Computing
https://doi.org/10.1007/s00607-022-01096-z

REGULAR PAPER

VAMPIRE: vectorized automated ML pre-processing and post-processing framework for edge applications

Ali W. Daher[1,2,3,4] · Enrico Ferrari[4] · Marco Muselli[3,4] · Hussein Chible[2] · Daniele D. Caviglia[1]

Received: 23 December 2021 / Accepted: 8 June 2022

Abstract
Machine learning techniques aim to mimic the human ability to automatically learn how to perform tasks through training examples. They have proven capable of tasks such as prediction, learning and adaptation based on experience and can be used in virtually any scientific application, ranging from biomedical, robotic, to business decision applications, and others. However, the lack of domain knowledge for a particular application can make feature extraction ineffective or even unattainable. Furthermore, even in the presence of pre-processed datasets, the iterative process of optimizing Machine Learning parameters, which do not translate from one domain to another, maybe difficult for inexperienced practitioners. To address these issues, we present in this paper a Vectorized Automated ML Pre-processIng and post-pRocEssing framework, approximately named (*VAMPIRE*), which implements feature extraction algorithms

Enrico Ferrari, Marco Muselli, Hussein Chible and Daniele D. Caviglia contributed equally to this work.

✉ Ali W. Daher
ali.daher@edu.unige.it

Enrico Ferrari
enrico.ferrari@rulex.ai

Marco Muselli
marco.muselli@ieiit.cnr.it

Hussein Chible
hchible@ul.edu.lb

Daniele D. Caviglia
daniele.caviglia@unige.it

1 Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture, University of Genoa, Via Opera Pia 11, 16145 Genoa, Liguria, Italy

2 Ph.D. School for Sciences and Technology, Lebanese University, Beirut 6573/14, Lebanon

3 Institute of Electronics, Computer and Telecommunication Engineering, Consiglio Nazionale delle Ricerche, Via De Marini 6, 16149 Genoa, Liguria, Italy

4 Rulex Innovation Labs, Rulex inc, Via Felice Romani 9, 16122 Genoa, Liguria, Italy

Published online: 22 June 2022

capable of converting large time-series recordings into datasets. Also, it introduces a new concept, the *Activation Engine*, which is attached to the output of a Multi Layer Perceptron and extracts the optimal threshold to apply binary classification. Moreover, a tree-based algorithm is used to achieve multi-class classification using the *Activation Engine*. Furthermore, the internet of things gives rise to new applications such as remote sensing and communications, so consequently applying Machine Learning to improve operation accuracy, latency, and reliability is beneficial in such systems. Therefore, all classifications in this paper were performed on the edge in order to reach high accuracy with limited resources. Moreover, forecasts were applied on three unrelated biomedical datasets, and on two other pre-processed urban and activity detection datasets. Features were extracted when required, and training and testing were performed on the Raspberry Pi remotely, where high accuracy and inference speed were achieved in every experiment. Additionally, the board remained competitive in terms of power consumption when compared with a laptop which was optimized using a Graphical Processing Unit.

Keywords Algorithms · Artificial intelligence · Multi Layer Perceptron · Classification · Edge computing · Feature extraction · Machine learning · Pre-processing · Post-processing · Signal processing

Mathematics Subject Classification 68W01 · 68W25 · 68W32 · 68W40 · 68T01 · 68T05 · 68T07 · 68T10 · 68T27 · 68T30 · 68T40 · 68Q30

1 Introduction

Machine learning (ML) has become a key technique that is used in many modern applications in many fields, such as busines, engineering, and healthcare. It is considered a form of artificial intelligence (AI), and because it possesses the ability to learn, adapt, and remember, it proved superior to classical hard-coded programming methods, which are application-specific in nature. Many ML algorithms can make predictions in multiple fields while generalizing and have a relatively high accuracy regarding a specific application. However, a ML process requires a workflow that usually consists of data collection, feature extraction, setting parameters, and applying the forecast on both training and test-sets. This process is iterative and, relying upon the application, some phases may require more tuning than others, depending on whether accuracy is reduced due to high bias or high variance.

ML algorithms usually require that the practitioner possesses domain knowledge in the field associated with the dataset that he is using, and most notably when feature extraction is required. Feature extraction is the process of conversion of data into a better format to use in ML setups. However, currently, many pre-processed datasets are available, where ML algorithm can be applied directly to the data without any domain knowledge. This can be the case in many medical, physical, and business applications.

However, even after acquiring pre-processed data, the ML algorithm hyperparameters still need to be tuned to optimize forecast accuracy. Moreover, this parameter tuning process can require a notable effort and sufficient experience in

ML in general, and even ML domain experience in that specific application. This is dominantly the case, since hyper-parameter tuning setups does not translate well from one domain to another.

In this paper, we present *VAMPIRE*, a vectorized automated ML pre-processing and post-processing framework. It consists of novel pre-processing and post-processing algorithms that we developed in the Python programming language. The first facilitates the feature extraction phase in case the user is dealing with time-series data, where the algorithm can be applied to fully annotated and semi-annotated datasets. Furthermore, a post-processing algorithm has been developed that implements *Activation Engine*s, which are applied to the outputs of a multi layer perceptron (MLP) and extracts the optimal threshold relevant to the test-set based on training accuracy.

The *VAMPIRE* framework achieves robust performance, since it can be applied to multiple types of time-series for feature extraction and can be implemented to all applications performed using MLP through the *Activation Engine* concept. Also, the main computational blocks have been implemented using Python Numpy vectors to considerably increase the speed of feature extraction, dataset generation, and post-processing substantially. Moreover, a subset of the extracted features from the time-series has been applied using the Rulex software in an edge computing arrangement. Rulex [1] is a general-purpose ML platform that operates through a graphical user interface (GUI); It can operate in a client/server setup where the ML forecasts are applied on the general-purpose Raspberry Pi IoT device [2]. Rulex applies Switching Neural Networks [1] and Positive Boolean Function Reconstruction [3] to implement explainable AI which provide a white-box learning approach to ML.

Multiple classification forecasts were applied using various datasets taken from various fields in a mesh setup, where some datasets were pre-processed using the algorithms developed in this paper, and others were already pre-processed using different techniques. Also, for the forecasts, Rulex was applied to a subset of the applications, while MLP with and without the new post-processing block was applied to the remaining datasets. Additionally, some forecasts were performed using just one of the ML setups. Also for four of the used datasets, a comparison with the literature is outlined and discussed since there exists viable related work suitable for comparison. Moreover, a complete description is provided in the experimental part of this paper. All ML training and testing were applied in an edge computing setup with limited resources by applying training and testing outside a cloud server. Therefore, ML workflows were either applied using Rulex running on the Raspberry Pi and in a client/server arrangement, or by running a MLP remotely on the Raspberry Pi with *VAMPIRE*'s *Activation Engine* placed at its output. However, since the *Activation Engine* is a basic binary classification block, a multi-class tree-based technique which was developed in [4] was incorporated into the *VAMPIRE* Framework to implement multi-class classification. Furthermore, the inference time of the framework was compared to other edge setups and so was its power consumption with a laptop having an onboard graphical processing unit (GPU).

This paper is organized as follows: Section 2 reviews the literature while discussing various automated ML techniques, related pre-processing and post-processing algorithms, and introduces the Rulex [5] and Raspberry Pi [2] platforms used in this paper. The new *VAMPIRE* pre-processing algorithms are presented in detail in sect. 3 and

VAMPIRE's *Activation Engine* block is described in sect. 4. Section 5 presents the data sources used for feature extraction and classification to test the *VAMPIRE* framework's robustness. All experimental results achieved using the data sources and the performance comparisons are provided in Sect. 6. Also, we draw a conclusion in Sect. 7, and finally an appendix explaining version two of the pre-processing algorithm in detail concludes the paper.

2 Literature review

Feature extraction or pre-processing is used to convert raw data into a more convenient format to apply various ML algorithms. On the other hand, post-processing is the act of applying an optimization block at the output of a ML workflow to improve accuracy.

2.1 Feature extraction and pre-processing of time-series

A certain dataset in its available format may not be directly applicable to a ML algorithm. Therefore, feature extraction is usually applied to time-series data, and most notably biomedical datasets.

In [6] authors propose a Nature–Inspired Differential–Evolution for feature selection which was applied as a pre-processing method using the Logistic Regression algorithm. The algorithm selects features based on the accuracy threshold applied taken from the output of Logistic Regression.

A web platform for biomedical time-series pre-processing [7] was applied to Electrocardiogram (ECG) signals for cases of myocardial ischemia, atrial fibrillation, and congestive heart failure. Biomedical signals are usually noisy and so filtering is an important issue in time-series pre-processing. In [8] an ECG signal is pre-processed using an adaptive filter to remove noise before applying discrete wavelet transform to extract features for classification using support vector machines (SVM). In [9] authors describe the process of applying wavelet transform to extract features from an Electroencephalogram (EEG) signal for classification using MLP.

In [10] authors review feature extraction and feature selection in ML techniques. They describe how feature extraction is used to reduce the dimensionality of a dataset through the transformation of the feature space, and also discuss how feature selection reduces the dimensionality as it points out the subset of features that possess the most significant effect on forecast accuracy.

A feature extraction Python pre-processing Library based on nature-inspired optimization algorithms has been developed in [11]. The EvoPreprocess library is compatible with the ML Scikit–Learn Library and performs well compared to other frameworks.

A time-series is a waveform that corresponds to physical, biological, or business data, that changes over time and is found in various ML applications. In [12], a Bag-of-Words representation for biomedical time-series is presented. The methods presented treat the time-series as text documents and extract segments as words. In that paper, the Bag-of-Word's methods were applied on two ECG datasets taken from [13] and an

EEG dataset from [14]. An ensemble learning approach is also applied to biomedical datasets in [15] which use the Chinese cardiovascular disease database [16]. Shortfuse is presented in [17], which is a biomedical time-series feature extraction method that implements Long Short–Term Memory to outperform competing models by 3%, in terms of accuracy.

In the case of biomedical applications, ECG, EEG, and Photoplethysmogram (PPG) are the most common measurements. ECG consists of measuring the electrical activity of the heart, PPG consists of the measurement of blood pressure and heart rate using light emitting diode, and an EEG provides the electrical signals taken from the scalp and usually have a much wider frequency spectrum than ECG and PPG signals. These three signal categories are used in this paper as data sources for feature extraction using the *VAMPIRE* framework.

2.2 Automated ML and post-processing

A ML workflow includes also a hyper-parameter tuning, such as setting class-weights and feature-weights, which are used as parameters in a loss function to optimize classification accuracy regarding the training set. In some applications, expert knowledge may not be available to tune the ML parameters, and so, automated ML techniques and software packages have been developed to facilitate the process of implementing forecasts. However, this can be a time-consuming process and requires much more processing power compared to classical workflows that are performed by experts in the field.

Post-processing is the process of applying a cognitive block at the output of a ML algorithm such as MLP or Logistic Regression. This is applied for each sample and may consist of applying multiple ML forecasts in parallel before combining all the outputs to optimize the overall accuracy.

In [18] authors apply ensemble learning using MLP on weather forecasts in different configurations. Ensemble learning is a technique that applies multiple ML forecasts in parallel and on the same dataset. Then, the outputs are brought together, and an optimal output is selected using a voting scheme.

Guidelines are provided by authors in [19] to select the best ML scheme in the case of biomedical application. They have performed forecasts on 31 datasets and applied various ML algorithms to select the best configuration.

Another approach to ML automation is meta-learning, which consists of iterating not just parameter tuning but also ML algorithms in the quest for optimization. This can be perceived as an optimization problem per algorithm and a collective optimization problem which attempts to iterate between different ML techniques. According to this approach, the Auto–WEKA package [20] has been presented as a modification of the original WEKA workbench [21], conceived as a toolset for data analysis and predictive modeling. Auto–WEKA applies Bayesian optimization to automatically select the algorithm to be used, along with its tuned parameters. It is a tool dedicated to non-experts to apply ML forecasts and to achieve good performance. In [22] authors present the AUTO–SKLEARN framework which is a Python implementation based on the Scikit–Learn Library. The framework applies ensemble learning at the output of

a meta-learning configuration to automate parameter tuning while choosing the most suitable ML algorithm.

A situation where expert knowledge is not always available is presented in [23] which consists of the prediction of the performance of a tunnel boring machine. So, a Bayesian optimization scheme is applied for hyper-parameter tuning and algorithm selection, and a neural architecture search for MLP optimization.

