Mid-year Progress Report

Part IV Project
Group 34

Research Problem:There is already two different types of model developed and proposed for pacemaker i.e open-loop and closed-loop.

We are building upon Luman Wang's model, which theoretically works, but it fails to consider the effects of pacing on the intestine. To address this limitation, our focus shifted towards creating a closed-loop pacemaker system. The main challenge we encountered was dealing with noise signals that were close to the original input signal. However, we developed an algorithm based on bullframework2011 that effectively mitigates this issue. The algorithm proposed by Larisa and Daryl proved to be promising, but it required implementation on an embedded platform and integration with Wang's design wang2022framework. While the algorithm functions adequately, it still yields a sensitivity score of 0.8, leaving room for improvement.

Research Plan

TASK TITLE	PROGRESS
Basic Embedded	
MATLAB model to C/C++	80%
WCET/Histogram creation (run N number of times)	
Machine Learning	
ML Discussion (time-series) for detection (classification), filtering if possible	100%
ML Design and Implementation	60%
WCET/Histogram Implementation (run N number of times)	
Final Output	
Feed to real-time moving graph of inputs and marked outputs for pace detection	
Comparison data of accuracy between embedded processor and MATLAB model in graphs and data	
Comparison data of performance between embedded processor and ML algorithms in graphs and data	

A. **Current Work**

We have converted the matlab model to C embedded Although it is not fully refined and functional as expected.

Some problems are solving data types passing from one function to another . There is also some pointers handling required so all functions could be runned from a call of the main function. There were various issues in implementing it such as neo transform, spline in artifact removal, and filter.

(i) We implemented C code for "***NEO transform***" algorithm, which takes an input array of doubles and calculates a new array by squaring each element and subtracting the product of its adjacent elements (excluding the first and last elements). The algorithm allocates memory for the output array, performs the transformation in a loop, and returns a pointer to the new array.

(ii) We implemented ***artifact removal*** in following way:-

Slope Calculation:
The algorithm first computes the start and end slopes of the artifact within the window. These slopes are used to determine the cubic spline coefficients for interpolation.

Cubic Spline Coefficients:
Using the calculated start and end slopes, along with the artifact's length, the algorithm determines the coefficients required for cubic spline interpolation. These coefficients are used to construct a smooth interpolation function that replaces the artifact region.

Generating x-values for Interpolation:
The algorithm generates a set of x-values in the interval [artifact_start, artifact_end] with an increment of 1. These x-values represent the positions within the artifact region where the interpolation function needs to be computed.

Interpolation and Artifact Replacement:
The cubic spline interpolation is performed using the generated x-values and the calculated coefficients. The interpolated values are then stored in a temporary array. Subsequently, the artifact region in the original signal window is replaced with these interpolated values.

Copying Remaining Values:
Before and after the artifact region, the algorithm copies the values from the original signal window to the output, ensuring that the entire window is now artifact-free.

(iii) we implemented ***artifact detect*** with help of math library in following manner :-

Absolute Values Calculation:
The algorithm first calculates the absolute values of all elements in the input signal window and stores them in the array absolute_signal_window. This step ensures that only the magnitudes of the signal samples are considered, regardless of their positive or negative values.

Finding the First Non-NaN Value:

Next, the algorithm searches for the index of the first non-NaN value in the absolute_signal_window. If the first element is not NaN, the index is set to 1. Otherwise, it iterates through the array until it finds the first non-NaN value.

Finding the Maximum Value and its Index:
The algorithm proceeds to find the maximum value within the absolute_signal_window and its corresponding index. It starts searching from the previously determined index of the first non-NaN value and iterates through the array to find the peak (maximum) value.

Artifact Detection:
Once the maximum value and its index are found, the algorithm compares the maximum value with the provided threshold. If the maximum value exceeds the threshold, it indicates the presence of an artifact. In this case, the function returns the index of the artifact, representing its location within the signal window. If the maximum value is not greater than the threshold, the function returns NaN, indicating that no artifact was detected in the given window.

There were some other functions as well which were moving average which was easy to implement. There were also filters to implement which were some hardcoded code matla with that still need to be fixed.

Machine Learning Techniques:
We also explored various techniques like logistic regression, decision trees, and others. However, we found that these methods were unsuitable for our task because they treated data points as independent entities, neglecting the crucial time-dependent patterns within the gut data. To address this issue, we ultimately settled on two algorithms: Support Vector Machines (SVM) and Long Short-Term Memory (LSTM) networks. Both of these techniques are well-suited for time series data, making them ideal candidates for our real-time pacemaking project.

Specifically, the **sliding window approach** is an essential aspect of both SVM and LSTM. It involves breaking down the time series data into smaller overlapping windows, allowing the algorithms to capture temporal patterns effectively.

However, the key difference lies in how each algorithm handles these windows: With SVM, the sliding window approach helps extract relevant features from each window, and these features are then used to make predictions. On the other hand, LSTM, as a type of recurrent neural network, uses the sliding window to maintain memory of past observations, enabling it to learn complex temporal patterns in the gut data. These algorithms are going to be used for binary classification of data on whether pacing should happen or not.

At this state, both algorithms have already been written out, the only issue lies with the sort of data that should be fed into these algorithms as the inputs as our group is still waiting for the model to execute through an embedded processor fully such that data can be extracted from it for training and testing after it has been processed.

B. **Future Work**

Our group wants to create a histogram that shows how long it takes for a real-world application to run in the worst-case scenario, where it illustrates the worst-case execution times of an embedded

processor when responding to gut activity. We will run the application many times under different conditions to collect data on its performance and compare its accuracy to the original MATLAB model. This information will help us understand how the application behaves in critical situations and where it may have problems.

As far as machine learning goes, our last steps involve testing and improving their performance. We will carefully examine how well the algorithms make predictions by using different sets of data (from the different channels) to see how accurate they are. We want to make sure they work well in various situations. When it comes to evaluating the machine learning techniques, relevant evaluation metrics for machine learning algorithms (such as accuracy, precision, recall, F1-score , etc.) will be accounted for. This will be drawn from a confusion matrix that provides a clear summary of the model's performance, showing how many predictions were correct (TP and TN) and how many were incorrect (FP and FN).

Something that our group is considering once the above are completed is whether the model can handle larger and more complex datasets that test its scalability. By handle, as to whether the accuracy will remain more or less the same or if there will be a large decrease.