorporating recurrent layers (e.g., long short-term memory or gated recurrent units) or others is a potential direction for future investigations.

5. Conclusions

This study makes a significant contribution to the application of neural networks for VF detection in AEDs by presenting a methodological workflow for the efficient deployment of pre-trained DNN models across diverse hardware platforms. This approach enables their practical use in OHCA scenarios where computational resources are inherently limited.

A high-performance TensorFlow CNN model trained for VF detection on a GPU-based workstation was successfully converted into a LiteRT model with a compact size of 40 kB and deployed on a Raspberry Pi 4 microcontroller. The LiteRT model achieves the same high accuracy as the original TensorFlow DNN model on a very large cardiac arrest ECG test set (Se = 98.6% for VF, Sp = 99.5% for asystole, Sp = 99.6% for normal sinus rhythms, Sp = 98.9% for other non-shockable rhythms) while delivering a rapid and highly reproducible inference time of 1.064 ± 0.025 ms—comparable to the GPU platform.

Further optimization was achieved by quantizing and calibrating the LiteRT model using representative datasets with diverse sizes and content. This process included 3×1300 calibration tests, incorporating between 40 and 4000 cardiac arrest ECG signals, ensuring robust performance across varied scenarios. The optimization process resulted in two lightweight, integer-based models (IntQ and Full-IntQ) that are specifically tailored for resource-constrained hardware, such as the STM32F7 ARM Cortex-M7 microcontroller. These models, with sizes of 20 kB and 214 kB SRAM usage and inference times of 615 ± 0.5 ms, reproduce the accuracy of the original TensorFlow model (Se = 98.6%, Sp = 99.3–99.5%). Notably, quantization reduces the energy consumption of DNN model execution by approximately 7.85 times, significantly enhancing the longevity of edge-computing devices in real-world deployments.

The demonstrated resource efficiency enables the deployment of these quantized LiteRT models on a wide range of edge devices with stringent flash memory and SRAM constraints. Moreover, their high accuracy supports reliable and real-time VF detection in

AEDs, contributing to improved patient outcomes in critical OHCA scenarios. This study underscores the potential of optimized DNN models for advancing the functionality and accessibility of life-saving medical devices.

Author Contributions: Conceptualization, V.K., T.S. and I.J.; methodology, V.K., T.S. and I.J.; software, V.K. and T.S.; validation, V.K. and I.J.; formal analysis, V.K., T.S. and I.J.; investigation, V.K. and I.J.; resources, V.K., T.S. and I.J.; data curation, T.S.; writing—original draft, V.K. and I.J.; writing—review and editing, V.K., T.S. and I.J.; visualization, V.K., T.S. and I.J.; project administration, V.K.; funding acquisition, V.K. and I.J. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Bulgarian National Science Fund, grant number КП-06-H42/3: "Computer-aided diagnosis of cardiac arrhythmias based on machine learning and deep neural networks".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Restrictions apply to the availability of the data. Out-of-hospital cardiac arrest AED databases used in this study were obtained from a third party (Schiller Médical SAS, Wissembourg, France) and are available on request from the corresponding author with the permission of the third party.

Acknowledgments: The authors would like to thank Ramun Schmid from Schiller AG, Baar, Switzerland for his valuable advice and discussions on the topic; Jean-Philippe Didon and Sarah Ménétré from Schiller Médical SAS, Wissembourg, France; and Daniel Jost, Benoît Frattini, Clément Derkenne, Vivien Hong Tuan Ha from the Brigade des Sapeurs-Pompiers de Paris (BSPP), who provided and annotated the OHCA ECG databases under evaluation in this study.

