

Deep Learning Based Arrhythmia Classification On Low-cost And Low-compute MCU

by

Md Abu Obaida Zishan

18201214

H M Shihab

19101585

Gazi Mashrur Rahman

18241003

Sabik Sadman Islam

18301029

Maliha Alam Riya

19101270

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
January 2023

© 2023. Brac University
All rights reserved.

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing our Computer Science and Engineering degree at BRAC University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material that has been accepted or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

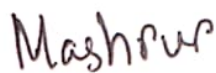
Student's Full Name & Signature:



Md Abu Obaida Zishan
18201214



H M Shihab
19101585



Gazi Mashrur Rahman
18241003



Sabik Sadman Islam
18301029



Maliha Alam Riya
19101270

Approval

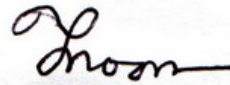
The thesis titled “Deep Learning Based Arrhythmia Classification On Low-cost And Low-compute MCU” was submitted by

1. Md Abu Obaida Zishan(18201214)
2. H M Shihab(19101585)
3. Gazi Mashrur Rahman(18241003)
4. Sabik Sadman Islam(18301029)
5. Maliha Alam Riya(19101270)

Of Fall, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on January 19, 2023.

Examining Committee:

Supervisor:
(Member)



Jannatun Noor Mukta
Designation
Department of Computer Science and Engineering
BRAC University

Thesis Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
BRAC University

Abstract

According to WHO, cardiovascular disease (CVD) is the leading cause of death globally. Unfortunately, these diseases are difficult to diagnose without proper equipment which is not cheap. One of the reasons for such a high cost of treatment is the use of expensive technologies like ECG or electrocardiograph monitoring systems. These monitoring systems are usually implemented using expensive high-compute hardware and proprietary algorithms. Conventional ECG systems cost between \$2000 and \$10,000. But in theory, these systems can also be developed through low-compute hardware (such as microcontrollers or FPGA) and machine learning. This paper performs a comparative study on the implementation of low-cost, low-power, and low-compute-based ECG systems and analyzes better approaches for future design. Additionally, it implements an ECG monitoring system based on that approach.

Keywords: ECG; arrhythmia; low-cost; low-compute; low-power; machine learning; micro-controller

Acknowledgement

Firstly, we praise Allah for whom our thesis was completed without any major interruption.

Secondly, to our supervisor Jannatun Noor Mukta madam, for her kind support and advice in our work. She helped us whenever we needed help.

And finally to our parents. With their kind support and prayer, we are now on the verge of graduation.

Table of Contents

Declaration	i
Approval	ii
Abstract	iv
Acknowledgment	v
Table of Contents	vi
List of Figures	viii
List of Tables	ix
Nomenclature	xi
1 Introduction	1
1.1 Background And Motivation	1
1.2 Problem Statement	1
1.3 Research Contributions	3
1.4 Thesis Organization	3
2 Overview Of ECG Monitoring System	4
2.1 Electrodes Or Leads	4
2.2 Analog Circuitry	5
2.3 Detection Algorithms	5
3 Literature Review and Comparative Analysis	7
3.1 Comparison Metrics	7
3.2 Analysis	8
3.3 Comparative Results And Discussion	10
4 System Design	11
4.1 Hardware	11
4.1.1 Arduino Nano	11
4.1.2 AD8232 SparkFun Single-Lead Heart Rate Monitor	12
4.2 Software	13
4.2.1 Heartbeat Detection Module (HDM)	13
4.2.2 Arrhythmia Detection Module (ADM)	13

5	Heartbeat Detection Module	15
5.0.1	Pan-Tomkins Algorithm	15
5.0.2	Our Implementation Of The PT Algorithm	15
6	Arrhythmia Detection Module	18
6.1	Deep Neural Network (DNN)	18
6.1.1	Perceptron	18
6.1.2	Multi Layered Perceptron	20
6.2	Implemented DNN	22
6.2.1	Dataset	22
6.2.2	Data Preprocessing	23
6.2.3	Neural Network	25
6.2.4	Sigmoid-Sigmoid	26
6.2.5	ReLU-Sigmoid	27
6.2.6	ReLU-Softmax	28
6.2.7	Sigmoid-Softmax	29
6.2.8	Error Function	30
6.2.9	Optimizer	30
6.2.10	Quantization And Model Compression	31
6.2.11	Temporary Dequantization And Model Selection	34
6.2.12	Performance	35
6.3	Porting	44
7	Commercial Low-cost ECG monitoring Systems	45
8	Conclusion & Future Work	47
A	Code	50

List of Figures

1	Single-lead ECG devices [4].	2
2	Basic ECG signal processing chain	4
3	Types of modern ECG monitoring system	5
4	A voltage vs time representation of a heartbeat / QRS complex	6
5	System hardware setup.	11
6	System flow-diagram	13
7	Code excerpt from the PT algorithm[21].	17
8	Gradient descent	20
9	Single layered perceptron	20
10	Multi-layered perceptron	22
11	Normal beat preprocessed and unchanged	24
12	Supraventricular ectopic beat preprocessed and unchanged	24
13	Ventricular ectopic beat preprocessed and original	25
14	Fusion beat preprocessed and original	25
15	Sigmoid-sigmoid based network architecture	26
16	Sigmoid-sigmoid based network training convergence	27
17	ReLU-sigmoid based network architecture	27
18	ReLU-sigmoid based network training convergence	28
19	ReLU-softmax based network architecture	28
20	ReLU-softmax based network training convergence	29
21	Sigmoid-softmax based network architecture	29
22	Sigmoid-softmax based network training convergence	30
23	Quantizing for compression	32
24	Fog computing	47
25	Edge computing	48

List of Tables

i	QRS wave analysis	6
ii	Comparative analysis based on cost, compute, power and detection algorithm	8
iii	Arduino Nano specifications	12
iv	Sparkfun ECG AD8232 Sensor board [20]	12
v	Differentiating our implementation and the PT algorithm	17
vi	Classification of Arrhythmia according to AAMI [24]	23
vii	Accuracy drop in (appx. in %) when zero is mapped to zero vs when not while quantizing	33
viii	Accuracy (appx.in %) comparision between model types in default, qunatized and, dequantized state	34
ix	Default, quantized, dequantized state of weight-bias for each layer and corresponding accuracy (appx in %) for the sigmoid-sigmoid model	34
x	Performance of dequantized sigmoid-sigmoid model (selected)	35
xi	True vs False positives and negatives	36
xii	Train-test split of the used dataset	37
xiii	FLOPs & memory breakdown per layer	38
xiv	System SRAM consumption breakdown in MCU	38
xv	Performance comparison with the best arrhythmia classification models studied	39
xvi	Relu-sigmoid dequantized model	39
xvii	Sigmoid-sigmoid quantized model	40
xviii	Sigmoid-softmax default model	40
xix	Sigmoid-softmax dequantized model	41
xx	Sigmoid-softmax quantized model	41
xxi	Sigmoid-sigmoid default model	42
xxii	Relu-sigmoid default model	42
xxiii	Relu-sigmoid quantized model	43
xxiv	Relu-softmax default model	43
xxv	Relu-softmax dequantized model	44
xxvi	Relu-softmax quantized model	44
xxvii	Cost comparison of low-cost commercially available ECG monitoring systems	45

Listings

1	Code excerpt from our modification of the PT algorithm	50
2	Softmax implementation in cpp	51
3	Temporary dequantization implementation	51
4	Excerpt of the model ported as header	52

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

AAMI Association for the Advancement of Medical Instrumentation

ADM Arrhythmia Detection Module

ANN Artificial Neural Network

CVD Cardio Vascular Disease

DNN Deep Neural Network

ECG Electro-cardiogram

HDM Heart beat Detection Module

MCU Micro Controller Unit

PT Pan-Tomkins

SBC Single Board Computer

WHO World Health Organization

Chapter 1

Introduction

1.1 Background And Motivation

According to WHO [1], An estimated 17.9 million people died from CVDs in 2019, representing 32% of all global deaths. Of these deaths, 85% were due to heart attack and stroke. This implies that a significant portion of global deaths is caused by CVDs. One might assume this portion of death is only from high-income and high-middle-income groups, but the leading cause of death in low and middle-income countries is CVD [1].

One of the most vital devices used to diagnose such diseases is the ECG or Electrocardiogram machine. Electrocardiography is the process of producing an electrocardiogram (ECG or EKG), a recording of the heart's electrical activity [2]. An electrocardiogram is a voltage versus time graph of a heart's electrical activity [3]. The ECG system simply draws a graphical representation of the heartbeat on a screen and/or paper using probes that are attached to the patient's torso by which physicians can analyze the state of the patient's heartbeat patterns. Modern ECG systems also provide alerts, whenever the patient is having irregular heartbeat patterns or arrhythmia.

1.2 Problem Statement

Although recent studies provide successful approaches for developing ECG monitoring systems, not all of them are cost-effective. Since any ECG monitoring system would require running 24/7, power consumption is also a factor overlooked by most studies. Also, ECG monitoring systems are used once for every required patient in a hospital, which further multiplies the importance of low-power and low-cost systems.

Chen et al.[5], Sakib et al.[6], Ahsanuzzaman et al.[7]and Hartman et al.[8] developed ECG monitoring system on single board computers (SBC). Faraone et al.[9], Scirè et al.[10] and Raj et al.[11] developed the same but with mere micro-controller units. Additionally, all of them implemented machine-learning-based arrhythmia detection on their devices including studies where MCUs were used as the main

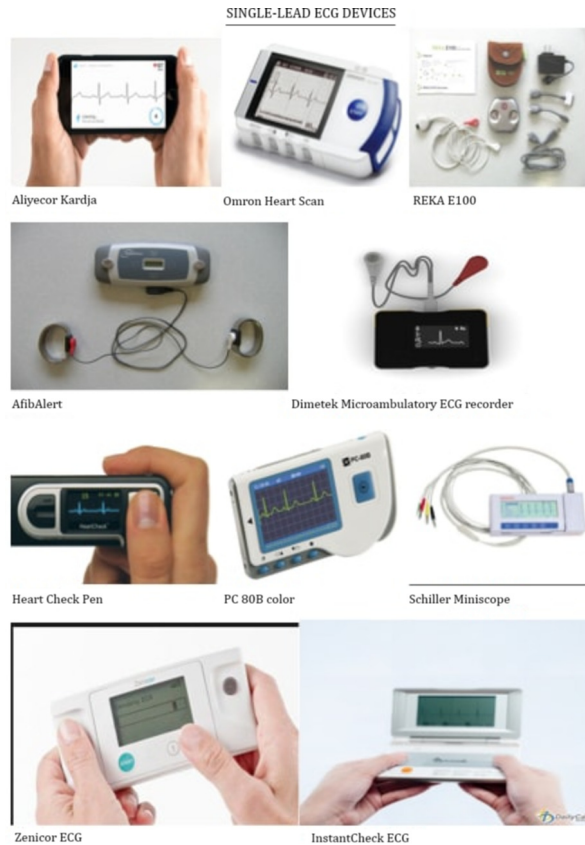


Figure 1: Single-lead ECG devices [4].

device.

Using MCUs instead of SBCs lowers the power consumption by a factor of 1000 at the very least since MCUs operate within milliwatt ranges. Moreover, MCUs are comparatively cheaper than SBCs. Hence, for broad-scale deployment and 24/7 duty, MCUs are a much better choice than SBCs for ECG monitoring system development. Although, development with MCUs does not necessarily mean sacrificing any features available in modern ECG monitoring systems since some of the above-mentioned studies already proved that.

However, one might assume that arrhythmia detection on ECG monitoring systems may require cycle expensive algorithm like machine learning which might not be feasible for MCUs. But inference or forward pass of a small enough machine learning model is possible. Since they are primarily simple matrix multiplication whose values are passed through activation functions of a few varieties. Even training machine learning models have been possible in extremely low constraints as MCU[12]. Therefore, with an efficient enough machine learning model for arrhythmia detection, one can develop a cost, compute, and power-efficient ECG Monitoring System based on MCUs with little to no drawbacks compared to industry-grade systems.

Moreover, the high cost of treatment for CVD has created a demand for cheaper and more efficient devices in developing and underdeveloped countries. This means developing cost-effective and efficient ECG monitoring systems could save more lives (potentially millions) by lowering CVD treatment costs. In addition, the global

market will prefer systems that are cost, power, and compute efficient.

1.3 Research Contributions

Therefore, it is clear that we need a cheaper, low-cost alternative for modern ECG monitoring systems for developing and underdeveloped countries. At the same time, we mustn't trade modern features like arrhythmia detection for cost. Based on our study, we make the following set of specific contributions to this paper-

- We perform a comparative analysis of recent studies on their approach to developing an ECG system.
- We analyze the papers on four metrics - cost, compute, power, and the use of detection algorithms
- Subsequently, we propose an implementation based on the approach inferred from the comparative analysis. Our implementation superseded most other approaches reviewed.

1.4 Thesis Organization

First, this paper provides an overview of an ECG monitoring system (section II). In section- III, we perform a comparative analysis of 9 studies conducted from 2018-2021 and propose the best approach. In section - IV we provide our system's hardware and software design. In sections - V, and VI we provide details of our main two software modules, HDM and ADM respectively. In section VII, we provide technical details on how our software was ported to MCU. Section VIII performs a cost comparison of some commercial low-cost ECG monitoring systems with our proposed implementation. Finally, we conclude this paper by discussing future directions of research and their impact.