In [24] authors claim that modern MLP is poorly calibrated since increasing the depth improves accuracy, however, may affect the calibration negatively. Furthermore, a domain-specific ML framework dedicated to predicting the properties of inorganic materials has been presented in [25].

2.3 Platforms used on the edge

IoT encompasses many fields of application which demand the monitoring and management of power, bandwidth, and reliability. For example, authors in [26] propose an IoT framework for resource management with the goal of optimal task offloading, where the framework is intended for healthcare systems. Power consumption during ML forecasts on edge is a critical issue due the limited power availability for IoT nodes. Therefore, power consumption optimization on the edge using ML is investigated in [27–29]. Also, in [30] ML is applied on time-series data from IoT sensors in order to predict failure in a slitting machine. Additionally, in [31, 32] MLP's are used to improve security in IoT networks.

Therefore, with the vast scope of ML application in IoT along with the existing benefits of ML solutions, we've developed the present framework to perform ML pre-processing and prediction using limited resources. Consequently, the hardware platform chosen in this paper is the multi-purpose and widely popular Raspberry Pi [33]. The Raspberry Pi is used as an edge computing node where the act of storing and making a cognitive decision on a local node rather than making computation and exchanging big data with a cloud server [34].

The Raspberry Pi is ideal for edge computing applications since it possesses adequate processing power and is able to manage enough data for most ML applications. As for the software part, The ML platform used is Rulex [5] which is a ML package directed towards non-domain-experts and which has been ported to the Raspberry Pi as reported in [2]. The user does not need to write any code to carry on ML forecasts thanks to an easy-to-use GUI. It allows to easily manipulate the data, apply one among many ML algorithms, and visualize the output. Following this approach, a subset of the forecasts applied in this paper were performed using Rulex in an edge computing environment. The client/server arrangement that has been implemented with Rulex is presented in Fig. 1, where as shown, the Rulex engine runs on the Raspberry Pi and is operated remotely while relying on a database server as a common storage point. Also, forecasts were performed using *VAMPIRE*'s post-processing algorithm, where an SSH connection [35] across the public network was established through the interface of the VSCode editor (without a third-party database), which was used for remote development.

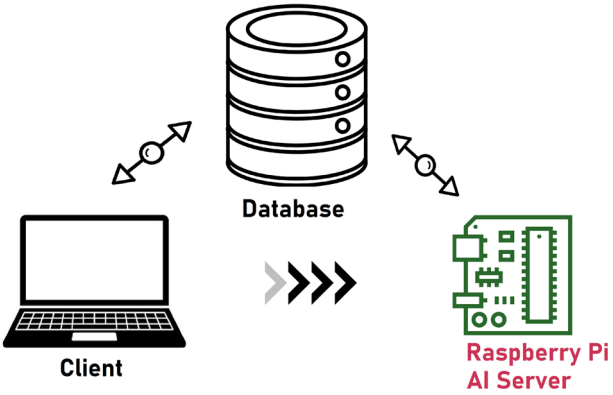

Fig. 1 ML forecasts applied using Rulex software over the public network in a client/server setup

In case of using Rulex as classifier, the client consists of an HP laptop with an Intel code-i7 1.8 GHz processor having 16GB of RAM running Windows 10 and without a GPU. The AI server is a Raspberry Pi 3B+ model with a ARM Cortex-A53 1.4GHz processor and running Raspbian as its operating system. The Databse used is a PostgreSQL 12 docker deployed server operated through a dedicated ODBC driver while running on Azure Cloud. The Client consists of a graphical interface developed in Python 2.7, and the AI Engine is written in C/C++ vectors for optimal performance. The connection between client and server is an encrypted SSH connection over the public or private network. In case of operating *VAMPIRE* remotely, again, an SSH connection locally or through the local or public network is applied where the code is written in Python 3.8 and the ML models and post-processing are developed using Tensorflow and Numpy libraries respectively. As for the dataset generation and feature extraction algorithms, they are also written in the Python 3.8 Numpy library and are performed outside the Raspberry Pi on an MSI laptop with an Intel 7-10750H CPU 2.60GHz processor with 16GB of RAM and a 6GB 2060 Nvidia GPU and on the same HP laptop used as a client. Consequently, the datasets were generated simultaneously on both computers with roughly the same time and efficiency.

2.4 Motivation for ML edge computing frameworks and automated dataset generation

Applying ML on edge devices and under performance constraints requires the development of light data processing programs which are not much of a burden under limited resources. However, specification restrictions are not the only issue faced in edge paradigms. Usually, training takes place on a remote cloud server [36] and so are the online training updates due to continuously changing data from the real world. The former demands bandwidth resources while compute-heavy ML tasks can cause the edge node to lag which results in undesired downtime. Consequently, both of these issues are common design goals in an industrial ML system deploy-able on the edge.

Therefore in [37] Edge2Train is presented, which is a an edge computing setup that allows for online training on the edge with automatic dataset generation capability. The framework uses an instrumentation feedback system which monitors changes in the real world in order to update and validate the model through training on the edge. Even so, this arrangement does not attempt to tune the parameters of ML model but rather takes them as inputs from an external application. Furthermore, currently this setup only deals with binary classification applications. Also in [38] authors implement a pipeline that executes convolutional neural networks (CNN) on edge devices. The pipeline aims to reduce the size and structure of the network and the volume of data such that they can be deployed on a unit with limited resources, while preserving accuracy. However, this method trains the model outside an edge node before deployment and does not include feature extraction which is missing from the framework.

For final or industrial implementation of edge-based AI systems, one fundamental characteristic is automated feature generation regardless if training is applied in one shot or whether online updates are applied. Time-series forecasting setups on the other hand usually include feature extraction in order to reduce the dimensionality of data and since it may improve prediction accuracy. Therefore, automating this process is desirable considering the large amounts of data that can be collected by biomedical edge devices and from multiple patients. Authors in [39] review the automation of time-series ML processes which encompass: Pre-processing, feature engineering, hyper-parameter optimization, model selection and ensembling. They reveal that the majority of publications only cover three out of the five pipelines mentioned. Also, based on their report, ensembling and parameter optimization are not suitable to be applied on the edge due to high computation requirements and since multiple complete training runs are required twice for each block. EEG automated feature extraction is reported in [40] where a GUI named Training Builder was developed which generates a dataset automatically based on predefined computations and relies on a windowing technique to split an unbounded time-series into smaller finite sets. However, the length of sets and the overlap times needs to be defined manually. Consequently, a practitioner may need to experiment with these time values since they affect the forecast accuracy which limits the automation attribute.

The *VAMPIRE* framework described in this paper avoids hyper-parameter tuning and Automated-ML which are increasingly compute-expensive but relies on a post-processing block that can run to a finite number of times where the runtime is adjustable for a loss in accuracy, where this loss is usually very small or improbable. Furthermore, the framework adds a multi-class classification feature using the tree-based method which is described in Sect. 6. As for the automated dataset generation part of the *VAMPIRE* framework, the process is completely automated since in the case of dealing with streams of data, the algorithm will set a variable length for each finite set which in continuously checked and updated without any predefined parameters and variables. The feature extraction algorithms provided in this framework may be implemented in an online learning approach due to their automated nature since they do not require parameters from an operator or from any additional programs. Also, the same statement can be declared regarding the post-processing technique which is adopted.

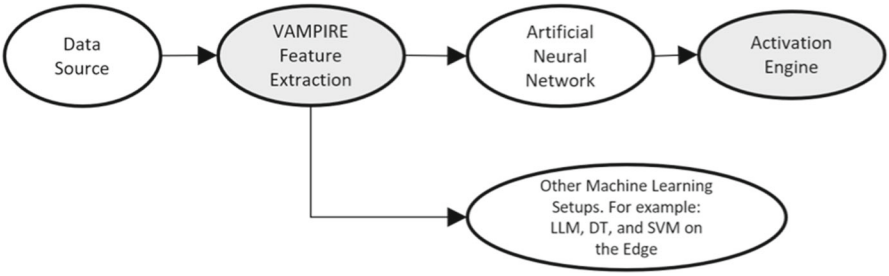

Fig. 2 Workflow of the *VAMPIRE* framework with various possible setups

3 Novel *VAMPIRE* pre-processing algorithms

The paper presents a new feature extraction method which takes a time-series as an input, and can automatically generate a dataset in comma separated values (CSV) format that can be processed by any ML algorithm. This process is automated and may only require the user to apply some filtering technique to the input signal to reduce noise. Furthermore, the key computational blocks in the code were implemented using Python Numpy to increase processing speed by a considerable factor.

The pre-processing algorithm is applied in two arrangements. The first operates directly by applying multiple algorithms in case the signals are annotated by experts in the field. The second version has been developed to deal with poorly annotated datasets.

Preliminary operations consist in subdividing the whole time-series in portions, each one with start and end clearly defined, and to annotate each portion according to a classification provided by an expert in the field. A semi-annotated dataset consists of a time-series that is classified over a longer period of time, meaning larger portion size, and may be classified by non-domain-experts with limited accuracy, such as a patient annotating her or his own state. The *VAMPIRE* pre-processing algorithms are implemented in two setups, being capable of extracting datasets without any excessive intervention by the ML practitioner. An illustration of the overall workflow of the *VAMPIRE* framework is provided in Fig. 2, where the shaded areas represent to parts where the *VAMPIRE* framework algorithms are being implemented.

In case of time-series data, the *VAMPIRE* pre-processing algorithm converts the recordings into datasets, which can be applied to a MLP with a post-processing block (or any other ML setup). Furthermore, a MLP with an *Activation Engine* at the output may be applied to any pre-processed datasets.

3.1 *VAMPIRE* pre-processing algorithm: *VAMPIRE FE1*

The novel feature extraction algorithm *VAMPIRE FE1* applies fast fourier transform (FFT) [41] to every record in the waveform recordings and stores frequency response information. The FFT input and the output signal are normalized on a scale of *0-N* where *N* is maximum value of the normalized signal. Then, the frequencies from the FFT, and the voltages from the original signal are correlated and multiplied, to form the

Algorithm 1 Algorithm that converts a time-varying voltage signals into the D-Domain

Require: Waveform x indexed by i
Ensure: VHz which is the D-Domain representation of x
$\alpha 1 \Leftarrow lower\ threshold$
$\alpha 1 \Leftarrow upper\ threshold$
$x \Leftarrow input\ waveform$
$x \Leftarrow$ normalize(x)
for *i for every waveform in x from the dataset* **do**
  $FFT \Leftarrow$ frequency response$(x[i])$
  $FFT \Leftarrow$ normalize(FFT)
  $VHz \Leftarrow$ empty array of size$(x[i])$
  for $j < size(x[i])$ **do**
    $temp \Leftarrow x[i, j]$
    for $k <$ size(FFT) **do**
      if $FFT[k] > \alpha 1 * temp$ **then**
        if $FFT[k] < \alpha 2 * temp$ **then**
          $Freq$.insert(k)
        end if
      end if
    end for
    if size$(Freq) > 0$ **then**
      $max \Leftarrow \max(Freq)$
      $VHz[i] \Leftarrow max * x[i, j]$
      $Freq \Leftarrow 0$
    end if
  end for
end for
Return VHz

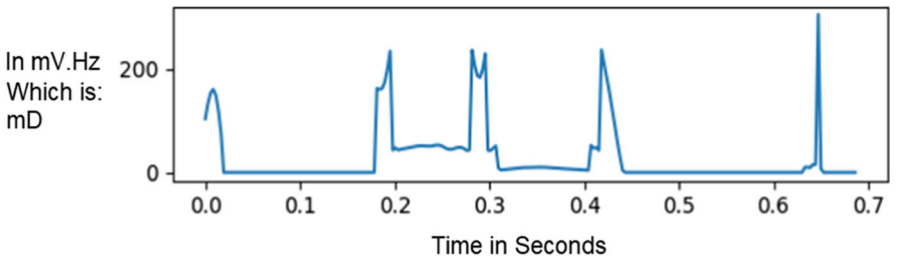

Fig. 3 A waveform used for feature extraction after being converted to the D-Domain. This is the result of applying the feature extraction method describes in Sect. 3, which correlates between time-varying signals and their frequency response

novel V.Hz curve, which is a new unit of representation proposed in this paper, where for every biomedical signal the curve is extracted and plotted versus the time axis as shown in Fig. 3. The process of generating the curve presented in Fig. 3, is illustrated in Fig. 4. As shown, the voltages from the original time-series are correlated with the maximum frequency value taken from the FFT output, in order to generate the V.Hz waveform. Algorithm 1 presents the coding steps needed to convert a time-varying voltage into the D-Domain in order ro further process the signal and extract features. It consists of taking FFT of signal x and normalizing its voltages to the same range, and then multiplying each maximum frequency with the identified voltage to convert x into the V.Hz Domain, which we will refer to as the D-Domain for simplicity.