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

References

1. Subasi, A. Chapter 3—Machine learning techniques. In *Practical Machine Learning for Data Analysis Using Python*, 1st ed.; Subasi, A., Ed.; Academic Press: Cambridge, MA, USA, 2020; pp. 91–202, ISBN 9780128213797. [CrossRef]
2. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; The MIT Press: Cambridge, UK, 2016.
3. Lucan Orasan, I.; Seiculescu, C.; Caleanu, C.D. A Brief Review of Deep Neural Network Implementations for ARM Cortex-M Processor. *Electronics* **2022**, *11*, 2545. [CrossRef]
4. Diab, M.S.; Rodriguez-Villegas, E. Embedded Machine Learning Using Microcontrollers in Wearable and Ambulatory Systems for Health and Care Applications: A Review. *IEEE Access* **2022**, *10*, 98450–98474. [CrossRef]
5. Warden, P.; Situnayake, D. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Lowpower Microcontrollers*, 1st ed.; O'Reilly Media: Boston, MA, USA, 2019; pp. 1–405.
6. Kim, E.; Kim, J.; Park, J.; Ko, H.; Kyung, Y. TinyML-Based Classification in an ECG Monitoring Embedded System. *Comput. Mater. Contin.* **2023**, *75*, 1751–1764. [CrossRef]
7. Farag, M.M. A Tiny Matched Filter-Based CNN for Inter-Patient ECG Classification and Arrhythmia Detection at the Edge. *Sensors* **2023**, *23*, 1365. [CrossRef]
8. Immonen, R.; Hämäläinen, T. Tiny Machine Learning for Resource-Constrained Microcontrollers. *J. Sens.* **2022**, *2022*, 7437023. [CrossRef]
9. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
10. Berthelier, A.; Chateau, T.; Duffner, S.; Garcia, C.; Blanc, C. Deep Model Compression and Architecture Optimization for Embedded Systems: A Survey. *J. Signal Process. Syst.* **2020**, *93*, 863–878. [CrossRef]
11. Busia, P.; Scrugli, M.A.; Jung, V.J.-B.; Benini, L.; Meloni, P. A Tiny Transformer for Low-Power Arrhythmia Classification on Microcontrollers. *IEEE Trans. Biomed. Circuits Syst.* **2024**, *99*, 1–11. [CrossRef]
12. Chen, J.; Jiang, M.; Zhang, X.; da Silva, D.S.; de Albuquerque, V.H.C.; Wu, W. Implementing ultra-lightweight co-inference model in ubiquitous edge device for atrial fibrillation detection. *Expert Syst. Appl.* **2023**, *216*, 119407. [CrossRef]
13. Gragnaniello, M.; Borghese, A.; Marrazzo, V.R.; Maresca, L.; Breglio, G.; Irace, A.; Riccio, M. Real-Time Myocardial Infarction Detection Approaches with a Microcontroller-Based Edge-AI Device. *Sensors* **2024**, *24*, 828. [CrossRef]