Chapter 2

Overview Of ECG Monitoring System

ECG stands for Electrocardiography system which records the voltage versus time series data of the heart's electrical activity. The continuous depolarization followed by repolarization of the heart spreads electrical charges throughout the body as an electric volume conduit [13]. Electrical charges create a potential difference between pair(s) of electrodes attached to the skin of a subject. The signals from leads are passed through analog circuitry to amplify and filter the signal. The component breakdown and their function are as follows:

2.1 Electrodes Or Leads

The potential difference between any two electrodes can be detected. When the heart continuously depolarizes and repolarizes sequentially (or beats), pair(s) of electrodes attached to the skin, can detect the changes in potential difference(s). Such pairs are known as leads. Leads or pairs of electrodes are placed across the torso & limbs to detect heartbeats. The ECG systems are made of single or multiple

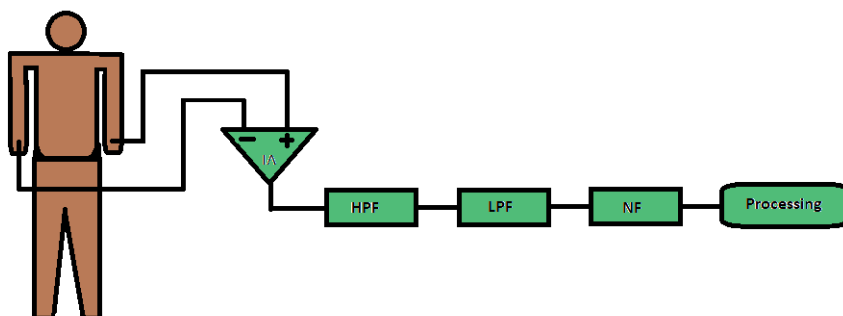


Figure 2: Basic ECG signal processing chain

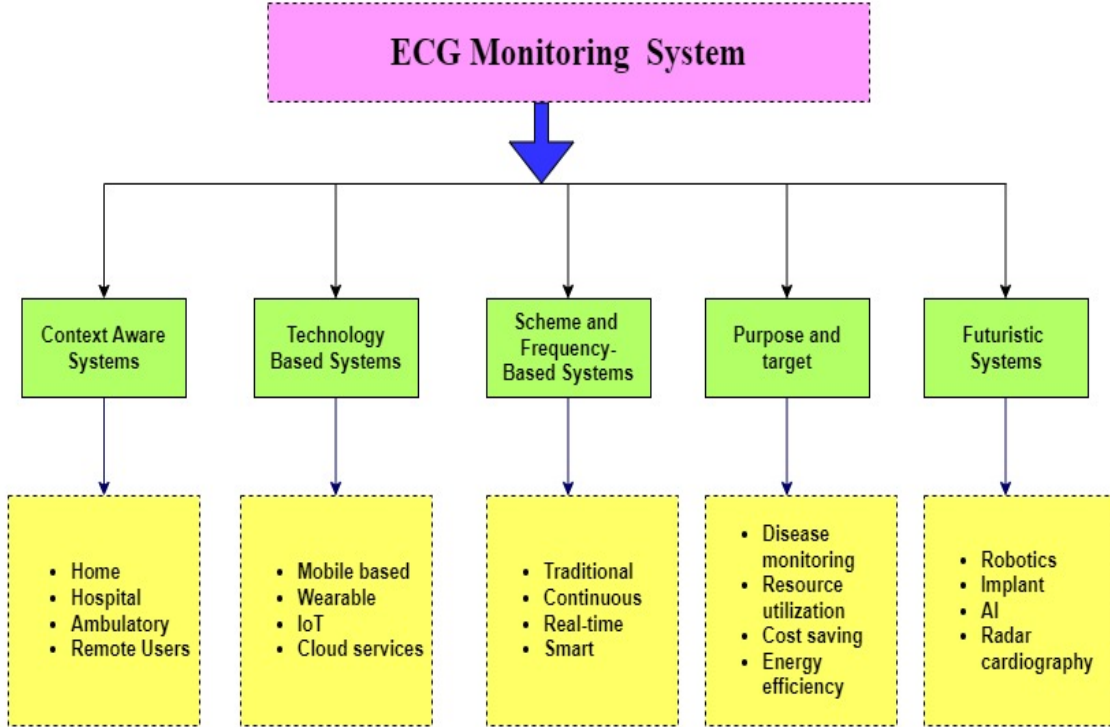


Figure 3: Types of modern ECG monitoring system

leads [14]. Although the most common system is a 12-lead ECG system.

2.2 Analog Circuitry

ECG signals are within the 0.5 - 4mV range [15], for which amplification is important. Also ECG signal contains noise, since, the entire human body is operated via electrical signals sent from the nervous system. Moreover, there is 50/60Hz interference of AC electricity running through almost every building. Therefore, ECG signals must be passed through the analog circuitry in the following order:

- Amplification: The first step is instrumental amplification[14]. The signal from leads is amplified to the desired voltage for easier reading.
- Filtration: As mentioned before, ECG signals contain noise. Therefore, the signals from amplifiers are passed through the analog low pass, high pass, and notch filters for noise removal [14]. Instead of using analog filters, one can also use digital ones implemented via FFT (Fast Fourier Transform) on computing systems.

2.3 Detection Algorithms

The filtered signal now can be passed through computing units to detect a heartbeat. After a heartbeat window is detected from an ECG signal stream, it can be passed to the arrhythmia detection module to detect arrhythmia. Most modern ECG monitoring system contains this feature. The component breakdown of

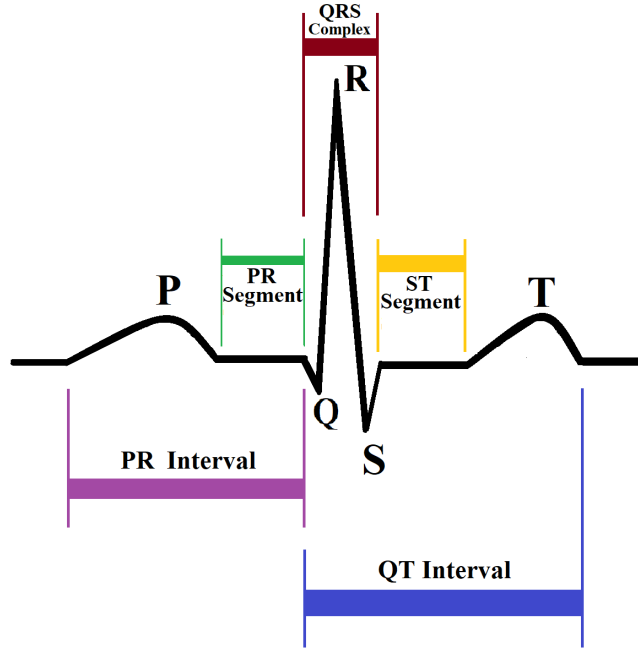


Figure 4: A voltage vs time representation of a heartbeat / QRS complex

detection algorithms is as follows:

- Heartbeat or QRS detection module (HDM): The heartbeat windows of an ECG signal wave are labeled as PQRST peaks. Detecting Q, R, and S peaks allow one to separate a heartbeat from an input data stream.

Table i: QRS wave analysis

Wave	Deflection	Action	Heart portion
Q	downwards	depolarization	interventricular septum
R	upwards	depolarization	main mass of the ventricles
S	downwards	depolarization	Purkinje fibres

- Arrhythmia detection module (ADM): The window separated as QRS or heartbeat can be passed through machine learning algorithms to classify arrhythmia.

Here, table I explains Fig-4 and relates signal deflection to the corresponding action of the heart.

Chapter 3

Literature Review and Comparative Analysis

In this section, we perform a comparative analysis of previous studies. The analysis is performed on four metrics. After discussing the comparison metrics, we analyze the studies based on those metrics. Finally, we reveal our findings in the comparative results and discussion subsection.

3.1 Comparison Metrics

1. *Cost*: Since the implementation of most ECG systems is expensive, a cheaper ECG system without significant setbacks should be considered superior. Therefore, we believe cost should be one of the deciding factors for comparison. In this study, only the cost of the development board (which has the processing unit embedded) is considered. Since, sensors, display units, etc. other components are available at variable costs and are interfaceable with any MCU/MPU, it may be better to leave them out for comparative analysis.
2. *Compute*: Proprietary ECG systems are expensive, which implies the hardware used may be high-compute based. Here, computing power implies the operating clock cycle frequency of a processor. Typically lower clock cycle frequency machines are cheaper than higher ones albeit slower. Therefore, the lower compute processor used, the cheaper the cost of the system will be. But it is important that the system designed must not have caveats that may lead to an inferior treatment or death of the patient.
3. *Power*: Power hungry system eventually will raise the cost of treatment, since healthcare complexes have to pay for electricity. Therefore low-power systems are better than power-hungry ones.
4. *Detection algorithm*: Irregular heartbeat pattern detection algorithms are used in most commercial ECG monitoring systems. Pattern detection is important as it alerts healthcare professionals to provide proper attention to the patient having an irregular heartbeat pattern. Therefore, this is a significant factor for comparison.

3.2 Analysis

Chen et al. [5] present a real-time electrocardiogram (ECG) analysis system that can detect atrial fibrillation (AF) using a Neural Network algorithm (NN). The authors used the PYNQ-Z2 development board which contains both a programmable chip (MPU) and programmable logic (FPGA). They used NN for arrhythmia/atrial fibrillation detection on the FPGA but used the MCU for feature extraction. The cost of their development board was around USD 149. Since the processor is 650 MHz dual-core, the total computing power used is 1300 MHz. The total power consumed by their board during the running of the algorithm was not disclosed.

Table ii: Comparative analysis based on cost, compute, power and detection algorithm

Study	Cost (USD)	Compute (MHz)	Power (mW)	Detection Algorithm
Chen et al. [5]	149	1300	-	NN
Sakib et al. [6]	59-99	5720	-	CNN
Ahsanuzzaman et al.[7]	35	4800	-	LSTM
Faraone et al.[9]	39	64	20.65	R-CNN
Hartman et al.[8]	35	≥ 700	-	DistilledDNN
Scirè et al. [10]	39.95	32	44.08	KNN + LSTM

Sakib et al.[6] tested their algorithm on Raspberry Pi 3, Raspberry Pi 4, and Jetson Nano. The best performance was on Jetson Nano. Since, Jetson Nano had the best performance, in this study, the rest are excluded. The cost of their development board was USD 59-99. The computing power for each core of ARM A57 on board stood at 1430 MHz, for four cores, it was 5720MHz. The authors

did not disclose the power consumption of their algorithm. The paper implemented CNN (Convolutional Neural Network) on single-channel raw ECG data and detected arrhythmia.

Ahsanuzzaman et al.[7] developed an LSTM (Long Short-Term Memory) algorithm for arrhythmia detection on Raspberry Pi 3. The cost of their main development board stood at USD 35. The power consumption of their algorithm was not discussed. The computing power for each core of ARM A53 stood at a minimum of 1200 MHz, for four cores it was 4800MHz. They also used other processing units like the ATmega328P on the Arduino Uno development board, but it was only used for interfacing and preprocessing ECG data from the ECG AD8232 sensor.

Faraone et al.[9] adapted a convolutional-recurrent neural network, designed to detect and classify cardiac arrhythmia from a single lead electrocardiogram to the low-power embedded System-onChip nRF52 from Nordic Semiconductor with an ARM CortexM4 processing core. The cost of nRF52 - DK stood at USD 39. Power consumption during the running of the algorithm on a loop stood at 20.65mW.

Hartman et al.[8] implemented distilled deep learning(dDNN) based arrhythmia detection classification on a Raspberry Pi-based platform and compared their performance with respect to three key performance indicators (KPI) of interest for health care applications: accuracy, energy efficiency, and latency. The cost of their development board stood at USD 35. The minimum computing power of the Raspberry Pi-based platform is 700MHz. They did not disclose the specific model of the development board used. The power consumption of their algorithm was not disclosed.

Rakin et al.[16] and Alam et al. [17] did not implement any detection algorithm for arrhythmia. They also used analog circuitry to develop a low-cost ECG system. Processing units were used for display purposes only. Which, was difficult for us to compare with other studies.

Scirè et al.[10] implemented heartbeat detection using KNN (k-Nearest Neighbor) and arrhythmia detection using an LSTM (Long Short Term Memory) classifier on an asymmetric embedded processor - the Intel Curie module and NXP MPC8572E module which provides a dedicated core for hardware-assisted pattern matching. The cost of their development board stood at USD 39.95. The authors reported a total of 44.08 mW of power consumed while detecting the heartbeat (15.24 mW) and classifying arrhythmia (28.84mw).

Raj et al.[11] implemented the ECG signal analysis method via discrete cosine Stockwell transform for feature extraction and artificial bee colony (ABC) optimized least-square twin support vector machines (TSVM) as classifiers on commercially available 32-bit microcontroller test platform to detect real-time heart-beat anomalies. They did not disclose the specific 32-bit microcontroller platform used or its power consumption. Which, was difficult for us to compare with other studies.

3.3 Comparative Results And Discussion

From table-II we can observe that Farone et al.[9] and Scirè et al.[10] has the lowest compute-powered processor running R-CNN and RNN algorithm respectively. Farone et al.[9] and Scirè et al.[10] reported 20.65 mW and 44.08 mW power consumed respectively. This indicates that the low-compute approach is highly power efficient which in turn will reduce the overall cost incurred of the operation of the ECG monitoring systems via electric utility bill.

In contrast, the development boards used by Chen et al.[5], Sakib et al.[6], Ahsanuzzaman et al.[7], and Hartmann et al.[8], which usually run a Linux-based operating system, consume at least 700 mW of power at idle mode with no peripherals connected. To put into perspective, that is ~ 34 devices of Farone et al.[9] or ~ 16 devices of Scirè et al.[10] can be run by the same power.

Therefore, it is clear that for low-cost, low-compute, and low-power with detection algorithms, using micro-controllers with no overhead of operating systems, might be the best approach. To provide a clearer picture of our suggested approach, we design a system in the next section.

