Journal of Systems Architecture 154 (2024) 103240

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Real-time intelligent on-device monitoring of heart rate variability with PPG sensors

Jingye Xu *, Yuntong Zhang, Mimi Xie, Wei Wang, Dakai Zhu

Department of Computer Science, The University of Texas at San Antonio, One UTSA Circle, San Antonio, 78249, TX, United States

ARTICLE INFO

Keywords:
PPG
HRV
On-device monitoring
Neural network
Neural architecture search

ABSTRACT

Heart rate variability (HRV) is a vital sign with the potential to predict stress and various diseases, including heart attack and arrhythmia. Typically, hospitals utilize electrocardiogram (ECG) devices to capture the heart's bioelectrical signals, which are then used to calculate HRV values. However, this method is costly and inconvenient due to the requirement for stable connections to the body. In recent years, photoplethysmography (PPG) sensors, which collect reflective light signals, have gained attention as a cost-effective alternative for measuring heart health. However, accurately estimating HRV using PPG signals remains a challenging task due to the inherent sensitivity of PPG sensors. To address the challenges, this paper presents an on-device, low-cost machine learning-based system that aims to achieve high-accuracy HRV estimation in real-time. Firstly, we propose a novel unified performance and resource-aware neural network (UP-RaNN) search method that leverages grid search techniques to identify a neural network model that can deliver both high HRV accuracy and smooth operation on resource-limited devices. Secondly, we design a real-time HRV monitoring system using a resource-limited, ultra-low-power microcontroller unit (MCU). This system utilizes the neural network model obtained through the UP-RaNN to provide HRV readings from PPG data in real-time. Thirdly, we evaluate the proposed UP-RaNN method and the real-time HRV monitoring system by comparing its performance to state-of-the-art studies. Moreover, the system is enhanced with adaptive reconfiguration capability, enabling it to improve energy efficiency and adapt to varying demands during runtime. The results demonstrate that when deployed on an MSP430FR5994 development board running at 8 MHz, the trained deep neural network model obtained through our proposed UP-RaNN achieves HRV estimation in just 0.3 s per inference. Additionally, the model exhibits a better mean absolute percentage error (~ 5.8%) than the state-of-the-art HRV estimation methods using PPG, while significantly reducing model complexity and computational time.

1. Introduction

Heart rate variability (HRV) is widely recognized as a crucial indicator of an individual's health, measuring the time interval between consecutive heartbeats [1]. Its analysis has been proven valuable in cardiology, with links observed between HRV and heart rate turbulence, maximal oxygen uptake, inflammatory response, and exercise capacity [2,3]. Traditionally, HRV assessment has relied on electrocardiogram (ECG) devices, which employ electrodes attached to the body to record the heart's bioelectrical signals [4]. ECG devices offer accurate HRV monitoring, making them the preferred choice for patients requiring intensive care in hospitals. However, these devices present certain limitations. Firstly, the requirement for electrode attachment to the skin imposes constraints on the applicability of ECG devices in various scenarios. Secondly, professional-grade ECG devices tend to be expensive, rendering them inaccessible to social-economically disadvantaged patients.

While ECG technology has been integrated into consumer electronics like the Apple Watch for everyday HRV monitoring [5–7], it still poses certain limitations due to specific operational requirements. For example, users of the Apple Watch's ECG module are instructed to rest their arms on a table or their lap and keep their fingers touching the Digital Crown (the button on the watch) to initiate HRV monitoring. This approach is inconvenient and impractical for continuous long-term monitoring [8]. In contrast, photoplethysmography (PPG) sensors have emerged as a promising alternative for measuring heart health [9]. These sensors capture reflective light signals from blood vessels and offer advantages such as low cost and convenience compared to ECG devices [10]. While PPG sensors do not directly provide the R–R

* Corresponding author.
E-mail address: jingye.xu@my.utsa.edu (J. Xu).

https://doi.org/10.1016/j.sysarc.2024.103240
Received 8 March 2024; Received in revised form 21 June 2024; Accepted 14 July 2024
Available online 18 July 2024

interval values needed for calculating HRV, they can extract "peak-to-peak" interval values that can be interpreted as the cardiac R–R interval [11]. Each peak in the PPG signal represents the occurrence of a heartbeat. However, estimating HRV from PPG sensor data remains challenging due to various sources of noise, particularly those related to motion artifacts. Accurately identifying the peak locations in the PPG signals is crucial to successful HRV estimation.

Two main approaches have been utilized to address the challenge of noise in data sensing and HRV estimation with PPG: *signal processing* [12,13] and *machine learning (ML)* [14,15]. While both approaches have demonstrated effectiveness in noise reduction and achieving high accuracy in HRV estimation, they have several limitations which are summarized as follows:

- One notable limitation of existing studies is their exclusive focus on improving HRV accuracy without considering the resource-heavy or costly implementation. Consequently, these approaches face challenges when it comes to being implemented in resource-constrained devices. Given that HRV is typically calculated within a time window ranging from half to five minutes [16], and PPG signals are commonly sampled at rates exceeding 100 Hz, a significant number of signals (3000~30,000) need to be stored to obtain a single HRV reading. This considerable memory requirement poses applicability issues for many devices constrained by limited memory capacity. An illustrative example is the widely recognized TROIKA framework [17], which has demonstrated high accuracy in heart rate (HR) estimation and presented the potential for accurate HRV estimation. However, the framework's reliance on a PPG sampling rate of 125 Hz implies that at least 3750 raw PPG signals are required to compute a single HRV reading. This substantial memory burden poses significant challenges for resource-constrained devices, making it difficult to implement the TROIKA framework in such environments.
- Another noteworthy observation is that existing ML-based HRV estimation methods often rely on specific platforms with neglecting compatibility considerations for deploying the proposed algorithms on various hardware, particularly embedded devices. For instance, Zhang et al. [18] introduced a compound method that combines signal processing and ML techniques to achieve highly accurate HRV estimation within a fast inference time. However, this method heavily depends on high-level tools like Python, along with relevant ML packages (PyTorch[1]), which necessitates additional transformations to execute the algorithm on different embedded devices. As a result, further analysis is required to assess the performance of these methods in real-world settings.
- Existing studies usually concentrate more on improving HRV estimation accuracy while less on considering the delay to generate an HRV reading. Considering the time-sensitiveness of real-world applications that need HRV monitoring such as driving fatigue detection [19], it is necessary to obtain real-time HRV estimation. Hence, it is imperative to evaluate the delay of the designed HRV monitoring system.

Considering that the PPG sensor is normally equipped on mobile devices that have restrained computing resources, limited memory, and precious energy, the approach we use for noise cancellation and HRV estimation should be practical and resource-aware of the device we deploy on.

To address existing works' limitations and make PPG-based HRV estimation feasible, we need to start from practice and design an intelligent HRV monitoring framework that could fit most embedded devices and provide a fast and high-accurate HRV estimation. In this work, we focus on **resource-aware deep-learning models** for HRV estimation with PPG sensing data and their deployment on ultra-low-power devices for **real-time monitoring**. Specifically, we first propose a unified performance and resource-aware neural network search method, named UP-RaNN that takes resource usage and performance into account. The search algorithm takes the deep learning (DL) model latency and model size into account to search for a fast and accurate HRV model that satisfies hardware memory constraints. Then, we design a real-time HRV monitoring system considering the limited resource of an ultra-low-power microcontroller unit (MCU). This system collects raw data from the PPG sensor and conducts several transforms before feeding the data into the HRV model to achieve real-time resource-saving HRV monitoring. In the end, we take an MSP430FR5994 development board as a case study to evaluate the monitoring system and the trained model's performance and energy efficiency. Moreover, the energy efficiency of the deep learning models with different operation modes on the device is also analyzed to support online adaptation. The experimental results show that the overall optimal model within the size limit can achieve a low Mean Absolute Percentage Error (MAPE) of 5.8% for HRV estimation, which has an inference time as low as 0.303 s on the MSP430FR5994 development board to support real-time monitoring of HRV.

The contributions of this work are summarized as follows:

- The UP-RaNN method is proposed to derive DL models for HRV estimation using PPG sensing data where the accuracy is comparable to the results obtained by the state-of-the-art methods;
- A real-time HRV monitoring system is designed for embedded devices considering their limited resources and their development environment. The designed system implements the trained low-resource occupation model obtained by our proposed UP-RaNN method to provide a low-latency and high-accurate HRV estimation;
- To accommodate various performance and energy efficiency demands (e.g., battery capacity) during runtime, we further enhance the system with adaptive reconfiguration capability based on the evaluation and analysis of the energy efficiency of different modules in the HRV monitoring system and the trained DL models under different operating modes.

The remainder of this paper is organized as follows. Section 2 presents some preliminary and background. Section 3 introduces our proposed unified neural network search method and real-time on-device HRV monitoring system. Section 4 discusses the experimental results and Section 5 presents our conclusion and future works.