14. Zhao, X.; Xu, R.; Guo, X. Post-training Quantization or Quantization-aware Training? That is the Question. In Proceedings of the China Semiconductor Technology International Conference (CSTIC), Shanghai, China, 26–27 June 2023; pp. 1–3. [CrossRef]
15. Novac, P.-E.; Boukli Hacene, G.; Pegatoquet, A.; Miramond, B.; Gripon, V. Quantization and Deployment of Deep Neural Networks on Microcontrollers. *Sensors* **2021**, *21*, 2984. [CrossRef] [PubMed]
16. De Melo Ribeiro, H.; Arnold, A.; Howard, J.P.; Shun-Shin, M.J.; Zhang, Y.; Francis, D.P.; Lim, P.B.; Whinnett, Z.; Zolgharni, M. ECG-based real-time arrhythmia monitoring using quantized deep neural networks: A feasibility study. *Comput. Biol. Med.* **2022**, *143*, 105249. [CrossRef] [PubMed]
17. Wei, L.; Ma, Z.; Yang, C.; Yao, Q. Advances in the Neural Network Quantization: A Comprehensive Review. *Appl. Sci.* **2024**, *14*, 7445. [CrossRef]
18. Maly, J.; Saab, R. A simple approach for quantizing neural networks. *Appl. Comput. Harmon. Anal.* **2023**, *66*, 138–150. [CrossRef]
19. Tang, W.; Hua, G.; Wang, L. *How to Train a Compact Binary Neural Network with High Accuracy?* AAAI: Menlo Park, CA, USA, 2017; pp. 2625–2631.
20. Pal Chowdhury, A.; Kulkarni, P.; Nazm Bojnordi, M. MB-CNN: Memristive Binary Convolutional Neural Networks for Embedded Mobile Devices. *J. Low Power Electron. Appl.* **2018**, *8*, 38. [CrossRef]
21. TensorFlow. Pruning Comprehensive Guide. Available online: https://www.tensorflow.org/model_optimization/guide/pruning/comprehensive_guide (accessed on 28 November 2024).
22. Yu, J.; Lukefahr, A.; Palframan, D.; Dasika, G.; Das, R.; Mahlke, S. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. *Sigarch. Comput. Archit. News* **2017**, *45*, 548–560. [CrossRef]
23. Google. TensorFlow Model Optimization. 2020. Available online: https://www.tensorflow.org/model_optimization/guide (accessed on 28 November 2024).
24. Misko, J.; Jadhav, S.S.; Kim, Y. Extensible Embedded Processor for Convolutional Neural Networks. *Sci. Program.* **2021**, *2021*, 6630552. [CrossRef]
25. Altman, M.B.; Wan, W.; Hosseini, A.S.; Nowdeh, S.A.; Alizadeh, M. Machine learning algorithms for FPGA Implementation in biomedical engineering applications: A review. *Heliyon* **2024**, *10*, e26652. [CrossRef]
26. Mohan, N.; Hosni, A.; Atef, M. Neural Networks Implementations on FPGA for Biomedical Applications: A Review. *SN Comput. Sci.* **2024**, *5*, 1004. [CrossRef]
27. Sanaullah, A.; Yang, C.; Alexeev, Y.; Yoshii, K.; Herbordt, M.C. Real-time data analysis for medical diagnosis using FPGA-accelerated neural networks. *BMC Bioinform.* **2018**, *19* (Suppl. 18), 490. [CrossRef]
28. CMSIS NN Software Library. Available online: https://arm-software.github.io/CMSIS-NN/latest/index.html (accessed on 28 November 2024).
29. LiteRT for Microcontrollers. Available online: https://ai.google.dev/edge/litert/microcontrollers/overview (accessed on 28 November 2024).
30. uTensor: TinyML AI Inference Library. Available online: https://github.com/uTensor/uTensor (accessed on 28 November 2024).
31. PyTorch Mobile: End-to-End Workflow from Training to Deployment for iOS and Android Mobile Devices. Available online: https://pytorch.org/mobile/home/ (accessed on 28 November 2024).
32. STMicroelectronics. X-CUBE-AI—AI Expansion Pack for STM32CubeMX. Available online: http://www.st.com/en/embedded-software/x-cube-ai.html (accessed on 28 November 2024).
33. Xu, X.; Liu, H. ECG heartbeat classification using convolutional neural networks. *IEEE Access* **2020**, *8*, 8614–8619. [CrossRef]
34. Shaker, A.M.; Tantawi, M.; Shedeed, H.A.; Tolba, M.F. Heartbeat Classification Using 1D Convolutional Neural Networks. *Adv. Intell. Syst. Comput.* **2020**, *1058*, 502–511. [CrossRef]
35. Fan, X.; Yao, Q.; Cai, Y.; Miao, F.; Sun, F.; Li, Y. Multiscaled Fusion of Deep Convolutional Neural Networks for Screening Atrial Fibrillation from Single Lead Short ECG Recordings. *IEEE J. Biomed. Health Inf.* **2018**, *22*, 1744–1753. [CrossRef] [PubMed]
36. Rubin, J.; Parvaneh, S.; Rahman, A.; Conroy, B.; Babaeizadeh, S. Densely connected convolutional networks for detection of atrial fibrillation from short single-lead ECG recordings. *J. Electrocardiol.* **2018**, *51*, S18–S21. [CrossRef]
37. Parvaneh, S.; Rubin, J.; Rahman, A.; Conroy, B.; Babaeizadeh, S. Analyzing single-lead short ECG recordings using dense convolutional neural networks and feature-based post-processing to detect atrial fibrillation. *Physiol. Meas.* **2018**, *39*, 084003. [CrossRef]
38. Zhao, Z.; Särkkä, S.; Rad, A.B. Kalman-based spectro-temporal ECG analysis using deep convolutional networks for atrial fibrillation detection. *J. Signal Process. Syst.* **2020**, *92*, 621–636. [CrossRef]
39. Acharya, U.R.; Fujita, H.; Oh, S.L.; Hagiwara, Y.; Tan, J.H.; Adam, M. Automated detection of arrhythmias using different intervals of tachycardia ECG segments with convolutional neural network. *Inf. Sci.* **2017**, *405*, 81–90. [CrossRef]
40. Fujita, H.; Cimr, D. Computer aided detection for fibrillations and flutters using deep convolutional neural network. *Inf. Sci.* **2019**, *486*, 231–239. [CrossRef]