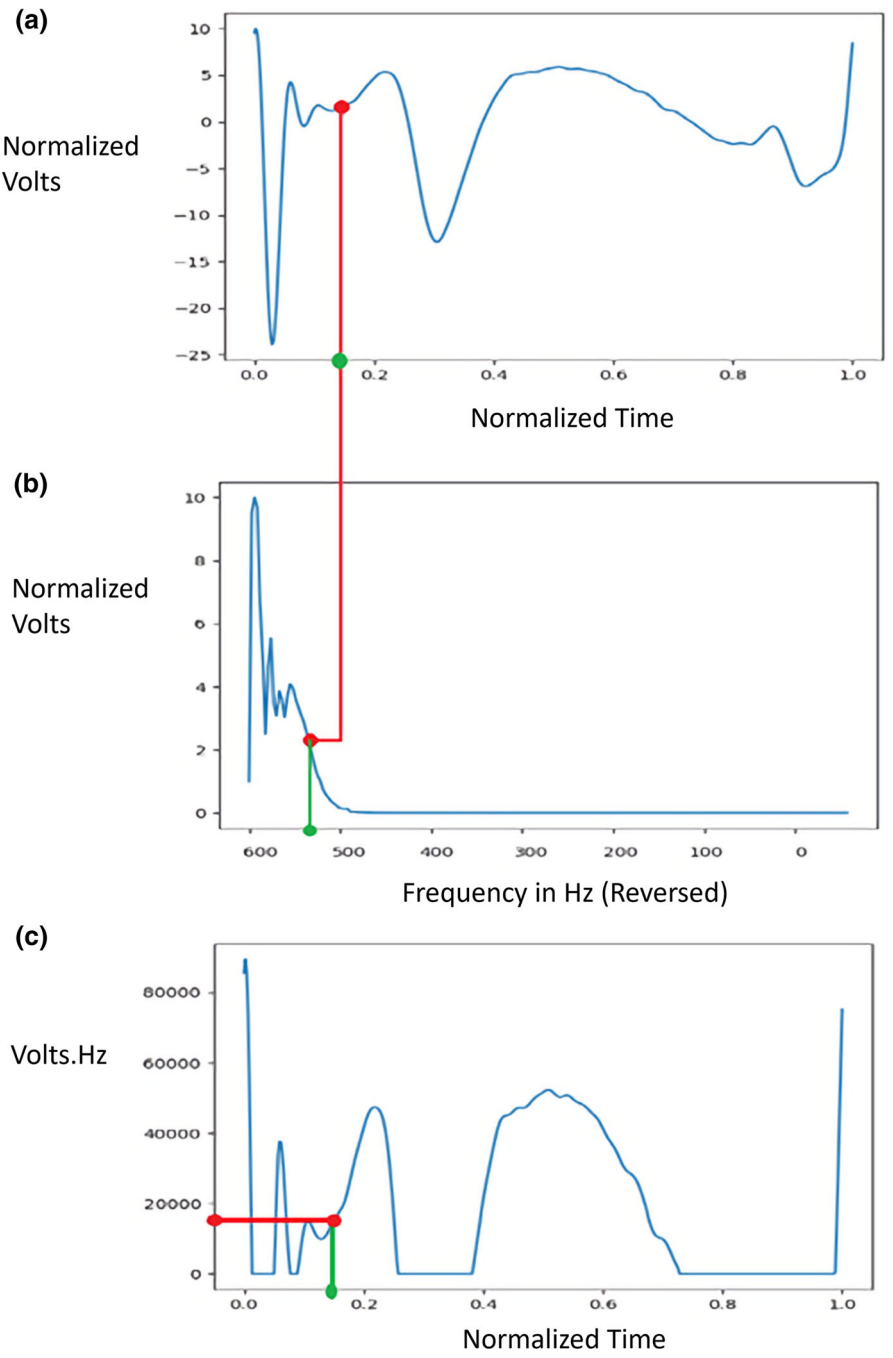

Fig. 4 **a** The original time-series from the recording with normalized voltages. **b** The FFT output for the time-series in **a**. **c** After extracting the maximum frequency from **b** related to a particular voltage in **a**, the normalized voltages and frequencies are multiplied generating the curve in **c**

Algorithm 2 Algorithm that extracts static base variables from the D-Domain signals

Require: D-Domain signals and VHzcross indexes
Ensure: Arrayss a1, b1, Sum1, Diff1, IntegVHz, PeakVhz used for feature extraction
for every *s crossings* in *VHz* **do**
  $VHza \Leftarrow VHz[s]$
  while $k <$ size(*VHzcross*) **do**
    for *j* from *VHzcross*[*s,k*] $\Rightarrow$ *VHzcross*[*s,k+1*] **do**
      if $VHza[j] == peak$ **then**
        ipeak[*s*].insert(*j*)
      end if
    end for
    peak $\Leftarrow$ max(*VHza*[*VHzcross*[*s,k*]:*VHzcross*[*s,k+1*]])
    PeakVHz[*s*].insert(*peak*)
    Area $\Leftarrow$ integral(*VHza*[*VHzcross*[*s,k*]:*VHzcross*[*s,k+1*]])
    IntegVHz[*s*].insert(*Area*)
    k=*k*+2
  end while
  while $d <$ size(*VHzcross*) **do**
    a1[*s*].insert(*ipeak*[*s,d/2*] - *VHzcross*[s,d])
    b1[*s*].insert(*VHzcross*[*s,d+1*] - *ipeak*[*s,d/2*])
    Diff1[*s*].insert(tail(*b1*[*s*])-tail(*a1*[*s*]))
    Sum1[*s*].insert(*VHzcross*[*s,d+1*] - *VHzcross*[*s,d*])
    d = *d* + 2
  end while
end for
Return $a1,\ b1,\ Sum1,\ Diff1,\ IntegVHz,\ PeakVHz$

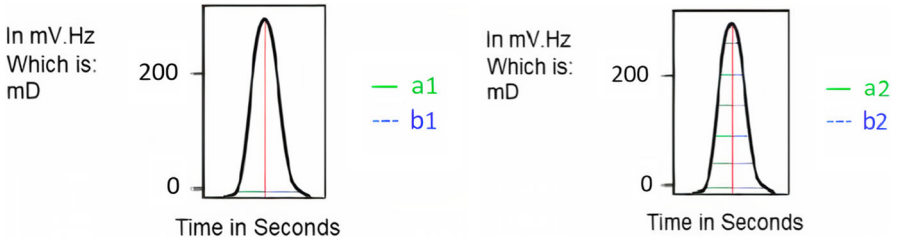

Fig. 5 **a** Base variable a1 and b1 used to derive the first set features. **b** Variable a2 and b2 used to derive second set of features. These variables are used to derive the features used in ML training, by implementing basic statistical operations

After extracting the signal in the D-Domain which is a V.Hz curve, the spikes from this signal are detected and two variables *a1* and *b1* are computed. The *a1* variable represents the time that is taken from the first zero-crossing of the V.Hz signal till the peak of the same signal. Variable *b1* represents the time taken from the peak till the second crossing. Their difference *Diff1* and their summation *Sum1* are computed. *mean*, *standard deviation (STD)*, *variance*, *median*, *range*, *maximum* and *minimum* values are derived for *a1*, *b1*, *Sum1*, and *Diff1*, and are saved as features for that record. Variables *a1* and *b1* are presented in Fig. 5a. Also, for every spike, the *peak*, as well as the *area* or *integral* values can be used to generate the same set of feature. Algorithm 2 describes the procedure that is used to extract *a1*, *b1*, *Sum1*, and *Diff1* by detecting the peak of each spike along with the zero-crossing pairs needed to calculate the base

Algorithm 3 Algorithm that extracts dynamic base variables from the D-Domain signals

Require: D-Domain signals and VHzcross indexes
Ensure: Base arrays a2, b2, Sum2, Diff2 used for feature extraction
for every *s crossings* in *VHz* **do**
  $VHza \Leftarrow VHz[s]$
  for *w* till size(*VHzcross*) with *step-size = 2* **do**
    $h \Leftarrow w/2$
    $v0 \Leftarrow VHzcross[s,w]$
    $v1 \Leftarrow VHzcross[s,w+1]$
    for *n* from *vo* till *ipeak*[*s,h*] **do**
      index1 ⇐ index(*VHza*[*n*])
      difference1 ⇐ (*ipeak*[*s,h*] - *index1*)
      a2[*s,h*].insert(*difference1*)
    end for
    for *n1* from *ipeak*[*s,h*] till *v1* **do**
      index2 ⇐ index(*VHza*[*n1*])
      difference2 ⇐ (*index2* - *ipeak*[*s,h*])
      b2[*s,h*].insert(*difference2*)
      Sum2[*s,h*].insert(*difference1*+*difference2*)
      Diff2[*s,h*].insert(absolute(*difference1*-*difference2*))
    end for
  end for
end for
Return $a2,\ b2,\ Sum2,\ Diff2$

variables along with the *peak* and *area* values. The algorithm requires as input the V.Hz waveform along with the indexes determining the start and end of each spike.

Another class of variables is based on variables *a2* and *b2* presented in Fig. 5b, which consist of the of sets of the averaged triangular function from zero-crossings to the maximum of the spike. Also, corresponding values for *Sum2* and *Diff2* are computed. Afterwords, statistical function of arrays *a2*, *b2*, *Sum2*, and *Diff2* are computed, where again, they consist of the *range*, *mean*, *variance*, *STD*, *median*, *maximum*, and *minimum* values. Algorithm 3 outlines the process used to append all measurements taken for every spike to arrays relating to the reported base variables, where it takes the D-Domain representation an the spike crossings as input, in order to generate the feature arrays that in turn are used to build the dataset in CSV format.

3.2 *VAMPIRE* pre-processing algorithm: *VAMPIRE FE2*

The goal of the second version of the pre-processing algorithm *VAMPIRE FE2* is to extract features from a semi-annotated dataset as in the case of the PPG dataset. It allows to detect the status of time-series in real-time, in case the classes are distributed over long periods of time such as the case in some biomedical applications. The classes in the PPG dataset are taken over an hour and are not period-specific such is the case with the ECG and EEG datasets. So, we could take a fixed period length distributed over an hour and classify the periods with their overall hourly class. However, a different approach is adopted, consisting in the detection of the uniformity of the positively rectified signal and to sense a re-occurrence in the voltages or peaks. This

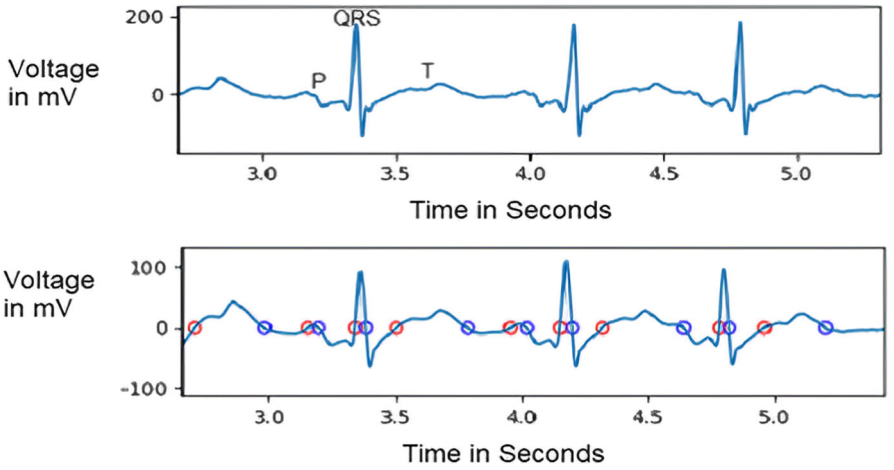

Fig. 6 ECG Waveform before and after passing through the novel moving average algorithm, which generates a more defined signal where it is easier to extract features in the case of a semi-annotated data source

Algorithm 4 Modified moving average

Require: Waveform x
Ensure: avgs which is the moving average results from input x
1: $avgs \Leftarrow x$
2: $step \Leftarrow step\ size$
3: $j \Leftarrow step$
4: **while** $j < size(x)$ **do**
5: $k \Leftarrow j$
6: **while** $k > j - step$ **do**
7: $avgs[j] \Leftarrow avgs[j] + avgs[k-1]$
8: k−−
9: **end while**
10: $avgs[j] \Leftarrow avgs[j]/step$
11: j++
12: **end while**
13: **Return** $avgs$

is performed while determining the window size that is continuously updated. So, child records that are encompassed by a parent record are assigned, but with different window sizes which are equal to the number of the child record's zero-crossing pairs (corresponding to the rectified signal). Therefore, To correctly rectify the input signal, a novel moving average algorithm has been developed which defines the time-series signal as it can be easily detected as shown in Algorithm 4.