Chapter 4

System Design

Following the revealed best approach of this paper, we implemented our own design. The outline of the entire system and the cost, compute and power parameters are discussed as follows:

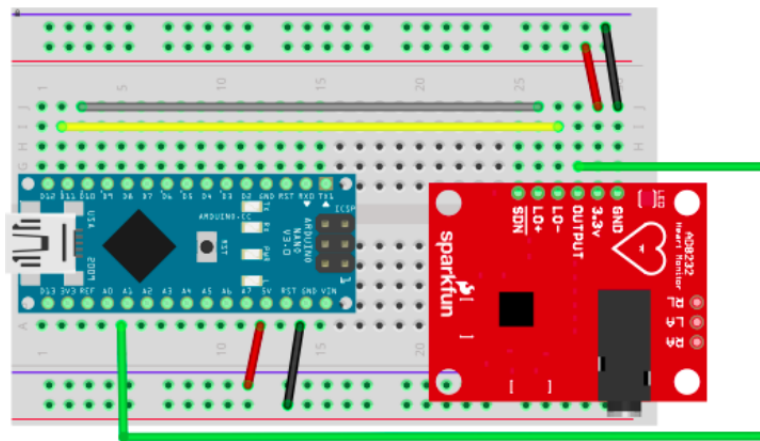


Figure 5: System hardware setup.

4.1 Hardware

For our development board chose Arduino Nano. Additionally, we used the AD8232 SparkFun Single-Lead Heart Rate Monitor sensor to read the ECG data stream. The details are as follows:

4.1.1 Arduino Nano

Based on the ATmega328P, the Arduino Nano is a compact, comprehensive, and breadboard-friendly board that was introduced in 2008. The MCU's processor, ATmega328P, is clocked at 16MHz and has a 32-bit data bus along with a 24-bit address bus [18]. Moreover, it has 2KB SRAM and 32KB Flash memory where 2KB is used by the bootloader. It has an operating voltage of 5V and current consumption of 19mA (Power consumption of 95 mW) [19]. It can be programmed using the Arduino Software integrated development environment (IDE), which is available

both online and offline and is shared by all Arduino boards. The board is affordable and widely available making it one of the most popular starter development boards for learning embedded systems programming for engineers, non-professionals, and children alike. Because of such ease of development, availability, low cost, low-compute, and low-power consumption, we chose this board as our main and only computing device.

Table iii: Arduino Nano specifications

Cost	USD 24.90
SRAM	2 KB
Flash memory	32 KB (2 KB used by bootloader)
Compute	16 MHz
Operating voltage	5.0 V
Current consumption	19 mA
DC current per I/O pins	40 mA
Power	95 mW (no peripherals)

4.1.2 AD8232 SparkFun Single-Lead Heart Rate Monitor

The AD8232 SparkFun Single-Lead Heart Rate Monitor is a low-cost board for measuring the electrical activity of the heart. This electrical activity is recorded as an ECG or electrocardiogram and output as an analog measurement. The AD8232 single-channel heart rate monitor acts as an operational amplifier to easily obtain clear signals from the PR and QT intervals [20].

Table iv: Sparkfun ECG AD8232 Sensor board [20]

Cost	USD 21.50
Leads	Single
Operating voltage	3.3 V
Current consumption	170 μ A
Power	0.561 mW

4.2 Software

There are two steps to detecting Arrhythmia from raw ECG data:

4.2.1 Heartbeat Detection Module (HDM)

The heartbeat detection module detects R peaks ECG data stream. We chose the Pan-Tompkins (PT) algorithm [21] which is highly respected and established in the bibliography.

4.2.2 Arrhythmia Detection Module (ADM)

After detecting the heartbeat, we need to classify Arrhythmia into five classes only one of which is normal via a densely connected neural network.

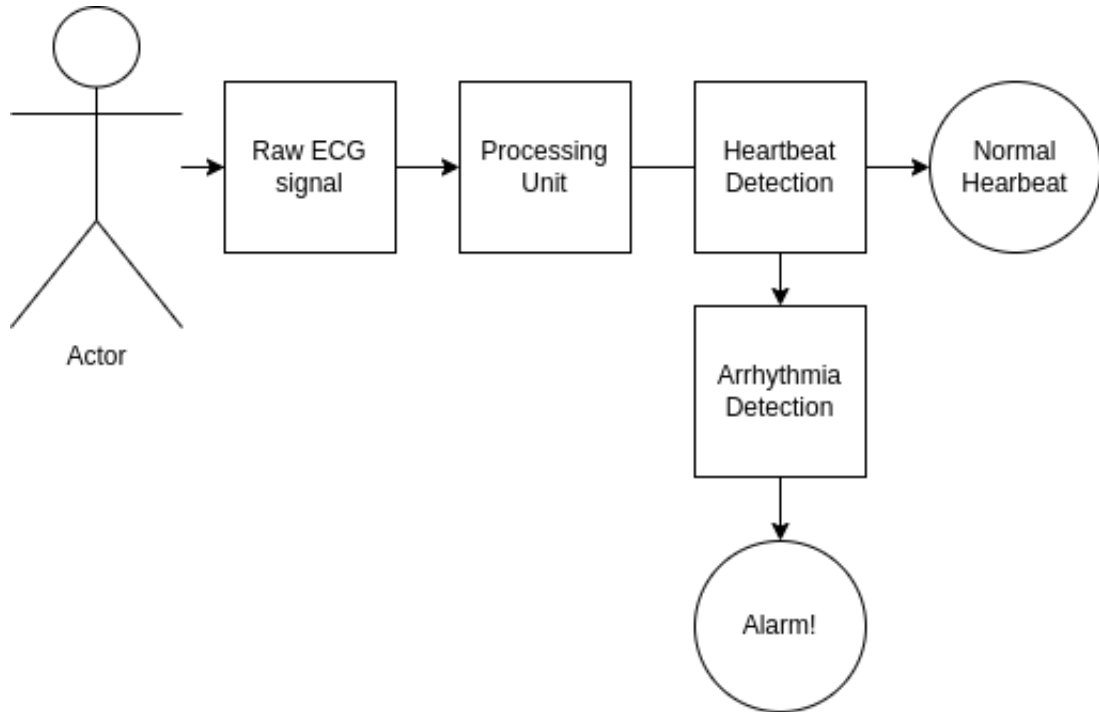


Figure 6: System flow-diagram

In figure 6, we can see the end to end flow-diagram of our system design. Raw ECG signal passes from the actor to the processing unit, where the signal is pre-processed. Afterward, the preprocessed signal is passed to the heart-beat detection module. If the signal is not a heartbeat the algorithm continues otherwise it passes the stream to the Arrhythmia detection module which if detects an abnormal beat, sets off the on-device alarm. Here, the simple alerting system is only for demonstration, a much more sophisticated alert system can be implemented if needed.

Table III presents the specifications of the Arduino Nano board. We can see that the power consumption by the board only is 95mW. However, adding peripherals that draw power may increase power consumption.

Table IV depicts the specifications of the Sparkfun ECG AD8232 Sensor board

we used. We can see that the power consumption by the board is close to 0.561 mW which barely affects the total power consumption of our system when connected.

Chapter 5

Heartbeat Detection Module

We will only discuss the software part of our implementation since the hardware used was merely connected but not modified. The discussion is as follows -

As discussed earlier we used the Pan-Tomkins algorithm in our system. However, we performed some modifications in our implementation of the PT algorithm. Hence a brief description of the PT algorithm and our implementation are provided in Algorithm 1.

5.0.1 Pan-Tomkins Algorithm

The PT algorithm passes an ECG data stream through a series of mathematical pre-processing modules: bandpass filter, differentiation, squaring, and moving window integration/one-dimensional convolution. Afterward, two sets of dual thresholding techniques are used to detect the R-peaks/heart-beat peaks. This means in each set of thresholds there are two thresholds. The first set of thresholds is applied to the data stream from the bandpass filtering module and the second set is applied to the data stream from the moving window integration module. If any heartbeat peak/R peak is not detected within the 166 percent of the average R-R peak interval applying the first threshold, the algorithm searches back through the buffer applying the second threshold. The second threshold is half of the first threshold. That way, false negatives might be eliminated and memory saved requiring a short buffer.

5.0.2 Our Implementation Of The PT Algorithm

Listing 1 of Appendix A is an excerpt from our code from the implemented system and figure 7 is the corresponding implementation of the counterpart of the PT algorithm by the authors on the Zilog Z80 microprocessor. Comparing both we observe, the Pan-Tomkins algorithm uses two thresholds, whereas we implemented one. That also makes performing the search-back algorithm and calculating the R-R interval average redundant.

Our reasoning for such modification is simply the reduction of inference time. Since we lose time inferring the beat type after detection (refer to Algorithm 1), we could miss newer beats or parts of them while performing the search-back algo-

Algorithm 1 Pseudocode of the implemented system

```
data  $\leftarrow$   $\emptyset$ 
sampling  $\leftarrow$  250 Hz
wait  $\leftarrow$  0.6 seconds
period  $\leftarrow$  wait  $\cdot$  sampling (150 samples)
while sensor_value is available at sensor do
  data.append(sensor_value)
  if data.length = period then
    start_time  $\leftarrow$  current_time_in_seconds()
    beat  $\leftarrow$  data
    beat  $\leftarrow$  bandpass_filter(beat  $\leftarrow$  beat, high  $\leftarrow$  15Hz, low  $\leftarrow$  5Hz)
    beat  $\leftarrow$  derivate(beat  $\leftarrow$  beat)
    beat  $\leftarrow$  square(beat  $\leftarrow$  beat)
    beat  $\leftarrow$  moving_window_integration(beat  $\leftarrow$  beat, window  $\leftarrow$  15)
    beat  $\leftarrow$  apply_threshold(beat)
    if beat =  $\emptyset$  then
      end_time  $\leftarrow$  current_time_in_seconds()
      data.remove(start  $\leftarrow$  1, end  $\leftarrow$  (end_time - start_time)  $\cdot$  wait)
      continue (Noise detected, hence we continue)
    end if
    index  $\leftarrow$  get_peak_index(beat  $\leftarrow$  beat)
    beat  $\leftarrow$  beat.get_values(start  $\leftarrow$  index - 30, end  $\leftarrow$  index + 30)
    type  $\leftarrow$  infer_beat_type(beat  $\leftarrow$  beat)
    if type  $\neq$  N then
      alarm(state  $\leftarrow$  ON)
    end if
    end_time  $\leftarrow$  current_time_in_seconds()
    data.remove(start  $\leftarrow$  1, end  $\leftarrow$  (end_time - start_time)  $\cdot$  wait)
  end if
end while
```

$$\text{SPKI} = 0.125 \text{ PEAKI} + 0.875 \text{ SPKI}$$

(if PEAKI is the signal peak) (12)

$$\text{NPKI} = 0.125 \text{ PEAKI} + 0.875 \text{ NPKI}$$

(if PEAKI is the noise peak) (13)

$$\text{THRESHOLD I1} = \text{NPKI} + 0.25 (\text{SPKI} - \text{NPKI})$$
(14)

$$\text{THRESHOLD I2} = 0.5 \text{ THRESHOLD I1}$$
(15)

Figure 7: Code excerpt from the PT algorithm[21].

rithm. Since more recent beats depict a clearer picture of a patient’s condition, we emphasized analyzing newer beats over older ones.

Furthermore, the PT algorithm used two sets of dual thresholds. The first was set used on the data stream from the bandpass filter and the second one was used on the data stream from the moving window integration module. In our implementation, we only applied the second set of thresholds. This too was done to save time by not missing newer beats. The modification of the PT algorithm was adopted from Michal et al. [22].

Table v: Differentiating our implementation and the PT algorithm

Our Implemetation	PT algorithm
1 set of threshold used	2 sets of thresholds used
Search-back was not implemented	Search-back was implemented
Total thresholds used was 1	Total thresholds used was 4

Chapter 6

Arrhythmia Detection Module

The Arrhythmia detection module reads a detected heart beat from the HDM module and classifies heart beats into Normal, Supraventricular, Ventricular, and Fusion beats (please refer to Table III). For classification, we used Deep Neural Network (DNN). Below is a description of DNN:

6.1 Deep Neural Network (DNN)

A deep neural network is a multi-layered artificial neural network (ANN) where each layer could consist of 1 or more neurons. It mimics the workings of the human brain. A neuron is also known as a perceptron.

6.1.1 Perceptron

A perceptron or an artificial neuron is a function that consists of a weight/kernel, bias/threshold, and an activation function. The output of a perceptron is usually between 0 and 1. The function is as follows:

$$F(x) = \sigma(w * x + b) \tag{i}$$

Here σ is the activation function, which is responsible for taking a number and turning it into a decision (0 or 1). The x here is our input data for learning, the w is weight/kernel and b is the bias or threshold. For now, we can update the values of w and b such that it can learn any linear sequence of data. For example, given a sequence of values of x of the equation $y = 3 * x + 6$, it can eventually come up with a much closer equation, provided w and b can be updated properly. To do that we need to know what amount to update the weight and bias to and when we shall stop updating (know when we are close enough).

For this, an error function is necessary that will let us know how far we are from the linear equation. One such error function is the mean squared error algorithm:

$$E = MSE = \sum_{i=1}^n \frac{(y_i - (w * x_i + b))^2}{n} \quad (\text{ii})$$

Here, x_i is one of the given range of values and y_i is one of the corresponding values derived from the equation given to predict for the given x_i . And n is the total number of samples provided to predict. The equation takes the square of the difference of y_i and its corresponding predicted value. This way it gets a squared error for each (y_i, x_i) pair. To get the mean squared error, all errors are summed and divided by n .