2. Background and closely related work

2.1. HRV with ECG

The ECG is a traditional medical device widely used for obtaining accurate and real-time human vital signs. It provides real-time electrical signals of the human body, which depict the propagation of a stimulus through the ventricles [4]. Different methods of HRV quantification, both in the time-domain and frequency-domain, are employed to analyze human health based on the R–R intervals obtained from ECG devices [20]. Among them, the Root Mean Square of Successive Differences (RMSSD), a time-domain method, is one of the most widely used [21]. Eq. (1) shows the definition of RMSSD, where RR_i denotes the ith R–R interval, and N denotes the total number of R–R intervals in a given period.

$$\mathrm{RMSSD} = \sqrt{\frac{\sum_{i=1}^{N-1}(RR_{i+1} - RR_i)^2}{N-1}} \tag{1}$$

While ECG devices can offer precise HRV readings, their application is restricted due to stringent development requirements. Connecting electrodes to the skin is necessary for an ECG device, which restricts the

[1] https://pytorch.org/

subject's movements since any vigorous activity can result in unstable and inaccurate HRV readings. Additionally, professional ECG devices are often expensive, making them accessible only within a hospital setting.

2.2. PPG and HRV estimation with learning models

In contrast to ECG devices that directly provide R–R intervals for HRV calculation using Eq. (1), PPG sensing data only allows the derivation of "peak"-"peak" intervals between pulsations, which can be used as an approximation of cardiac R–R intervals [11]. Therefore, an alternative method to obtain HRV is by estimating these PPG "peak"-"peak" intervals and subsequently deriving R–R intervals for HRV calculation, as represented by the RMSSD formula in Eq. (1). Indeed, Eq. (1) demonstrates that even slight variations in R–R interval values can result in significant differences in HRV. Consequently, minor noise in "peak"-"peak" intervals derived from PPG sensing can lead to varying R–R intervals and ultimately substantial discrepancies in calculated HRV [22,18]. For example, a small 1% MAPE in R–R interval estimations amplifies to over 10% error for RMSSD based on our previous study [18]. Furthermore, PPG "peak"-"peak" intervals are susceptible to interference from motion artifacts, environmental lighting, and other sources of noise [17]. Therefore, accurately estimating HRV using PPG sensors remains a challenging task.

ML models have been utilized in existing studies to estimate HRV [23–27]. Wittenberg et al. [23] tried a few neural network models in locating peaks on TBME [28] and TROIKA [17] datasets and found that the 3-layer gated recurrent units (GRU) outperform 1-layer convolutional neural network (CNN). Alqaraawi et al. [24] used Bayesian learning to mitigate the effects of human artifacts when detecting the PPG peaks. However, all these studies focused on peak detection from PPG signals.

Rather than peak detection, Xu et al. [29] applied a bidirectional long short-term memory (biLSTM) model on accurate PPG cardiac period segmentation and pulse rate variability (PRV) estimation under strenuous physical exercise where PPG waveforms are contaminated by strong motion artifacts. Moreover, Hong-Yu et al. [15], Luke et al. [14] employed ML to reconstruct ECG-like signals through PPG signals.

The aforementioned PPG-based HRV estimation studies have focused on predicting HRV **indirectly**. Previous studies have primarily focused on predicting R–R intervals from detected peaks or attempting to construct ECG-like signals. Besides, the evaluation of their proposed approaches has not included performance metrics specifically related to HRV prediction, such as RMSSD. However, performance in predicting R–R intervals does not directly reflect the effectiveness of their methods in predicting HRV. Therefore, it is crucial to directly predict HRV instead of solely focusing on R–R interval prediction. Zhang et al. [18] proposed a comprehensive approach that combines signal processing and machine learning for direct HRV estimation. They proved that the direct HRV estimation outperforms the indirect HRV estimation. However, their method was implemented using Python and scikit-learn, which may not be directly deployable on devices with limited resources and software packages, such as MSP430FR5994. In contrast, our method does not require the use of high-level programming languages or machine learning packages, offering a more accessible and efficient approach.

Compared to existing works, our proposed HRV monitoring approach takes into full consideration the limitations of resources, and we have evaluated the UP-RaNN models and the real-time HRV monitoring system after deploying the system on a resource-constrained device. Furthermore, our method directly predicts HRV values rather than calculating the HRV with predicted HRs, which can avoid amplifying the errors and is a notable advantage compared to other approaches.

2.3. Network architecture search for resource-limited devices

Network Architecture Search (NAS) is an advanced technique in the field of machine learning and artificial intelligence that focuses on automating the design of neural network architectures, which could significantly reduce human efforts when the network design space is tremendously large [30]. NAS aims to streamline this process by leveraging algorithms that can systematically explore and evaluate a vast space of potential architectures, identifying the most effective configurations with minimal human intervention. Unlike early NAS methods [31] that mainly focus on searching for a global optimal architecture in terms of accuracy, NAS for resource-limited devices requires additional attention on model size, model latency, and energy cost.

Several researchers have studied NAS for resource-limited devices. On-NAS [32] is proposed to provide a memory-efficient on-device NAS, which can reduce the massive memory requirement of NAS on the device. However, the On-NAS could deploy on a Jetson Nano equipped with 2 GB memory, but cannot fit the embedded system. LC-NAS [33] is proposed specifically for RISC-V devices to achieve a training-free NAS method. To achieve this, LC-NAS utilizes a lookup table and latency predictor to provide precise latency measurement. Yang [34] employed adaptive dataflow patterns to achieve hardware-aware search, and utilized the latency as a constraint directly, to reduce the number of sampled useless networks and improve the searching efficiency. FastStereoNet [35] is based on late acceptance hill-climbing, followed by simulated annealing, and considers the estimated network inference time along with accuracy as the search objectives to discover resource-efficient architectures. LightNAS [36] is proposed to try to find the required architecture that satisfies various performance constraints through a one-time search. However, there are still some limits when employing those frameworks on embedded systems. Some of the aforementioned frameworks are specifically for one platform, making them infeasible for other platforms. For example, LC-NAS is designed specifically for the RISC-V platform. Some of them seem to reduce memory utilization, but the memory requirement is still so high that cannot fit embedded systems. For example, the smallest memory utilization of On-NAS is above 20 MB, while the embedded system may only have tens of KB available. Moreover, although these frameworks indeed considered the model latency, they did not consider the model size, which is also an important factor that affects the deployment of the model on an embedded system. Therefore, we want to design a general algorithm that can work upon existing NAS methods, making the NAS process more efficient. The basic idea here is that our proposed UP-RaNN method is a multi-objective method that considers both model size and model latency, which can help to reduce model candidates significantly without training those useless models, making the NAS process more efficient.

2.4. Challenges in real-time on-device monitoring

Existing works for estimating HRV provide good performances in predicting R–R intervals, or ECG-like signals. However, these models are generally too complex to be adopted in resource-constrained devices. For instance, the model of BioTranslator [14] has more than 40,000 parameters, which is too large for wearable devices. In addition, several works use the Long Short-Term Memory (LSTM) architecture that is energy and computation-hungry [23,29], which is not suitable for wearable devices with limited battery energy as well. Furthermore, existing works ignored the algorithm's compatibility. The TROIKA framework [17] and Zhang et al. [18] utilize specific platforms with high-level packages, like Python and MATLAB that require additional conversion to adapt to resource variations on different embedded devices.

Besides ignoring the resource-constrained hardware and algorithm compatibility, these offline methods normally cannot provide real-time monitoring of HRV. There are cases where it is necessary to

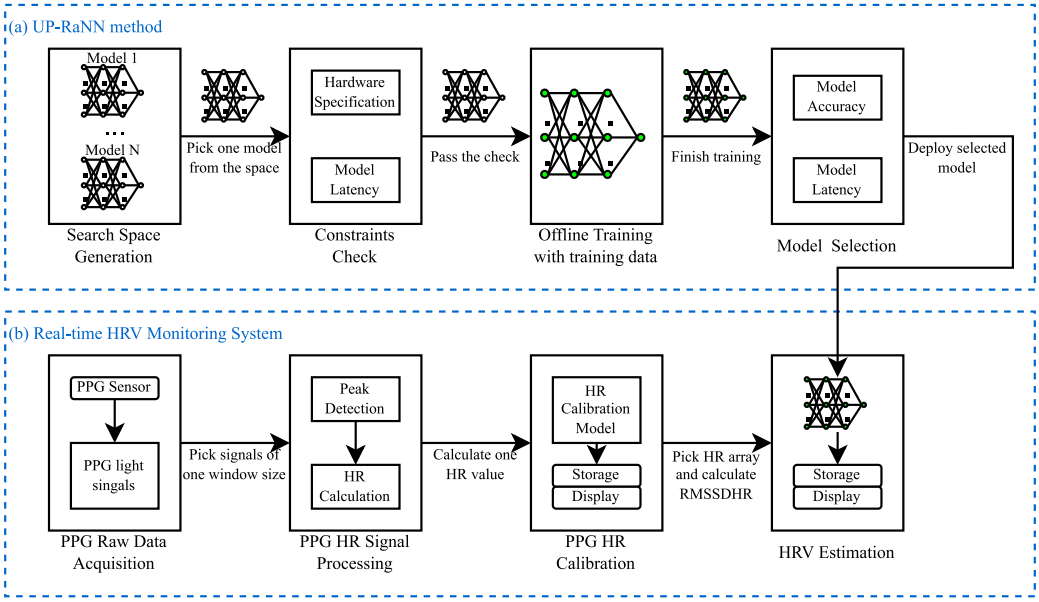


Fig. 1. Overview of the PPG-based Real-time HRV Monitoring.

provide real-time HRV monitoring, for instance, to support self-health monitoring or timely alerts in driving fatigue detection [19].

