*tiny*Care: A *tiny*ML-based Low-Cost Continuous Blood Pressure Estimation on the Extreme Edge

Khaled Ahmed
McMaster University, Canada
ahmedk1@mcmaster.ca

Mohamed Hassan
McMaster University, Canada
mohamed.hassan@mcmaster.ca

***Abstract*—We propose a solution that deploys Machine Learning (ML) techniques on resource-constrained edge devices (*tiny*ML) for the *healthcare* domain. In particular, we construct a complete end-to-end prototyped system that conducts ML inference with various ML techniques on microcontroller unit (MCU)-powered edge devices to predict blood-pressure-related vital metrics such as systolic (SBP), diastolic (DBP), and mean arterial (MAP) blood pressures using electrocardiogram (ECG) and photoplethysmogram (PPG) sensors. The proposed solution is trained and tested using over 500 hours of $12,000$ real intensive care unit data instances. Despite running on an extremely limited computation, power and memory budget, the proposed solution achieves comparable results to server-based state-of-the-art solutions. Furthermore, it meets the British Hypertension Society (BHS) standard for grade B (C in extremely-constrained devices). This is achieved by careful investigation of the correlation between a wide-set of ECG and PPG features and BP. Afterwards, we compress the ML inference models by only incorporating the minimal features that meet i) the edge constraints from one side, and ii) the standard's acceptable accuracy from the other side. Unlike existing solutions, the inference is entirely conducted on MCU-based edge devices without depending on any cloud-based infrastructure. Hence, the proposed solution improves robustness, accessibility, reliability, security, as well as data privacy.**

***Index Terms*—blood pressure, machine learning, tinyML, edge, monitoring, regression,BP, DBP, SBP, ECG, PPG**

I. Introduction

Healthcare sector is at the forefront of the domains that are envisioned to witness significant transformations by utilizing Machine Learning (ML), Internet-of-Things (IoT), and Cyber-Physical Systems (CPS) technologies [1], [2], with potential benefits spanning technological, economic, and social dimensions [3]. To enable these transformations, several solutions were proposed in the past few years to utilize the aforementioned technologies in a vast multitude of health sectors both from industry [4] and academic researchers [5]–[11]. This includes healthcare epidemiology [5], data analysis of Electronic Health Records (EHRs) [6], drug management [7] and discovery [12], Ambient-Assisted Living (AAL) [8], and healthcare monitoring [9]–[11].

One observation about most of these works is that they depend on traditional ML infrastructure, which are resource demanding and require powerful computation and memory capabilities as well as a theoretically unlimited power budget. Advances in cloud computing infrastructure is a key enabler towards the utilization of traditional ML for applications with non-safety and non-real-time requirements (e.g. the analysis of EHRs [6] and drug discovery research [12] from the healthcare domain). However, cloud-based solutions are ill-suited for applications associated with life-critical and time-sensitive conditions. Examples of these applications form the healthcare domain are AAL and healthcare monitoring. Robustness, service accessibility and response-time requirements of these applications along with security and privacy concerns entail the sole dependence on network connectivity to access the cloud a non-ideal solution that might lead to severe consequences [13]. Recent efforts in edge computing aim at mitigating these concerns by moving the required computations to the embedded systems that have a close proximity to the sensors and the controlled environment. The computing elements in these systems are usually referred to as *edge* devices. By doing so, continuous connectivity to the cloud is no longer required, which in turn mitigates the aforementioned concerns. As a result, the edge computing market is envisioned to reach 1.12 trillion dollars by 2023 [14].

Only recently, the deployment of ML inference on severely-constrained embedded edge devices has gained a considerable attention [15]–[18]. Unlike mobile phones and other relatively-larger edge hardware, these severely-constrained devices are mainly microcontroller (MCU)-based with significantly less computation and memory capabilities and smaller batteries (hence they have to run in ultra-low power). These stringent constraints motivated the field of embedded ML or *tiny*ML, which explores solutions to empower these embedded edge devices by ML inference capabilities. Healthcare is identified as one of the sectors that will have the greatest growth potential when adopting *tiny*ML [18].

In this paper, we propose *tiny*Care: a solution that leverages *tiny*ML techniques to enable continuous health-care monitoring on the extreme edge; i.e., with the close proximity to the sensors attached to the patient body without any dependency on network connectivity or cloud infrastructure. In particular, we make the following contributions. We propose an end-to-end solution for estimating vital signals (namely, systolic (SBP), diastolic (DBP) blood pressures, and mean arterial pressure (MAP)) using only electrocardiogram (ECG) and photoplethysmogram (PPG) sensors. More discussion about the background and importance of this problem in healthcare domain is provided in Section II. Six ML algorithms are considered in our case-study: Linear Regression (LR), Support Vector Machine (SVM), Polynomial Regression (PR),

Decision Tree (DT), Random Forest (RF), and Ada Boost (AB). The ML training phase is conducted offline on a desktop machine (or cloud-based servers) using the Physionet's multi-parameter Intelligent Monitoring in Intensive Care (MIMIC) dataset, which is a widely-accepted massive database comprising real collected records of patients admitted to critical care units [19]–[21]. The inference phase, unlike existing BP estimation solutions, is entirely conduced on a severely-constrained MCU-based embedded edge devices without depending on any cloud-based infrastructure. Hence, the proposed solution improves robustness, accessibility, reliability, security, as well as data privacy. Section IV covers the details of the proposed solution. The solution is prototyped and tested using medical grade sensors and several MCUs as discussed in Section V. The proposed solution is trained and tested using over 500 hours of $12,000$ real intensive care unit data instances from the MIMIC dataset. Despite running on an extremely limited computation, power and memory budget, the proposed solution achieves comparable results to server-based state-of-the-art solutions. Furthermore, it meets the British Hypertension Society (BHS) standard grades.

II. Background

One of the major causes of early deaths is the elevated blood pressure. It has no warnings nor symptoms until it damages organs. Unfortunately it is quite common as it hits 1 in 4 men and 1 in 5 women according to the the world health organization. Therefore, there is a desperate need for monitoring people with hypertension to report any anomalies and signs of risk to the healthcare facilities. Continuous and remote monitoring of BP not only mitigates risks of sudden health conditions, but also allows for better diagnosis by continuously collecting data which is practically impossible in normal daily life.

Mercury sphygmomanometer has been used for over a century to measure blood pressure. The process of measuring BP using a mercury sphygmomanometer starts with wrapping a rubber cuff around the arm, inflating the cuff to apply enough pressure to stop the flow of the blood in artery. Then, slowly deflating the cuff and observing a pounding sound. BP values when the sound starts and ends are named systolic SBP and diastolic DBP respectively. While this being the most accurate noninvasive BP measurement device, it is uncomfortable and not practical for continuous BP monitoring. Therefore, there has been a growing interest in finding noninvasive cuff-less BP estimation.

