# APPENDIX D - CODE DOCUMENTATION IN C++

Code development in C++:

For the development of the embedded code, we wanted to reapply all the steps carried out in the training of the model written in Python, as this is extremely important for the model to be able to perform well and work under the same conditions in which it was created and trained;

In this way, the code development in C++ was divided into steps that would make the work easier and could help maintain the same conditions that were applied at a high level with Python. The chosen steps were then: data acquisition and data processing in real time, separation of beats, introduction of the signal in the model and classification. All these steps will be opened and worked on below.

Acquisition and training of data in real time:

In the acquisition step, the Adafruit_ADS1x15 library was used to configure and read the ADC, where 3 interrupts were used to manage the code's operation. The first interrupt monitored the change in a push button, and triggers a debounce routine to confirm that the switch was pressed. Then, two interrupts were triggered, in which one is triggered every 0.01s and is responsible for releasing the data acquisition flag and keeping the sampling frequency at 100 Hz, and a second interrupt is responsible for signaling the end of the data acquisition time which is 10s.

In the data processing process in Python, the library used was ***scipy***, so that in Python two filters were defined: a high-pass with a cut-off frequency of 0.5 Hz and a low-pass with a cut-off frequency of 41 Hz In order to replicate data processing in C++, the libfilter library was used, which has very similar implementations to those used in the process carried out at a high level for model training, where this library enabled the creation of a low-pass filter of 3rd order with a cutoff frequency of 41 Hz, and a 2nd order high pass filter with a cutoff frequency of 0.5 Hz, thus forming a bandpass filter that is applied in real time as the data is acquired from the ADC.

All read and processed data were stored in an array declared as static in the microcontroller's memory that is created at the time of processor initialization. These data were stored to be used in the following steps, for the creation of data that are delivered with the objective that the classification of the model is carried out.

Separation of heartbeats:

In the step of separating the heartbeats in Python, the biosppy library was used to normalize the heartbeats and create the images, where at first we tried to make a direct translation of this library to C++, which was not very practical, leading to the creation of our own code for peak detection that was first applied in Python and then translated to C++, so with a few minor adaptations this routine managed to be translated and tested inside the microcontroller.

After all the signal peaks are identified, it goes to the heartbeat separation step. With the QRS point indices in hand, it is possible to assemble a matrix containing all the heartbeats present in the signal, taking 60 points in front of the QRS peak and 40 points behind the QRS peak,

thus forming the heartbeats. Here, care is only taken to avoid picking up points that were on the edge of the signal, which can configure signals without correct identification.

Input of the signal in the model and final classification:

In this step, the deployment is carried out using the Edge Impulse tool, which was responsible for optimizing and converting the model, thus reducing both the memory expenditure used by the model and the processing expenditure necessary for it to be able to run on a processor with fewer resources, such as embedded processors. In exchange for that, the model loses some of its precision, which in the case of this project proved to be imperceptible.

After processing the model by the tool, a library is obtained with the model application ready, where both the model and some tools that were also delivered by Edge Impulse are extracted, such as the feature vector where the data that will be processed is introduced. by the model. With this in mind, we went through the classification stage, where the valid heartbeats were introduced in the features vector provided by the Edge Impulse library.

From there, the classification function is called, which will analyze and classify each heartbeat individually, so that after classifying all the beats, a comparison is made between the number of heartbeats classified as AFIB and SR. The one with the highest number of ratings is indicated as the heart rhythm identified in the 10s exam.

Trobleshoot:

Possible improvements of the work carried out would be: improvement in the quality of the filters performed, creation of a model capable of detecting a greater number of arrhythmias, in addition to an increase in the sampling rate, which would make it possible to acquire more accurate data with a smaller spacing between samples.