To overcome the challenges mentioned earlier, it is essential to develop a universal method that can be easily deployed on various devices without requiring additional effort. This method should offer efficient and accurate real-time on-device HRV monitoring capabilities. By developing such a method, we can ensure widespread accessibility, convenience, and reliability in monitoring HRV across different devices.

3. Intelligent real-time on-device HRV monitoring

3.1. Overview

Fig. 1 illustrates the overview of the proposed work, which consists of two main parts: the offline UP-RaNN method using a labeled dataset and the online real-time on-device HRV monitoring system. In Fig. 1(a), the UP-RaNN takes into account the constraints of the device, such as hardware-specific limitations (specified as model size) and performance requirements (specified as model latency). This method trains size-limited deep neural network (DNN) models to estimate HRV based on offline collected PPG sensing data, using HRV readings from ECG as the ground truth. In Fig. 1(b), we developed a real-time HRV monitoring system. The real-time HRV monitoring system works with the model retrieved from our UP-RaNN method and can be deployed on various devices. To maintain real-time monitoring, the system tries to reduce computing workload by reducing data size without sacrificing HRV model accuracy. In the end, we evaluate our UP-RaNN searched model and real-time HRV monitoring system by analyzing the energy consumption and latency on an MSP430FR5994 development board. Besides, we propose multiple working modes that allow for runtime system reconfiguration to adapt to various demands, like changing system frequency to reduce power consumption when the battery level is low. These modes ensure optimal performance while considering specific requirements and constraints.

3.2. Unified performance and resource-aware neural network search (UP-RaNN)

As mentioned above, we propose a unified neural network search method that guarantees the trained neural network model not only fits the various hardware but also provides a fast and high-accurate HRV estimation. To achieve this, the NAS process will be guided by two constraints: **model size** and **model latency**.

3.2.1. Data preprocessing

In our study, we utilized an ECG device to acquire accurate HR/HRV measurements, which served as the ground truth for providing label information during model training and testing phases. To capture the PPG signals, a PPG sensor was attached to the finger and the light signals will be obtained synchronously along with the ECG R–R interval values. Instead of feeding the light signals into our model directly, we use a signal processing algorithm [37] to estimate peaks in four seconds to predict an HR every second with a sliding window of one-second step. This strategy brings two distinct advantages. Firstly, we enhance computational efficiency by significantly reducing the input data size. Secondly, our approach enables adaptation to various sampling rates of data, making it versatile and applicable across different scenarios and datasets. Then the signal-processed HR (calculated HR) will be the input of the HR calibration model. The HR model's labels are calculated through Eq. (2), where $RR_{average}$ denotes the average interval between that period in milliseconds obtained from ECG. Therefore, it estimates how many beats there are in one minute (60,000 ms) when the interval time between every two beats is $RR_{average}$. The HR model is utilized to mitigate the bias between HR and ECG-based HR. In this work, we mainly focus on the HRV estimation model and thus the HR model can be treated as a preprocessing phase for "calibrating" the HRV model's features (historical HR values).

$$HR = 60,000/RR_{average} \tag{2}$$

After that, the Root Mean Square of Successive Differences in HR (RMSSDHR) is extracted through Eq. (3), where HR_i denotes the ith HR, and N denotes the total number of HR in a given period. The period can be 30–300 s for different demands. Eq. (3) is inspired by combining Eq. (1) and (2). Although PPG sensors cannot directly retrieve R–R intervals, using Eq. (3) allows the HRV model to learn the relations between HRV and the variations in HR directly. The label, which is ECG-based HRV, is calculated with Eq. (1).

$$RMSSDHR = \sqrt{\frac{\sum_{i=1}^{N-1}(60,000/HR_{i+1} - 60,000/HR_i)^2}{N-1}} \tag{3}$$

3.2.2. Constraints definition for UP-RaNN

Model size and **model latency** are two main constraints to guide the neural network search. Among them, the model size is constrained by the hardware's storage capacity and can be estimated by Eq. (4), where S_m denotes the model size, N_{para} denotes the number of trainable

parameters of the neural network, and U denotes the occupied size of one parameter (e.g., four bytes). When conducting the neural network search, we pass a specific model size for reference so that every to-be-trained model that exceeds the model size will be rejected to save time and guarantee the model's deployability. The model latency can be estimated by Eq. (5), where T denotes the model latency, $T_{preprocess}$ denotes the central processing unit (CPU) cycles used to calculate the $RMSSDHR$, T_n denotes the model forward propagation CPU cycles in layer n, and $FREQ$ denotes the CPU frequency. For a model having two hidden layers, the N equals three (T_1 denotes input → hidden layer1, T_2 denotes hidden layer1 → hidden layer2, T_3 denotes hidden layer2 → output). When we set a computation latency as a threshold, Eq. (5) estimates whether the model could fulfill the latency requirement. With these two constraints, we can make sure that the trained models can be deployed on the MCU and provide real-time HRV monitoring.

$$S_m = N_{para} \times U \tag{4}$$

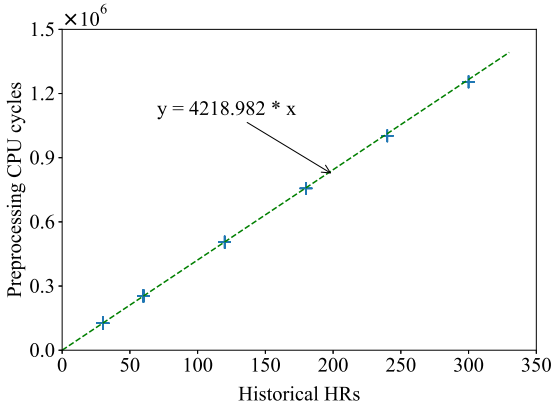
$$T = (T_{preprocess} + \sum_{n=1}^{N} T_n)/FREQ \tag{5}$$

To obtain the $T_{preprocess}$ and T_n, which are highly related to specific system settings of the target device, experiments are required to be conducted on the target device. The basic concept entails deploying multiple random models on the target device and measuring the CPU cycles consumed during different phases, specifically preprocessing and forward propagation. By analyzing the statistics of these CPU cycle measurements, we can predict the CPU cycles required for both preprocessing and forward propagation tasks. This approach allows us to estimate the computational resources needed for these phases and optimize the deployment of models on the target device.

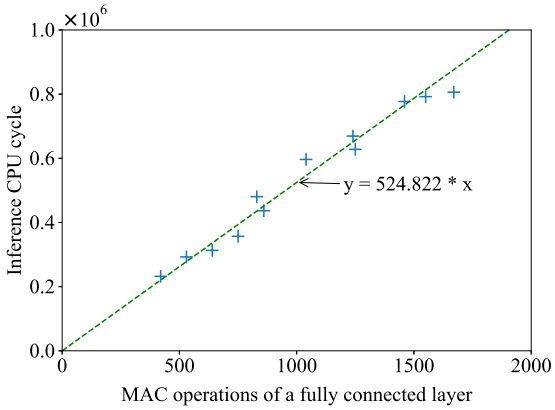
Fig. 2 shows the CPU cycles of the neural network model running on an MSP430FR5994 development board as an example. Fig. 2(a) depicts the CPU cycles elapsed for calculating the $RMSSDHR$, and Fig. 2(b) and (c) show the CPU cycles when inference conducted through a fully connected layer and a convolutional layer. It can be found that both preprocessing and MAC operations have an approximately linear relation with the CPU cycles and therefore we can estimate the preprocessing time through the number of historical HRs and estimate the forward propagation inference time through the number of multiply–accumulate (MAC) operations. Consequently, the total computational time can be obtained through Eq. (5). Here we do not need to conduct a complete accurate CPU cycle estimation as an approximate value is sufficient. Please note that, in our case, a linear function is sufficient to predict the CPU cycles, but it may not be suitable for more complex model structures. Therefore, utilizing a multilayer perceptron (MLP) to predict the model latency appears to be more practical when model structures are more complicated [36].

3.2.3. Search space for UP-RaNN

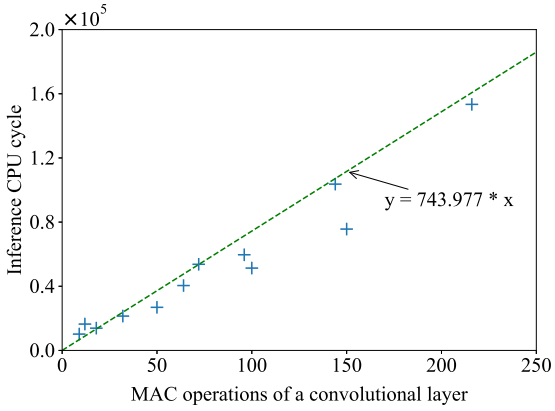
Grid search and random search are two common neural network search strategies for neural network exploration [38]. In this work, we choose grid search as the basic search strategy. Although a traditional grid search is time costly compared to a random search as it permutes all the possible combinations of a given search space, it benefits from finding the globally optimized neural network model. Therefore, to find the globally optimized neural network model while not spending much time, we can try to shrink the search space by setting thresholds, as mentioned in Section 3.2.2, thus making the grid search feasible and efficient in our work. When generating search space, the maximum possible hidden layers and the maximum possible neurons of each layer are set to determine the search space. The input size is determined by the application scenarios and can be any time in the set (30 s, 60 s, ..., 300 s) plus one calculated $RMSSDHR$. During the fine-tuning process of the neural network model, we can consider different batch sizes, learning rates, and maximum epochs to optimize the model's performance. This flexibility allows for customization and fine-tuning



(a) Preprocessing vs. Historical HRs.