As an alternative to the traditional way of measuring BP, researchers have been working on estimating BP from other quantities. In order to derive a mathematical expression for the BP value, the heart can be thought of as a pump displacing blood through the the vascular system which is modeled as a set of tubes. The laws of fluid mechanics can then model the relationship between BP and other parameters that may be easier to capture. The pulse transit time (PTT) is one of the quantities that have been extensively used in the literature to estimate BP. It is defined as the time elapsed between a heart beat and the arrival of that beat to a body peripheral. Using fluid mechanics, it can be shown that PTT is related to the pressure (P) in the artery as [22]:

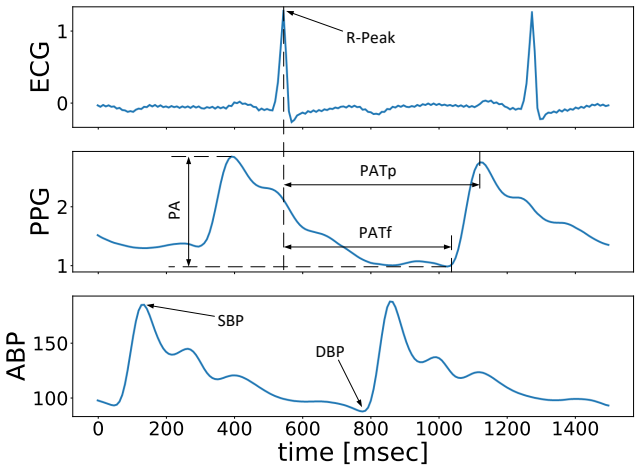


Fig. 1: ECG, PPG, and ABP signals.

$$\mathrm{PTT} = l\sqrt{\frac{\rho A_m}{\pi A P_1\left[1+\left(\frac{P-P_0}{P_1}\right)^2\right]}} \tag{1}$$

where ρ is blood density, P_0, P_1, A, and A_m are all person-dependent values, and l is the length of the artery considered for measuring PTT. Therefore, all parameters in Equation 1 depend on the person for whom PTT gets measured and BP gets estimated. Hence, this method of modelling BP as a function of PTT requires per-person calibration which is not approved by the medical standards. While Equation 1 over-simplifies the relation between PTT and BP and gives a person-specific model, it is still useful to declare a possible underlying relationship between PTT and BP while the generalization of this assumption is the unfinished work. For that kind of problems where data can be collected and a relation between some inputs and outputs is probable but mathematically intractable, machine learning is a very promising solution. The collected data can be fed to a model that explores the underlying relationship between inputs (features), and outputs (predictions).

An alternative to the PTT is the pulse arrival time (PAT) which is the time elapsed between the electrical pulse initiating the heart beat, instead of the heart beat itself as in PTT, and the arrival of the heart beat to a body peripheral. PAT is longer than PTT since it includes an extra interval between the electrical activation of the heart and the heart beat. Therefore, it is considered an approximation to the PTT that simplifies the measurement process. In order to capture PAT, 2 vital signals are required; the ECG and the PPG, Figure 1. The ECG signal is measured by placing a number of electrodes on the person's skin. It mainly shows the electrical activity of the heart and, therefore, used as a reference signal to obtain the PAT. The peak value, ECG spikes in Figure 1, is called the R-peaks

and the interval between two successive peaks indicate the time between one heart beat to another. Thus, the heart rate (HR) is the inverse of that interval. The PPG signal is usually measured by placing a clip on the person's finger tip. It uses light to indicate a change in the blood flow in the vessels. Therefore, it can capture when a heart beat's pressure reaches that finger tip. Several features can be crafted from the relative position of ECG and PPG maximums, minimums, inflection point, maximum slope, etc [23], [24]. We discuss the main features we adopt in our solution in Section IV-B. Moreover, in order to have a ground truth dataset to train machine learning models, the actual BP values have to be extracted. The BP metrics can be evaluated from the arterial blood pressure signal (ABP). Three important BP metrics are: 1) the SBP which is the peak value of the ABP. 2) the DBP which is the minimum value of the ABP. 3) the mean arterial pressure (MAP) which is the average of the ABP signal over a complete cycle.

III. Related Work

Due to the inconvenience of the traditional cuff-based blood pressure measurement technique as well as its inapplicability for continuous monitoring, there has been a significant recent attention in the research community to devise cuffless BP measurement methods [22], [24]–[34]. Early works [25]–[27] use non-ML mathematical formulas to calculate BP from pulse wave velocity (PWV) and Pulse transit time (PTT). Subsequent works deploy a preliminary fitting [28] or interpolation [29] approaches to empirically construct a mathematical function that fits the data of PWV/PPT and SBP. On the other hand, [22], [34] deploy machine learning algorithms to predict SBP, DBP, and MAP from pulse arrival time (PAT). In addition to PTT, [32] proposes utilizing the PPG intensity ratio (PIR) for better accuracy of BP estimation. These ML-based solutions assume that both training and inference are conducted on a powerful machine (e.g. cloud servers), and hence, require the ECG and PPG signals collected from sensors attached to the patient to be transmitted to the cloud to determine the corresponding BP values. In contrast, we propose a *tiny*ML methodology to conduct the BP estimation entirely on edge.

*tiny*ML is one of the fastest growing subfields of machine learning technologies [15], [16]. It aims at bringing ML capabilities to extremely resource-limited MCUs at Several solutions leverage *tiny*ML in real-world use-cases. While most of these solutions focus on audio [35]–[37] and vision [38], [39] applications, some recent efforts leverages *tiny*ML in domains such as autonomous vehicles [40], gaming [41], smart agriculture [42], and even offering *tiny*ML solutions as-a-service [43]. This work to the best of our knowledge, is the first to apply *tiny*ML techniques in a real-use case from the healthcare domain using a massive real dataset.

IV. Proposed Methodology

In this section, we introduce the flow of the proposed methodology to enable real-time machine learning inference on resource-constrained edge devices, which we delineate in Figure 2. As the figure illustrates, the proposed methodology consists of two phases; the training phase, and the inference phase. **The training phase** is usually conducted offline on a powerful computing machine or a cloud server. The main purpose of this phase is to use the training labeled data (that has both the input variables and their corresponding ground-truth outputs) to build an inference model that can be later used in the inference phase. **The inference phase** utilizes the learned inference model to predict the output from online "unlabeled" input data. Unlike the training phase, the inference phase has real-time requirements since it usually operates on a stream of incoming data. Additionally, as aforementioned, inference is desirable to be deployed on edge devices with both computational, memory, and power constraints. We now discuss in details the stages of both phases with a focus on the implications resulting from adopting resource-constrained edge devices for online inference.

A. Preprocessing

ECG and PPG vital signs normally have different kinds of artifacts caused by the patient, such as muscle movements and respiration, or due to the measuring devices and sensors that add noise and unwanted harmonics to the signals. During training, a well-known problem in the used MIMIC dataset is that PPG and ECG signals are not synchronized. Accordingly, we needed to synchronize both signals as part of the preprocessing to be able to capture the inter-signal features between them (Step ❷ in Figure 2). In our prototype on the other hand, since the pre-processing has to be conducted on the resource-constrained edge devices, we use medical grade-sensors that are equipped with noise mitigation capabilities to reduce the required on-edge processing power. In addition, the used sensor synchronizes both PPG and ECG capturing to avoid the aforementioned issue.

B. Feature Extraction

It is important to extract the relevant features to train the model (and conduct the inference) to achieve the best possible learning outcomes. In addition, feature extraction helps in reducing the size of the data fed to the model, which can significantly contribute to saving processing power on edge devices. To ease the required processing by the edge devices, and to reduce the complexity and size of the *tiny*ML models, we considered the features that can be detected from maximum and minimum points only. Therefore, the edge-devices can save the processing power required for detecting the inflection points and the maximum slope. Furthermore, by doing so, we reduced the number of extracted features to 4 features as follows (visualized in step ❸ in Figure 2).

1) Heart Rate (HR): Measured using the time interval between two successive R-Peaks in the ECG signal.
2) PAT Peak (PATp): The time interval between an R-peak, and the successive PPG maximum.
3) PAT Foot (PATf): The time interval between an R-peak, and the successive PPG minimum.
4) PPG Amplitude: The amplitude of the PPG systolic peak.