Figure 6 presents two example signals: The upper signal is the raw ECG signal which is difficult to process as it is. The novel moving average algorithm was applied to the signal while updating the voltages with the average of the current mean and the *n-1* previous means with a window of width *n*. In case a signal crosses zero the averaged output will remain in its vicinity for a longer time, thus allowing for zero-crossing detection more easily. Algorithm 4 describes the code used tom implement

Algorithm 5 *VAMPIRE* pre-processing algorithm for semi-annotated data

Require: Signal x
Ensure: Crosspx
$\beta 1 \Leftarrow starting\ window$
$\beta 1 \Leftarrow last\ window$
for *every M portions of x* **do**
  Add *N-to-P* and *P-to-N crossing index pairs of x* to *Cross*
  for *j* in *Cross* **do**
    $Patt[j] \Leftarrow \max(x[Cross[2j], Cross[2j+1]])$
  end for
  for *Window* from $\beta 1$ till $\beta 2$ **do**
    for *k* in *Window* till end of *Patt* **do**
      $Mean[k] \Leftarrow \text{Mean}(Patt[k] + \ldots + Patt[k - Window + 1])$
      if $\text{STD}(Mean[k]) < Threshold$ **then**
        Set *Window*
        Break
      end if
    end for
  end for
  while *i* with *every index in Cross* **do**
    if $Last\ index\ in\ Cross - i \geq Window*2$ **then**
      Crosspx.insert(*crossing pairs from Cross*)
      $i = i + Window*2$
    end if
  end while
  Shift *x* by *M*
end for
Return *Crosspx*

this algorithm where as presented returns the mean-based moving average. Figure 6, presents all the zero-crossings of an ECG signal after it has passed through Algorithm 4: the red dots represent zero crossings with positive slope and the blue dots represent zero crossings with negative slope.

The pre-processing algorithm presented provides a novel technique to generate a waveform from both frequency-dominant and voltage-dominant signals, such as EEG and ECG signals respectively. This output waveform exists in the new D-Domain where for every record in the used dataset the bounds of each record is clearly defined. However, the generalized version of the algorithm, which can be applied to poorly annotated datasets provides novel methods that expand the scope of operation into a wider range of applications. This technique is presented in Algorithm 5 where as demonstrated the waveform is subdivided into *M* portions whose window size is based on the maximum voltage of each Negative-to-Positive *(N-to-P)* and Positive-to-Negative *(P-to-N)* crossing pairs. The moving means of multiple maximum voltages are collected before calculating their *STD* which is compared to a predefined *Threshold* in order to set the current *Window* size. this *Window* size is continuously refreshed in order to subdivide the waveform *x* using *Crosspx* in real-time. Appendix A at the end of this paper presents practical details of the code execution of Algorithm 5, by posting data taken directly from the Python interpreter to further explain the operation used to pre-process a time-series, in case it is not meticulously annotated.

4 *VAMPIRE* post-processing algorithms - *Activation Engines*

In this paper, in addition to the pre-processing algorithms presented, we further propose a new post-processing technique we name the *Activation Engine*. It consists of applying an optimal thresholding technique at the output of a classification algorithm, aimed to significantly improve prediction accuracy. It is simple, yet capable of improving the accuracy without requiring too much effort on hyper-parameter tuning. This approach can lead to more accurate classification results, using a single ML setup, and without resorting to automated techniques which demand processing power, are time-consuming, and are challenging to implement. Additionally, the *Activation Engine* was implemented using Numpy vectors to allow for efficient post-processing on the edge.

The *Activation Engine* has been developed to improve the testing accuracy of a MLP with a binary *Softmax* output. Algorithm 6.1 determines the *Ref* variable which presents the factor between the occurrence for the two classes in the training set. Algorithm 6.2 may run the *For* Loop (Bordered in bold) from Algorithm 6.1 twice since there might be two activation functions at the output.

In this case, the algorithm swaps the first and second *Softmax* outputs for each iteration. Finally, in case there are two *Activation Engines*, the cluster that has the highest accuracy will be voted based on overall training accuracy and is then applied to the testing data. Furthermore, since there are two activation functions, there should be two possible clusters per function. So, there can be four values from two pairs that have to be considered for voting.

After determining values for array *Th*, the best version of the two thresholds arrays is extracted from the training data, and two indices are computed which point to the better version of the thresholds. Finally, the best threshold version is applied on the testing date resulting in improved prediction accuracy.

Algorithm 6.1 and Algorithm 6.2 as shown in Fig. 7 and 8 respectively, describe the steps and iterations taken to perform the binary classification versions of the *Activation Engine*. Moreover, a tree-based technique is used to perform multi-class classification using this approach. However, it is important to point out that since an output of two *Softmax* functions are used, the variable *Th* is a 2x2 matrix since it has two possible values for every *Softmax* threshold.

The feature extraction algorithms presented have been developed in order to convert time-series of various characteristics and notations into datasets directly applicable to various ML setups, and without resorting to any advanced and stressful pre-processing techniques, which are in terms difficult for inexperienced practitioners. Furthermore, since beginners in ML might find parameter tuning unreachable, the post-processing algorithm presented: The *Activation Engine*, may be used along with a MLP in order to optimize forecast performance without turning to automated ML methods, such as meta-learning or ensemble learning, which require considerable processing power and are time-consuming in nature.

The complexity of the *Activation Engine* can be expressed as shown in Eq. 1 where the factor 2 corresponds to the 2x2 matrix representing the two possible threshold pairs of each *Softmax* activation function. Also the factor 4 corresponds the counting of *True* and *False* values of each binary classification for the two cases of that threshold. the