(b) Inference vs. MAC Operations in FC layer.



(c) Inference vs. MAC Operations in CONV layer.

Fig. 2. CPU cycles estimation with historical HRs and MAC operations.

of the model based on specific requirements and datasets in different scenarios. Therefore, our UP-RaNN method could apply to other research fields besides HRV estimation. By incorporating model size and model latency constraints into the process, we can effectively narrow down the range of possible neural network configurations. This allows us to focus on a subset of neural network settings that meet the desired criteria for model size and model latency. By reducing the search space, we can streamline the neural network search process and improve the

Table 1
Search space attributes.

Attributes	Descriptions	Search Space 1	Search Space 2
maxConv	Maximum possible convolution layers	0	3
maxChannel	Maximum possible channels for each convolution layer	NA	5
kernelSize	Kernel size options, two dimensions	NA	[(1,2)]
strideSize	Stride size options, two dimensions	NA	[(1,2)]
padSize	Padding size options, two dimensions	NA	[(0,1)]
pooling	Pooling options	NA	["max"]
poolKernel	Pooling kernel size options, two dimensions	NA	[(1,2)]
poolStride	Pooling stride size options, two dimensions	NA	[(1,2)]
poolPad	Pooling padding size options, two dimensions	NA	[(0,1)]
maxLinear	Maximum possible fully connected layers	8	3
neurons	Neuron options for each fully connected layer	$2^n, n = 1..7$	
activation	Activation function options	relu	

efficiency of finding the optimal model configuration within the given constraints.

3.2.4. UP-RaNN search

Algorithm 1 UP-RaNN search

```
function UPRaNN(searchSpace, maxEpoch, sizeT, timeT)
    modelCandidates←modelGeneration(searchSpace)
    for model in modelCandidates do
        if size(model) > sizeT then
            continue
        else if latency(model) > timeT then
            continue
        end if
        TRAIN(model, maxEpoch)
    end for
end function
function TRAIN(model, maxEpoch)
    epoch ← 1
    while epoch ≤ maxEpoch do
        modelTraining(model)
        modelSave(model)
        if loss goes up and exceeds a threshold then
            break
        end if
        epoch += 1
    end while
end function
```

Algorithm 1 shows the pseudocode of the grid search phase. In the initial phase of our approach, several input parameters are given, namely the search space, maximum epoch, size threshold, and time threshold, which are represented as searchSpace, maxEpoch, sizeT, and timeT in the algorithm, respectively. Please note that unlike the other three parameters, the search space is a multi-dimensional parameter, and its attributes and descriptions are detailed in Table 1. The attributes are expandable so that the modelGeneration function in our algorithm can take care of more network structures in the future. In this paper, we only focus on the convolution and fully connected layers. We then iterate through all possible neural network models based on the given search space and for each generated neural network model, we assess its compatibility with the hardware and its ability to provide fast inference by comparing it against the provided size and time thresholds. If the model does not exceed all thresholds, we proceed with training. On the other hand, if the model exceeds either threshold, we discard it and move on to the next model configuration. Additionally, an early exit strategy is implemented during training. If the loss starts to increase and surpasses a predefined epoch threshold, the training process is terminated early.

3.2.5. Model selection

Our ultimate objective is to realize on-device real-time heart rate variability monitoring using models selected through our UP-RaNN model search method. The deployment of the model into the real-time monitoring system necessitates the selection of suitable models from the search outcomes. Model accuracy and model latency represent the two objectives of our research and how to calculate a unified score to help select the optimal model is shown in Eq. (6). Here, $Score$ denotes the overall score of a model, $loss_n$ denotes the test loss of the model n, $latency_n$ denotes the latency of the model n, $f1$ and $f2$ are two positive factors associated with model loss and model latency. They can help to normalize the model loss and model latency to make sure the two metrics are comparable. namely, the factors are used to adjust the weights of the two matrics. In addition, $b1$ and $b2$ are two biases. Eq. (6) illustrates that a lower loss corresponds to a higher score, while a shorter latency leads to a higher score as well.

$$\text{Score} = (b1 - loss_n/f1) + (b2 - latency_n/f2), n \in \{models\} \tag{6}$$

3.3. Real-time on-device HRV monitoring system

In this section, we will present the real-time on-device HRV monitoring system that we have developed. As previously mentioned, the design of this system incorporates the constraints of embedded systems, such as low CPU frequency and limited memory size. Our primary objective is to provide real-time HRV monitoring without compromising the accuracy of HRV estimation. Additionally, the system only requires common components, such as a float arithmetic logic unit, I2C communication unit, and non-volatile storage, to make it versatile and compatible with various devices. To fulfill the requirements, we design the HRV monitoring system as shown in Fig. 1(b).

3.3.1. Raw data transform

Initially, the light signals of the PPG sensor will be stored in a buffer on the MCU so that the signal data array can be used to estimate peaks by the signal processing function. The size of the buffer is determined by the sampling rate times four seconds. For example, when the sampling rate of the sensor is set to 25 Hz, the buffer size is 100 (25×4) with each value being a four-byte floating-point number. To achieve greater data size compression, data quantization is typically employed. However, in the context of HRV prediction, we refrain from using data quantization. Since HRV prediction is a regression problem, inappropriate data quantization could substantially amplify the error. The signal processing phase can effectively reduce the number of input features and thus reduce model sizes. Given original light signals in 300 s and 25 Hz sampling rate, then the input size becomes 7500, which is too huge for a resource-constrained device. With the help of signal processing, the number of inputs can be reduced from 7500 to 300. Moreover, this could also enable our method to adapt to various

sampling rates of data. Once we get the estimated peaks from signal processing, the calculated HR can be obtained by Eq. (7). The basic idea here is that once we estimate how many peaks there are in four seconds (the time is determined by buffer size divided by sampling frequency), we can calculate HR by estimating how many peaks there are in 60 s. We only store light signals in the RAM to ensure enough memory space for HR signal processing, model inference, and preprocessing in HRV estimation.

$$HR = peaks * 60/4 \tag{7}$$

3.3.2. HR "calibration"

According to Eq. (1), the RMSSD can change significantly by a minor change in the R–R interval, that is, any minor changes happening on HR values would lead to totally different RMSSD readings. Therefore, in our designed HRV monitoring system, the calculated HR will be passed through an HR "calibration" model, which is trained by taking ECG as the ground truth, to filter out the noise and minimize the bias between the PPG sensor and the ECG device. The HR "calibration" model has four hidden layers, each having 16, 32, 16, and 8 neurons, respectively. The effectiveness of combining signal process and machine learning in increasing HR estimation has already been proven by our previous work [18], thus we integrate the theory and implement it in our real-time monitoring system.

3.3.3. Historical HR/HRV storage

Given that the monitoring system is intended to operate on a resource-constrained embedded device with limited battery capacity, we have incorporated the use of non-volatile storage. This allows us to store the predicted HR and HRV data, ensuring that historical information is preserved even in the event of a power outage.

3.3.4. Real-time monitoring

When monitoring HRV in real-time, a N-second sliding window is determined and we will retrieve all historical HRs during that window to estimate an HRV. In our paper, since we focus on predicting RMSSD as it is the most used HRV indicator, the HR array will be sent to a preprocessing function to calculate the RMSSDHR based on Eq. (3). After that, we use the calculated RMSSDHR and the HR array as the input features to predict the HRV, which is RMSSD in our paper. Both the predicted HR and predicted HRV will be sent to a UART client to provide real-time monitoring. Moreover, to fulfill various needs, we could specify different step lengths for the sliding window.

3.4. Adaptive runtime reconfiguration

The performance of the model, battery life, and model latency are intuitively influenced by factors such as model size, CPU frequency, and sensor sampling rate. The memory capacity of the MCU limits the PPG sensor's sampling rate and model inference intermediate memory usage, while the model size determines the size of the model that can be deployed. Generally, larger model sizes require more computational time and result in shorter battery life as the CPU remains occupied.

Different models exhibit varying sizes and performance levels. Moreover, a higher sampling rate negatively impacts battery life and computational time, as it necessitates larger input sizes for signal processing. On the other hand, a higher CPU frequency accelerates computational time but consumes more power.

Consequently, various system configurations offer distinct performances and can adapt to different requirements. For instance, if users prioritize faster computational time and are willing to compromise on battery life, they would configure the MCU at a higher frequency. If energy conservation is of greater importance while accepting longer inference times, a lower frequency and lower sampling rate configuration can be chosen. Therefore, runtime reconfiguration enables users to switch between different operation modes without the need for reprogramming the device.