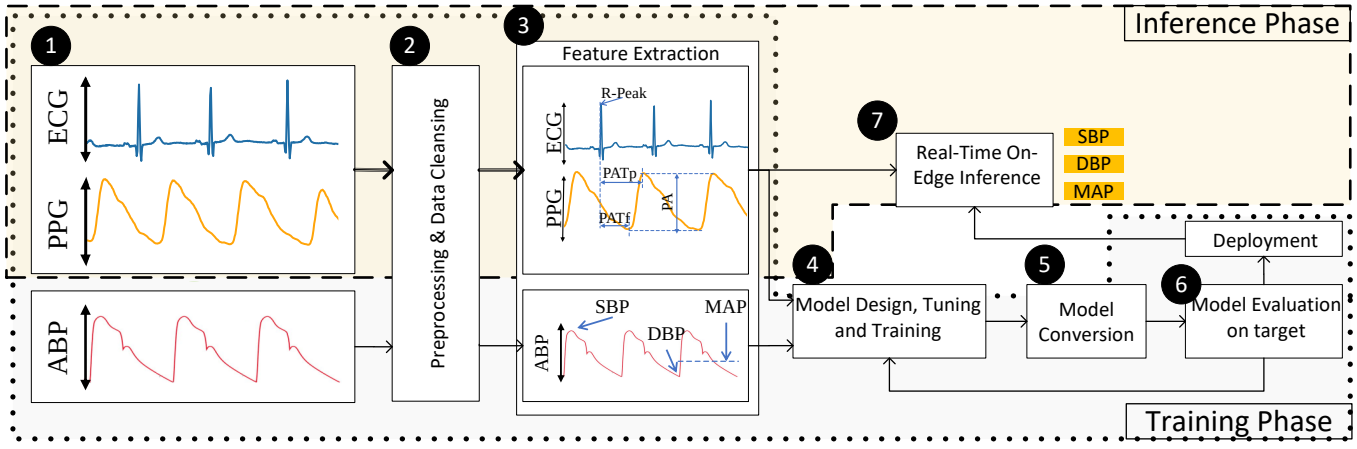


Fig. 2: Block diagram of training and edge-inference of cuffless blood pressure estimation.

For the prediction side, SBP, DBP, and MAP are extracted from the ABP signal which, again, depend only on the systolic and diastolic peaks. The effect of feature reduction will be revisited in the results in Section V. To increase reliability, the extracted values are averaged over a window of 8 seconds to reduce noise and possible preprocessing miss-detection of minimums and maximums.

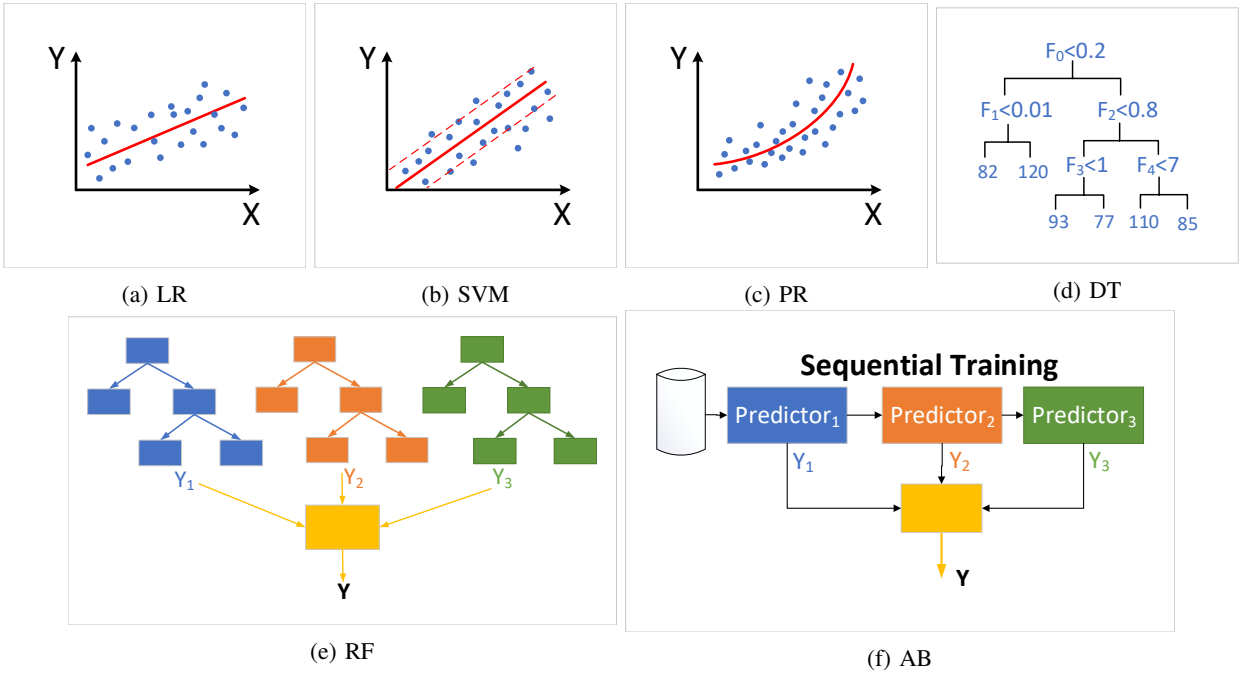
C. Model Training and Testing

Usually adopting a ML algorithm involves several trade-offs such as the desired accuracy of the model, the real-time requirements on the inference time, the training speed, the available computation resources for training, and the dependency of the algorithm on the size of the training data. Therefore, we explore different types of ML algorithms and discuss their associated trade-offs. One of the simplest models is to assume a linear relationship between the features and the corresponding output values. Examples of these models are LR (Figure 3a) and SVM (Figure 3b). The resulting linear model is a linear combination of the input features and, therefore, its complexity grows only with the number of features. Linear models are simple with relatively few inputs, and hence, are attractive for resource-constrained edge devices and are almost deployable in any MCU-based device. On the other hand, a linear relationship can be an oversimplification for the problem under consideration, which leads to accuracy degradation. For the later, more complicated models can capture the non-linearity of the problem, which in turn comes at the cost of the added computational complexity. Examples of these models are PR, DT, RF, and AB, which are shown in Figures 3c–3f, respectively. For comprehensiveness, we explore both linear and non-linear algorithms for estimating BP on the edge (Step ❹ in Figure 2).

Linear Regression and Support Vector Machine. Firstly, assuming a linear model, we consider LR and SVM with a linear kernel. LR models the output as a linear combination of the inputs. This can be illustrated, as shown in Figure 3a, by assuming a single input and drawing a line that best-represents the instances and, therefore, minimizes the prediction error. The SVM algorithm models the linear relationship in a different way. One way to visualize it, is to draw a street that includes as much instances as possible with limited violations. Since SVM allows some instances to be outside the modelled street, it handles outliers more efficiently than linear regression. Therefore, both LR and SVM consume the same memory and inference time T_{inf}, while the SVM has higher accuracy (i.e. less MAE) due to its immunity to outliers. This comparison is shown in Figure 4 where the memory and T_{inf} coincide while the SVM shows less MAE.

Polynomial Regression. PR can be thought of a LR of the polynomial generated from the inputs. Thus, it consider different inputs raised to different powers, as well as combinations and multiplications of those inputs. This behaviour is useful not only for modelling non-linearity (Figure 3c), but also to create new features that are the mixtures of the original ones raised to different powers. The complexity of PR increases with the number of features as well as the polynomial degree. Notice that the multiplication process, and raising a number to a certain power, are both resource-hungry arithmetic processes. Therefore, it is worth using on the edge only if its accuracy pays for the added computational cost. Further discussion about its performance is included in Section V.

Decision Tree. Another powerful algorithm that can model complex relations between inputs and outputs is DT. The DT algorithm is mainly a series of comparisons shaped as a tree as Figure 3d illustrates. At each node, one feature is checked and, based on the comparison, it goes to the right or the left branch and eliminates the other. Due to this structure, *DT has four particular advantages that are essential for edge devices.* 1) It is based on comparison which is very fast and simple arithmetic process. 2) It does not require any scaling to the inputs since the comparisons can be done with respect to the non-scaled inputs directly. This advantage reduces the required preprocessing significantly. 3) At each comparison, half of the tree is eliminated. Therefore, the maximum number of comparisons to make a prediction in



(a) LR (b) SVM (c) PR (d) DT

(e) RF (f) AB

Fig. 3: Illustrations of machine learning algorithms: LR, SVM, PR, DT, RF, AB.

