BiomedBench: A benchmark suite of TinyML biomedical applications for low-power wearables

Dimitrios Samakovlis[1], Stefano Albini[1], Rubén Rodríguez Álvarez[1], Denisa-Andreea Constantinescu[1], Pasquale Davide Schiavone[1], Miguel Peón-Quirós[2], David Atienza[1],[2]
[1] Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
[2] EcoCloud Center, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

***Abstract*—The design of low-power wearables for the biomedical domain has received a lot of attention in recent decades, as technological advances in chip manufacturing have allowed real-time monitoring of patients using low-complexity ML within the mW range. Despite advances in application and hardware design research, the domain lacks a systematic approach to hardware evaluation. In this work, we propose BiomedBench, a new benchmark suite composed of complete end-to-end TinyML biomedical applications for real-time monitoring of patients using wearable devices. Each application presents different requirements during typical signal acquisition and processing phases, including varying computational workloads and relations between active and idle times. Furthermore, our evaluation of five state-of-the-art low-power platforms in terms of energy efficiency shows that modern platforms cannot effectively target all types of biomedical applications. BiomedBench will be released as an open-source suite to enable future improvements in the entire domain of bioengineering systems and TinyML application design.**

***Index Terms*—Benchmarking, TinyML, biomedical applications, wearable, low-power, signal processing.**

I. Introduction

Wearable devices promise to improve preventive medicine through continuous health monitoring of chronic diseases. To this end, we face the challenge of increasing their computational capability and energy efficiency to implement advanced biomedical algorithms on the edge, increase patient privacy, and reduce response latency while enabling seamless operation with small batteries and sparse recharging cycles. Unfortunately, most low-power commercial platforms focus on Internet-of-Things (IoT) applications and are suboptimal for patient monitoring scenarios with stringent energy requirements.

We need a synergy between hardware and software development to achieve efficiency in the low-power wearables domain. Architectural design must be aligned with the characteristics of target applications to meet real-time and energy constraints, and application developers must be aware of the most common kernels and most suitable platforms in the domain to minimize software development and deployment time. However, for wearable applications targeting low-power platforms operating in the mW range, we are missing in the state-of-the-art (SoA) a set of representative end-to-end applications to guide the codesign process by standardizing hardware evaluation and unveiling the critical hardware and software design points.

In response to these needs, we propose BiomedBench, a biomedical benchmark suite composed of end-to-end applications aimed at low-power wearable devices. These applications feature diverse requirements during processing, idle, and signal acquisition that effectively represent the challenges in the domain. Furthermore, we demonstrate how to utilize BiomedBench to systematically evaluate and compare SoA platforms. To our knowledge, this is the first benchmark suite explicitly targeting the low-power biomedical domain, offering a systematic approach to software and hardware codesign. The contribution of BiomedBench is twofold:

- It offers a set of complete end-to-end biomedical applications, including the idle, acquisition, and processing phases. The variety of requirements present in the applications is representative of the challenges in the low-power wearables domain and makes BiomedBench a suitable benchmark suite for hardware evaluation.
- It serves as a baseline for future hardware and application design in the wearable and TinyML domains. Utilizing BiomedBench to compare SoA platforms unveils the critical design features that impact the energy footprint for hardware designers and hints at the deployment platform selection for application designers. Overall, open-sourcing of the code accelerates future application development efforts.

We organize this work as follows. In Section II, we compare BiomedBench with existing benchmark suites. In Section III, we propose a set of SoA wearable applications along with a systematic characterization of their key features. In Sections IV and V, we describe the setup of our experiments and analyze the results, respectively. Finally, we summarize the key findings of this work in Section VI.

II. Related Work

BCIBench [1] is a benchmark suite targeting electroencephalogram (EEG)-based kernels and applications in the low-power wearables domain. Hermit [2] targets biomedical workloads in the Internet-of-Medical-Things (IoMT). Hermit includes three kernels for monitoring common medical conditions (sleep apnea, heart-rate variability, and blood pressure), kernels for offline diagnosis using image processing, and encryption and compression algorithms. Finally, ImpBench [3] targets devices in the implantable biomedical domain. ImpBench features two synthetic benchmark applications for motion detection and drug delivery system simulation, and six lightweight kernels for data compression, encryption, and integrity.