To empower the system with adaptive reconfiguration capability at runtime to improve energy efficiency and adapt to different demands, we prepare different system settings on the MCU. When a reconfiguration is needed, the system will stop the data pipeline in Fig. 1, switch the configuration, and restart the pipeline to apply the new configuration, where the first HRV estimation would be available only after a certain time interval (e.g., 300 s) from the reconfiguration time point. In other words, our developed real-time HRV monitoring system is fully capable of supporting runtime reconfiguration without compromising its functionality, albeit resulting in only a brief service interruption.

4. Results evaluation

4.1. Experiments setup

For evaluating our proposed UP-RaNN method and the real-time HRV monitoring system, we select the MSP430FR5994 development board as our experimental platform. The MSP430FR5994 offers a comprehensive development platform for ultra-low-power MCU. It includes an on-device probe for programming and debugging, supports a variety of communication protocols, and contains all necessary hardware components, making it ideal for verification experiments. We selected the MSP430FR5994 primarily because it is the platform we currently have available and it has the most limited resources. Besides, this board is a typical embedded device which can represent a large number of embedded systems that has two layer on-chip memory and limited CPU frequency. Our method can be easily transferred to other platforms as long as the platform supports the necessary hardware components our method requires. Following the UP-RaNN method, we obtained suitable HRV models and integrated them into our real-time HRV monitoring system. Therefore, the evaluation contains two parts: evaluate the effectiveness and efficiency of the UP-RaNN method and evaluate the performance of online real-time HRV monitoring.

To prepare the dataset for the proof of concept, a MAXREFDES117 PPG sensor is selected and connected to a Raspberry Pi 4 to collect the PPG data, and a 3-lead TLC5007 dynamic ECG device is utilized to provide the data ground truth. The PPG sensor and ECG device are carefully synchronized in time to guarantee simultaneous data recording. To ensure comprehensive data collection and prevent underfitting, the subject is instructed to provide three sets of data, each corresponding to different activities: sitting, sleeping, and daily life. Each activity is recorded for two hours to capture sufficient data for analysis. During the training phase, the collected data is split into training and testing sets with a ratio of eight to two.

When applying our proposed UP-RaNN method, we set two constraints based on the hardware. The selected MSP430FR5994 supports an MCU frequency of up to 16 MHz and is equipped with 256 KB FRAM and 8 KB SRAM. To make sure the explored model can fit the MSP430FR5994, the two constraints are set as follows: the model should not exceed 4000 parameters and the model latency should not exceed two seconds. When estimating the model latency, we set the CPU frequency to 8 MHz, that is the maximum number of allowed CPU cycles in computation is 16,000,000 ($8\ \text{MHz} \times 2\ \text{s} = 16,000,000$). We devised two search spaces to assess our UP-RaNN method thoroughly: one includes solely linear layers, while the other incorporates both convolutional layers and linear layers. In our case, we focus on the global optimal network model composed of both linear and convolutional layers. Therefore, the two search spaces are sufficient as they can cover most combinations of these layers. A search space with only convolutional layers is impractical because we typically need to append several linear layers at the end of a CNN model to solve regression problems. The details can be seen in Table 1. We find that most of the models do not have more than three fully connected layers, such as ResNet [39], GoogLeNet [40], and VGG [41], thus we set the maximum hidden layer to be less than three linear layers in our Search Space 2.

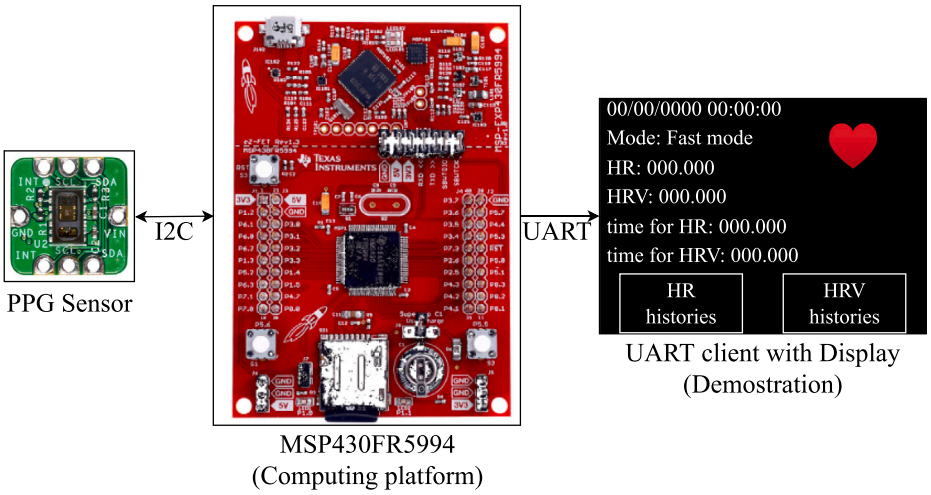


Fig. 3. Connections of the real-time on-device HRV monitoring.

Table 2
Search results of UP-RaNN method.

Search Spaces and methods	Search Space 1 (pure Linear)		Search Space 2 (Linear + Conv)	
	Grid Search	UP-RaNN	Grid Search	UP-RaNN
Settings	6,725,600	612,130	61,845	**45,047**
MAE of top 1 percent	3.2 ± 1.0	3.3 ± 1.1	**1.8 ± 0.4**	**1.8 ± 0.4**
Average MAE of top 1 percent	3.3	3.7	**2.0**	2.1
Best MAE	2.2	2.2	**1.4**	1.5
MAPE of top 1 percent	10.0 ± 3.1%	10.3 ± 3.3%	**5.5 ± 1.2%**	5.7 ± 1.1%
Average MAPE of top 1 percent	10.1%	11.4%	**6.4%**	6.5%
Best MAPE	6.9%	7.0%	**4.4%**	4.6%

For our Search Space 1, since we do not have any convolutional layers there, we slightly increased the maximum number of possible linear layers to eight. Moreover, we only provide one option for kernel size, stride size, padding size, and activation function for proof of concept, although we can add more possibilities. The reason is our data is time series one-dimensional data, padding in two dimensions seems not necessary. When training, the learning rate is set to 0.005, and the batch size is set to 32. We choose the mean squared error (MSE) as the loss function.

When evaluating the designed real-time on-device HRV monitoring system, the MSP430FR5994 development kit receives light signals from a PPG sensor, conducts HRV estimation, and then sends the results to a UART client for real-time monitoring. The UART client can decode the messages and show the monitoring results on a display with a resolution of 320 × 240. In our experiment, for the convenience of evaluating the power consumption, there is a multi-meter between the UART client and the MSP430FR5994 so that we can easily measure current in real-time. The schematic connection is shown in Fig. 3.

4.2. Effectiveness and efficiency of UP-RaNN method on HRV estimation

As mentioned above, we set the maximum number of allowed trainable parameters to 4000 and the estimated total CPU cycles to be no larger than 16,000,000 to reduce the search intensity and make the searched neural network compatible with the given development board. The search results of our UP-RaNN are shown in Table 2. In the table, Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) are provided to demonstrate the benefits of our UP-RaNN search method on two different search spaces. Our UP-RaNN method achieves notable acceleration compared to the grid search, attributed to our hardware-aware strategy. Moreover, the UP-RaNN method applied to the pure Linear search space experiences a speed-up of up to ten times due to its expansive search scope. This underscores that a larger search space correlates with greater speed-up, demonstrating the efficiency of our UP-RaNN method. Although the exhaustive grid search yields better results than our method, the difference is minimal. The grid search shows only a 0.1–0.2% improvement in best MAPE and at most a 0.1 improvement in best MAE compared to our method. This slight advantage does not justify spending ten times more training time, especially considering that the resulting models cannot be deployed on the chosen platform. Furthermore, the search results indicate that Search Space 2 surpasses Search Space 1 in both MAE and MAPE, demonstrating that the CNN network outperforms the pure linear network.

Table 3 shows the top three models from Search Space 1 and 2 when the sort keys are MAPE, MAE, and Total CPU cycles, respectively. To represent the model network, we add "l" for a linear layer followed by the number of neurons of that layer and add "c" for a convolutional layer followed by the number of channels of that layer. There are 301 features as the input (300 historical HR plus one RMSSDHR) and one neuron as the output which will predict an HRV value. We do not include input and output in the network representation to simplify the representation. As depicted in Table 3, models c1-l2 and l2-l2-l2 fail to converge during training, attributed to poor parameter initialization. Models c1-c1-l2 and c1-l2-l2 both demonstrate minimal total CPU cycles, indicating they demand limited computational resources for deployment. However, their performance in terms of MAE and MAPE is suboptimal due to the limited number of trainable parameters utilized. Additionally, an intriguing observation is that within the provided search spaces, networks comprising purely linear layers exhibit worse MAE and MAPE compared to those incorporating convolutional layers, while consuming only half of the CPU cycles. That is, convolutional layers help to increase accuracy in estimating HRV but require more computational resources. Furthermore, model c1-c3-c2-l32-l2-l2 as one optimal solution is appended to the table. It can be observed that model

Table 3
HRV estimation models results.