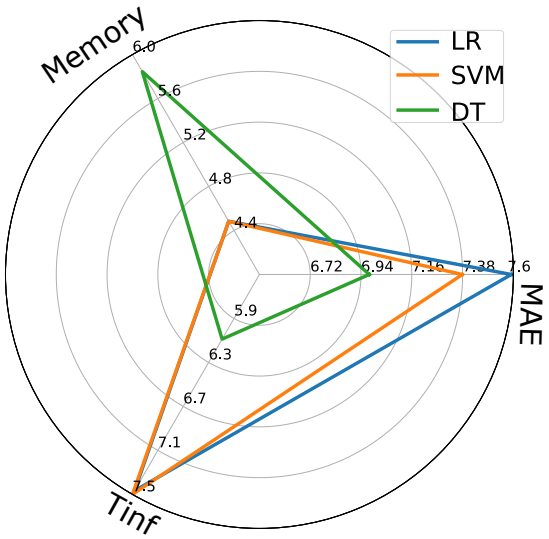


Fig. 4: Memory, MAE, and T_{inf} evaluation of DBP estimation using LR, SVM, and DT on PyBadge board.

a tree of N nodes is $\log(N)$. 4) The memory requirements can be easily controlled by tuning its hyper-parameters; the number of leaf nodes and/or the maximum depth. On the other hand, DT is sensitive to input data and changing the training data leads to a completely different tree. Additionally, the tree is built in a stochastic way with random features checked at each node. Therefore, unless a pre-defined random seed is given, the same training data may result into a different DT model at every run. In conclusion, the DT has lots of advantages especially for the edge-related applications, but it requires further tuning to overcome its drawbacks.

Ensemble Algorithms: Decision Tree and AdaBoost. Ensemble algorithms are strong predictors that are based on a group of weaker predictors. They can boost the performance of the sub-predictors by collecting their estimates and making a final prediction out of them. Therefore, we consider two ensemble algorithms based on DTs to improve the DT performance and tackle its challenges. First, we consider RF, which is a group of DTs, each making a prediction, and the final prediction is the average of all of them (Figure 3e). The averaging step reduces noise, improves the stability of DT, and reduces its dependency on the training data. Second, we use AB, which is also a set of DTs similar to RF but learns the model differently. It uses sequential learning in which every predictor learns to correct its predecessor as if each DT is focusing on one aspect of the data and trying to master it. Figure 3f delineates this behavior. While both RF and AB improve the performance, their memory requirements increase with the number of DTs and the size of each tree. Figure 4 visualizes the different trade-offs of ML models by plotting the Mean Absolute Error (MAE), inference time T_{inf}, and the memory requirements for LR, SVM, and DT. T_{inf} is measured in micro-seconds and the memory axis represents the occupied percentage of the available memory. The results in Figure 4 are collected from our setup using the PyBadge MCU estimating the DBP using three different algorithms; LR, SVM, and DT. More details about the setup are provided in Section V.

D. Feature Scaling

Another important stage to reconsider for real-time edge inference is *feature scaling*. ML algorithms generally can learn faster when the input data is scaled, i.e. all data has the same scale. On the other hand, since the same pipeline shall be executed in the inference stage, this will be a burden for a resource-constrained edge device. This is because scaling requires shifting and normalizing the inputs, which is a subtraction operation followed by a division. The processing power required for those operations can be more than the one required for the prediction stage itself (e.g. in linear models). Moreover, the scaling factors are data dependent. For instance, standardization, which is one of the most commonly used scaling approaches [44] requires to calculate the mean and standard deviation of the input data to use them for scaling, which requires a frequent re-evaluation. Therefore, a reasonable decision for real-time edge inference is to feed the data to the model with no scaling. This may result in spending more time in the training process, but it reduces the required processing on the edge dramatically.

E. Model Conversion

The outcome from the training phase is a trained and tested model. Since we are targeting embedded MCU-based devices, the generated models need to be converted to their equivalent C codes to fit the micro-controllers. This is shown in Figure 2 as Step ❺. For *tiny*Care, we use the *m2cgen* [45] since it supports the models we are using and it can convert them to the equivalent C codes. Additionally, we use the converted C models on the PC as a baseline to compare the edge performance against, and to help in quick evaluation of the generated Models to decide their applicability to the target device's resources as well as their need for further tuning.

F. Model Assessment and Tuning

As discussed in Section IV-C, the size of the produced model may vary depending on the given model parameters. As a result, depending on the edge memory constraints, the generated model needs to be tested to check that it can fit on the target edge device (Step ❻ in Figure 2). This check needs to be done before the deployment phase and if the models do not fit within the edge constraints (e.g. they are bigger than the available memory), different model parameters may be used to produce smaller models that may fit. This is represented by the feedback path to Step ❹ in Figure 2. Clearly, this comes at the cost of compromised accuracy and Section V discusses.

In our healthcare use-case, and as we will discuss in more details in Section V, LR, SVM, and DT for instance fit in all considered devices, RF fits only on one of the edge devices and does not fit on the other two due to memory constraints, while AB did not fit in any of the target devices. Moreover, model parameter tuning is required for DT to determine the number of nodes that represents the best compromise between the memory footprint and the desired accuracy. In addition, RF required further parameter tuning to determine the best number of ensembles (i.e. parallel DTs) to address the same memory-accuracy trade-off.

G. Inference Phase on Edge Devices

The inference phase on edge devices as shown in Figure 2 also has the same Steps ❷ (preprocessing) and ❸ (feature extraction) as in the training phase. Therefore, the discussion in Sections IV-A and Section IV-B applies. The main difference of those blocks in the training-versus-inference is that the ground truth for the target (values to be predicted) appears only in the training phase. On contrary, in the inference phase those predictions are the output and should not appear at any of the inputs. Therefore, in our use-case, the inputs to the preprocessing block are the ECG and PPG only, excluding the ABP. Consequently, as shown in Figure 2, the extracted features do not include SBP, DBP, nor MAP in the inference phase. Those three appear as outputs (predictions) of the edge device instead. Step ❼ in Figure 2 contains the trained models after being converted to the proper language that the edge device can understand. For a complete implementation, both preprocessing and feature extraction should be deployed on the edge device, as a complete end-to-end pipleline. The whole pipeline needs to fulfill the constraints enforced by both the application (e.g. maximum latency) and the implementation (e.g. memory constraints).

Finally, a verification step is necessary to complete the evaluation of the whole system. This can be done by closing the loop by collecting the predicted values from the edge device, compare it with the ground truth (i.e. original features-to-target values), and evaluate the accuracy of the edge device and compare it with the accuracy obtained on the PC. A lower accuracy is expected at the far end of the system. This degradation is an unavoidable result of all the compromises being done to match the edge constraints. Whether this was done by simplifying the preprocessing block, sacrificing some features, tuning models to fit, or quantizing variables on extremely-constrained MCUs compared to powerful server machines usually used for ML. Those limitations trigger several new trade-offs that are problem-specific and open the door for a new research direction to address them. We elaborate more on the discussion of these trade-offs and introduce several insights from the healthcare use-case we consider to address them in our evaluation in Section V.

V. Evaluation

We now evaluate the proposed methodology by applying it to the *tiny*ML healthcare usecase and discuss the different trade-offs affecting the results upon deploying real-time inference on resource-constrained edge devices.

A. Experimental Setup

Dataset. We use the MIMIC database, which is a publicly available medical database provided by PhysioNet [19]–[21]. The latest version of the MIMIC database, MIMIC-IV [21], has critical care data for more than 40,000 de-identified patients that were admitted to intensive care units. The collected

MCU	processor	cores	width	Program Storage	SRAM	Clock
Arduino uno	ATmega328P	1	8 bit	32KB/31.5KB	2KB/2KB	16MHz
ESP32	Xtensa LX6	2	32 bit	4MB/1.25MB	520KB/320KB	80-240MHz
PyBadge	ATSAMD51	1	32-bit	512KB/ 496KB	192KB/NA	120MHz

TABLE I: Technical Specifications of Edge Devices.

data includes various vital signs, laboratory measurements, clinical information, medical diagnoses, and other personal information. The variety of the recorded information, the large size, and its public availability make it an excellent option for a practical, reliable, and reproducible model. Our proposed model for BP estimation requires only three vital signs from the MIMIC database namely; ECG, PPG, and ABP. We extract the features from the ECG and PPG signals, while the ABP provides us with the corresponding SBP, DBP, and MAP values. Therefore, we find the MIMIC database to be an excellent choice for training our models with enormous amounts of data from thousands of patients over hundreds of hours, which is one of the distinguishing elements for this work compared to the state-of-the-art *tiny*ML works, which generally use small data for training and/or testing.