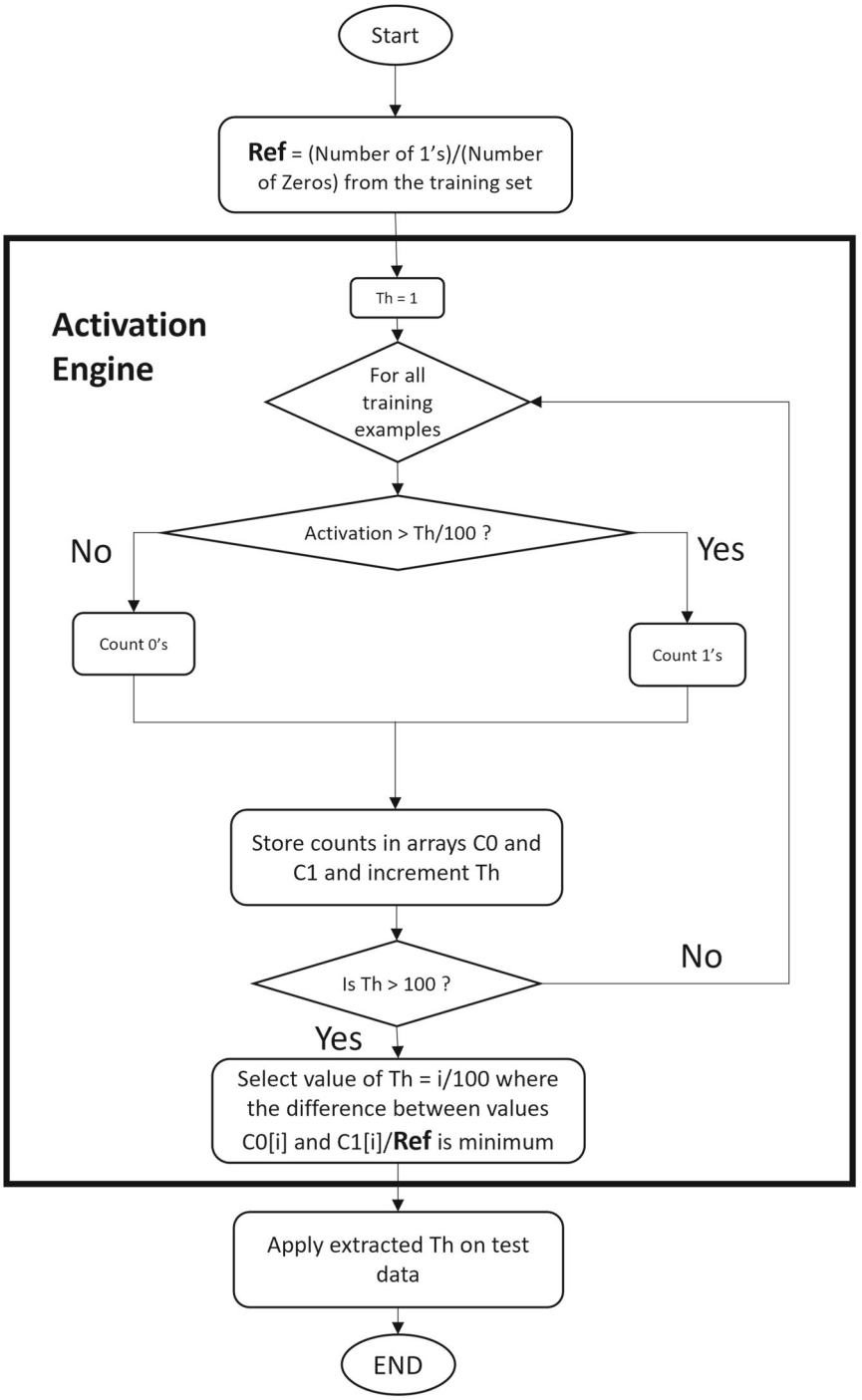

Fig. 7 Algorithm 6.1: *VAMPIRE Activation Engine*

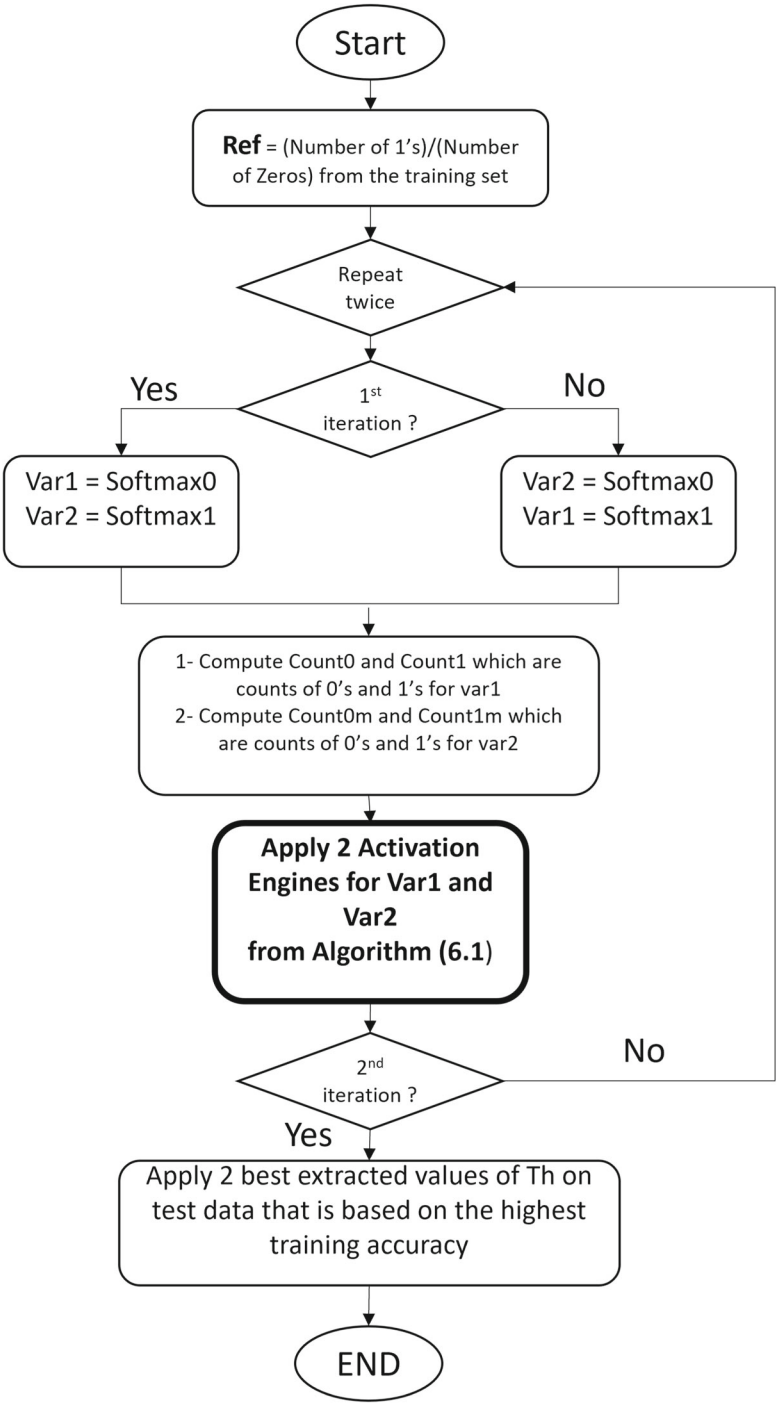

Fig. 8 Algorithm 6.2: Combining two *Activation Engines*

factor 6 represents he extraction of the indices and the evaluation of the *True* and *False* for the optimal threshold extraction. *T1* and *T2* represent the sizes of the training set and test-set respectively. Consequently, *N* corresponds to the value 100 found in Algorithm 6.1 and Algorithm 6.2 from Figs. 7 and 8 which sets the resolution for the threshold used by the *Activation Engine* inference. Additionally, *C* corresponds to the number of class labels used in the dataset which is included since the post-processing algorithm uses a tree-based approach to achieve multi-class classification. Although, setting *C=2* implies that the tree-based method is not used and so Eq. 1 is valid also for binary classification:

$$Complexity = \sum_{0<=i<C-1} (1 - \frac{i}{C}).(2.(N.4.T1 + 2.T1) + 2.6.T1 + 2.6.T2) \quad (1)$$

Furthermore if *N.T1* is large enough, which represents the product of the training set size and threshold precision, Eq. 1 can be approximated in terms of Eq. 2:

$$Complexity = \sum_{0<=i<C-1} (1 - \frac{i}{C}).8.N.T1 \quad (2)$$

5 Data sources

In order to test the performance of the *VAMPIRE* Framework, five data sources taken from five unrelated applications were adopted to perform forecasts using its pre-processing algorithms and *Activation Engines*. Three biomedical time-series datasets (Related to ECG, EEG, and PPG data) were used to implement *VAMPIRE FE1* and *FE2*. Therefore, the biomedical datasets along with the pre-processed datasets were employed to test Rulex and the *Activation Engine* on data that was pre-processed using *VAMPIRE FE1 and FE2* and using different feature extraction techniques from the literature. All tests were performed through a general-purpose laptop as a client and the Raspberry Pi as an AI applications server.

5.1 ECG dataset

For the ECG application, the database used is the MIT-BIH arrhythmia database, [14, 42]. It consists of 48 30 min ECG recordings which pair with 48 annotation files that contain the classification of each sample. Therefore, the*VAMPIRE FE1* was used which deals with fully annotated datasets. In this dataset, there are 20 arrhythmia classes, however, we have chosen to consider five that have enough samples for feasible ML training and testing. Also, we have applied multiple forecasts for the three most dominant classes and with four most dominant classes to demonstrate that the algorithm is more accurate in case a larger number of samples is available. Therefore, a collective subset of 35989 samples was used in the forecasts having 73 features.

5.2 EEG dataset

The EEG recordings were taken from the CHB-MIT scalp EEG database [14] which consists of records from pediatric subjects with intractable seizures. Moreover, in accordance with the previous section, *VAMPIRE FE1* was implemented to generate the features. The dataset analyzed in this work contains records from 22 subjects, sampled at a rate of 256 samples per second with 16-bit resolution. Most files contain 23, 24, or 26 EEG signals. As a whole, a random subset from all patient was used with 17978 records having 180 features for the train/test split.

5.3 PPG dataset

The PPG data was taken from part of the predicting cognitive fatigue with Photoplethysmography project [43] and is taken from the Kaggle dataset repository [44]. The recordings consist of participants taking a 22 hour-shifts of gaming. These are used to classify the Stanford Sleepiness Scale (1-7). However, due to the ambiguity of the data, since it is a self-assessment and not a clinical diagnosis, we have split the classes into two class groups *True* and *False*, which signify *Sleepy* or *Not Sleepy*. Furthermore, the classification is taken over long periods of time, so, in order to classify the gamer's fatigue level in real-time, *VAMPIRE FE2* was employed. Two gamers records were chosen for the present ML workflow with a total of 31355 instances with 64 features.

5.4 Radar dataset

The radar dataset from [45] consists of features extracted from a double FFT applied to radar measurements. These include statistical quantities of this signal such as *mean*, *STD*, *variance*, *range*, and others.

There are four classes, where there is one for *Humans* and three others for *Vehicles*. A MLP with an *Activation Engine* at its output with K-folder cross-validation splits was implemented, which achieves high forecast accuracy. This dataset consists of 120 records equally split over its labels having 10 features.

5.5 Activity dataset

The activity detection dataset from [46] consists of features taken from the accelerometer and gyroscope embedded in a smartphone. These include statistical quantities of this signal such as *mean*, *STD*, *variance*, and others. Six classes describe the activity performed by a subject such as *Laying*, *Standing*, *Sitting*, *Walking*, *Walking Upstairs*, and *Walking Downstairs*. This dataset consists of 561 pre-processed features including a user field while containing 7352 samples.

6 Experimental results

All related experimental results will be presented in detail in this section. The biomedical time-series described in the previous section were used to test *VAMPIRE* framework's performance where the recordings were pre-processed to generate corresponding datasets, and were either applied using Rulex in a client/server arrangement on the Raspberry Pi, or by remotely employing a MLP with an *Activation Engine* on the same board. In case of the EEG and PPG datasets, both of *VAMPIRE*'s pre-processing algorithms and *Activation Engines* were employed, in addition to applying the EEG and ECG datasets using Rulex on the Raspberry Pi.

Concerning the pre-processed datasets, both the activity detection and radar classification datasets were implemented with a MLP having an *Activation Engine* at its output. Furthermore, the activity detection dataset was applied using Rulex on the edge. In every case where ML was applied on the edge using Rulex running on the Raspberry Pi, holdout validation was applied while implementing a K-folder setup for the EEG, PPG, and radar forecasts. Additionally, the power consumption and iference time will be evaluated by comparing with the performance of other ML setups.

6.1 Results for ECG forecasts

In the case of the ECG application, we used 73 Features for the generated dataset, which were extracted using *VAMPIRE FE1* from the MIT BIH arrhythmia database. The features consist of the *mean*, *variance*, *STD*, *median*, *minimum*, and a *maximum* of the variables *a1*, *b1*, *Sum1*, *Diff1*, the *integral* of the spike, the *area* of the spikes, and the same statistical functions for the sets of *a2*, *b2*, *Sum2*, and *Diff2* as presented in Fig. 5.

We have applied ML algorithms decision trees (DT), and SVM in their default format using the Rulex ML platform running on the Raspberry Pi in a client/server setup [47]. Three arrangements were adopted with 3, 4, and 5 classes which take the more dominant classes in the dataset into consideration. Just five classes have been used in the simulations which are the forward-slash / which represents a Paced Beat, *N* for Normal, *L* and *R* which are the left and right bundle branch respectively, and *V* which means Premature Ventricular Contraction.

VAMPIRE FE1 requires a large time-series dataset as input, mainly since it transforms the input data from one domain to another, which is then used for training and testing. However, we were still able to achieve highly accurate forecasts using classical ML algorithms.

Figure 9a, b represent the accuraies for the training and testing using DT respectively, however, due to the lack of samples in classes A and V, the accuracy is less than for the other classes in the testing. In the case of 5 classes, the accuracies for decision trees (DT) and SVM in testing are 95.55% and 96.24% respectively.

In Fig. 9c, d the accuracies for both training and testing using DT with 4 classes is presented, respectively. DT and SVM perform more accurately in the case of 4 classes. class V which has the worst performance has been omitted. The overall accuracies in the case of DT and SVM testing are 96.60% and 97.62% respectively.

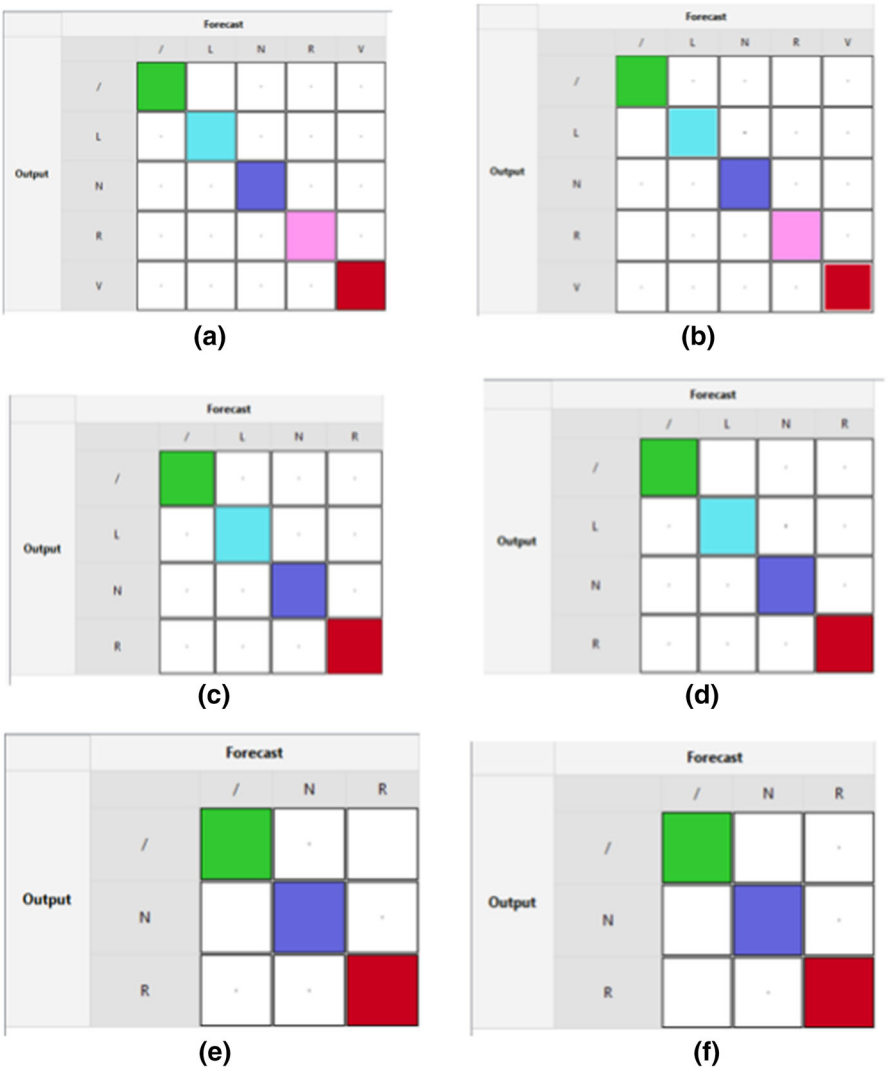

Fig. 9 Training and testing accuracies for ECG forecasts:(**a**) and (**b**) DT training and testing accuracy with 5 classes respectively. (**c**) and (**d**) DT training and testing accuracy with 4 classes respectively.(**e**) and (**f**) SVM training and testing accuracy with 3 classes respectively

Figure 9e, f present the training and testing forecast accuracy for SVM using a 3-class arrangement. For all algorithms using the same datasets, the performance improved in terms of accuracy as the fewer dominant classes were omitted. This is due to the lack of samples where the algorithm does not generate enough information in the features for accurate prediction. The overall accuracies in the case of 3 classes for DT and SVM are 97.99% and 98.73% respectively.