41. Tutuko, B.; Nurmaini, S.; Tondas, A.E.; Rachmatullah, M.N.; Darmawahyuni, A.; Esafri, R.; Firdaus, F.; Sapitri, A.I. AFibNet: An implementation of atrial fibrillation detection with convolutional neural network. *BMC Med. Inf. Decis. Mak.* **2021**, *21*, 216. [CrossRef]
42. Nurmaini, S.; Tondas, A.E.; Darmawahyuni, A.; Rachmatullah, M.N.; Partan, R.U.; Firdaus, F.; Tutuko, B.; Pratiwi, F.; Juliano, A.H.; Khoirani, R. Robust detection of atrial fibrillation from short-term electrocardiogram using convolutional neural networks. *Future Gener. Comput.* **2020**, *113*, 304–317. [CrossRef]
43. Fan, X.; Hu, Z.; Wang, R.; Yin, L.; Li, Y.; Cai, Y. A novel hybrid network of fusing rhythmic and morphological features for atrial fibrillation detection on mobile ECG signals. *Neural Comput. Appl.* **2020**, *32*, 8101–8113. [CrossRef]
44. Xia, Y.; Wulan, N.; Wang, K.; Zhang, H. Detecting atrial fibrillation by deep convolutional neural networks. *Comput. Biol. Med.* **2018**, *93*, 84–92. [CrossRef] [PubMed]
45. Attia, Z.I.; Noseworthy, P.A.; Lopez-Jimenez, F.; Asirvatham, S.J.; Deshmukh, A.J.; Gersh, B.J.; E Carter, R.; Yao, X.; A Rabinstein, A.; Erickson, B.J.; et al. An artificial intelligence enabled ECG algorithm for the identification of patients with atrial fibrillation during sinus rhythm: A retrospective analysis of outcome prediction. *Lancet* **2019**, *394*, 861–867. [CrossRef] [PubMed]
46. Lai, D.; Bu, Y.; Su, Y.; Zhang, X.; Ma, C.S. Non-standardized patch-based ECG lead together with deep learning based algorithm for automatic screening of atrial fibrillation. *IEEE J. Biomed. Health Inform.* **2020**, *24*, 1569–1578. [CrossRef]
47. Zhang, C.-J.; Lu, Y.; Tang, F.-Q.; Cai, H.-P.; Qian, Y.-F.; Wang, C. Heart failure classification using deep learning to extract spatiotemporal features from ECG. *BMC Med. Inf. Decis. Mak.* **2024**, *24*, 17. [CrossRef]
48. Liu, N.; Wang, L.; Chang, Q.; Xing, Y.; Zhou, X. A Simple and Effective Method for Detecting Myocardial Infarction Based on Deep Convolutional Neural Network. *J. Med. Imaging Health Inform.* **2018**, *8*, 1508–1512. [CrossRef]
49. Hammad, M.; Alkinani, M.H.; Gupta, B.B.; Abd El-Latif, A.A. Myocardial Infarction Detection Based on Deep Neural Network on Imbalanced Data. *Multimed. Syst.* **2022**, *28*, 1373–1385. [CrossRef]
50. Rawal, V.; Prajapati, P.; Darji, A. Hardware Implementation of 1D-CNN Architecture for ECG Arrhythmia Classification. *Biomed. Sig. Proc. Contr.* **2023**, *85*, 104865. [CrossRef]
51. Aruna, V.B.K.L.; Chitra, E.; Padmaja, M. Accelerating deep convolutional neural network on FPGA for ECG signal classification. *Microprocess. Microsyst.* **2023**, *103*, 104939. [CrossRef]