Model	Trainable Parameters	Total CPU cycles	MAE	MAPE	FRAM	Model	RAM
c1-l2	160	1,719,519	4.389	13.444%	35,094B	9,388B	4,670B
l2-l2-l2	619	1,609,636	4.442	13.770%	33,350B	10,348B	4,670B
c1-c1-l2	51	1,723,239	3.160	9.564%	35,210B	9,188B	4,670B
c1-l2-l2	166	1,723,983	2.931	9.178%	35,140B	9,428B	4,670B
c2-c4-c4-l64-l16	2,719	4,791,401	1.459	4.631%	35,384B	22,372B	4,670B
c3-c5-c3-l128-l4-l128	3,794	6,327,713	1.507	4.826%	35,446B	29,188B	4,670B
c3-c5-c3-l64-l4-l128	2,322	5,232,579	1.568	4.959%	35,446B	22,788B	4,670B
l4-l64-l4-l32-l4-l32-l4-l128	3,141	2,861,951	2.224	6.956%	33,604B	22,564B	4,670B
l4-l16-l64-l8-l16-l4-l4-l64	3,513	3,119,519	2.224	6.964%	33,604B	23,316B	4,670B
l4-l4-l64-l4-l64-l8-l2-l64	2,923	2,773,134	2.228	6.975%	33,598B	21,228B	4,670B
l4-l32-l32-l8-l32-l8-l2-l64	3,515	3,119,519	2.225	6.978%	33,604B	23,340B	4,670B
c1-c3-c2-l32-l2-l2	517	2,343,716	1.860	5.792%	35,430B	11,932B	4,670B

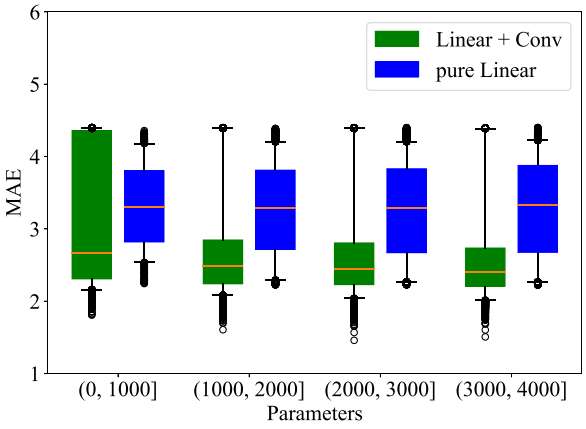
c1-c3-c2-l32-l2-l2 sacrifices 25% accuracy but reduces the CPU cycles by about 50%, compared to model c2-c4-c4-l64-l16.

Additionally, Table 3 provides the actual memory consumption when models are deployed on our tested platform. In the table, the FRAM size primarily indicates how much memory our Real-time HRV monitoring system occupies when integrating various models. Thus, we can observe that the program size changes minimally when switching to a similar model. The Model size reflects how much memory is utilized to store the model itself, and it increases according to the number of trainable parameters in a model. However, the RAM size remains consistent across all models. This is because the RAM size represents the size of global and static variables and the software system stack in our monitoring system. Since the stack size is pre-set and the global and static variables do not change when different models are used, the RAM size remains the same.

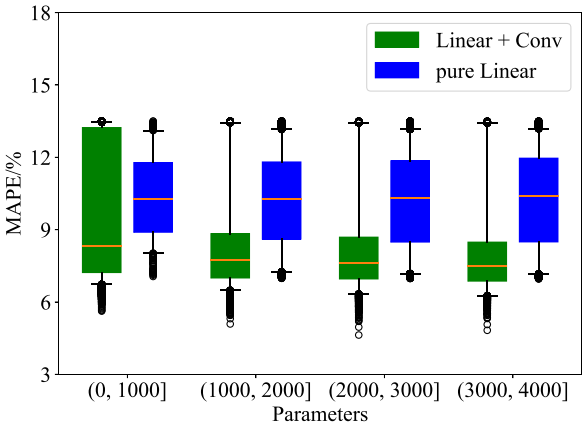
To conduct a more in-depth analysis and comparison of our UP-RaNN method results across two search spaces, Fig. 4 displays the performance distributions of MAE and MAPE, respectively. Box plots have been utilized in the figures to illustrate the positions of the first quartile (Q1), third quartile (Q3), and the mean of the results, while circles denote outlier points. Significantly, the search space incorporating both linear and convolutional layers demonstrates superior performance compared to the search space containing pure linear layers, as evidenced by both MAE and MAPE metrics. Furthermore, both search spaces exhibit loss reduction as the number of trainable parameters increases. However, it is important to note that enhancing performance is not indefinite with the continual increase of trainable parameters, as excessive parameter inflation can result in overfitting issues.

When selecting the optimal models, Eq. (6) is converted to Eq. (8). Here we chose MAPE to represent the model loss and CPU cycles to represent the model latency, respectively. As MAPE typically ranges from 0 to 100, we set $f1$ to 100 to scale the range between 0 and 1. Considering that CPU cycles are related to hardware constraints, we set $f2$ to 16,000,000 to normalize the range between 0 and 1. Both $b1$ and $b2$ are assigned values of 1. Then, we can calculate the score to select the optimal models. To be noticed that the unified score could help us to find the model that has the highest score, while could not provide the alternatives unless two optimal models have the same score. To better explain the model selection phase, the Pareto front results are shown in Fig. 5. The Pareto front, also referred to as the Pareto frontier or Pareto curve, represents the set of all Pareto-efficient solutions. In the accompanying figure, each dot corresponds to a neural network model within the Pareto front. The bottom-left point signifies the ideal point as our objective is to attain the lowest MAPE while minimizing time expenditure. The figure illustrates that the search space incorporating both linear and convolutional layers outperforms the search space containing only linear layers. Furthermore, the optimal model, identified as model c1-c3-c2-l32-l2-l2, is readily discernible in the figure. Alternatively, the model c2-c4-c4-l64-l16, which exhibits the lowest MAPE, can be selected when aiming for the highest model accuracy.

$$\text{Score} = (1 - MAPE_n/100) + (1 - CPUcycle_n/16,000,000) \quad (8)$$



(a) MAE vs. Parameters.



(b) MAPE vs. Parameters.

Fig. 4. Performance distribution of UP-RaNN search method on two search spaces.

4.3. Comparison with existing studies in HR/HRV prediction

Table 4 shows the performance of our UP-RaNN method compared with that of existing frameworks. TROIKA [17] and BioTranslator [14] both use ISPC dataset [17]. We implement our method on the ISPC dataset first for a direct comparison, and then we introduce our method with our dataset. In the comparison, model c2-c4-c4-l64-l16 is selected. The results show that the TROIKA framework has the best accuracy in HR estimation. However, it relies on singular spectrum analysis that is very complicated [42] and takes almost more than 1.9 s to estimate one HR when we simulate this framework using MatLab on Windows 10 powered by i7-9700k PC (eight cores, 3.6GHz, and 32 GB

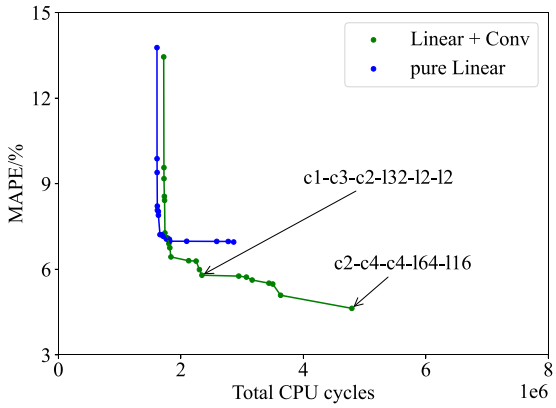


Fig. 5. Pareto frontiers of UP-RaNN search method on two search spaces.

Table 4
Comparison with Existing Framework.

Framework	HR		HRV (RMSSD)		
	MAPE	Latency/Platform	MAPE	MACs	Para.
TROIKA [17]	**1.8%**	1.9s/PC	na	na	na
BioTranslator [14]	7.2%	na	52.59%	158M	42,657
UP-RaNN Model (ISPC dataset)	8.2%	0.0002s/PC	na	na	na
UP-RaNN Model (Own dataset)	4.6%	**0.056s/MCU**	**5.79%**	**1,449**	**517**

RAM). Therefore, it is impractical to deploy this framework on an MCU. Moreover, both BioTranslator and our UP-RaNN search method have a MAPE of around 8% for HR estimation, whereas our HR model only takes 0.0002s to obtain one HR estimation on the PC and 0.056 s on the MCU at 8 MHz. In addition, when training the HR model on our dataset, the HR estimation can reach a much lower MAPE.