However, the perceptron still needs a system to update its weight and bias. That is where the optimizer function comes in. One such optimizer function is the gradient descent function. In this function, we take the partial derivative of the error function with respect to each weight and bias. Then we subtract the partial derivative times the learning rate from respective the weight or bias. Here, subtracting allows error minimization as MSE is a quadratic equation: the lower we move in the function, the less error we have.

$$w = w - \alpha * \frac{\partial E}{\partial w} \quad (\text{iii})$$

$$b = b - \alpha * \frac{\partial E}{\partial b} \quad (\text{iv})$$

Learning rate or α is simply a rate chosen to update the parameter at each step. A large rate would allow the model to converge faster but it might start oscillating and never reach the minimum it could if a smaller rate was chosen. Smaller rates, however, require more time but can reach go lower compared to their larger counterparts.

The loss/error/cost and optimizer function discussed are an example only. One may use another loss-optimizer pair for their use case.

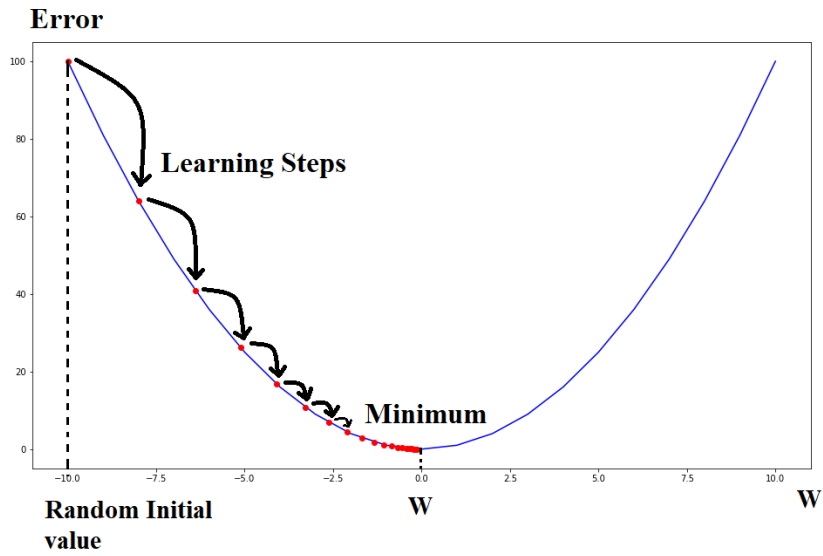


Figure 8: Gradient descent

However, the perceptron still alone cannot learn non-linear data. But multiple layered perceptrons can do so.

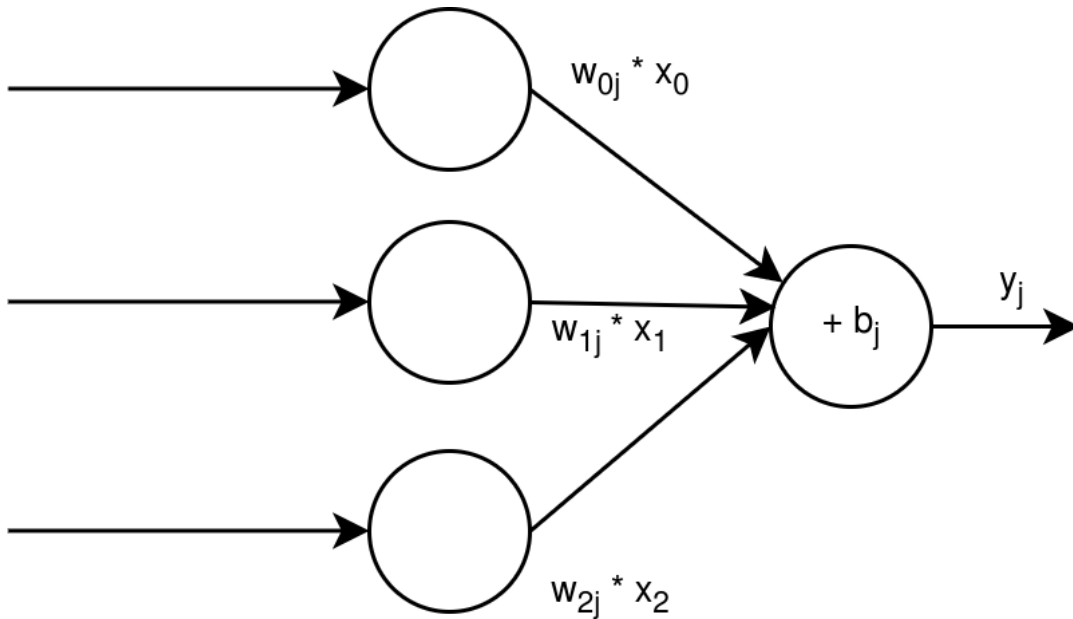


Figure 9: Single layered perceptron

6.1.2 Multi Layered Perceptron

It is not possible for a single-layered perceptron to learn non-linear data. However, as layers increase, they can eventually catch on to non-linear data.

Each layer in a single-layered perceptron can be imagined as two layers, where

one layer passes the data to the next, which processes the data and provides output [23].

Hence, each layer in a multilayered perceptron can be imagined in the same way. Let i be the neuron of the previous layer and j be the neuron of the next layer. The output y_j of j th neuron can be depicted as -

$$y_j = \sigma\left(\sum_{i=1}^n w_{ij} * x_i + b_j\right) \quad (\text{v})$$

Here, w_{ij} represents the strength of the synaptic connection between the i and j th neuron. b_j is the threshold of the j th layer.

We observe that output from every neuron of the previous layer is multiplied by w_{ij} , summed, and the threshold b_j is added. The result goes through the activation function, hence converted to a binary value of 0 or 1. This goes on for as many neurons as that layer has.

In this way, a complete neural network can be built with a varying number of layers and neurons per layer. The learning process of applying error and optimizer functions should be done as well for the entire network. For this, we need to calculate the total error E_t and update the weights/biases for each and every neuron -

$$E_t = MSE = \sum_{l=1}^L \sum_{j=1}^k \sum_{i=1}^n \frac{(y_i - (w_{ij} * x_i + b_{ij}))^2}{n} \quad (\text{vi})$$

$$w_{ij} = w_{ij} - \alpha * \frac{\partial E_t}{\partial w_{ij}} \quad (\text{vii})$$

$$b_{ij} = b_{ij} - \alpha * \frac{\partial E_t}{\partial b_{ij}} \quad (\text{viii})$$

Here, k can assume any pattern of natural numbers. Which allows k neurons in any l -th layer. For example, for 3 layered perceptron the values of k could be -

$$k = 1, 4, 2$$

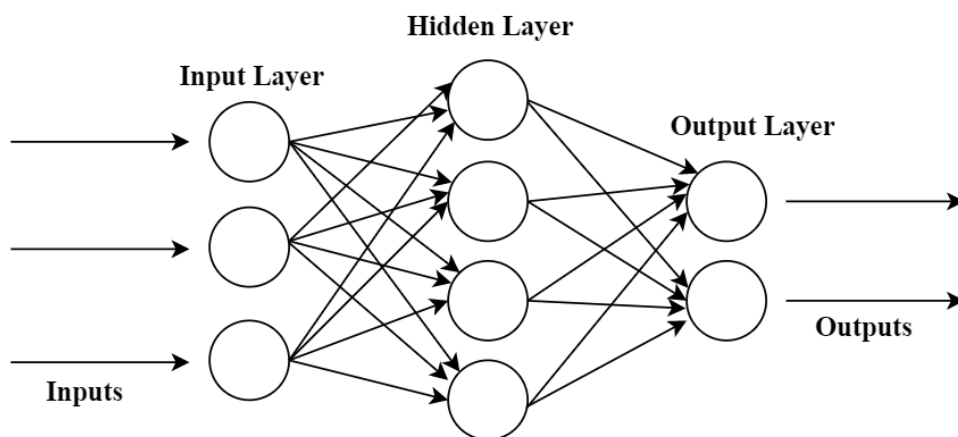


Figure 10: Multi-layered perceptron

6.2 Implemented DNN

In order to explain the implemented DNN, the training dataset needs to be discussed.

6.2.1 Dataset

The research on arrhythmia analysis and related topics has been supported since 1975 by labs at MIT and Boston’s Beth Israel Hospital (now the Beth Israel Deaconess Medical Center). The MIT-BIH Arrhythmia Database, which was finished and started disseminating in 1980, was one of the first significant outcomes of that work. The database, which was used in about 500 locations around the world, was the first generally accessible collection of standard test materials for arrhythmia detector validation. In addition to the assessment of arrhythmia detection, it was also used for fundamental study into cardiac dynamics. Initially, the dataset was delivered on a quarter-inch IRIG-format FM analog tape and 9-track half-inch digital tape at 800 and 1600 bpi. A CD-ROM version of the database was in August 1989.

48 half-hour snippets of two-channel ambulatory ECG recordings from 47 people who participated in BIH Arrhythmia Laboratory research between 1975 and 1979 are available in the MIT-BIH Arrhythmia Database. 23 recordings were chosen from the set of 4000 24-hour ambulatory ECG recordings made from a mixed population of inpatients (about 60%) and outpatients (about 40%) at Boston’s Beth Israel Hospital. The remaining 25 recordings were selected from the same set in order to include less common however clinically significant arrhythmia.

The recordings were digitalized over a 10 mV range at 360 samples per second per channel with 11-bit resolution. Each record was separately annotated by two or more cardiologists; differences were settled to produce the computer-readable reference annotations for each beat, which are supplied with the database and total

over 110,000 annotations.

Since the launch of PhysioNet in September 1999, around half of this database—25 of 48 complete records, and reference annotation files for all 48 entries—has been freely accessible <https://physionet.org/content/mitdb/1.0.0/>. In February 2005, the final 23 signal files that had previously only been accessible via the MIT-BIH Arrhythmia Database CD-ROM was posted.

Table vi: Classification of Arrhythmia according to AAMI [24]

N	Normal and bundle branch block beat
S	Supraventricular ectopic beats
V	Ventricular ectopic beats
F	Fusion of N and V
Q	undefined or paced beats

The Association for the Advancement of Medical Instrumentation (AAMI) standard for testing and reporting performance results of cardiac rhythm and segment measurement algorithms states that the arrhythmia classification performance should be based on five major categories of heartbeats (Table VI) to evaluate an arrhythmia detection algorithm.

6.2.2 Data Preprocessing

We passed the heart-beat data through the same preprocessing as the HDM (heart-beat detection module). At first, we passed the data through the bandpass filter to filter out noise. This is important since ECG data could be attenuated by 50-60 Hz alternating current (AC) current running through housing infrastructure along with interference from other sources. Taking the derivative and consequently performing moving window integration allows a beat’s feature to be concentrated in a smaller temporal space. Preprocessing and training algorithm is provided in Algorithm 2. The example for each type of beat is provided from figure 11 to figure 14.