52. Sá, P.; Aidos, H.; Roma, N.; Tomás, P. Heart Disease Detection Architecture for Lead I Off-the-Person ECG Monitoring Devices. In Proceedings of the 27th European Signal Processing Conference (EUSIPCO), A Coruna, Spain, 2–6 September 2019. [CrossRef]
53. Zhao, Y.; Shang, Z.; Lian, Y. A 13.34 μW Event Driven Patient-Specific ANN Cardiac Arrythmia Classification for Wearable ECG Sensors. *IEEE Trans. Biomed. Circ. Sys.* **2020**, *14*, 186–197. [CrossRef]
54. Lu, J.; Liu, D.; Hu, A.; Zhang, C.; Mo, C.; Guo, R.; Li, H. A Low-cost and Configurable Hardware Architecture of Sparse 1-D CNN for ECG Classification. In Proceedings of the 2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT), Nangjing, China, 25–28 October 2022. [CrossRef]
55. Ran, S.; Yang, X.; Liu, M.; Zhang, Y.; Cheng, C.; Zhu, H.; Yuan, Y. Homecare-Oriented ECG Diagnosis With Large-Scale Deep Neural Network for Continuous Monitoring on Embedded Devices. *IEEE Trans. Instr. Meas.* **2022**, *71*, 2503113. [CrossRef]
56. Lu, J.; Liu, D.; Cheng, X.; Wei, L.; Hu, A.; Zou, X. An Efficient Unstructured Sparse Convolutional Neural Network Accelerator for Wearable ECG Classification Device. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2022**, *69*, 4572–4582. [CrossRef]
57. Faraone, A.; Delgado-Gonzalo, R. Convolutional-recurrent neural networks on low-power wearable platforms for cardiac arrhythmia detection. In Proceedings of the 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Genova, Italy, 31 August–2 September 2020; pp. 153–157.
58. Ingolfsson, T.M.; Wang, X.; Hersche, M.; Burrello, A.; Cavigelli, L.; Benini, L. ECG-TCN: Wearable cardiac arrhythmia detection with a temporal convolutional network. In Proceedings of the IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), Washington, DC, USA, 6–9 June 2021; pp. 1–4. [CrossRef]
59. Kim, S.; Chon, S.; Kim, J.-K.; Kim, J.; Gil, Y.; Jung, S. Lightweight Convolutional Neural Network for Real-Time Arrhythmia Classification on Low-Power Wearable Electrocardiograph. In Proceedings of the 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), Scottish Event Campus, Glasgow, UK, 11–15 July 2022; pp. 1915–1918. [CrossRef]
60. Huang, Z.; Herbozo Contreras, L.F.; Leung, W.H.; Yu, L.; Truong, N.D.; Nikpour, A.; Kavehei, O. Efficient Edge-AI Models for Robust ECG Abnormality Detection on Resource-Constrained Hardware. *J. Cardiovasc. Transl. Res.* **2024**, *17*, 879–892. [CrossRef] [PubMed]
61. Sivapalan, G.; Nundy, K.K.; Dev, S.; Cardiff, B.; John, D. ANNet: A Lightweight Neural Network for ECG Anomaly Detection in IoT Edge Sensors. *IEEE Tran. Biomed. Circuits Syst.* **2022**, *16*, 24–35. [CrossRef]
62. Elola, A.; Aramendi, E.; Irusta, U.; Picón, A.; Alonso, E.; Owens, P.; Idris, A. Deep Neural Networks for ECG-Based Pulse Detection during Out-of-Hospital Cardiac Arrest. *Entropy* **2019**, *21*, 305. [CrossRef]