Our method demonstrates superior performance over BioTranslator in terms of MAPE, MACs, and Trainable Parameters for HRV estimation. A significant difference can be observed when comparing our method to BioTranslator. BioTranslator has over 100,000 times more MACs and 80 times more parameters, making it impractical to deploy on a memory-constrained MCU. It is worth noting that BioTranslator exhibits a considerably high MAPE for HRV estimation. There are two main reasons for the high MAPE observed in HRV estimation using BioTranslator. Firstly, the ISPC dataset utilized in the evaluation contains numerous motion artifacts that can affect the accuracy of HRV estimation. Secondly, the ISPC dataset only provides 300 s of data for each subject, which is insufficient for directly predicting an HRV value over such a short period. As an alternative approach, BioTranslator chooses to first predict the R–R interval and then calculate the corresponding HRV values based on that information. This method is proved inaccurate as we mentioned in Section 2.2: Even minor variations in R–R interval values would lead to a major difference in HRV. As a comparison, our proposed solution has a pretty low MAPE of 5.79% with model c2-c4-c4-l64-l16. In summary, our method offers significant advantages for deployment on a MCU due to its optimized resource usage. Despite being designed for MCU constraints, our method maintains comparable performance in terms of latency and accuracy when compared to the state-of-the-art methods for HRV estimation. This combination of efficient deployment and competitive performance makes our method highly favorable for practical implementation.

4.4. Energy consumption of the HRV monitoring system

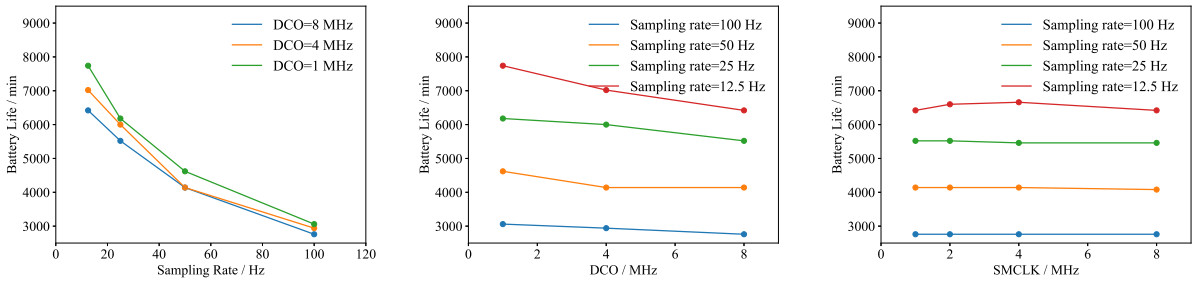
In addition to the neural network search method and the real-time on-device HRV monitoring system, three key configurations have impacts on the power consumption of the entire system: the Digitally Controlled Oscillator (DCO), the sub-main system clock (SMCLK), and the sampling rate (SR). The DCO determines the main clock frequency of the chip, which in turn affects the speed at which the code can execute. By adjusting the DCO configuration, we can control the overall performance and power consumption of the system. The SMCLK relies on the DCO as its source and drives the I2C and UART protocols. By modifying the SMCLK configuration, we can influence the data transfer rate and power consumption of these communication interfaces. The sampling rate has implications for memory usage, computational time, and battery life. The choice of sampling rate determines the amount of sensor data that needs to be processed and stored. Different sampling rates may require adjustments to the configurations of I2C, UART, and buffer size used for storing the sensor data. These variations directly impact the overall power consumption of the system. Therefore, careful consideration and optimization of the DCO, SMCLK, and sampling rate configurations allow us to fine-tune power consumption while balancing the system's memory usage, model latency, and battery life.

Fig. 6 shows the MCU battery life results under different system configurations. The battery life is calculated by measuring energy consumption using EnergyTrace [43], a built-in tool for MSP430 developing software, and assumes the MCU is powered by a CR2032 battery. Increasing the sampling rate has a significant impact on reducing battery life. For instance, when the DCO is set to 1 MHz, the battery life decreases by 2.5 times when the sampling rate is increased from 12.5 Hz to 100 Hz. Although the DCO does affect energy consumption, it does not have a notable impact on the sampling rate itself. However, reducing the DCO will result in longer inference times for HR and HRV estimation. Therefore, if the objective is to conserve energy, reducing the sampling rate is a more effective approach than decreasing the DCO. The effect of the SMCLK on energy consumption is less straightforward. The SMCLK primarily influences the configurations of the I2C and UART protocols, affecting the execution speed of the MSP430 chip. Therefore, when designing the HR/HRV monitoring system, we can adjust the SMCLK based on the DCO and other specific requirements. However, it is not expected that reducing the SMCLK will provide significant improvements in power savings. In conclusion, to optimize power consumption in the HR/HRV monitoring system design, it is recommended to reduce the sampling rate rather than the DCO. The impact of the SMCLK on energy consumption is indirect, primarily affecting protocol configurations, and any power-saving enhancements from reducing the SMCLK are minimal.

4.5. Evaluation of runtime reconfigurations

Based on the aforementioned analysis, we propose designing multiple configurations that can adapt to different demands or scenarios during runtime. The core concept is to pre-define various configurations on the MCU and utilize different signals to trigger reconfiguration when necessary. These triggering signals can include a button press, a timer reaching a specific interval, or reaching a certain battery level threshold.

Table 5 presents the analysis results for different HRV monitoring modes. From the analysis, it can be determined that the regular mode is the recommended choice for regular users who prioritize quick computational time. In this mode, each computation takes only 0.302 s, while still providing relatively high accuracy in HRV estimation. Additionally, the Energy (saving) mode is available for users who need to conserve energy, particularly when the battery has low energy levels. This mode reduces the DCO frequency and sampling rate and employs the same model with regular mode to optimize energy efficiency while still providing acceptable accuracy in HRV estimation. Furthermore, the Accuracy mode is designed to maximize accuracy in HRV estimation by utilizing a larger model, c2-c4-c4-l64-l16. However, it may require more energy consumption due to the increased computational requirements. These preset modes allow users to switch between them at runtime based on their specific needs.



(a) Battery life vs. SR. (b) Battery life vs. DCO. (c) Battery life vs. SMCLK.

Fig. 6. Battery life results.

Table 5
Current under various system modes.

Mode	Model	MAPE/%	DCO/MHz	SR/Hz	Latency/s	Current/mA
Regular	c1-c3-c2-l32-l2-l2	5.79	8	25	0.302	8.7
Energy	c1-c3-c2-l32-l2-l2	5.79	1	12.5	2.456	8.2
Accuracy	c2-c4-c4-l64-l16	4.63	8	100	0.556	12.2

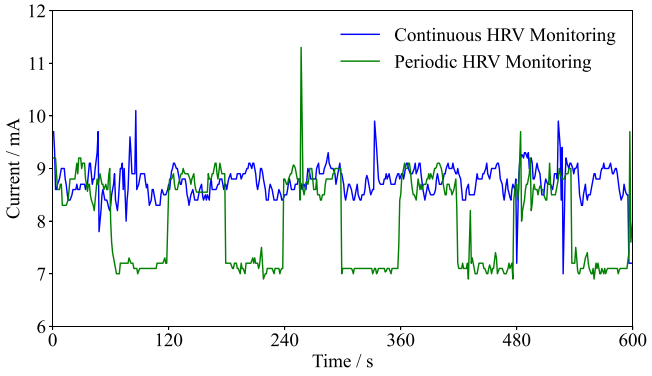


Fig. 7. Power consumption comparison in different work schedules. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Fig. 7 displays the results of a current comparison on the MSP430FR 5994 development board using different schedules. Power consumption is measured in milliamps (mA). In the figure, the blue line represents continuous monitoring of the HRV for 10 min, while the green line represents a periodic schedule of "work 1 min - rest 1 minute". The periodic monitoring reduces the current by approximately 22% compared to continuous monitoring. This feature enables effective power-saving when required, especially when combined with an accelerator to intelligently and efficiently monitor an individual's heart health during intense activities.

5. Conclusion and future works

In this study, we introduce a novel approach that addresses the challenges by proposing a unified performance and resource-aware neural network search method: UP-RaNN. This method aims to find the optimal neural network model for HRV estimation using PPG sensing data while considering specific resource limitations. Following that, we develop a real-time on-device HRV monitoring system for ultra-low-power devices using the trained models. Additionally, we analyze the performance and energy efficiency of different operation modes with DNN models to support runtime configurations.

To evaluate the proposed UP-RaNN method and the real-time HRV monitoring system associated with the adaptive runtime configuration switch mechanism, we choose an MSP430FR5994 development board as the test platform. The results show that our proposed UP-RaNN method can effectively and efficiently conduct the network architecture search and explore suitable models for a given platform compared to the traditional grid search method. In addition, the explored models show comparable performance to state-of-the-art HRV estimation methods using PPG. Moreover, our real-time HRV monitoring system outperforms state-of-the-art HRV methods in computational time and the runtime configuration switch mechanism provides the possibility of changing operation mode for various demands.

However, there are still some limitations that need to be addressed in our future works. First of all, the current real-time HRV monitoring system does not take advantage of hardware accelerator and low power modes. We plan to explore how utilizing these hardware accelerators and enabling low power mode can enhance the monitoring system, improve power consumption, and speed up the system. Second, we aim to analyze the integration of other sensors to increase HRV estimation accuracy by further eliminating motion artifacts. Third, the accuracy of the HRV model varies for different individuals, thus creating the need for personalized models. Lastly, quantized neural networks have not been explored in this paper. Given that quantized neural networks can save storage and speed up inference time, their implementation is worth investigating.