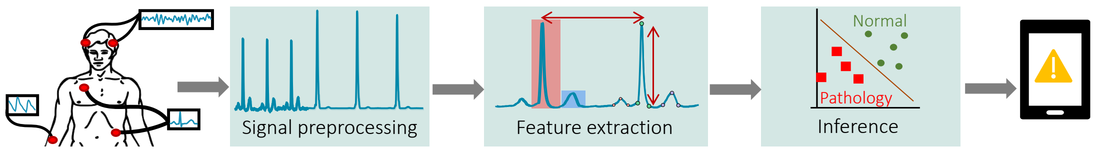

Fig. 1: Typical modules of biomedical applications [5].

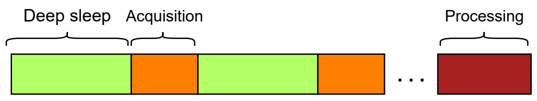

Fig. 2: micro-controller unit (MCU) operating phases

BiomedBench differs from existing biomedical benchmark suites in two ways. First, BiomedBench integrates the idle and acquisition phases in complete end-to-end applications and highlights the impact of these phases in the application and platform design. Second, BiomedBench includes a wider variety of processing kernels stemming from a larger set of input signals, like electrocardiogram (ECG), surface electromyography (sEMG), and Photoplethysmography (PPG), thus covering a wider spectrum of workloads, as shown in Section III-B.

III. Applications

Fig. 1 shows a typical biosignal monitoring application pipeline. Sensors capture the biosignal and send it to the processing device for analysis. Typically, the processing step consists of signal preprocessing (i.e., filtering), feature extraction (i.e., time or frequency characteristics), and inference (i.e., machine learning (ML) model). However, applications can exhibit a wide range of workloads and computational requirements. For example, feature extraction can be implemented explicitly (i.e., features designed manually) or implicitly (e.g., convolutional neural network (CNN)). Similarly, the inference step can use a lightweight machine learning method, such as a random forest or a computationally intensive deep neural network (DNN).

From an implementation point of view, the MCU interchanges among idle, acquisition, and processing, as presented in Fig. 2. Assuming an external analog-to-digital converter (ADC) with a buffer, MCU collects the data through a communication protocol such as serial peripheral interface (SPI) upon buffer filling. The acquisition is typically served by the direct memory access (DMA). The MCU is in low-power modes during idle and acquisition (i.e., deep sleep for idle and light sleep for acquisition). The duration of the low-power modes can vary significantly between applications and can dominate the system's energy consumption. Considering the variety of workloads, input bandwidths, and idle-to-active ratios present in low-power wearable applications, a benchmark suite that covers a wide range of applications is an essential tool for hardware evaluation in the domain.

A. *Metrics for application characterization*

We propose a characterization of the application by metrics. The metrics must suffice to describe the computing profile, impact of the idle period, acquisition intensity, and memory requirements of the applications. This information is critical for deployment platform selection and application-driven hardware design. To this end, we have selected the following five metrics: *main operations*, *duty cycle*, *input bandwidth*, *dynamic data*, and *static data*.

1) *Main operations:* Identifies whether the dominant operations are branches, logical operations, fixed-point (FxP) or floating-point (FP) computations, among others. This metric hints at the microarchitectural design needed to efficiently handle different computational workloads and achieve high performance.

2) *Duty cycle:* Represents the ratio between CPU active cycles and total cycles. A low duty cycle means that the low-power modes of a platform dominate the energy footprint. We use the following scale: "very low" (less than $0.1\,\%$), "low" (between $0.1\,\%$ to $1\,\%$), "medium" (between $1\,\%$ to $15\,\%$), "high" (between $15\,\%$ to $60\,\%$), and "very high" (above $60\,\%$). Since this metric is platform-dependent, we calculate it running on an ARM Cortex-M4.

3) *Input bandwidth:* Measured in $\mathrm{B/sec}$, is the product of the sensor's sampling rate, the size per sample, and the number of channels used. It specifies the intensity and energy impact of the signal acquisition phase.

4) *Static data:* Measured in KiB, it quantifies the memory required for the code and read-only data, such as pre-trained parameters. Hence, it defines the amount of memory to be retained, on RAM or FLASH, during idle phases.

5) *Dynamic data:* Measured in KiB, defines how much memory the application requires during runtime for the stack and the heap. Hence, it specifies the minimum amount of RAM required.