Edge Devices. As discussed in Section IV, the constraints of edge devices impact all stages in the pipeline, from dataset selection all the way to model deployment (Figure 2). Therefore, we implement our *tiny*ML models on three edge devices with different specifications; namely Arduino uno, ESP32 Wrover Board, and AdaFruit PyBadge. Each of those devices introduces a different set of constraints on memory and clock frequency (hence inference time), which in return dictates the deployable models and the resulting accuracy. Table I shows the specifications of those edge devices in terms of flash memory, SRAM, and clock frequency. The training phase is always conducted on a PC machine with a quad-core Intel Core i7-9700 processor running on a 3 GHz clock and equipped with a 64 GB DRAM. The inference phase is conducted on all the three edge devices. We also run the inference on the aforementioned PC machine and use it as a baseline for comparison purposes only. The accuracy is quantified in terms of different metrics such as mean absolute error (MAE), mean error (ME), and standard deviation (Std). We also compute the cumulative distribution function (CDF) of prediction errors for each scenario to compare against the BHS standard and categorize the performance into BHS grades.

Machine Learning Algorithms. The experiments we ran on different devices are as follows. We use in total six machine learning algorithms (LR, SVM, PR, DT, RF, AB) to predict SBP, DBP, and MAP, on the PC using the Scikit-learn framework [46]. Then, we convert the resulting model out of the training phase to C. The rational behind this conversion is that 1) we want to assess and tune the resulting models for edge devices (which are programmed in C) as we discuss later in Section V-C, and 2) we want to use the inference on PC as a baseline to compare the edge performance against.

Dataset. We use over 500 hours of the MIMIC dataset for the training and testing process on both the PC and the edge devices. More than 12,000 signals are processed to extract the features (HR, PATp, PATf, PA) and the corresponding BP metrics.

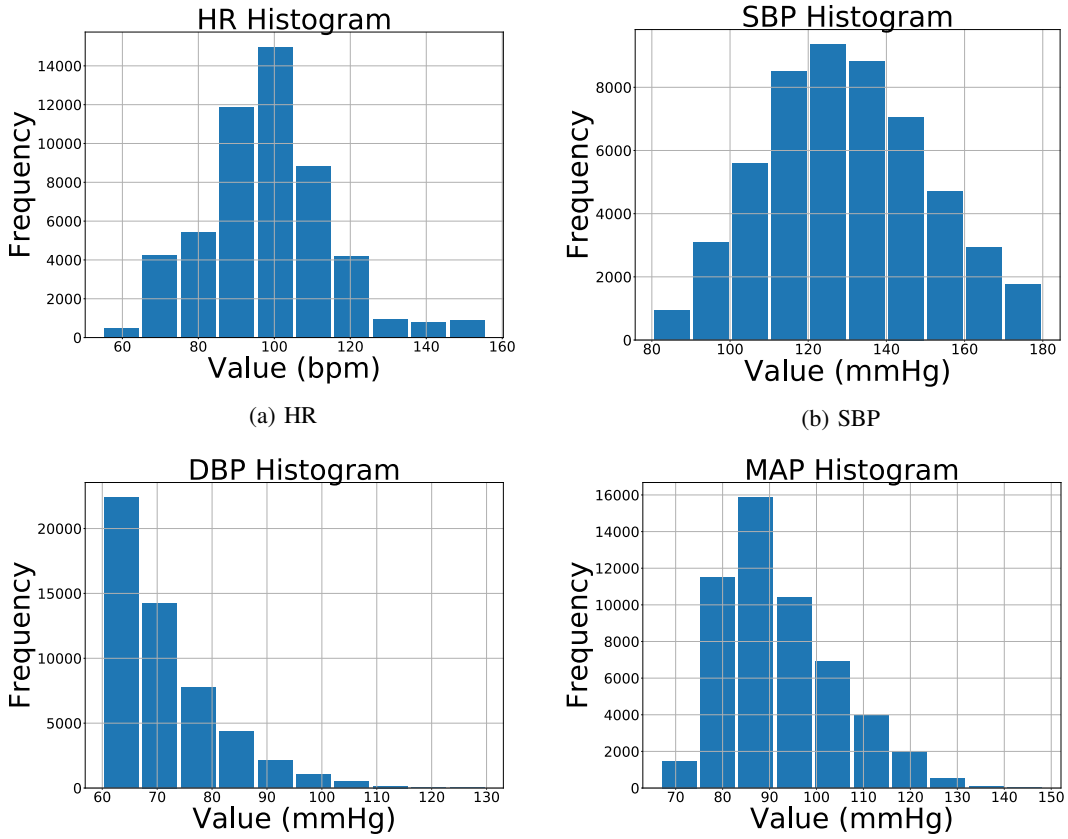
B. Data Preparation and Feature Extraction

The preprocessed data is fed to the feature extraction block in the training stage. Extracting the features starts with the detection of the ECG, PPG, and ABP peaks (minimums and maximums). Then, the HR is evaluated as the inverse of the interval between two successive R-peaks from the ECG signal. PATp, PATf, and PA are then determined based on peak values and their relative locations as discussed in Section II. Similarly, the SBP, DBP, and MAP are extracted from the ABP signal as a ground truth for our predictions.

As a further data cleansing step, all extracted features and BP values are averaged over a window of 8 seconds to tolerate possible peak misdetection. Additionally, we remove outliers by dropping the extreme values of BP and HR that are far from normal physiological records. In particular, we consider $80 \leq \text{SBP} \leq 180$ mmHg, $60 \leq \text{DBP} \leq 130$ mmHg, and $54.4 \leq \text{HR} \leq 155.8$ bpm. The distributions of HR, SBP, DBP, and MAP after cleansing are shown in Figures 5a–5d, respectively. After removing outliers, we shuffle the resulting dataset to include different samples from different patients' instances. Shuffling data exposes the model to samples from a variety of patients with different conditions which leads to more general models and reduces over-fitting. Finally, the shuffled dataset is then divided into training and testing datasets with a ratio of 80%-20%, respectively.

C. Model Assessment and Tuning

After obtaining the inference model, we assess its applicability to the specific edge device constraints and whether the model parameters can be tuned to address the trade-offs of this device (e.g. performance vs memory vs accuracy). For the considered six algorithms, LR and SVM fit on all edge devices and do not require any parameter tuning. This was expected since the resulting inference model is a simple linear equation. For PR, despite fitting in all devices and having a slightly better accuracy, this comes at the cost of model complexity. For PR to achieve this slightly better accuracy (Results for PR on PC are in Table III), we had to use a polynomial of order 10. Therefore, PR was deemed non beneficial for the usecase. DT shows significant improvement in performance; however, it is an example of a model that requires parameter tuning. In order to select the DT's parameters (maximum number of leaf nodes, minimum number of leaf samples), we conduct a grid search to the performance change with these parameters. Figure 6a shows how MAE decreases while increasing the



(a) HR

(b) SBP

(c) DBP

(d) MAP

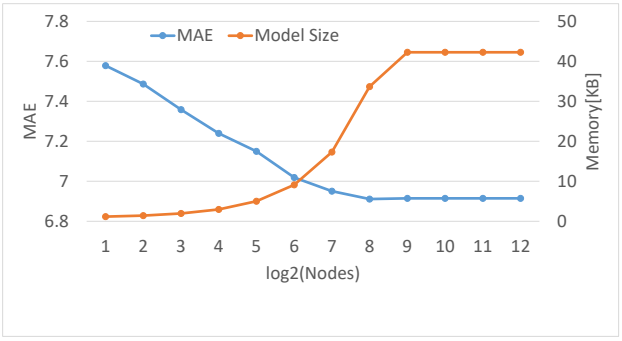
Fig. 5: Distribution histograms after data cleansing

maximum number of nodes until it almost saturates. For MAE to keep decreasing while increasing the model complexity, number of nodes in this case, proper regularization needs to be applied to prevent the model from overfitting the training data during the training phase. This is done by increasing the number of minimum samples per node. After this tuning stage, DT fits in all edge devices and achieves the maximum possible accuracy with the minimum memory footprint. For RF, since it is naturally composed of several DTs, we use the DT with the obtained parameters to construct the RF model, whom the preliminary performance analysis on PC shows that it achieves better performance than DT. In addition to the DT parameters tuning, another RF parameter that can be tuned is the number of estimators (i.e. DTs). As Figure 6b shows, this parameter affects both the model performance (accuracy) and its complexity (memory footprint). Finally, AB boosts the performance by sequentially training DTs to correct predictions errors. With that technique, it reaches BHS grade B for DBP estimation. For AB to achieve grade B requirements, 1024 DTs were deployed and the resulting model size was over 1 GB of memory. Therefore, it is more practical to be run on the cloud rather than on the edge.