63. Nguyen, M.T.; Kiseon, K. *Feature* Learning Using Convolutional Neural Network for Cardiac Arrest Detection. In Proceedings of the 2018 International Conference on Smart Green Technology in Electrical and Information Systems (ICSGTEIS), Bali, Indonesia, 25–27 October 2018; pp. 39–42. [CrossRef]
64. Acharya, U.R.; Fujita, H.; Oh, S.L.; Raghavendra, U.; Tan, J.H.; Adam, M.; Gertych, A.; Hagiwara, Y. Automated identification of shockable and non-shockable life-threatening ventricular arrhythmias using convolutional neural network. *Future Gener. Comput. Syst.* **2018**, *79*, 952–959. [CrossRef]
65. Irusta, U.; Aramendi, E.; Chicote, B.; Alonso, D.; Corcuera, C.; Veintemillas, J.; Larrea, A.; Olabarria, M. Deep learning approach for a shock advise algorithm using short electrocardiogram analysis intervals. *Resuscitation* **2019**, *142*, e28–e114. [CrossRef]
66. Picon, A.; Irusta, U.; lvarez-Gila, A.; Aramendi, E.; Alonso-Atienza, F.; Figuera, C.; Ayala, U.; Garrote, E.; Wik, L.; Kramer-Johansen, J.; et al. Mixed convolutional and long short-term memory network for the detection of lethal ventricular arrhythmia. *PLoS ONE* **2019**, *14*, e0216756. [CrossRef]
67. Krasteva, V.; Ménétré, S.; Didon, J.-P.; Jekova, I. Fully Convolutional Deep Neural Networks with Optimized Hyperparameters for Detection of Shockable and Non-Shockable Rhythms. *Sensors* **2020**, *20*, 2875. [CrossRef]
68. Isasi, I.; Irusta, U.; Aramendi, E.; Eftestøl, T.; Kramer-Johansen, J.; Wik, L. Rhythm Analysis during Cardiopulmonary Resuscitation Using Convolutional Neural Networks. *Entropy* **2020**, *22*, 595. [CrossRef]
69. Isasi, I.; Irusta, U.; Aramendi, E.; Olsen, J.-Å.; Wik, L. Detection of shockable rhythms using convolutional neural networks during chest compressions provided by a load distributing band. In Proceedings of the 2020 Computing in Cardiology Conference (CinC), Rimini, Italy, 13–16 September 2020; Volume 47, pp. 1–4. [CrossRef]
70. Hajeb, M.S.; Cascella, A.; Valentine, M.; Chon, K.H. Deep Neural Network Approach for Continuous ECG-Based Automated External Defibrillator Shock Advisory System During Cardiopulmonary Resuscitation. *J. Am. Heart Assoc.* **2021**, *10*, e019065. [CrossRef] [PubMed]
71. Gong, Y.; Wei, L.; Yan, S.; Zuo, F.; Zhang, H.; Li, Y. Transfer learning based deep network for signal restoration and rhythm analysis during cardiopulmonary resuscitation using only the ECG waveform. *Inf. Sci.* **2023**, *626*, 754–772. [CrossRef]
72. Jekova, I.; Krasteva, V. Optimization of End-to-End Convolutional Neural Networks for Analysis of Out-of-Hospital Cardiac Arrest Rhythms during Cardiopulmonary Resuscitation. *Sensors* **2021**, *21*, 4105. [CrossRef]
73. Krasteva, V.; Didon, J.-P.; Ménétré, S.; Jekova, I. Deep Learning Strategy for Sliding ECG Analysis during Cardiopulmonary Resuscitation: Influence of the Hands-Off Time on Accuracy. *Sensors* **2023**, *23*, 4500. [CrossRef] [PubMed]
74. Shen, C.P.; Freed, B.C.; Walter, D.P.; Perry, J.C.; Barakat, A.F.; Elashery, A.R.A.; Shah, K.S.; Kutty, S.; McGillion, M.; Ng, F.S.; et al. Convolution Neural Network Algorithm for Shockable Arrhythmia Classification Within a Digitally Connected Automated External Defibrillator. *J. Am. Heart Assoc.* **2023**, *12*, e026974. [CrossRef]