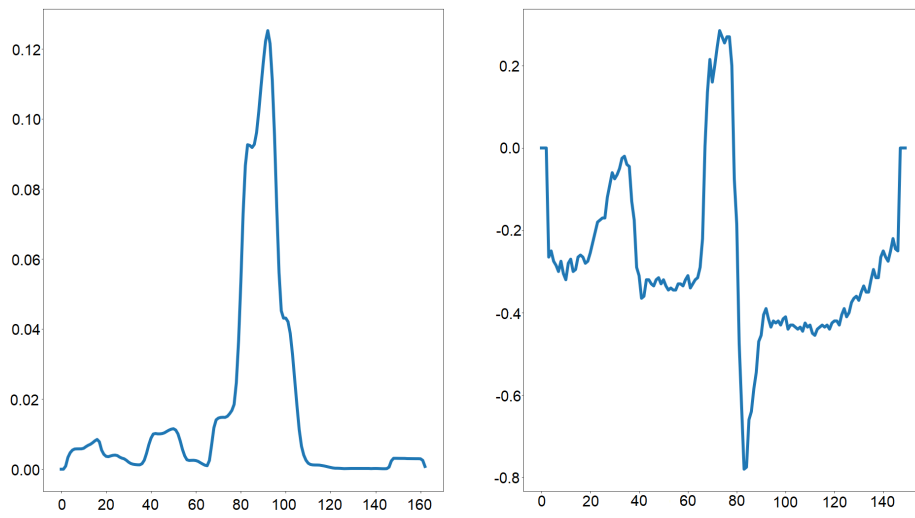


Figure 11: Normal beat preprocessed and unchanged

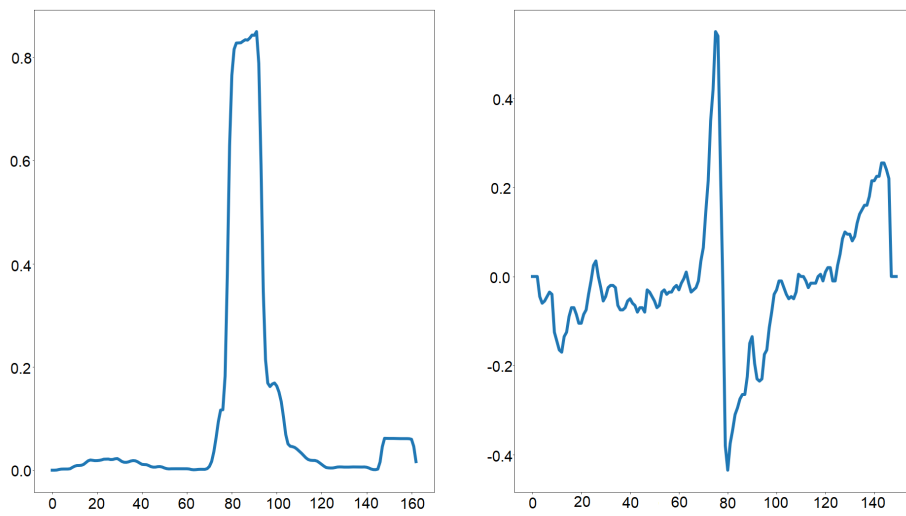


Figure 12: Supraventricular ectopic beat preprocessed and unchanged

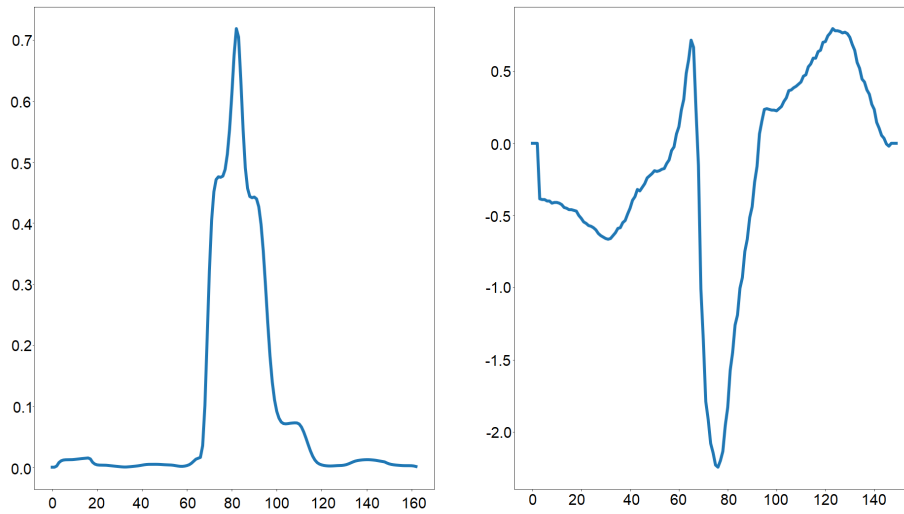


Figure 13: Ventricular ectopic beat preprocessed and original

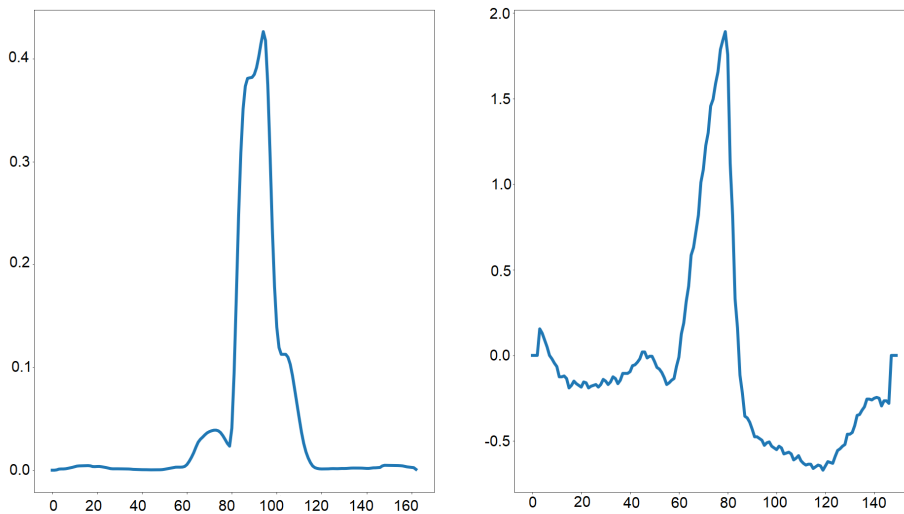


Figure 14: Fusion beat preprocessed and original

6.2.3 Neural Network

The structures of our deep learning networks and training convergence graphs are depicted from figure 15 through 22. Please note that the input layer was not considered the first layer, since it had no activation function. By first layer we meant the first layer after the input layer.

Hence, we used two layers, with 10 neurons in the first and 4 in the last. However,

Algorithm 2 Algorithm for training DNN for classification of arrhythmia

Ensure: *data* from MIT - BIH dataset is initialized

```
preprocessed_beats  $\leftarrow$   $\emptyset$ 
while data.length > 0 do
  beat, type  $\leftarrow$  data.get()
  beat  $\leftarrow$  bandpass_filter(beat  $\leftarrow$  beat, high  $\leftarrow$  15Hz, low  $\leftarrow$  5Hz)
  beat  $\leftarrow$  derivate(beat  $\leftarrow$  beat)
  beat  $\leftarrow$  square(beat  $\leftarrow$  beat)
  beat  $\leftarrow$  moving_window_integration(beat  $\leftarrow$  beat, window  $\leftarrow$  15)
  beat  $\leftarrow$  beat.get_values(start  $\leftarrow$  index - 30, end  $\leftarrow$  index + 30)
  preprocessed_beats.append((beat, type))
end while
train_and_validate(preprocessed_beats)
```

we tried more than one approach to find the best network. The network structures are discussed below -

6.2.4 Sigmoid-Sigmoid

In this network we used sigmoid activation function in all 14 neurons in the first and the last layer. Training achieved the highest accuracy and convergence was also the quickest.

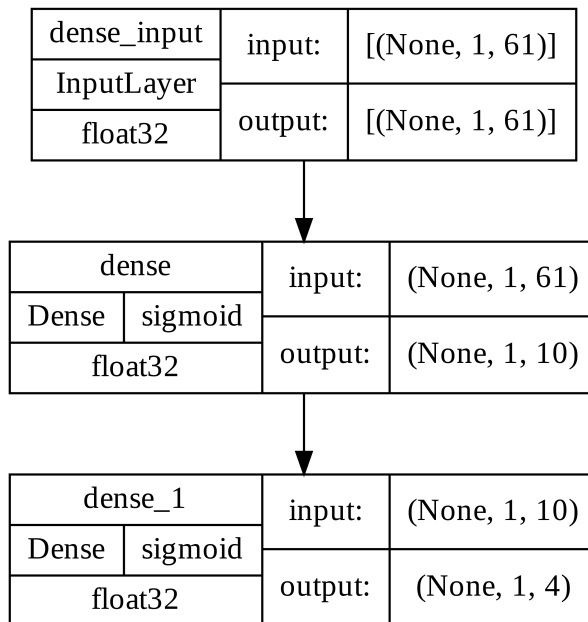


Figure 15: Sigmoid-sigmoid based network architecture

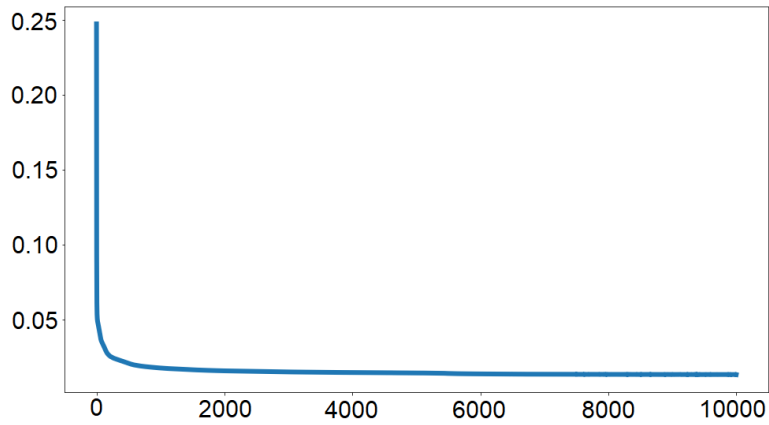


Figure 16: Sigmoid-sigmoid based network training convergence

6.2.5 ReLU-Sigmoid

Here, we used ReLU (all 10 neurons) in the first layer and all 4 sigmoid in the last layer.

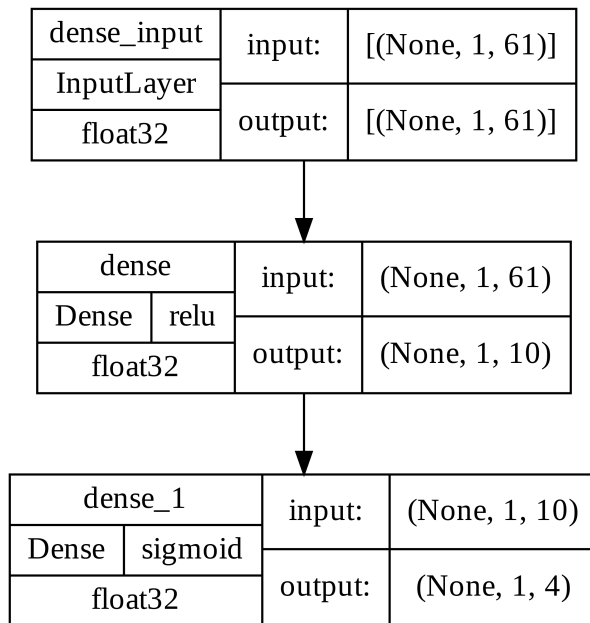


Figure 17: ReLU-sigmoid based network architecture

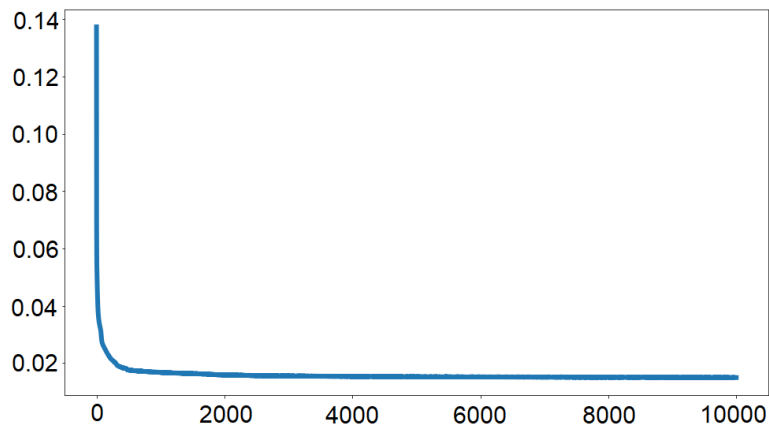


Figure 18: ReLU-sigmoid based network training convergence

6.2.6 ReLU-Softmax

We used softmax activation function for the last 4 neurons. The first 10 neurons however used ReLU.

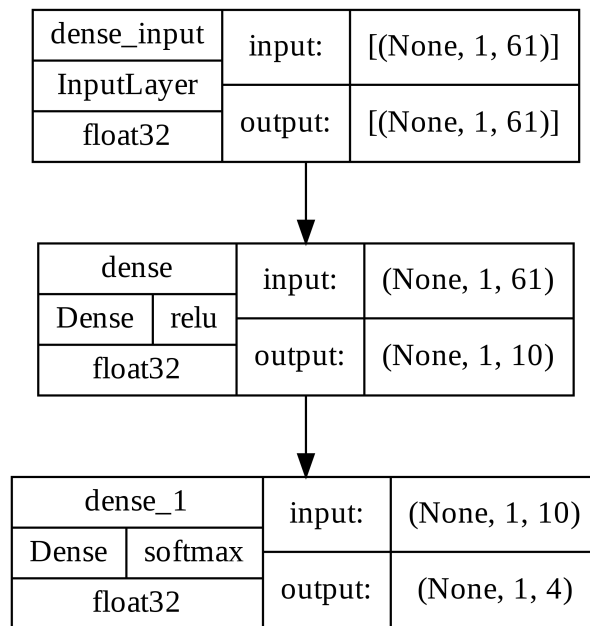


Figure 19: ReLU-softmax based network architecture

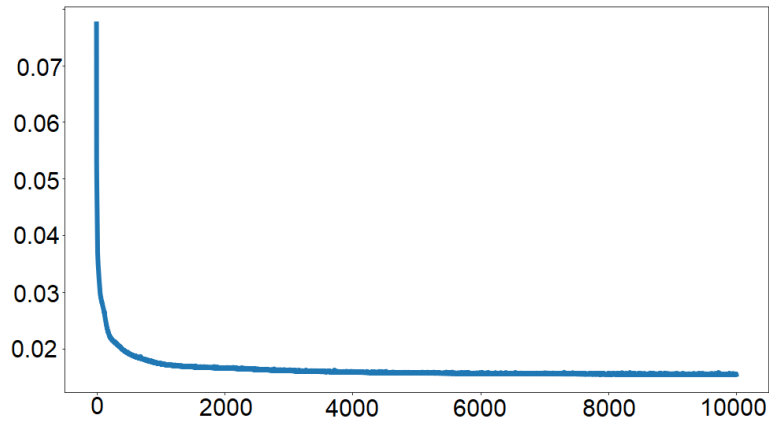


Figure 20: ReLU-softmax based network training convergence

6.2.7 Sigmoid-Softmax

We also tried sigmoid in the first layer and softmax activation function in the last.