CRediT authorship contribution statement

Jingye Xu: Writing – review & editing, Writing – original draft, Validation, Software, Methodology. **Yuntong Zhang:** Writing – review & editing, Writing – original draft, Validation. **Mimi Xie:** Writing – review & editing, Writing – original draft, Supervision, Methodology. **Wei Wang:** Writing – review & editing, Writing – original draft, Supervision, Methodology. **Dakai Zhu:** Writing – review & editing, Writing – original draft, Supervision, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgements

This work is funded by the National Science Foundation (Award Number: NSF 2306596).

References

[1] A.J. Camm, M. Malik, J.T. Bigger, G. Breithardt, S. Cerutti, R.J. Cohen, P. Coumel, E.L. Fallen, H.L. Kennedy, R.E. Kleiger, et al., Heart rate variability: Standards of measurement, physiological interpretation and clinical use. Task force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology, Circulation 93 (5) (1996) 1043–1065.

[2] G.E. Billman, H.V. Huikuri, J. Sacha, K. Trimmel, An introduction to heart rate variability: Methodological considerations and clinical applications, Front. Physiol. 6 (2015) 55.

[3] J. Achten, A.E. Jeukendrup, Heart rate monitoring: Applications and limitations, Sports Med. 33 (2003) 517–538.

[4] D.B. Geselowitz, On the theory of the electrocardiogram, Proc. IEEE 77 (6) (1989) 857–876.

[5] A.N. Ganesan, D.P. Chew, T. Hartshorne, J.B. Selvanayagam, P.E. Aylward, P. Sanders, A.D. McGavigan, The impact of atrial fibrillation type on the risk of thromboembolism, mortality, and bleeding: A systematic review and meta-analysis, Eur. Heart J. 37 (20) (2016) 1591–1602.

[6] N. Isakadze, S.S. Martin, How useful is the smartwatch ECG? Trends Cardiovascul. Med. 30 (7) (2020) 442–448.

[7] C. Wang, L. Xie, W. Wang, Y. Chen, Y. Bu, S. Lu, RF-ECG: Heart rate variability assessment based on COTS RFID tag array, Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2 (2) (2018) http://dx.doi.org/10.1145/3214288.

[8] A. Inc, Take an ECG with the ECG app on Apple Watch, 2022.

[9] S.S.S. Das, S.K. Shanto, M. Rahman, M.S. Islam, A.H. Rahman, M.M. Masud, M.E. Ali, BayesBeat: Reliable atrial fibrillation detection from noisy photoplethysmography data, Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 6 (1) (2022) http://dx.doi.org/10.1145/3517247.

[10] J. Allen, Photoplethysmography and its application in clinical physiological measurement, Physiol. Meas. 28 (3) (2007) R1.

[11] C.G. Scully, J. Lee, J. Meyer, A.M. Gorbach, D. Granquist-Fraser, Y. Mendelson, K.H. Chon, Physiological parameter monitoring from optical recordings with a mobile phone, IEEE Trans. Biomed. Eng. 59 (2) (2011) 303–306.

[12] B. Coffen, P. Scott, M.S. Mahmud, Real-time wireless health monitoring: An ultra-low power biosensor ring for heart disease monitoring, in: 2020 International Conference on Computing, Networking and Communications, ICNC, IEEE, 2020, pp. 626–630.

[13] Q. Wang, Z. Wang, X. Dai, S. Song, T. Xing, S-HRVM: Smart watch-based heart rate variability monitoring system, in: Ewsn, 2019, pp. 178–183.

[14] L. Everson, D. Biswas, B.-E. Verhoef, C.H. Kim, C. Van Hoof, M. Konijnenburg, N. Van Helleputte, BioTranslator: Inferring R-peaks from ambulatory wrist-worn ppg signal, in: 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC, IEEE, 2019, pp. 4241–4245.

[15] H.-Y. Chiu, H.-H. Shuai, P.C.-P. Chao, Reconstructing QRS complex from PPG by transformed attentional neural networks, IEEE Sens. J. 20 (20) (2020) 12374–12383.

[16] U.R. Acharya, K.P. Joseph, N. Kannathal, C.M. Lim, J.S. Suri, Heart rate variability: A review, Med. Biol. Eng. Comput. 44 (12) (2006) 1031–1051.

[17] Z. Zhang, Z. Pi, B. Liu, TROIKA: A general framework for heart rate monitoring using wrist-type photoplethysmographic signals during intensive physical exercise, IEEE Trans. Biomed. Eng. 62 (2) (2015) 522–531.

[18] Y. Zhang, J. Xu, M. Xie, D. Zhu, H. Song, W. Wang, Efficient and direct inference of heart rate variability using both signal processing and machine learning, in: The IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies, CHASE, 2023.

[19] K. Lu, A.S. Dahlman, J. Karlsson, S. Candefjord, Detecting driver fatigue using heart rate variability: A systematic review, Accid. Anal. Prev. 178 (2022) 106830.

[20] R.J. Oweis, B.O. Al-Tabbaa, QRS detection and heart rate variability analysis: A survey, Biomed. Sci. Eng. 2 (1) (2014) 13–34.

[21] R.E. Kleiger, P.K. Stein, J.T. Bigger Jr., Heart rate variability: Measurement and clinical utility, Ann. Noninvasive Electrocardiol. 10 (1) (2005) 88–101.

[22] Y. Zhang, J. Xu, M. Xie, W. Wang, K. Ye, J. Wang, D. Zhu, PPG-based heart rate estimation with efficient sensor sampling and learning models, in: 2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application, HPCC/DSS/SmartCity/DependSys, IEEE, 2022, pp. 1971–1978.

[23] T. Wittenberg, R. Koch, N. Pfeiffer, N. Lang, M. Struck, O. Amft, B. Eskofier, Evaluation of HRV estimation algorithms from PPG data using neural networks, Curr. Dir. Biomed. Eng. 6 (3) (2020) 505–509.

[24] A. Alqaraawi, A. Alwosheel, A. Alasaad, Heart rate variability estimation in photoplethysmography signals using Bayesian learning approach, Healthc. Technol. Lett. 3 (2) (2016) 136–142.

[25] E.K. Naeini, F. Sarhaddi, I. Azimi, P. Liljeberg, N. Dutt, A.M. Rahmani, A deep learning–based PPG quality assessment approach for heart rate and heart rate variability, ACM Trans. Comput. Healthc. 4 (4) (2023) 1–22.

[26] P. Jain, C. Ding, C. Rudin, X. Hu, A self-supervised algorithm for denoising photoplethysmography signals for heart rate estimation from wearables, 2023, arXiv preprint arXiv:2307.05339.

[27] F. Esgalhado, A. Batista, V. Vassilenko, M. Ortigueira, Real-time PPG-based HRV implementation using deep learning and Simulink, in: Doctoral Conference on Computing, Electrical and Industrial Systems, Springer, 2022, pp. 103–111.

[28] W. Karlen, S. Raman, J.M. Ansermino, G.A. Dumont, Multiparameter respiratory rate estimation from the photoplethysmogram, IEEE Trans. Biomed. Eng. 60 (7) (2013) 1946–1953.

[29] K. Xu, X. Jiang, H. Ren, X. Liu, W. Chen, Deep recurrent neural network for extracting pulse rate variability from photoplethysmography during strenuous physical exercise, in: 2019 IEEE Biomedical Circuits and Systems Conference, BioCAS, 2019, pp. 1–4.

[30] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: A survey, J. Mach. Learn. Res. 20 (55) (2019) 1–21.

[31] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, 2018, arXiv preprint arXiv:1806.09055.

[32] B. Kim, S. Lee, On-NAS: On-device neural architecture search on memory-constrained intelligent embedded systems, 2023.

[33] M. Xiang, R. Ding, H. Liu, X. Zhou, Latency-constrained neural architecture search method for efficient model deployment on RISC-V devices, Electronics 13 (4) (2024) 692.

[34] Z. Yang, Q. Sun, Toward efficient neural architecture search with dynamic mapping-adaptive sampling for resource-limited edge device, Neural Comput. Appl. 35 (7) (2023) 5553–5573.

[35] M. Loni, A. Zoljodi, A. Majd, B.H. Ahn, M. Daneshtalab, M. Sjödin, H. Esmaeilzadeh, Faststereonet: A fast neural architecture search for improving the inference of disparity estimation on resource-limited platforms, IEEE Trans. Syst. Man Cybern. A 52 (8) (2021) 5222–5234.

[36] X. Luo, D. Liu, H. Kong, S. Huai, H. Chen, W. Liu, You only search once: On lightweight differentiable architecture search for resource-constrained embedded platforms, in: Proceedings of the 59th ACM/IEEE Design Automation Conference, 2022, pp. 475–480.

[37] G. Palshikar, et al., Simple algorithms for peak detection in time-series, in: Proc. 1st Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence, vol. 122, 2009.

[38] P. Liashchynskyi, P. Liashchynskyi, Grid search, random search, genetic algorithm: A big comparison for NAS, 2019, arXiv preprint arXiv:1912.06059.

[39] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[40] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.

[41] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.

[42] G.H. Golub, C.F. Van Loan, The singular value decomposition and unitary matrices, Matrix Comput. (1996) 70–71.

[43] B. Finch, W. Goh, MSP430 advanced power optimizations: ULP advisor software and energy trace technology, Appl. Rep. Texas Instrum. (2014).