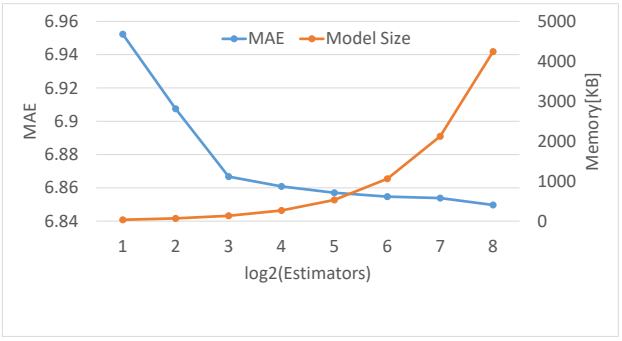
D. Performance of ML Algorithms

We compare predictions (SBP, DBP, MAP) from the trained models to the ground truth values, and calculate the MAE, ME, and Std values for accuracy evaluation.

Meeting Medical Grades for BHS Standard. Furthermore, CDF of those errors are numerically calculated and plotted to judge which of those algorithms belong to which BHS medical grades. Figure 7 shows the CDFs of DBP prediction errors of various algorithms on Python, C, and on edge devices. A set of markers are added to the graph to specify different BHS grades (A, B, C) where grade D classifies anything worse than C. For a CDF graph to match a certain grade, it has to be higher than all of the 3 markers defining that grade. This is equivalent to having probability of BP estimation error of 5, 10, and 15 mmHg within the acceptable limits of that grade as shown in Table II. As discussed in Section IV-C, we carefully chose a diverse set of algorithms starting from a simple LR and boosting the performance using more complicated models up to AB. Hence, as shown in Figure 7, those algorithms achieved a wide range of BHS grades, from grade B, and approaching grade A, to grade D. We now detail a complete comparison between ML and *tiny*ML algorithms on PC and edge devices for DBP estimation. LR was not able to capture the input-output relationship precisely enough to achieve any



(a) DT accuracy and model size versus number of leaf nodes.



(b) RF accuracy and model size versus number of estimators.

Fig. 6: The effect of model tuning for DT and RF

	Cumulative Error[mmHg]		
	≤ 5	≤ 10	≤ 15
Grade A	60%	85%	95%
Grade B	50%	75%	90%
Grade C	40%	65%	85%
Grade D	worse than C		

TABLE II: BHS grades and associated error percentages.

of the top three grades and, therefore, is categorized as grade D. SVM, same inference complexity (linear equation) but with more complicated training, managed to reach grade C. This is an important advantage especially for edge applications since the training complexity is done on powerful machines while the inference complexity matters the most. While PR, DT, and RF come at the same level, grade C, as SVM, their performance differs in terms of MAE and Std as shown in Table III. Lastly, AB built on the top of 1024 DTs achieved grade B and approached grade A. Despite achieving different grades, it is remarkable that CDFs of prediction errors from different algorithms, on PC and edge devices, are relatively close to each other and our trained models are still satisfying medical grades on the edge.

Inference Accuracy on Edge: MAE. While BHS medical grades provide a standardized method to judge the accuracy, they do not differentiate between models belonging to the same grade. For a higher-resolution comparison, MAE is used to quantify the accuracy and reflect the superiority of models over each other. Table III shows the MAE of different algorithms implemented on PC and the edge devices. The two linear algorithms, LR and SVM, come at the tail of the list. Notice that, due to the stochastic nature of training SVM, sometimes its produced models can perform worse than LR. This can be addressed by repeatedly training SVM models and saving them, then picking the one with the best performance. As highlighted earlier, PR is slightly better than LR and SVM but this comes at the cost of model complexity and, therefore, is excluded from the models deployed on edge for this usecase. DT, RF, and AB show significant improvement in accuracy. Reported results for DT and RF are after the parameter tuning,

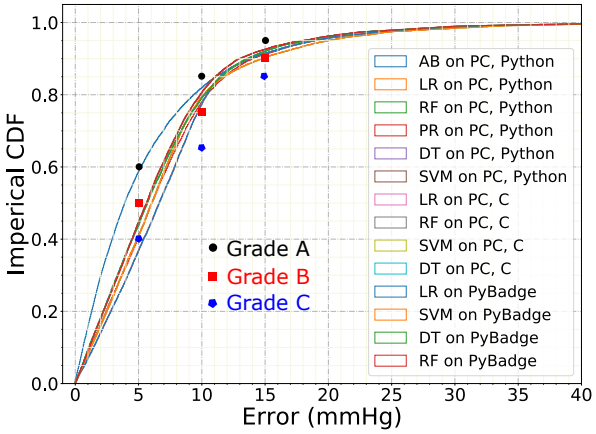


Fig. 7: CDFs and BHS grades for DBP estimation on PC and edge device (AdaFruit PyBadge).

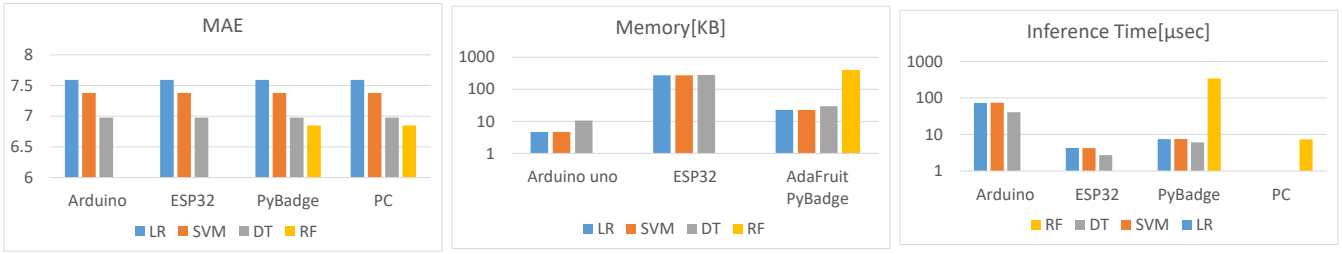
	PC			Edge		
	DBP	MAP	SBP	DBP	MAP	SBP
LR	7.59	9.66	16.7	7.59	9.66	16.7
SVM	7.38	10.98	16.92	7.38	10.98	16.92
PR	7.35	9.04	15.35	NA	NA	NA
DT	6.98	8.51	14.49	6.98	8.51	14.49
RF	6.85	8.34	14.08	6.85	8.34	14.08
AB	6.06	7.49	12.69	NA	NA	NA

TABLE III: MAE for DBP, MAP, SBP estimation

while AB has no edge results since it did not fit on any of the edge devices.