In all the forecasts, the data which were used for prediction were taken from the files which are dominated by the targeted classes. For instance, if there are the files

Table 1 Five classes for ECG classification using Rulex on the edge

Classes	/	L	N	R	V
SVM	99.57%	94.61%	98.80%	96.88%	89.11%
Decision Trees	98.19%	93.82%	97.10%	95.12%	89.98%

ECG Classification using Rulex running on the Raspberry Pi with 5 classes

Table 2 Four classes for ECG classification using Rulex on the edge

Classes	/	L	N	R
SVM	99.24%	94.77%	98.93%	97.14%
Decision Trees	99.24%	94.45%	97.20%	96.40%

ECG Classification using Rulex running on the Raspberry Pi with 4 classes

Table 3 Three classes for ECG classification using Rulex on the edge

Classes	/	L	N
SVM	99.86%	99.82%	95.93%
Decision Trees	99.57%	98.73%	95.80%

ECG Classification using Rulex running on the Raspberry Pi with 3 classes

that predict classes *F* or *S* in a large enough number, these files were omitted to focus on the targeted classes without adding any excessive data. So, even classes containing labels such as *N* which represent a normal beat and exist in almost all files, the samples are only taken from the files which are rich in the target classes.

This methodology proved to be fair since it generates very good results consistently as the number of classes varied. This is also valid for various ML algorithms. A complete set of results for forecast accuracy using the described Feature Extraction method is presented in Tables 1, 2, and 3.

In [48] and ensemble learning approach was applied to the same ECG dataset which is used in this paper and another much smaller data source with the aim of improving prediction accuracy. The system described is not fully automated since it requires careful tuning while it relies on grid-search to tune an SVM from the ensemble. The best accuracy achieved was realized by an MLP which is 98.06% while the overall ensemble accuracy was recorded at 97.78%. However, the best accuracy was taken without considering additional forecasts in a K-folder setup which provides more reliable and unbiased results. In [49] a machine learning framework dedicated to ECG classification is presented which partly used the dataset used in this paper found in [13]. Even though an overall accuracy of 98.2% was achieved, this is due to the addition of synthetic data which might have caused the ML model to over-fit to the test-et. Furthermore, when using real world testing data, the accuracy dropped to 81% using Random Forest and with a two-second window 85% using a five-second window for a four-class forecast.

Table 4 EEG forecast using Rulex on the edge

Classes	Non-Seizure	Seizure
LLM	85.087%	85.992%
SVM	88.289%	81.493%
DT	84.355%	84.05%

EEG Classification using Rulex running on the Raspberry Pi

6.2 Results for EEG forecasts with *VAMPIRE FE1* and Rulex

In the second forecast case, we have applied *VAMPIRE FE1* on time-series related to epileptic seizure Detection using EEG measurements, which are taken from subjects with intractable seizures. Forecasts were applied using Rulex ML software running on the Raspberry Pi in and edge computing setup. However, unlike the previous set of ECG Features, the total number of features for the EEG applications was multiplied by three, since brainwaves are frequency-dominant signals unlike voltage-dominant signals occurring in ECG waveforms. So, the EEG recording was split into three frequency bands using digital band-pass filters and the same number of features was generated for each frequency band.

The forecast accuracies using the Rulex implementation are presented in Table 4, where SVM, DT, and logic learning machine (LLM) algorithms were applied to the extracted features which provide accurate results across all the adopted algorithms.

6.3 EEG forecasts with *VAMPIRE FE1* and *Activation Engines*

As for the prediction results on the EEG data using *VAMPIRE*'s *Activation Engine* on the Raspberry Pi, a Tensorflow implementation of a MLP was build using Keras. It consists of a MLP trained using back-propagation. The MLP has an input layer with 219 inputs and a Rectified Linear Unit (*Relu*) activation function. The is proceeded by four hidden layers alternating between *Sigmoid* and *Relu* until it reaches 2 *Softmax* functions for a binary output. The corresponding transfer functions for the *Sigmoid*, *Relu*, and *Softmax* functions can be found in Eqs. 3–5:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

$$Relu(x) = \max(0, 1) \tag{4}$$

$$Softmax(xi) = \frac{e^{-xi}}{\sum_{j=1}^{k} e^{xj}} \tag{5}$$

The training using Tensorflow was run for 60 epochs with a K-folder split with $K = 5$. As for the proposed MLP with *Activation Engines*, the overall testing accuracy was 85.32% with the 2 output classes being predicted with accuracies of 87.58% and 83.07%. In [50] forecasts were applied partly on the same EEG dataset used in this paper using classical ML algorithms and fuzzy-based techniques where an accuracy of

85.6% was achieved using a deep CNN (which is computationally expensive) and 90% using Sparse Extreme Learning Machine [51]. However, the authors use a subset of five patients from the CHB-MIT dataset whereas in this paper a completely random subset from all patients is used which helps to avoid over-fitting and allows for more reliable generalization, while remaining competitive. Furthermore, the MLP was tested after removing the *Activation Engines* and relying on a one-vs-all approach where the new method outperforms the classical technique by 3% which has an overall accuracy of 82.2%.

6.4 PPG forecasts with *VAMPIRE FE2* and *Activation Engines*

Regarding the Sleepiness PPG recording dataset, we considered two subjects out of the existing five, specifically gamers 1 and 3. The 7 sleepiness classes were grouped into parent classes by equally distributing them into *Sleepy* and *Not Sleepy*. So, in this binary classification arrangement, we have made forecasts using the Python Tensorflow library through a MLP. Also, as in the EEG application, the *Activation Engine* technique was applied at the output of the MLP to improve testing performance.

VAMPIRE FE2 described in Algorithm 5 was used to extract the features from the PPG dataset, considering that the time-series is patient-annotated with no expert medical assessment present. Moreover, the *Activation Engine* described in Algorithm 6.1 and Algorithm 6.2 from Figs. 7 and 8 was applied at the output of a MLP to further improve accuracy on the edge.

The MLP implemented using Keras has an input layer with 63 inputs and a *Relu* activation function. The is proceeded by three hidden layers having a *Relu* as an activation function and has two *Softmax* binary outputs. We have used the above MLP arrangement with hidden layers having a size 70, and the algorithm was run for 20 epochs using Tensorflow.

For gamer 3, after 5 epochs,the testing accuracy as reported by the *Activation Engines* with a k-folfer setup having $K = 5$ was 89.23% and 79.77% for *Sleepy* and *Not Sleepy* classes, respectively. As for the overall accuracy, it was equal to 83.59% in general.

As for gamers 1 and 3 combined, after 5 epochs, the testing accuracy as reported by the *Activation Engine* arrangement with a K-folder setup having $k = 5$ was 85.43% and 76.76% for *Sleepy* and *Not Sleepy* classes, respectively. As for the overall accuracy, it was equal to 78.46% in general. Additionally, the MLP was test with no *Activation Engines* as its output where the performance degraded without the post-processing block having a unbalanced output testing accuracy of 90.93% and 38.58% for the two classes.

6.5 Urban classification using *Activation Engines*

A multi-class dataset for urban classification via radar was used to test the *Activation Engine* setup in Python in the case of four available classes. These are *Humans*, *Cars*, *Trucks*, and *Motorcycles*.

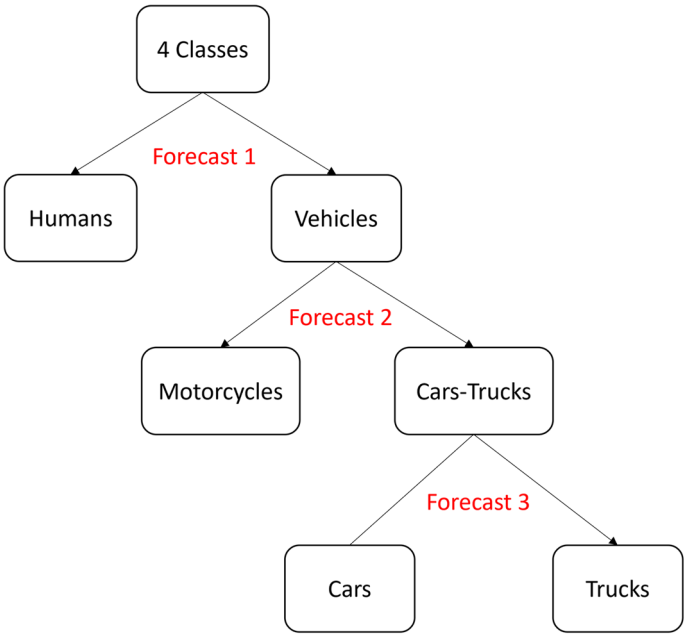

Fig. 10 Algorithm 7: Applying tree-based structure to perform classification on urban dataset

Table 5 Radar classification accuracy of pedestrians and vehicles using *VAMPIRE* on the edge with comparisons

Algorithms	Classic MLP	*VAMPIRE*	[52]	[53]
K-folds	Yes	Yes	No	No
Humans	94.78%	93.60%	100%	100%
Vehicles	–	99.40%	80%	82.24%
Motorcycles	41.6%	99.40%	83.33%	75%
Cars & Trucks	–	94.85%	80%	88.89%
Cars	97.22%	83.23%	75%	80%
Trucks	22.71%	93.75%	83.33%	100%

Urban classification using *Activation Engines*

The tree-based structure used in Algorithm 7 which is shown in Fig. 10, works similarly to the one-vs-one method for multi-class classification except that it combines child classes into parent classes, and once a child class is reached, its accuracy is multiplied with the prediction accuracies of the parent classes. With this technique, each forecast accuracy at a given level should be multiplied by the accuracy of their preceding parent forecasts.

This tree as presented in Algorithm 7 from Fig. 10, applies four forecasts using MLP with the *Activation Engines* method described in Algorithm 6.1 and Algorithm 6.2 from Figs. 7 and 8. The MLP consists of four *Relu* layers and a one *Softmax* output layer. After running the algorithms in K-folder arrangement with $K = 15$, the following results were obtained in Table 5 where they were compared with results from

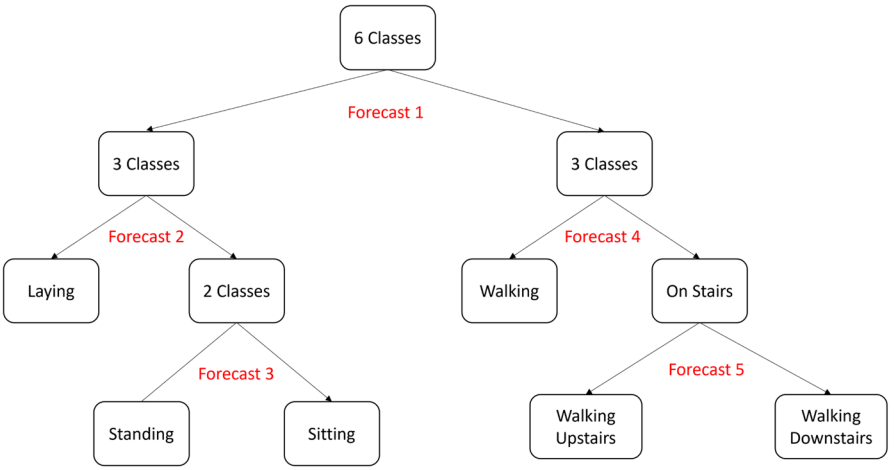

Fig. 11 Algorithm 7: Applying tree-based structure to perform classification on the human activity detection dataset

the literature where the dataset was first tested. Although the previous results don't use K-folders to generate reliable results in terms of test-set-variability, *VAMPIRE* was able to achieve an overall classification accuracy with an 8.8% improvement over previous techniques.

In the results from *VAMPIRE*, the *Motorcycles* detection accuracy shown is multiplied by that of *Vehicles*. Also, the accuracy for *Cars* and *Trucks* classes was multiplied with the *Vehicles* class accuracy, in addition to being multiplied with the accuracy of the combined *Cars-Trucks* parent class. Also, after removing the post-processing block and applying the MLP model using a one-vs-all approach, the performance degraded with the classic MLP which highlights the significance of the *Activation Engine* applied at the output.