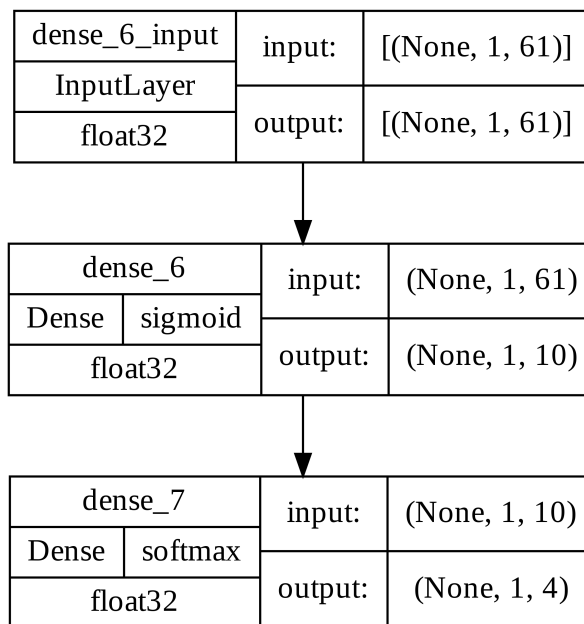


Figure 21: Sigmoid-softmax based network architecture

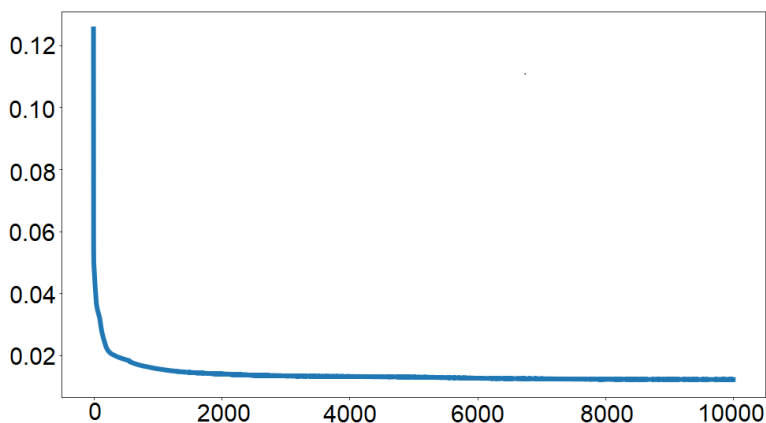


Figure 22: Sigmoid-softmax based network training convergence

Even though we are *classifying* arrhythmia in this system, we opted not to choose softmax activation for a reason aside from the inferior training performance. Which is that the softmax activation function involves two consecutive loop for which it might delay inference time (see listing 2 of Appendix A). Also, in an MCU, the clock is already slower, which makes saving time crucial. We chose the Sigmoid-sigmoid model for the ADM module for its best performance. We provide the variants of activation functions used from equation ix to equation xi.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (\text{ix})$$

$$\text{Relu}(z) = \max(0, z) \quad (\text{x})$$

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } i = 1, 2, \dots, K \quad (\text{xi})$$

6.2.8 Error Function

We chose the mean squared error for our error/loss function. This function has been described before in the deep neural network section under the perceptron and the multi-layered perceptron subsection through equations ii and vi.

6.2.9 Optimizer

For our optimizer algorithm, we chose the Adam optimizer developed by Kingma and Ba [25]. The name Adam is derived from adaptive moment estimation. This optimizer uses bias-corrected moment estimates of previous timesteps' gradients to update the weight of the current step, instead of directly using the current step's gradient. The complete algorithm is stated in Algorithm 3.

Algorithm 3 Algorithm for Adam

Require: α : Learning rate

Require: $E(\theta)$: Error function

Require: $\beta_1, \beta_2 \in [0, 1)$

Require: θ_0 : Initial weight

$m_0 \leftarrow 0$

$v_0 \leftarrow 0$

$t_0 \leftarrow 0$

while θ_t is not converge **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} E(\theta_{t-1})$ (Calculate gradient w.r.t. chosen weight θ)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Calculate biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Calculate biased second raw moment estimate)

$\hat{m}_t \leftarrow \frac{m_t}{(1 - \beta_1^t)}$ (Calculate unbiased moment estimate)

$\hat{v}_t \leftarrow \frac{v_t}{(1 - \beta_2^t)}$ (Calculate unbiased second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$ (Update weight, please note that g_t was not directly used as GD or SGD)

end while

return θ_t (Return updated weight)

6.2.10 Quantization And Model Compression

After training the size of the model was 2616 bytes. However, the MCU we decided to use only has 2000 bytes of SRAM. The model size exceeds the RAM. Moreover, even if the model fit within 2KB, we have the HDM module and the Arduino library to fit as well. For which, the model size should be less than 1000 bytes. To solve this problem, we introduced quantization.

Quantization or scaling a machine learning model simply means changing the scale of the weights and biases. Meaning if a model has weights 1, 2, 3 and bias 4, 5, and, 6. We can make another model with the exact same performance with weights 0.5, 1, 1.5 and bias 2, 2.5, and, 3. Notice that the weights/biases of the second model are half of the first. However, the performance will be the same since the proportion of each weight/bias with the other has been kept the same.

Another way to think of quantization is to expand or constrict any given range to another range. For example, the range from 1 to 6 is constricted to 0.5 to 3. Again, the same can be thought of in reverse (expanding 0.5 to 3 range into 1 to 6).

To quantize a range into another, we need to select the lower limit and upper limit of the quantized range, α_q , and β_q respectively, for any given upper and lower limit of the unquantized range, α and β respectively. Then, through equation xiv to xvii, we can get the quantized value given an unquantized number and vice versa.

Even though for quantization, the need to calculate the scale (equation xiv) seems self-explanatory, calculating the zero point (equation xv) might not seem so. But, we still need to calculate the zero point which is the middle point and offset of all number lines.

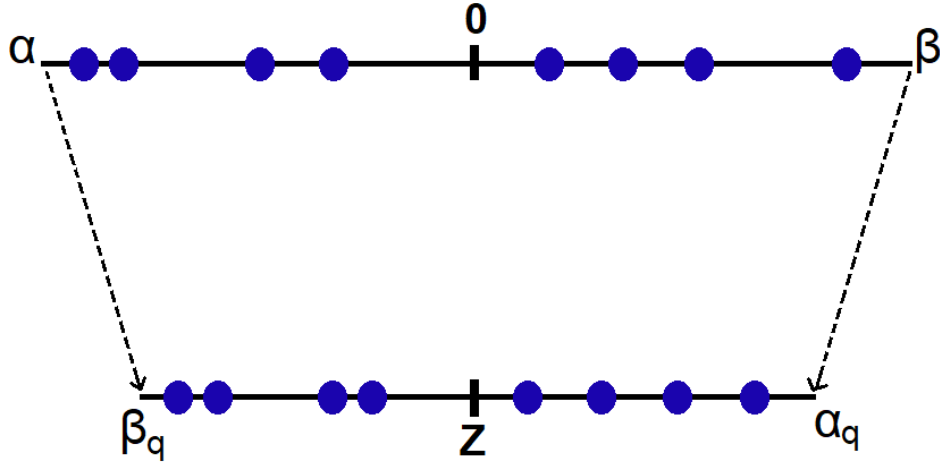


Figure 23: Quantizing for compression

This is necessary since not all quantized ranges will have zero points at zero. For instance, for $\alpha_q = -128, \beta_q = +127$ and $\beta = -\alpha = 1$, to keep the zero point in the middle it has to be a little to the right, or the positive side ($z = 0.5$). Since the range has more negative values than positive. That way, both the positive and negative side has the same length in a number line, keeping the quantized range balanced with the unquantized.

In order to calculate scale s and zero point z , we substitute x and x_q of xvii into equations xviii and xix and solve them as a system of two equations. Equations xvii and xviii are a given since the lower unquantized limit will always map to the lower quantized limit. The same goes for the mapping of the upper limits.

However, quantization is redundant for model compression if rounding is not applied (see equation xvi). As we know to train models we need to use floating point quantities that consume at least 4 bytes (IEEE 754 32-bit standard in most CPU architectures). Because iteratively updating weights and bias does not guarantee a change in whole numbers. But integers can be represented in only 1 byte with the cost of representing 256 values including 0. Hence, after training, if we can quantize weights into any single byte range of 256 values, we can save 4 times memory.

$$\alpha_q = -127 \tag{xii}$$

$$\beta_q = +127 \tag{xiii}$$

$$s = \frac{\beta - \alpha}{\beta_q - \alpha_q} \tag{xiv}$$

$$z = \frac{\alpha\beta_q - \beta\alpha_q}{\beta - \alpha} \quad (\text{xv})$$

$$x_q = \text{round}\left(\frac{1}{s}x - z\right) \quad (\text{xvi})$$

$$x = s(x_q + z) \quad (\text{xvii})$$

$$\alpha = s(\alpha_q + z) \quad (\text{xviii})$$

$$\beta = s(\beta_q + z) \quad (\text{xix})$$

But to keep symmetry, if the range to be quantized contains negative values, we should ensure that zero is quantized to zero. Not only that, we noticed significant accuracy improvement across all model types tested when that is done. This is what we did in our implementation of quantization.

Our α and β range is dynamically set as the maximum quantity achieved, $\beta = -\alpha = \text{max}$. However, we chose $\alpha_q = -127$ and $\beta_q = +127$. Such choices allowed zero to be mapped to zero. We still tested the case of zero not being mapped to zero, where $\alpha = \text{least}$, $\beta = \text{max}$ and α_q, β_q were kept the same.

Table vii: Accuracy drop in (appx. in %) when zero is mapped to zero vs when not while quantizing

Quantized model	$zero_{quant} \neq 0, A$	$zero_{quant} = 0, B$	Drop, $B - A$
Sigmoid-Sigmoid	80.023	93.74	13.717
ReLU-Sigmoid	89.33	95.40	6.07
ReLU-Softmax	89.33	94.27	4.94
Sigmoid-Softmax	7.13	95.23	88.1

6.2.11 Temporary Dequantization And Model Selection

Table viii: Accuracy (appx.in %) comparison between model types in default, qunatized and, dequantized state

Model	Default	Qunatized	Dequantized
Sigmoid-Sigmoid	97.09	93.74	96.84
ReLU-Sigmoid	96.68	95.40	96.53
ReLU-Softmax	96.63	94.27	93.33
Sigmoid-Softmax	97.21	95.23	95.51

In table VIII, *default* refers to the state after the model is trained. Quantized refers to quantizing the default model’s weight/bias to 8-bit integer from -127 to +127 in this case. Dequantized state means, scaling the weights/biases back to default. However, the weights/biases may not be equal to the default since quantizing involves rounding (see equation xvi). However, they will be approximate. Table IX includes some weights and biases in their default, quantized, and dequantized state from each layer for $\beta = -\alpha = 64.74442$ and $\beta_q = -\alpha_q = 127$ along with their corresponding accuracy. We can see that in the dequantized state, they do not recover their exact precision but are close.

Table ix: Default, quantized, dequantized state of weight-bias for each layer and corresponding accuracy (appx in %) for the sigmoid-sigmoid model

Weight/bias type	Layer 1 weight	Layer 1 bias	Layer 2 weight	Layer 2 bias	Accuracy
Default	-56.74	-1.58	14.58	-17.0	97.09
Quantized	-111	-3	29	-33	93.74
Dequantized	-56.59	-1.53	14.78	-16.82	96.84

Even though it is not possible to pack one of the default models to the MCU, since their size exceeds 2KB, we can pack the quantized and the dequantized weights/biases. Using quantized weights/biases will be the easiest since they will take the

least space and work seamlessly. However, packing dequantized state is also possible through *temporary dequantization*. That is why we decided to use the dequantized sigmoid-sigmoid model with the temporary dequantization technique since it has the highest accuracy among the dequantized and quantized models.

In this technique, weights can stay as 8-bit integers in memory (as quantized integers). However, when that specific weight/bias is used for calculation, it can be copied, dequantized, used, and released from memory.

Hence, we packed the sigmoid-sigmoid quantized weights/biases along with the calculated scale and zero point. Then when the time came for inference using any of the above weights, we dequantized only that specific weight/bias and caused inference.

That way at inference time, we are only using 4 bytes of additional memory. Which is tolerable, provided the increase in performance is achieved. Listing 3 of Appendix A shows the use of temporary dequantization in MCU code (cpp).

6.2.12 Performance

In this section, we showed the performance of the dequantized sigmoid-sigmoid model since it achieved the highest accuracy with the ability to be compressed to fit in MCU’s memory. The performance details are provided in table X.

Table x: Performance of dequantized sigmoid-sigmoid model (selected)

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.975297	0.783034	0.928136	0.937500	0.968398	0.905992	0.966362
Recall	0.992711	0.522876	0.892379	0.452830	0.968398	0.715199	0.968398
F1-score	0.983927	0.627041	0.909906	0.610687	0.968398	0.782890	0.965907
Samples	29908	918	2388	265	0.968398	33479	33479

To evaluate our metrics we used precision, recall, and F1 score along with accuracy since accuracy alone can be misleading. Even though an imbalanced dataset can cause a very high accuracy, the classifier may be wrongly detecting any sample as the highest available sample in the dataset. For instance, if a classifier that detects cats and dogs is trained on a dataset of dogs only and is tested on a dataset with 99 dogs and 1 cat, it may detect all the members as dogs. That way the classifier still has $99/100 = 99\%$ accuracy, which is a very desirable score. However, the model

has 0 classification ability. Therefore, we need other metrics besides accuracy that account for false positives (like the cat on the testing dataset) and false negatives.

The formula provided for the used metrics is provided from equations xx through xxiv. To understand them, some concepts are needed to be explained:

- *True positive*: Number of samples detected as positive and are also positive.
- *False positive*: Number of samples detected as positive but are actually negative.
- *True negative*: Number of samples detected as negative and are also negative.
- *False negative*: Number of samples detected as negative but are actually positive.

Table xi: True vs False positives and negatives

	Ground Truth	Predicted
TP	Positive	Positive
FP	Negative	Positive
TN	Negative	Negative
FN	Positive	Negative

Here, precision is a ratio of detected true positives for a specific class in a classification problem with a total number of testing samples detected as positive (both true and false). The recall is the ratio of true positives but with the total number of samples for that class. A fall in precision means an increase in false positives. On the other hand, a fall in recall means a rise in false negatives. The rise consequently means the opposite for both. For which high recall and precision are always desirable. However, in some cases, only one of them being higher than the other also works as long as both are high enough.

The F1 score is the harmonic mean of precision and recall. A harmonic mean for two numbers is only high when all of its members are high. Therefore using the F1 score, we can know whether both the recall and precision are high or not. That way we can use a single metric of performance instead of two.