E. Comparison with Server-Based Solutions

For edge consideration and in order to match the device constraints, we had to make trade-offs at each step in our proposed methodology in Figure 2. Thus, those compromises affected the performance and the edge-feasibility of the ML and *tiny*ML algorithms. **1)** By reducing the number of features



(a) MAE (b) Memory (c) Inference time

Fig. 8: DBP estimation results

to less than half of those in the literature, we simplified the feature extraction step and reduced the complexity of algorithms that use those features to make predictions. However, this came at the cost of the accuracy of the models. Table IV shows a comparison between our work, on PC and edge devices, and the work in the literature. Comparing the MAE and Std values, we can see, with the proposed feature reduction, the accuracy of our models are among the accuracy of those in [47] and [23]. For instance, our work on PC achieves lower MAE for DBP and MAP estimation compared with [47]. On the other hand, due to sacrificing more than half of the features, the work in [23] outperforms ours. However, our reduced-complexity models, both on PC and on edge devices, still satisfy BHS medical grades as aforementioned.

2) Due to the memory constraints, some algorithms do not fit on the edge devices. Figure 8b shows the required memory of different trained models on Arduino, ESP32, and the PyBadge. For the considered MCUs, AB did not fit on any of them, and RF fits on the PyBadge only. On the contrary, LR, SVM, and DT (with certain parameters) matched the requirements of all MCUs. Therefore, the best on-edge performance for the selected devices and the considered algorithms was obtained using RF. This is also reflected in Table IV as for the same number of features, MAE is higher on edge devices since our best model, i.e. AB, does not match the device constraints. MAE of DBP prediction using the selected *tiny*ML algorithms (LR, SVM, DT, RF) on edge devices is shown in Figure 8a.

3) The memory requirements of edge devices not only filtered out some algorithms, but also limited the range of parameters of other algorithms. More specifically, for DT and RF, where the complexity and model size are dependent on the pre-defined model parameters. DT's size increases with the number of nodes which can be limited by defining the maximum number of leaf nodes which impacts accuracy in return as shown in Figure 6a. By looking at the graph, Figure 6a, we can decide the range of number of nodes that satisfies the device constraints and, hence, the best achievable MAE accordingly. RF's size and accuracy are affected by both the number of DTs, its building blocks, and the size of those DTs. A grid search can be made to change both parameters and observe the corresponding MAE and memory requirements. However, for simplicity of the design process, we used the obtained DT structure from the previous step, and tuned the number of DTs in the RF to optimize for MAE and memory of the whole structure.

	DBP		MAP		SBP	
	MAE	Std	MAE	Std	MAE	Std
[23]	5.35	6.14	5.92	5.38	11.17	10.09
[47]	6.34	8.45	7.52	9.54	12.38	16.17
this work, PC	6.06	8.95	7.49	10.42	12.69	17.31
this work, edge	6.85	9.16	8.34	10.75	14.08	17.82

TABLE IV: Comparison with other work.

4) Since most of on-the-edge applications require real-time analyses, it is essential to investigate the inference time of different algorithms on different devices. The timing requirements are enforced by the application which, in our use-case, is BP estimation. Normally, BP is monitored on a scale of minutes which gives the edge devices plenty of time to produce their predictions. However, to generalize our outcomes, we evaluated the inference times on different devices and found it to be, by most, in μseconds as shown in Figure 8c. By comparing different algorithms on the same device, Figure 8c shows that LR and SVM have the same inference time while DT works faster. This is due to the similarity, computational-wise, between the linear equation used in both LR and SVM. On the other hand, the comparisons in DT are evaluated much efficiently that a tree with 128 nodes is traversed faster than a linear equation in LR and SVM. Figure 8c gives more insights by comparing the inference time of the same algorithm on different devices in which the architecture, number of cores, and clock frequency are the main factors affecting the inference time of that algorithm on different devices.

F. Prototype

In addition to the massive testing of the proposed approach using the MIMIC dataset on the MCUs, we also built a complete prototype that captures lively ECG and PPG signals from subjects and estimates the BP metrics, which we depict in Figure 9. It is important to emphasize that the proposed solution is not limited to the specific modules used in the prototype. Other sensors, MCUs, or displaying components than the deployed ones can be used. First the sensor captures ECG and PPG signals. In this prototype, we use the MAX86150, a medical-grade sensor with low-noise

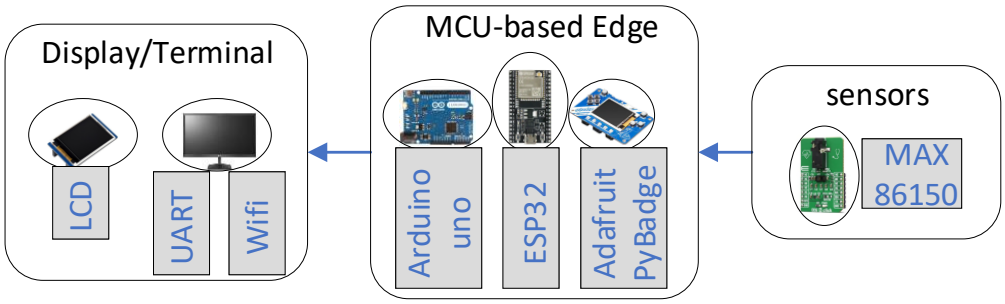
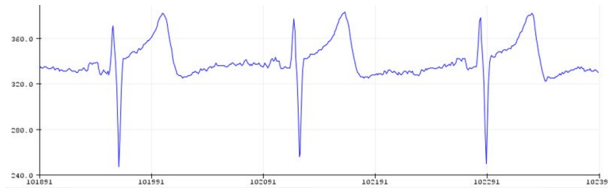
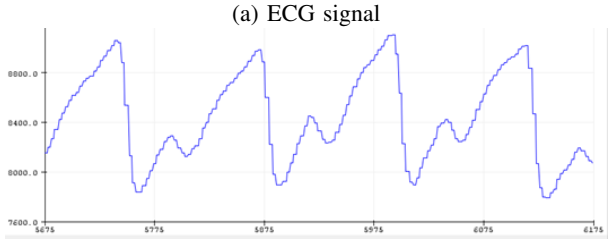


Fig. 9: Use-case prototype setup



(a) ECG signal



(b) PPG signal

Fig. 10: Captured signals from the MAX86150 sensor for one of the subjects

and ambient light rejection features for accuracy. It integrates both the PPG and ECG sensing modules into the same chip. The PPG is optically captured by measuring the changes in the volume of blood over the skin tissue. ECG sensor on the other hand detects the electrical activity of the heart. We leverage this capability to simultaneously sample ECG and PPG signals such that we get synchronised ECG and PPG values for better PTT feature extraction. We use the MikroElektronika ECG 6 to connect the electrodes to the sensor and to connect the sensor to the MCU through the standard I^2C bus protocol. We experimented on 10 different subjects varying on age and health conditions. We justify the limited number of subjects by 1) the constraints imposed by the pandemic and 2) the extensive testing done using the real MIMIC dataset. Figures 10a and 10b delineates an example of the sensed raw ECG and PPG signals respectively for one of the subjects. We observe that the ECG of this subject has a negative QRS Complex Polarity (sometimes referred to as rS complex), which in our case was due to the placement of the electrodes by the subject. This has no effect on BP estimation since peaks are still easily detectable. For the raw PPG signal, as the figure illustrates, it is horizontally mirrored/flipped. This is because the PPG sensor's LED emits an infrared light, which penetrates the skin and blood vessels. Afterwards, a photodectector captures the reflected light to measure the blood stream. This captured light is the mirror of the PPG signal. This mirroring again does not affect the feature extraction process, and hence the algorithms of predicting BP values. The inference is conducted entirely on the edge and the resulting predicted metrics are displayed either through an LCD or sent to a PC station through serial communication or WiFi.

VI. Conclusion

We propose a general methodology to systematically deploy real-time ML inference on extremely resource-constrained edge devices. In this methodology, we reconsider the full ML pipeline from the perspective of edge-related trade-offs. As a showcase for the methodology, we use it to propose an end-to-end solution for estimating blood pressure metrics from electrocardiogram (ECG) and photoplethysmogram (PPG) sensors. The proposed solution is prototyped and tested using a massive amount of data from real patients. Despite running the inference entirely on edge, the performance is comparative to server-based solutions and meets medical standard grades.