75. LiteRT Overview. Available online: https://ai.google.dev/edge/litert (accessed on 28 November 2024).
76. Model Conversion Overview. Available online: https://ai.google.dev/edge/litert/models/convert (accessed on 28 November 2024).
77. Post-Training Quantization. Available online: https://ai.google.dev/edge/litert/models/post_training_quantization (accessed on 28 November 2024).
78. LiteRT 8-Bit Quantization Specification. Available online: https://ai.google.dev/edge/litert/models/quantization_spec (accessed on 28 November 2024).
79. Raspberry Pi. Available online: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (accessed on 28 November 2024).
80. Discovery Kit with STM32F769NI MCU. Available online: https://www.st.com/en/evaluation-tools/32f769idiscovery.html (accessed on 28 November 2024).
81. Integrated Development Environment for STM32. Available online: https://www.st.com/en/development-tools/stm32cubeide.html (accessed on 28 November 2024).
82. Koster, R.; Baubin, M.; Bossaert, L.; Caballero, A.; Cassan, P.; Castrén, M.; Smyth, M.A.; Olasveengen, T.; Monsieurs, K.G.; Raffay, V.; et al. European Resuscitation Council Guidelines for Resuscitation 2010 Section 2. Adult basic life support and use of automated external defibrillators. *Resuscitation* **2010**, *81*, 1277–1292. [CrossRef]
83. Perkins, G.D.; Handley, A.J.; Koster, R.W.; Castrén, M.; Smyth, M.A.; Olasveengen, T.; Monsieurs, K.G. European Resuscitation Council Guidelines for Resuscitation 2015: Section 2. Adult basic life support and automated external defibrillation. *Resuscitation* **2015**, *95*, 81–99. [CrossRef]
84. Kerber, R.E.; Becker, L.B.; Bourland, J.D.; Cummins, R.O.; Hallstrom, A.P.; Michos, M.B.; Nichol, G.; Ornato, J.P.; Thies, W.H.; White, R.D.; et al. Automatic External Defibrillators for Public Access Defibrillation: Recommendations for Specifying and Reporting Arrhythmia Analysis Algorithm Performance, Incorporating New Waveforms, and Enhancing Safety. *Circulation* **1997**, *95*, 1677–1682. [CrossRef]
85. Ye, N.; Zeng, Z.; Zhou, J.; Zhu, L.; Duan, Y.; Wu, Y.; Wu, J.; Zeng, H.; Gu, Q.; Wang, X.; et al. OoD-Control: Generalizing Control in Unseen Environments. *IEEE Trans Pattern. Anal. Mach. Intell.* **2024**, *46*, 7421–7433. [CrossRef]
86. TensorFlow Lite for Microcontrollers. Available online: https://github.com/tensorflow/tflite-micro (accessed on 7 July 2023).

87. Huber, P.; Göhner, U.; Trapp, M.; Zender, J.; Lichtenberg, R. Analysis of Neural Network Inference Response Times on Embedded Platforms. In Proceedings of the 2024 Asian Conference on Communication and Networks (ASIANComNet), Bangkok, Thailand, 24–27 October 2024; pp. 1–7. [CrossRef]
88. Ultra-Low-Power with FPU Arm Cortex-M4 MCU 80 MHz with 1 Mbyte of Flash Memory, USB OTG, DFSDM. Available online: https://www.st.com/en/microcontrollers-microprocessors/stm32l475vg.html (accessed on 28 November 2024).
89. Nordic Semiconductor nRF52 Series SoCs. Available online: https://www.mouser.bg/new/nordic-semiconductor/nrf52-series-soc/ (accessed on 28 November 2024).
90. Nordic Semiconductor nRF52840 Multi-Protocol 2.4GHz SoC. Available online: https://www.mouser.bg/new/nordic-semiconductor/nordic-nrf52840-soc/ (accessed on 28 November 2024).