The macro average is the average of any of the classification scores (precision, recall, or f1). However, in the weighted average, each score is weighted by the no. of times it appears in the dataset. That way the mean score accounts for the success/failure it had for each class.

$$Precision = \frac{TP}{TP + FP} \quad (xx)$$

$$Recall = \frac{TP}{TP + FN} \quad (xxi)$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (xxii)$$

$$Macro\ average = \frac{\sum_{i=1}^{Number\ of\ classes} Score_i}{Number\ of\ classes} \quad (xxiii)$$

$$Weighted\ average = \frac{\sum_{i=1}^{Number\ of\ classes} Number\ of\ Samples_i * Score_i}{Total\ no.\ samples} \quad (xxiv)$$

Table xii: Train-test split of the used dataset

Samples	N	S	V	F
Training	60723	1863	4848	538
Testing	29908	918	2388	265
Total	90631	2781	7236	803

We can observe that the model has nearly perfect precision and an even recall for type N arrhythmia. Which provides a great F1 score. The second-best performance was achieved by type V arrhythmia. However, the scores for types S and F are not satisfactory. Although type F has a higher precision, the recall is less than 50%. On the other hand, the recall score for S type is slightly higher than type F. However both have close F1 scores. Which means the overall performance type S and F is the same. The reason for this is quite clear since the training samples of S and F are close to 2000 and 500 respectively. However, type N has nearly 60,000 samples and V has almost 5000 samples which is significantly higher than S and F.

Moreover, the weighted average for each score is very high, implying the success of classifying the N and V type arrhythmia correctly. However, in the case of the Macro average which implies as said before, the model would have performed better if a more balanced dataset was provided.

Furthermore, analyzing FLOPs (Floating Point Operations) we observe our operation count is 1314 operations per inference. Also, the model weights and biases consume 664 bytes of memory. But we at least need to keep 150 samples of ECG data in memory for the HDM module which is eventually preprocessed to 61 samples that the ADM module intakes for inference. Each of the 150 samples occupies 4 bytes, whereas the model weight/bias occupies 1 byte each. Also for temporary de-quantization we require an additional 3 bytes at any moment in time. That amounts to 1267 bytes or 1.267 KB of memory.

However, this calculation is of the *unreleasable* memory and an approximation. In practice, we had to allocate and deallocate some more memory. The calculations are provided in tables XIII and XIV.

Table xiii: FLOPs & memory breakdown per layer

Layer	FLOPs	Memory
1	1230	620
2	84	44
Total	1314	664

Table xiv: System SRAM consumption breakdown in MCU

	Memory (bytes)
Model size	$664 + 3 = 667$
ECG buffer	$150 \cdot 4 = 600$
Total	1267

Previously, we discussed about the best performance of Faraone et al [9]. and Scire et al. [10]. to achieve better performance than all other compared studies. Therefore, table XV contains the performance of their arrhythmia classification model vs ours. There, we notice that the FLOPs and memory calculation of Scire et. al is absent. The reason being we could not find explicit data or the number of weights or biases used by them per layer in the study. However, compared with the available data we notice that our model consumes more than 5 times less memory and 2500 times fewer operations than Faraone, where our performance exceeds theirs by 0.2% in terms of F1 score and 11% in terms of accuracy. However, we underperformed compared to Scire et al. by 6.3% in terms of F1 score where our accuracy was equal.

Therefore, this study shows that deep learning models using fewer weights/biases and primitive activations like sigmoid can outperform or come close to modern networks with a high parameter count using Dropouts and Convolution layers in some cases.

Table xv: Performance comparison with the best arrhythmia classification models studied

Model	Faraone et al.[9]	Scirè et al. [10]	Ours
FLOPs (MOps)	3.221	-	0.001314
Memory (KB)	6.8	-	1.267
Precision	0.795	0.805	0.905
Recall	0.776	0.891	0.715
F1-score	0.780	0.845	0.782
Accuracy	0.854	0.968	0.968

For sake of future research and reference, we are also including tables for the default, quantized, and dequantized models that we did not decide to use in table XVI through XXVI. We noticed the relu-sigmoid dequantized and sigmoid-sigmoid quantized model to provide the second and third-best performances whose F1 scores were 0.753 and 0.720 respectively whereas the selected sigmoid-sigmoid dequantized model has 0.782. Using different random seeds, one might get a higher performance for them since those models have very close scores. This might make temporary dequantization redundant (if the sigmoid-sigmoid quantized model gets a better f1 score) and ensure faster inference.

Table xvi: Relu-sigmoid dequantized model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.970984	0.817269	0.932057	0.780000	0.965292	0.875077	0.962481
Recall	0.994684	0.443355	0.855946	0.441509	0.965292	0.683874	0.965292
F1-score	0.982691	0.574859	0.892382	0.563855	0.965292	0.753447	0.961751
Support	29908	918	2388	265	0.965292	33479	33479

Table xvii: Sigmoid-sigmoid quantized model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.980569	0.369759	0.819113	0.558065	0.937394	0.681876	0.948960
Recall	0.953324	0.586057	0.904523	0.652830	0.937394	0.774183	0.937394
F1-score	0.966754	0.453434	0.859701	0.601739	0.937394	0.720407	0.942154
Support	29908	918	2388	265	0.937394	33479	33479

Table xviii: Sigmoid-softmax default model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.977666	0.809278	0.947323	0.878788	0.972072	0.903264	0.970102
Recall	0.993814	0.513072	0.911223	0.656604	0.972072	0.768678	0.972072
F1-score	0.985674	0.628000	0.928922	0.751620	0.972072	0.823554	0.969966
Support	29908	918	2388	265	0.972072	33479	33479

Table xix: Sigmoid-softmax dequantized model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.955181	0.937500	0.959082	0.894737	0.955106	0.936625	0.954496
Recall	0.998328	0.065359	0.804858	0.513208	0.955106	0.595438	0.955106
F1-score	0.976278	0.122200	0.875228	0.652278	0.955106	0.656496	0.943087
Support	29908	918	2388	265	0.955106	33479	33479

Table xx: Sigmoid-softmax quantized model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
precision	0.951744	0.800000	0.970757	0.857143	0.952328	0.894911	0.948190
Recall	0.999064	0.013072	0.778476	0.498113	0.952328	0.572181	0.952328
F1-score	0.974830	0.025723	0.864048	0.630072	0.952328	0.623668	0.938175
Support	29908	918	2388	265	0.952328	33479	33479

Table xxi: Sigmoid-sigmoid default model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.976478	0.805128	0.942054	0.931818	0.970907	0.913870	0.968971
Recall	0.993848	0.513072	0.898660	0.618868	0.970907	0.756112	0.970907
F1-score	0.985086	0.626747	0.919846	0.743764	0.970907	0.818861	0.968697
Support	29908	918	2388	265	0.970907	33479	33479

Table xxii: Relu-sigmoid default model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.972532	0.812741	0.928920	0.875969	0.966755	0.897541	0.964275
Recall	0.994416	0.458606	0.875628	0.426415	0.966755	0.688766	0.966755
F1-score	0.983352	0.586351	0.901487	0.573604	0.966755	0.761199	0.963384
Support	29908	918	2388	265	0.966755	33479	33479

Table xxiii: Relu-sigmoid quantized model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.955721	0.971429	0.947654	0.617188	0.953971	0.872998	0.952896
Recall	0.998796	0.074074	0.803601	0.298113	0.953971	0.543646	0.953971
F1-score	0.976784	0.137652	0.869703	0.402036	0.953971	0.596544	0.941587
Support	29908	918	2388	265	0.953971	33479	33479

Table xxiv: Relu-softmax default model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.973631	0.827826	0.906830	0.0	0.966277	0.677072	0.957161
Recall	0.993814	0.518519	0.900754	0.0	0.966277	0.603272	0.966277
F1-score	0.983619	0.637642	0.903782	0.0	0.966277	0.631261	0.960652
Support	29908	918	2388	265.0	0.966277	33479	33479

Table xxv: Relu-softmax dequantized model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.961679	1	0.685644	0.0	0.933301	0.661831	0.935428
Recall	0.969975	0.021786	0.927973	0.0	0.933301	0.479934	0.933301
F1-score	0.965809	0.042644	0.788612	0.0	0.933301	0.449266	0.920212
Support	29908	918	2388	265.0	0.933301	33479	33479

Table xxvi: Relu-softmax quantized model

	N	S	V	F	Accuracy	Macro avg	Weighted avg
Precision	0.960054	0.0	0.800739	0.0	0.942651	0.440198	0.914766
Recall	0.982781	0.0	0.907035	0.0	0.942651	0.472454	0.942651
F1-score	0.971284	0.0	0.850579	0.0	0.942651	0.455466	0.928354
Support	29908	918.0	2388	265.0	0.942651	33479	33479

6.3 Porting

In order to port to Arduino Nano and test them, we had to code deeply connected layers and activations into a cpp header file. Listing 4 of Appendix A includes a significant excerpt of that file along with the PT algorithm functions. However dense and matmul functions have already been shown in listing 3 of the same appendix, for which they were excluded.

Chapter 7

Commercial Low-cost ECG monitoring Systems

In this section, we discuss some commercially available low-cost ECG monitoring systems and their advertised features. We also compare their cost with the studies we analyzed and our designed system.

Alivecor KardiaMobile is a single-lead ECG device that connects wirelessly with a smartphone and transfers the ECG signals from the fingertips of the patient. It can record 30 seconds to 5 minutes of ECG data.

Omron Wireless Blood Pressure Monitor+ EKG is a hand-held blood pressure monitor with a built-in ECG device. It is portable and has an LCD display screen to check ECG data.

Table xxvii: Cost comparison of low-cost commercially available ECG monitoring systems

Name	Cost (USD)
Alivecor KardiaMobile	79.00
Omron Wireless Blood Pressure Monitor+ EKG	199.99
Instant Check	459.00
Heart Check CardiBeat	129.00
AFibAlert	179.00
ReadMyHeart	189.00
Studies reviewed	35.00 - 149.00
Implemented system	24.90

InstantCheck is a portable ECG device with an LCD screen to display ECG in real-time. It allows users to record their ECG signals conveniently.

HeartCheck CardiBeat is a small handheld ECG monitor that is capable of capturing a wide range of arrhythmia such as tachycardia, bradycardia, ventricular

premature beats (VPBs), pauses, and atrial fibrillation.

AFibAlert Heart Rhythm Monitor allows patients who have been diagnosed with AF or are at risk of developing AF or other arrhythmias, to take periodic readings and to transmit this data to their physician by email, fax, or secure online access.

ReadMyHeart Handheld ECG is a handheld standard heart monitoring device utilizing ‘Modified Lead 1 – ECG’ measurements that are normally obtained from the traditional single-lead ECG signal.

Table XXVII reveals that commercially available systems are much more expensive than the ones researchers implemented. The lowest cost for the systems mentioned here is USD 79 whereas for the studies we reviewed it was USD 35. The cost of our system is USD 24.90 which is even lesser. But this excludes the cost of analog circuitry, sensors, display elements, labor, and other miscellaneous costs, for both our system and the studies analyzed.

However, it is noteworthy that researchers themselves bought their components at retail prices. This implies if their systems go to production their cost of manufacturing would be much lower. Not to mention, a lot of the components on the development boards used are obsolete for this specific implementation, the removal of which could further lower the price.

Therefore, redesigning the schematics with the embedded computing unit could potentially lower the cost (by removing obsolete components and adding relevant ones) which in turn, could lower the manufacturing cost and make room for some profit margin.

Chapter 8

Conclusion & Future Work

The objective of this study was to identify the best approach to developing a low-cost, low-compute, low-powered ECG monitoring system and providing an implemented system. Analyzing 9 recent studies from 2018 to 2021 which attempted to develop a low-compute ECG monitoring system, we can conclude that the best approach might be using low-compute micro-controller units with no operating systems overhead combined with machine learning-based detection algorithms.

Consequently, we implemented a system using a widely used, low-cost, low-compute, and low-power microcontroller with the Pan-Tompkins algorithm for heart-beat detection and a deep neural network for arrhythmia detection.

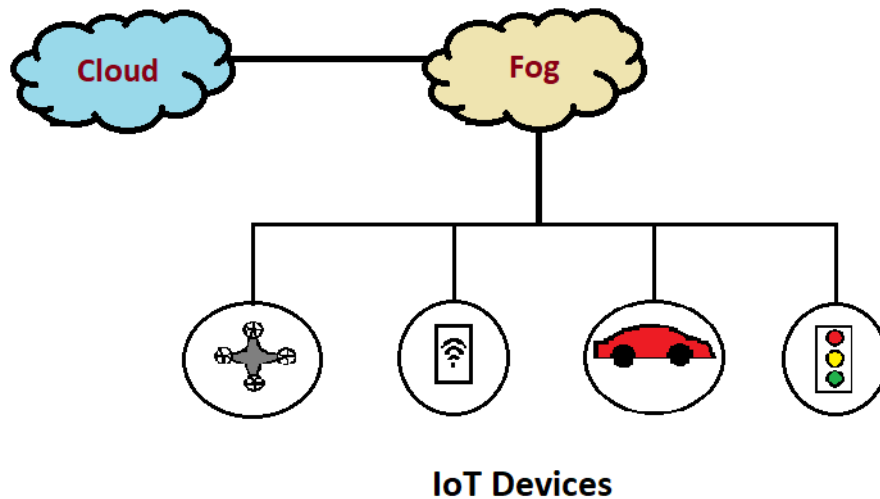


Figure 24: Fog computing

Future researchers could integrate IoT(Internet of Things) capability on such low-cost systems to increase usability, capability, robustness, and long-term hardware-software support. IoT capabilities can be incorporated in two ways :

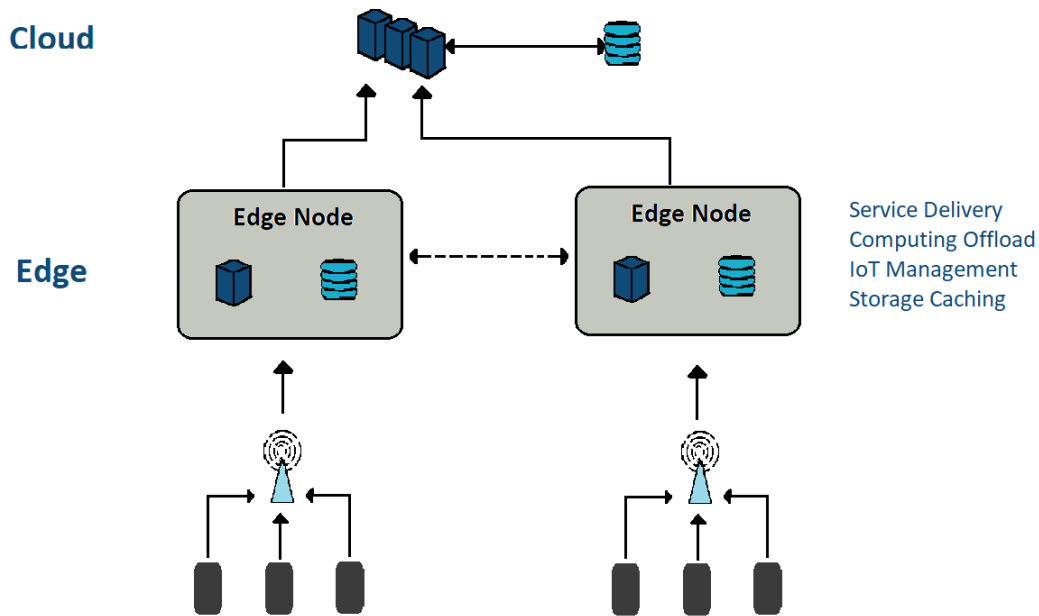


Figure 25: Edge computing

- *Edge computing based system:* Through this system, only actionable information should be sent to the cloud to alert hospitals and physicians on their phones in case of abnormalities like arrhythmia. That way medical professionals who might not be nearby could rush to help the patient. But data processing will occur at the *edge* or the ECG monitoring system.
- *Fog computing based system:* The edge computing system could be further developed to use fog computers with comparatively high-compute processors, to detect trends in patients. Which could allow the system to detect disease, heart failure, or stroke long before they occur. Also, the fog computers could be used to optimize the algorithm running on low-compute micro-controllers, allowing continuous and limitless improvement of the system as more and more data becomes available.