VII. Acknowledgment

This work has been partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the McMaster University Engineering Life Event Fund (ELEF).

References

[1] T. B. Murdoch and A. S. Detsky, "The inevitable application of big data to health care," *Jama*, vol. 309, no. 13, pp. 1351–1352, 2013.

[2] R. Bhardwaj, A. R. Nambiar, and D. Dutta, "A study of machine learning in healthcare," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2017, pp. 236–241.

[3] S. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K.-S. Kwak, "The internet of things for health care: a comprehensive survey," *IEEE access*, vol. 3, pp. 678–708, 2015.

[4] C. Insights, "From virtual nurses to drug discovery: 65+ artificial intelligence startups in healthcare," *CB Insights*, 2016.

[5] J. Wiens and E. S. Shenoy, "Machine learning for healthcare: on the verge of a major shift in healthcare epidemiology," *Clinical Infectious Diseases*, vol. 66, no. 1, pp. 149–153, 2018.

[6] B. Norgeot, B. S. Glicksberg, and A. J. Butte, "A call for deep-learning healthcare," *Nature medicine*, vol. 25, no. 1, pp. 14–15, 2019.

[7] D. S. A. Minaam and M. Abd-Elfattah, "Smart drugs: Improving healthcare using smart pill box for medicine reminder and monitoring system," *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 443–456, 2018.

[8] A. Vijayalakshmi and D. V. Jose, "Internet of things for ambient-assisted living—an overview," *Internet of Things Use Cases for the Healthcare Industry*, pp. 221–239, 2020.

[9] K. N. Swaroop, K. Chandu, R. Gorrepotu, and S. Deb, "A health monitoring system for vital signs using iot," *Internet of Things*, vol. 5, pp. 116–129, 2019.

[10] P. Gupta, D. Agrawal, J. Chhabra, and P. K. Dhir, "Iot based smart healthcare kit," in *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*. IEEE, 2016, pp. 237–242.

[11] M. R. Ma'arif, A. Priyanto, C. B. Setiawan, and P. W. Cahyo, "The design of cost efficient health monitoring system based on internet of things and big data," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2018, pp. 52–57.

[12] H. Bohr, "Drug discovery and molecular modeling using artificial intelligence," in *Artificial Intelligence in Healthcare*. Elsevier, 2020, pp. 61–83.

[13] I. Azimi, A. Anzanpour, A. M. Rahmani, T. Pahikkala, M. Levorato, P. Liljeberg, and N. Dutt, "Hich: Hierarchical fog-assisted computing architecture for healthcare iot," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–20, 2017.

[14] C. MacGillivray and M. Torchia, "Internet of Things: Spending trends and outlook," 2019. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US45161419

[15] tinyML, "tinyml foundation," 2021. [Online]. Available: https://www.tinyML.org

[16] P. Warden and D. Situnayake, *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[17] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov *et al.*, "Benchmarking tinyml systems: Challenges and direction," *arXiv preprint arXiv:2003.04821*, 2020.

[18] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020.

[19] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals," *circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[20] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-Wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[21] A. Johnson, L. Bulgarelli, T. Pollard, S. Horng, L. A. Celi, and R. Mark, "Mimic-iv," *circulation*, 2020.

[22] M. Kachuee, M. M. Kiani, H. Mohammadzade, and M. Shabany, "Cuffless blood pressure estimation algorithms for continuous health-care monitoring," *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 4, pp. 859–869, 2016.

[23] ——, "Cuffless blood pressure estimation algorithms for continuous health-care monitoring," *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 4, pp. 859–869, 2016.

[24] A. Gaurav, M. Maheedhar, V. N. Tiwari, and R. Narayanan, "Cuff-less ppg based continuous blood pressure monitoring — a smartphone based approach," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2016, pp. 607–610.

[25] C. Poon and Y. Zhang, "Cuff-less and noninvasive measurements of arterial blood pressure by pulse transit time," in *2005 IEEE engineering in medicine and biology 27th annual conference*. IEEE, 2006, pp. 5877–5880.

[26] D. B. McCombie, A. T. Reisner, and H. H. Asada, "Adaptive blood pressure estimation from wearable ppg sensors using peripheral artery pulse wave velocity measurements and multi-channel blind identification of local arterial dynamics," in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2006, pp. 3521–3524.

[27] M. Y.-M. Wong, C. C.-Y. Poon, and Y.-T. Zhang, "An evaluation of the cuffless blood pressure estimation based on pulse transit time technique: a half year study on normotensive subjects," *Cardiovascular Engineering*, vol. 9, no. 1, pp. 32–38, 2009.

[28] H. Gesche, D. Grosskurth, G. Küchler, and A. Patzak, "Continuous blood pressure measurement by using the pulse transit time: comparison to a cuff-based method," *European journal of applied physiology*, vol. 112, no. 1, pp. 309–315, 2012.

[29] S. Ahmad, S. Chen, K. Soueidan, I. Batkin, M. Bolic, H. Dajani, and V. Groza, "Electrocardiogram-assisted blood pressure estimation," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 3, pp. 608–618, 2012.

[30] L. Peter, N. Noury, and M. Cerny, "A review of methods for non-invasive and continuous blood pressure monitoring: Pulse transit time method is promising?" *Irbm*, vol. 35, no. 5, pp. 271–282, 2014.

[31] R. Mukkamala, J.-O. Hahn, O. T. Inan, L. K. Mestha, C.-S. Kim, H. Töreyin, and S. Kyal, "Toward ubiquitous blood pressure monitoring via pulse transit time: theory and practice," *IEEE Transactions on Biomedical Engineering*, vol. 62, no. 8, pp. 1879–1901, 2015.

[32] X. Ding, B. P. Yan, Y.-T. Zhang, J. Liu, N. Zhao, and H. K. Tsang, "Pulse transit time based continuous cuffless blood pressure estimation: A new extension and a comprehensive evaluation," *Scientific reports*, vol. 7, no. 1, pp. 1–11, 2017.

[33] J. Solà and R. Delgado-Gonzalo, *The Handbook of Cuffless Blood Pressure Monitoring*. Springer, 2019.

[34] E. Monte-Moreno, "Non-invasive estimate of blood glucose and blood pressure from a photoplethysmograph by means of machine learning techniques," *Artificial intelligence in medicine*, vol. 53, no. 2, pp. 127–138, 2011.

[35] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio set: An ontology and human-labeled dataset for audio events," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 776–780.

[36] Y. Vaizman, K. Ellis, and G. Lanckriet, "Recognizing detailed human context in the wild from smartphones and smartwatches," *IEEE pervasive computing*, vol. 16, no. 4, pp. 62–74, 2017.

[37] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[38] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," *arXiv preprint arXiv:1906.05721*, 2019.

[39] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[40] M. de Prado, M. Rusci, A. Capotondi, R. Donze, L. Benini, and N. Pazos, "Robustifying the deployment of tinyml models for autonomous mini-vehicles," *Sensors*, vol. 21, no. 4, p. 1339, 2021.

[41] S. Lobov, N. Krilova, I. Kastalskiy, V. Kazantsev, and V. A. Makarov, "Latent factors limiting the performance of semg-interfaces," *Sensors*, vol. 18, no. 4, p. 1122, 2018.

[42] C. Vuppalapati, A. Ilapakurti, K. Chillara, S. Kedari, and V. Mamidi, "Automating tiny ml intelligent sensors devops using microsoft azure," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 2375–2384.

[43] H. Doyu, R. Morabito, and J. Höller, "Bringing machine learning to the deepest iot edge with tinyml as-a-service," *IEEE IoT Newsl*, 2020.

[44] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[45] m2cgen, "Model 2 Code Generator," Mar. 2021. [Online]. Available: https://github.com/BayesWitnesses/m2cgen

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[47] M. Kachuee, M. M. Kiani, H. Mohammadzade, and M. Shabany, "Cuff-less high-accuracy calibration-free blood pressure estimation using pulse transit time," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015.