6.6 Human activity classification using *Activation Engines*

A multi-class dataset for human activity detection using smartphone sensors was used which has six classes that were also predicted using the same approach. As a first step, the classes were allocated to two parent groups, *Sitting*, *Laying*, and *Standing* as the first, and the three classes related to *Walking* in the other. Moreover, these two classes were applied using a ML algorithm, and were subsequently split into sub-classes following the same procedure as in the radar classification application, and are similarly presented in Fig. 11. Again, as pointed out in the previous application, each accuracy achieved through a child forecast is multiplied with the preceding parent class accuracy. Forecasts were performed using a MLP with of four *Relu* layers and a two *Softmax* outputs functions having two *Activation Engines* connected at the MLP output. Furthermore, the human activity detection dataset was applied on the edge using Rulex with multiple ML algorithms where the accuracies for LLM, KNN, SVM and VAMPIRE are provided in Table 6. Moreover, the MLP was tested in a one-vs-all

Table 6 Human activity forecasts using Rulex and *VAMPIRE* on the edge

Algorithm	LLM	KNN	SVM	[54]	Classic MLP	*VAMPIRE*
User field	No	No	No	Yes	No	No
Laying	100%	100%	100%	100%	89.11%	100%
Standing	86.80%	94.40%	98.40%	96%	93.82%	99.81%
Sitting	88.48%	91.86%	97.63%	97%	99.92%	99.81%
Walking	94.74%	100%	100%	99%	98.57%	100%
Walking Upstairs	89.89%	99.47%	100%	100%	99.36%	100%
Walking Downstairs	82.93%	100%	100%	99%	89.01%	100%

Human activity classification using Rulex

Table 7 Inference and specifications using *VAMPIRE* and other IoT setups on the edge

Board	Cores	Frequency	RAM	GPU	ML setup	Inference (msec.)
Raspberry Pi 3B+	4	1.4GHz	1GB	NO	PPG	**3.1**
Raspberry Pi 3B+	4	1.4GHz	1GB	NO	EEG	**6.1**
Raspberry Pi 3B+	4	1.4GHz	1GB	NO	Activity	25
MacBook	4	2.7GHz	8GB	NO	AlexNet	29
Jetson TX2	6	2GHz	8GB	YES	AlexNet	13.5
FogNode	4	3.2GHz	32GB	NO	AlexNet	27
Raspberry Pi [34]	4	0.9GHz	1GB	NO	SqueezeNet	2080

Inference time for IoT boards

setup without an *Activation Engines* (with K equal to 5) to validate its effectiveness in tuning the model. Moreover, the forecasts applied in [54] are included in Table 6 to compare the accuracies achieved in every case.

In [54], an KNN ensemble classifier with 30 instances was used to classify human activity using the same dataset where an overall accuracy of 98.6% was achieved. This accuracy is substantially less than that achieved using both SVM in the Rulex client/server forecast, and when using a MLP with a *VAMPIRE Activation Engine* edge setup, as shown in Table 6. In the original dataset, a user field was included which identifies the person who is holding the smartphone. However, in this experiment the field was removed for more generality where *VAMPIRE* outperforms the literature with a lesser degree of information.

6.7 Latency and power consumption on the edge

To evaluate the performance of *VAMPIRE* on the Raspberry Pi and to ensure its efficiency in addition to providing accurate predictions, a comparison has been made with a laptop (using the same data sources) and other edge setups described in the literature. Therefore, in [34], various ML libraries have been tested on multiple IoT modules including the Raspberry Pi. Consequently, Training was applied outside the edge node and inference was performed on each board. Table 7 presents a comparison between

Table 8 Power consumption comparison between Raspberry Pi 3B+ and a MSI laptop using *VAMPIRE*'s *Activation Engine*

Platform	Dataset	Kfolds	GPU	DC Power (Wh)	AC Power (Wh)	inference (msec.)
Rasp-Pi	Activity	1	NO	1.28	1.63	25
Rasp-Pi	EEG	5	NO	7.1	10.43	6.1
Rasp-PI	PPG	5	NO	0.76	1.06	0.1
MSI-PC	Activity	1	NO	–	3.2	3.1
MSI-PC	EEG	5	NO	–	13.78	1.4
MSI-PC	PPG	5	NO	–	5.2	1.5
MSI-PC	Activity	1	Yes	–	0.92	4
MSI-PC	EEG	5	Yes	–	8.38	0.24
MSI-PC	PPG	5	Yes	–	2.29	0.2

Power consumption of the Raspberry Pi and MSI laptop running *VAMPIRE*

VAMPIRE on the Raspberry Pi and the additional setups. As shown, *VAMPIRE*'s optimized *Activation Engine* post-processing block outperforms the other setups presented in the literature where the inference/task is presented along with the specifications of all hardware platforms. Furthermore, in [55] authors present performance measurements taken from the Raspberry Pi on various classification and regression datasets using different ML classifiers. During the classifications, the inference time on the board depend on the dataset where these inferences are compared with those applied on a personal computer (PC). the ratio between the Raspberry Pi's inference divided by the PC's inference varies between 175 and 306 depending on the dataset. However, using *VAMPIRE* on the Raspberry pi, this ratio varies only between 2 and 8 without a GPU, and between 6.25 and 25.4 with a GPU, as shown in Table 8. Additionally the Raspberry pi's performance using *VAMPIRE* is compared with that of the MSI laptop (described in sect. 2.3) where the same datasets were used for the comparison in terms of power consumption (For training and testing combined) and inference time. Also, when using the laptop two forecasts were applied per dataset where the first was performed without including the onboard GPU and the second where a GPU was used to optimize the latency and power consumption. Table 8 illustrates the results for the EEG, activity detection and PPG datasets using *VAMPIRE*. As illustrated, when running training and inference, the Raspberry Pi outperformed the case with the laptop running without GPU optimization by 3-4 orders of magnitude regarding the energy consumption. Moreover, the board remains competitive with the PC even when it is optimized using the 2060 Nvidia GPU which is designed to increase the power and decrease the overall energy in ML workflows.

7 Conclusion

To summarize, this paper presents the *VAMPIRE* framework, which implements novel pre-processing algorithms and introduces *Activation Engines* that can extract features from any time-series data and can be applied to any other type of application, through

the *Activation Engine* concept. The pre-processing algorithms consist of converting time-varying signals into the proposed D-Domain, which is a V.Hz co-representation the original signal and its FFT. The novel waveform is used to derive statistical information obtained from the shape of the rectified time-series. Furthermore, the pre-processing can be applied in two modes: either using an annotated time-series where a classification exists for every event in the signal, or when the time-series is poorly annotated where the classification is taken for a large span of time. In the second case, an adaptive algorithm was implemented to detect an event in real-time by continuously classifying the current segment of a time-series before producing the V.Hz waveform and extracting statistical features. Also, the paper introduces the *Activation Engine* which is a post-processing block which improves testing accuracy by optimizing the classification threshold at the output of an MLP based on the training performance. The *Activation Engine* achieves high testing accuracy by relying on an optimal threshold extraction method that is based on the training accuracy by employing a pair of *Activation Engines* and relying on a clustering approach. Additionally, *VAMPIRE*'s core pre-processing and post-processing operations were implemented in Python Numpy vectors in order to the improve performance considerably. Also, every ML forecast was carried out in an IoT setting, either using Rulex running on the Raspberry Pi and in a client/server setup, or by performing forecasts using a MLP with an *Activation Engine* running on the Raspberry Pi remotely through an SSH tunnel.

In regards to the pre-processing algorithms developed in this paper, experiments were carried out by generating three datasets from biomedical time-series automatically before applying ML forecasts. The forecasts were performed on these datasets in an edge computing setup where our results confirm that high accuracy was achieved in every experiment. Furthermore, in the cases of the EEG and PPG datasets, in addition to *VAMPIRE FE1* and *VAMPIRE FE2*, *Activation Engines* were employed. In these two cases, the novel post-processing method lead to better testing accuracy than in training due to the influence of the threshold clustering technique of Algorithms 6.1 and 6.2. Also, the forecast accuracy for the two already pre-processed datasets is significantly higher than the accuracies achieved in the original publications. This is the case, again, thanks to the addition of *Activation Engines* at the *Softmax* outputs of the MLP.

Feature extraction was performed on biomedical time-series such as ECG, EEG, and PPG signals where a dataset can be generated in an automated manner and the operator only needs to remove noise and offset from the input signal, before applying ML algorithms on the Raspberry Pi. Also, two pre-processed datasets related to urban classification and human activity detection were also used to perform ML forecasts on the edge.

Furthermore, the framework's accuracy, latency and power consumption on the Raspberry Pi was also evaluated where the *Activation Engine* outperforms most previously published results in terms of inference speed and accuracy. Moreover, regarding the power consumption of the Raspberry Pi, a comparison with a PC was employed where the board consumes less energy than the laptop in case the GPU is not included in the training and remains competitive in comparison with the GPU being used to optimize the ML workflow. In the end, accurate results across all experiments prove that *VAMPIRE* is easy to implement and demonstrate the robustness of the framework by

competing with different techniques from the literature (Which are compute-expensive in most cases) while maintaining generality and so it can be used virtually in any ML application.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s00607-022-01096-z.

Acknowledgements Not applicable.

Funding Open access funding provided by Università degli Studi di Genova within the CRUI-CARE Agreement.

Appendix A: Further explaining Algorithm 5 by example

This appendix aims to provide a step-by-step application of Algorithm 5 from *VAMPIRE FE2* in order to clarify the pseudo-code by supplying real data taken from the Python interpreter and to clear up any ambiguity. Initially, in the array below, we can find all the zero-crossings of an input signal. To determine the crossings, the minimum local maxima and the maximum local minima are used as reference points to detect crossings rather than zero. The name of the array from Algorithm 5 to be computed is *Cross*.

Cross:
*[**100, 169, 201, 269, 293, 335, 361, 379**, 405, 484, 506, 576, 604, 682, 708, 777, 811, 880, 909, 990, 1022, 1081, 1116, 1182, 1211, 1279, 1309, 1373, 1402, 1475, 1503, 1570, 1600, 1683, 1705, 1779, 1809, 1884, 1914,…]*

As shown in Cross for every Cross[i]:
Cross [0] = 100 Cross [2] = 201
Cross [1] = 169 Cross [3] = 269 …

In the array *Cross*, *Cross[i]* represents a Negative-to-Positive crossing *(N-to-P)* if the index is zero or is an even number, and *Cross[i]* represents a Positive-to-Negative crossing *(P-to-N)* if the index i is odd. For every pair of crossings, the maximum voltage should be computed. The array *Patt* is used for storing the maximum Value in the signal between every *N-to-P* and *P-to-N* pair. *Patt []* represents the maximum voltage between every crossing pairs:

Patt:
*[**57.878, 64.533, 64.724, 3.538**, 42.306, 45.081, 51.199, 49.983, 42.531, 48.166, 28.105, 39.79, 53.489, 62.38, 61.229, 54.553, 60.818, 56.603, 54.73, 51.671,*

57.029, 66.834, 65.955, 32.123, 34.94, 30.187, 55.639, 43.408, 49.639, 62.696, 59.278, 64.183, 48.3, 65.897, 142.026, 72.131, 511.249, 338.739, 367.466,10.085, 101.134, 105.386, 47.537, 28.63, 52.903, 43.38]

Consequently, after computing the maximum voltages between crosses in the *Patt []* array, the window-size of every beat or repeating series of crossings needs to be determined. In order to do this, we can rely on the mean for every possible window size. The array *Mean []* below contains all the computed means:

Mean:
[0.0, 0.0, 0.0, ***47.668***, textit43.775, 38.912, 35.531, 47.142, 47.198, 47.97, 42.196, 39.648, 42.388, 45.941, 54.222, 57.913, 59.745, 58.301, 56.676, 55.956, 55.008, 57.566, 60.372, 55.485, 49.963, 40.801, 38.222, 41.044, 44.718, 52.846, 53.755, 58.949, 58.614, 59.414, 80.102, 82.089, 197.826, 266.036 …]

For example:
or Window = 4
Mean [3] = Mean of: Patt [3], Patt [3-1], Patt [3-2], Patt [3-3]
Mean [3] = Mean of: Patt [3], Patt [2], Patt [1], Patt[0]
Mean [3] = (57.878 + 64.533 + 64.724 + 3.538)/ (4 = Window Size)
Mean [3] = 47.668

Therefore, after computing the mean for some window size, as in Algorithm 5, the standard deviation is computed and if *STD* is higher a certain *Threshold*, the *Window* variable is incremented by one. This will go on recursively until the correct *Window* size is determined. It should also be pointed out that multiples of a correct window-size hold and may incorrectly detect *Window*. In case *Window = 3*, Windows of sizes 2, 4, and 5 will give a higher *STD* for *Mean []*, which implies that this is an incorrect *Window*.