Developing such ECG monitoring systems could allow developing and underdeveloped countries to acquire and operate ample systems for their hospitals, which could lower deaths from CVD diseases. This could also help patients with CVD to receive better care at home at a much lower cost by combining telemedicine and low-cost systems.

Following the process discussed, many other biomedical devices, that rely on signal processing could be developed. Recently, the TinyML community is helping developers around the world to flash machine learning algorithms to edge devices like micro-controllers and other embedded processors and move machine learning away from the cloud. Therefore, working on such a research topic could be comparatively easier than before.

Overall, attempting to develop more power efficient, zero operating system overhead, machine-learning-based systems could make countless lives simpler and reduce the cost of development for under-developed and developed countries. Future research may delve more into systems like the ECG monitoring system, following the approach discussed in this study.

Appendix A

Code

Listing 1: Code excerpt from our modification of the PT algorithm

```
if (last_qrs_index == -1 ||
i - last_qrs_index >
refractory_period)
{
float current_peak_val = res[i];

// Comparing peaks with threshold values

if (current_peak_val > threshold)
{
last_qrs_index = i;

qrs_indices[*qcount] = i;
*qcount = *qcount + 1;

//Running QRS peak estimate

qrs_peak_est = qrs_filt * current_peak_val
+ (1 - qrs_filt) * qrs_peak_est;
}
else
{
noise_indices[*ncount] = i;
*ncount = *ncount + 1;

//Running Noise peak estimate

noise_peak_est = noise_filt * current_peak_val
+ (1 - noise_filt) * noise_peak_est;
}

//Note that only one threshold is being used
```

```

threshold = noise_peak_est + qrs_noise_diff_weight
* (qrs_peak_est - noise_peak_est);
}

```

Listing 2: Softmax implementation in cpp

```

void softmax(float *x, int rows, int cols)
{
    float sum = 0;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            x[i * cols + j] = expf(x[i * cols + j]);
            sum += x[i * cols + j];
        }
    }
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            x[i * cols + j] = x[i * cols + j] / sum;
        }
    }
}

```

Listing 3: Temporary dequantization implementation

```

void matmul(float *result, float *x,
uint8_t *y, int r1, int c1, int r2, int c2)
{
    if (c1 != r2)
        printf("c1 and r2 should be equal");

    for (int i = 0; i < r1; i++)
    {
        for (int j = 0; j < c2; j++)
        {
            for (int k = 0; k < c1; k++)
            {
                result[i * c1 + j] +=
                x[i * c1 + k] *
                dequantize(y[k * c2 + j]);
// Temporary dequantization is applied here
            }
        }
    }
}

```

```

void dense(float *output, float *input,
uint8_t *kernel, uint8_t *bias, int ir,
int ic, int kr, int kc)
{
    matmul(output, input, kernel, ir, ic, kr, kc);
    for (int i = 0; i < ir; i++)
    {
        for (int j = 0; j < kc; j++)
        {
            output[i * kc + j] +=
                dequantize(bias[i * kc + j]);
            // Temporary dequantization is applied here
        }
    }

    sigmoid(output, ir, kc);
}

```

Listing 4: Excerpt of the model ported as header

```

void sigmoid(float *x, int rows, int cols)
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            x[i * cols + j] = 1
                / (1 + expf(x[i * cols + j]));
        }
    }
}

int argmax(float *output, int len)
{
    int max_ind = 0;
    float max = output[max_ind];

    for (int i = 1; i < len; i++)
    {
        if (output[i] > max)
        {
            max = output[i];
            max_ind = i;
        }
    }
    return max_ind;
}

```



```

float dequantize(int8_t x_q)
{
    float scale = 0.5097986056110052;
    float zero = 0.0;
    return scale * (x_q + zero);
}

int pan_tom_detect_qrs()
{
    int len = 150;
    float y[len] {0.0};
    float x_2 = 0.0;
    float x_1 = 0.0;
    float y_1 = 0.0;
    float y_2 = 0.0;

    const float a0 = 1.0;
    const float a1 = -1.73356294;
    const float a2 = 0.77567951;
    const float b0 = 0.11216024;
    const float b1 = 0;
    const float b2 = -0.11216024;

    for (int i = 0; i < len;)
    {
        if ((digitalRead(10) == 1) || (digitalRead(11) == 1)) {
        }

        else {
            float x_i = (analogRead(A0) / 511.5) - 1;
            y[i] = (b0 * x_i
                + b1 * x_1
                + b2 * x_2
                - (a1 * y_1 + a2 * y_2)) / a0;
            x_2 = x_1;
            x_1 = x_i;
            y_2 = y_1;
            y_1 = y[i];
            ++i;
        }
    }

    for (int i = 0; i < 5; i++)
        y[i] = y[5];

    for (int i = 0; i + 1 < len; i++)

```

```

{
y[i] = y[i + 1] - y[i];
y[i] *= y[i];
}

len--;

int integration_window{15};
int n = len + integration_window - 1;
int max_len = len >
integration_window ?
len :
integration_window;
for (int i = 0; i < n; i++)
{

int kMax = i < max_len ? i : max_len;
for (int k = 0; k <= kMax; k++)
{
    if (k < len && (i - k) < integration_window)
    {
        res[i] += y[k];
    }
}
}

int spacing{50};
float limit{0.35};
float qrs_peak_est{0.0};
float noise_peak_est{0.0};
float noise_peak_val{0.0};
float qrs_filt{0.125};
float noise_filt{0.125};
float qrs_noise_diff_weight{0.25};
float last_qrs_index{-1};
int refractory_period{120};
float threshold{0.0};

for (int i = 0; i < n; i++)
{
int lookup = (i - 1) -
spacing < 0 ? i : spacing;
bool is_broken = false;
for (int j = 1; j <= lookup; j++)
{
    if (!(res[i] > res[i - j]))
    {
        is_broken = true;
    }
}
}

```

```

        break;
    }
}
if (is_broken)
    continue;

lookup = (i + 1) +
spacing > n - 1 ?
(n - 1) - i : spacing;
for (int j = 1; j <= lookup; j++)
{
    if (!(res[i] > res[i + j]))
    {
        is_broken = true;
        break;
    }
}
if (!is_broken && res[i] > limit)
{
    if (last_qrs_index == -1
        || i - last_qrs_index >
        refractory_period)
    {
        float current_peak_val = res[i];
        if (current_peak_val > threshold)
        {
            last_qrs_index = i;
            return i;
        }
    }
    else
    {
        noise_peak_est = noise_filt *
            current_peak_val + (1 - noise_filt)
            * noise_peak_est;
    }
    threshold = noise_peak_est +
        qrs_noise_diff_weight * (qrs_peak_est -
        noise_peak_est);
}
}
}
return -1;
}

```

Bibliography

- [1] W. H. Organization *et al.*, *World health statistics 2020*, <https://apps.who.int/iris/rest/bitstreams/1277753/retrieve>, [Online; accessed September 2, 2022], 2020.
- [2] N. H. Bunce, ; Robin, and H. Patel, *Kumar and Clark's Clinical Medicine*. Elsevier, 2020, pp. 1033–1038.
- [3] L. S. Lilly, *Pathophysiology of Heart Disease: A Collaborative Project of Medical Students and Faculty, 6th Edition*. Lippincott Williams & Wilkins, 2016, pp. 70–78.
- [4] A. Bansal and R. Joshi, “Portable out-of-hospital electrocardiography: A review of current technologies,” en, *J. Arrhythm.*, vol. 34, no. 2, pp. 129–138, Apr. 2018.
- [5] C. Chen, C. Ma, Y. Xing, *et al.*, “An atrial fibrillation detection system based on machine learning algorithm with mix-domain features and hardware acceleration,” in *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, Mexico: IEEE, Nov. 2021.
- [6] S. Sakib, M. M. Fouda, Z. M. Fadlullah, N. Nasser, and W. Alasmary, “A proof-of-concept of ultra-edge smart IoT sensor: A continuous and lightweight arrhythmia monitoring approach,” *IEEE Access*, vol. 9, pp. 26 093–26 106, 2021.
- [7] S. M. Ahsanuzzaman, T. Ahmed, and M. A. Rahman, “Low cost, portable ECG monitoring and alarming system based on deep learning,” in *2020 IEEE Region 10 Symposium (TENSYP)*, Dhaka, Bangladesh: IEEE, Jun. 2020.
- [8] M. Hartmann, H. Farooq, and A. Imran, “Distilled deep learning based classification of abnormal heartbeat using ECG data through a low cost edge device,” in *IEEE Symposium on Computers and Communications (ISCC)*, 2019, pp. 1068–1071.
- [9] A. Faraone and R. Delgado-Gonzalo, “Convolutional-recurrent neural networks on low-power wearable platforms for cardiac arrhythmia detection,” in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Genova, Italy: IEEE, Aug. 2020.
- [10] A. Scirè, F. Tropeano, A. Anagnostopoulos, and I. Chatzigiannakis, “Fog-computing-based heartbeat detection and arrhythmia classification using machine learning,” *Algorithms*, vol. 12, no. 2, p. 32, 2019.

- [11] S. Raj and K. C. Ray, “A personalized point-of-care platform for real-time ECG monitoring,” *IEEE trans. consum. electron.*, vol. 64, no. 4, pp. 452–460, Nov. 2018.
- [12] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, “On-device training under 256kb memory,” *arXiv preprint arXiv:2206.15472*, 2022.
- [13] S. M. Lobodzinski, “ECG instrumentation: Application and design,” in *Comprehensive Electrocardiology*, London: Springer London, 2010, pp. 427–480.
- [14] N. Faruk, A. Abdulkarim, I. Emmanuel, *et al.*, “A comprehensive survey on low-cost ECG acquisition systems: Advances on design specifications, challenges and future direction,” in *Biocybern. Biomed. Eng.*, vol. 41, no. 2, pp. 474–502, Apr. 2021.
- [15] Webster, *Medical instrumentation - application and design, fifth edition*, 5th ed., J. G. Webster and A. J. Nimunkar, Eds. Hoboken, NJ: Wiley-Blackwell, 2020.
- [16] R. H. Rakin, A. Siam, M. R. Hossain, and H. U. Zaman, “A low-cost and portable electrocardiogram (ECG) machine for preventing diagnosis,” in *2019 International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, Dhaka, Bangladesh: IEEE, Jan. 2019.
- [17] S. Alam, M. Hossain, M. Rahman, and M. K. Islam, “Towards development of a low cost and portable ECG monitoring system for Rural/Remote areas of bangladesh,” *International Journal of Image, Graphics and Signal Processing*, vol. 10, pp. 24–32, 2018.
- [18] *8-bit avr microcontroller with 32k bytes in-system programmable flash*, ATmega328P, [Accessed December 31, 2022], Atmel Corporation. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.
- [19] *Arduino nano*, V2.3, [Accessed December 31, 2022], Arduino. [Online]. Available: <https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf>.
- [20] *Single-lead, heart rate monitor*, AD8232, [Accessed December 31, 2022], Analog Devices. [Online]. Available: <https://cdn.sparkfun.com/datasheets/Sensors/Biometric/AD8232.pdf>.
- [21] J. Pan and W. J. Tompkins, “A real-time QRS detection algorithm,” in *IEEE Trans. Biomed. Eng.*, vol. 32, no. 3, pp. 230–236, Mar. 1985.
- [22] M. Sznajder and M. Łukowska, *Python Online and Offline ECG QRS Detector based on the Pan-Tompkins algorithm*, version v1.1.0, Jul. 2017. DOI: 10.5281/zenodo.826614. [Online]. Available: <https://doi.org/10.5281/zenodo.826614>.
- [23] V. Golovko, “Deep learning: An overview and main paradigms,” *Optical memory and neural networks*, vol. 26, no. 1, pp. 1–17, 2017.
- [24] “ANSI/AAMI EC57:2012-Testing and Reporting Performance results of cardiac rhythm and ST segment measurement algorithms,” *American National Standard*, 2013.
- [25] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.