For example:

$$\textit{Mean (Window size of 3)} = \textit{Mean (Window size of 3*N)} \tag{A1}$$

If we choose for *Window* a starting point of 4, instead of starting at 2, and the correct Window is 3, *Window* may be detected as 6. This is true since the mean of array *Mean []* with *Window* size 3 is almost equal to the array *Mean []* of *Window* size *3*N*. A description of this is presented in Eq. (A1).

After detecting *Corss* and *Window* for that series, *Crosspx* is generated from the collection of *Cross* pairs of size *2*Window*. Algorithm 5 demonstrated how to collect the first and last crossings from *Cross* into *Crosspx* of size *Window*2*. The *Crosspx []* array is used to transform the signal into the D-Domain which in terms is used to extract the features for ML classification.

Crosspx:
*[****100, 379****, 405, 777, 811, 1182, 1211, 1570, 1600, 1978, 2012, 2329, 2365, 2726, 2761, 3097, 3137, 3508, 3541, 3853, 3886,...]*

References

1. Muselli M (2005) Switching neural networks: A new connectionist model for classification. In: Neural Nets, pp. 23–30. Springer
2. Daher AW, Rizik A, i, M., Chible H, Caviglia DD (2020) Porting rulex machine learning software to the raspberry pi as an edge computing device. In: International Conference on Applications in Electronics Pervading Industry, Environment and Society, pp. 273–279. Springer
3. Muselli M, Ferrari E (2011) Coupling Logical Analysis of Data and Shadow Clustering for Partially Defined Positive Boolean Function Reconstruction. IEEE Trans Knowl Data Eng 23(1):37–50. https://doi.org/10.1109/TKDE.2009.206
4. Daher AW, Rizik A, Randazzo A, Tavanti E, Chible H, Muselli M, Caviglia DD (2020) Pedestrian and multi-class vehicle classification in radar systems using rulex software on the raspberry pi. Appl Sci 10(24):9113
5. Muselli M (2012) Extracting knowledge from biomedical data through Logic Learning Machines and Rulex. EMBnet. J 18(B):56–58. https://doi.org/10.14806/ej.18.B.549
6. Fister D, Fister I, Jagrič T, Fister I, Brest J (2018) A novel self-adaptive differential evolution for feature selection using threshold mechanism. In: 2018 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 17–24. https://doi.org/10.1109/SSCI.2018.8628715
7. Jovic A, Kukolja D, Friganovic K, Jozic K, Car S (2017) Biomedical time series preprocessing and expert-system based feature extraction in MULTISAB platform. In: 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 330–335 . https://doi.org/10.23919/MIPRO.2017.7973444
8. Venkatesan C, Karthigaikumar P, Paul A, Satheeskumaran S, Kumar R (2018) Ecg signal preprocessing and svm classifier-based abnormality detection in remote healthcare applications. IEEE Access 6:9767–9773
9. Kalayci T, Ozdamar O (1995) Wavelet preprocessing for automated neural network detection of EEG spikes. IEEE Eng Med Biol Magazine 14(2):160–166. https://doi.org/10.1109/51.376754
10. Khalid S, Khalil T, Nasreen S (2014) A survey of feature selection and feature extraction techniques in machine learning. In: 2014 Science and Information Conference, pp. 372–378. https://doi.org/10.1109/SAI.2014.6918213
11. Karakatič S (2020) EvoPreprocess–Data Preprocessing Framework with Nature-Inspired Optimization Algorithms. Math 8(6):900. https://doi.org/10.3390/math8060900
12. Wang J, Liu P, She MFH, Nahavandi S, Kouzani A (2013) Bag-of-words representation for biomedical time series classification. Biomed Signal Process Control 8(6):634–644. https://doi.org/10.1016/j.bspc.2013.06.004
13. Moody GB, Mark RG (2001) The impact of the mit-bih arrhythmia database. IEEE Eng Med Biol Magazine 20(3):45–50
14. Goldberger AL, Amaral LA, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE (2000) Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. circulation 101(23):e215–e220
15. Jin Lp, Dong J (2016) Ensemble deep learning for biomedical time series classification. Computational intelligence and neuroscience **2016**
16. Zhang Jw, Wang Lp, Liu X, Zhu Hh, Dong J (2010) Chinese cardiovascular disease database (ccdd) and its management tool. In: 2010 IEEE International Conference on BioInformatics and BioEngineering, pp. 66–72. IEEE
17. Fiterau M, Bhooshan S, Fries J, Bournhonesque C, Hicks J, Halilaj E, Re C, Delp S (2017) ShortFuse: Biomedical Time Series Representations in the Presence of Structured Information. In: Machine Learning for Healthcare Conference, pp. 59–74. PMLR. http://proceedings.mlr.press/v68/fiterau17a.html
18. Rasp S, Lerch S (2018) Neural networks for postprocessing ensemble weather forecasts. Monthly Weather Review 146(11):3885–3900
19. Tanwani AK, Afridi J, Shafiq MZ, Farooq M (2009) Guidelines to select machine learning scheme for classification of biomedical datasets. In: European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, pp. 128–139. Springer
20. Thornton C, Hutter F, Hoos HH, Leyton-Brown K (2013) Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 847–855

21. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1):10–18
22. Feurer M, Klein A, Eggensperger K, Springenberg JT, Blum M, Hutter F (2019) Auto-sklearn: efficient and robust automated machine learning. In: Automated Machine Learning, pp. 113–134. Springer, Cham
23. Zhang Q, Hu W, Liu Z, Tan J (2020) TBM performance prediction with Bayesian optimization and automated machine learning. Tunnelling and Underground Space Technol 103:103493. https://doi.org/10.1016/j.tust.2020.103493
24. Guo C, Pleiss G, Sun Y, Weinberger KQ (2017) On Calibration of Modern Neural Networks. In: International Conference on Machine Learning, pp. 1321–1330. PMLR. http://proceedings.mlr.press/v70/guo17a.html
25. Ward L, Agrawal A, Choudhary A, Wolverton C (2016) A general-purpose machine learning framework for predicting properties of inorganic materials. npj Comput Materials 2(1):1–7
26. Al-Khafajiy M, Webster L, Baker T, Chible H, Waraich A (2018) Towards fog driven IoT healthcare: challenges and framework of fog computing in healthcare. In: Proceedings of the 2nd international conference on future networks and distributed systems, pp. 1–7. Springer
27. Cecilia JM, Cano JC, Morales-García J, Llanes A, Imbernón B (2020) Evaluation of clustering algorithms on GPU-based edge computing platforms. MDPI sens 20(21):6335
28. Lapegna M, Balzano W, Meyer N, Romano D (2021) Clustering Algorithms on Low-Power and High-Performance Devices for Edge Computing Environments. MDPI sens 21(16):5395
29. Novac PE, Hacene GB, Pegatoquet A, Miramond B, Gripon V (2021) Quantization and Deployment of Deep Neural Networks on Microcontrollers. MDPI sens 21(9):2984
30. Kanawaday A, Sane A (2017) Machine learning for predictive maintenance of industrial machines using IoT sensor data. In: 8th International Conference on Software Engineering and Service Science (ICSESS), pp. 87–90. IEEE
31. Canedo J, Skjellum A (2016) Using machine learning to secure IoT systems. In: 14th annual conference on privacy, security and trust (PST), pp. 219–222. IEEE
32. Hodo E, Bellekens X, Hamilton A, Dubouilh PL, Iorkyase E, Tachtatzis C, Atkinson R (2016) Threat analysis of IoT networks using artificial neural network intrusion detection system. In: International Symposium on Networks, Computers and Communications (ISNCC) , pp. 1–6.IEEE
33. Vujović V, Maksimović M (2014) Raspberry Pi as a Wireless Sensor node: Performances and constraints. In: 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1013–1018. https://doi.org/10.1109/MIPRO.2014.6859717
34. Zhang X, Wang Y, Shi W (2018) pCAMP: Performance Comparison of Machine Learning Packages on the Edges. https://www.usenix.org/conference/hotedge18/presentation/zhang
35. Čeleda P, Velan P, Král, B, Kozák O (2019) Enabling SSH Protocol Visibility in Flow Monitoring. In: Symposium on Integrated Network and Service Management (IM), pp. 569–574, IFIP/IEEE
36. Murshed MS, Murphy C, Hou D, Khan N, Ananthanarayanan G, Hussain F (2021) Machine learning at the network edge: A survey. In: ACM Computing Surveys (CSUR), 54(8), 1–37. ACM New York
37. Sudharsan B, Breslin J, Ali MI (2020) Edge2train: A framework to train machine learning models (svms) on resource-constrained iot edge devices. In: Proceedings of the 10th International Conference on the Internet of Things, pp. 1–8
38. Sudharsan B, Breslin J, Ali MI (2020) RCE-NN: a five-stage pipeline to execute neural networks (cnns) on resource-constrained iot edge devices. In: Proceedings of the 10th International Conference on the Internet of Things, pp. 1–8
39. Meisenbacher S, Turowski M, Phipps k, Ratz M, Muller D, Hagenmeyer V (2022) Review of automated time series forecasting pipelines. In: arXiv preprint arXiv:2202.01712
40. Martone A, Zazzaro G, Pavone L (2019) A Feature Extraction Framework for Time Series Analysis. In: ALLDATA, pp. 13
41. Takahashi D (2019) Fast fourier transform. In: Fast Fourier Transform Algorithms for Parallel Computers, pp. 5–13. Springer
42. Moody GB, Mark RG (2001) The impact of the MIT-BIH Arrhythmia Database. IEEE Eng Med Biol Magazine 20(3):45–50. https://doi.org/10.1109/51.932724
43. Finean R. Predicting Cognitive Fatigue with Photoplethysmography Project. https://www.researchgate.net/project/Predicting-Cognitive-Fatigue-with-Photoplethysmography-PPG

44. Finean R. PPG Heart Beat for Cognitive Fatigue Prediction Dataset. https://kaggle.com/canaria/5-gamers
45. Rizik A, Randazzo A, Vio R, Delucchi A, Chible H, Caviglia DD (2019) Feature Extraction for Human-Vehicle Classification in FMCW Radar. In: 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 131–132. https://doi.org/10.1109/ICECS46596.2019.8965072
46. Anguita D, Ghio A, Oneto L, Parra X, Reyes-Ortiz JL (2013) Human activity recognition with smartphones. p. 3. https://kaggle.com/uciml/human-activity-recognition-with-smartphones
47. Hajdarevic K, Konjicija S, Subasi A (2014) A low energy aprs-is client-server infrastructure implementation using raspberry pi. In: 2014 22nd Telecommunications Forum Telfor (TELFOR), pp. 296–299. IEEE
48. Ahamed MA, Hasan kA, Monowar kF, Mashnoor N, Hossain MA (2020) ECG heartbeat classification using ensemble of efficient machine learning approaches on imbalanced datasets. In: 2nd International Conference on Advanced Information and Communication Technology (ICAICT) IEEE pp. 140–145
49. Sree V, Mapes J, Dua S, Lih O, Koh J, Ciaccio EJ, Acharya UR (2021) A novel machine learning framework for automated detection of arrhythmias in ECG segments. J Ambient Intell Humanized Comput 12(11):10145–10162
50. Qureshi MB, Afzaal MQ, Muhammad S, Fayaz M (2021) Machine learning-based EEG signals classification model for epileptic seizure detection. Multimedia Tools Appl 80(12):17849–17877
51. Bai Z, Huang G, Wang D, Wang H, Westover MB (2014) Sparse extreme learning machine for classification. In: IEEE transactions on cybernetics, 44(10), 1858–1870, IEEE
52. Rizik A, Tavanti E, Vio R, Delucchi A, Chible H, Randazzo A, Caviglia DD (2020) Single Target Recognition Using a Low-Cost FMCW Radar Based on Spectrum Analysis. In: IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 1–4 IEEE
53. Rizik A, Tavanti E, Chible H, Caviglia DD, Randazzo A (2021) Cost-Efficient FMCW Radar for Multi-Target Classification in Security Gate Monitoring. IEEE Sens J 21(18):20447–20461
54. Anguita D, Ghio A, Oneto L, Parra X, Reyes-Ortiz JL et al (2013) A public domain dataset for human activity recognition using smartphones. In: Esann, vol. 3, p. 3
55. Yazici MT, Basurra S, Gaber MM (2018) Edge machine learning: Enabling smart internet of things applications. In:Big data and cognitive computing, **2**(3), 26, MDPI