B. *SoA wearable applications*

We have selected eight biomedical wearable applications that offer representative workloads and varied profiles for the processing, idle, and acquisition phases. The applications are complementary and enable evaluation of different architectural parts (e.g., sleep mode, digital signal processing). BiomedBench will be launched with eight applications but is open to future additions that present new challenges in any of the three phases.

All applications are coded in C or C++. Four applications are implemented in FxP arithmetic, targeting low-end MCUs, and four applications are implemented in 32-bit FP arithmetic. Four of the eight applications also include a multi-core implementation that enables acceleration in multicore platforms.

Table I summarizes the main metrics of the applications, illustrating the wide spectrum of computational workloads, active-to-idle ratios, and acquisition and memory requirements

Application	**Main operations**	**Duty Cycle**	**Input bandwidth** (B/sec)	**Static data** (KiB)	**Dynamic data** (KiB)
HeartBeatClass	Branches (FxP min/max search)	Low	1536	25	30
SeizureDetSVM	32-bit FxP multiplications	Very Low	128	40	40
SeizureDetCNN	16-bit FxP MAC	High	11776	350	120
CognWorkMon	32-bit FxP multiplications	Medium	4096	90	50
GestureClass	32-bit FP MAC	Very High	192000	50	110
CoughDet	32-bit FP multiplications	Very High	64400	568	160
EmotionClass	Branches (FP sorting)	Low	822	16	4
Bio-BPfree	32-bit FP MAC	-	-	1300	2600

TABLE I: Benchmark applications - A characterization by metrics

covered by the benchmark suite. This variety of requirements is key to a complete evaluation of low-power platforms, as illustrated in Section V. The benchmarks are explained below.

1) Heartbeat classifier (HeartBeatClass): Detects abnormal heartbeat patterns in real time for common heart diseases using the ECG signal [4]. The input signal is sampled by three different ECG leads at 256 Hz with 16-bit accuracy for 15 s. The input signal is processed through morphological filter (MF), and the root-mean-square (RMS) combines the three signal sources before enhancing the signal through relative energy (Rel-En). In feature extraction, the relative-energy-based wearable R-peak detection (REWARD) algorithm detects the R peaks before delineating the other fiducial points of ECG. Finally, a neuro-fuzzy classifier using random projections (RP) of the fiducial points classifies the heartbeats as abnormal or not.

The application uses 16-bit fixed-point arithmetic. MF is the dominant kernel, accounting for more than 80 % of the execution time. The MF implementation involves a queue to perform dilation and erosion, translating into data movements and min/max search. We also include the multicore version of the application [5], having improved the parallelization strategy for the delineation and classification phases with dynamic task partition instead of static.

2) Seizure detector support vector machine (SeizureDetSVM): Works on ECG input and recognizes epileptic episodes in real time [6]. The ECG signal is sampled from a single lead at 64 Hz with 16-bit accuracy for 60 s. The preprocessing phase consists of a simple moving average (MAVG) subtraction. For feature extraction, the R-peak interval (RRI) and ECG-derived respiration (EDR) time series are calculated from the ECG. From RRI, heart-rate variability (HRV) features and Lorenz plot features are extracted. From EDR, the linear predictive coefficients and the power spectral density of different frequency bands are calculated. For the frequency feature extraction (FFE) of RRI and HRV, the Lomb-Scargle periodogram (PLOMB) algorithm, which involves a fast Fourier transform (FFT), is used. For inference, a support vector machine (SVM) uses all the extracted features to classify the patient's state.

PLOMB is the dominant kernel, accounting for more than 75 % of the execution time. Since the implementation is in 32-bit FxP arithmetic, the main operations are 32-bit integer multiplications with a 64-bit intermediate result followed by a shift. This application originally contained a self-aware mechanism to determine the number of features and the complexity of the SVM. For this benchmark, we only use the full pipeline to avoid variability among executions and test the most complete version. Finally, we designed a parallel version of this application since it features a high degree of parallelism.

3) Seizure detector convolutional neural network (SeizureDetCNN): Based on EEG data, detects in real time epileptic seizure episodes [7]. The signal is sampled from 23 leads at 256 Hz with a 16-bit accuracy for 4 s. This application does not feature any signal preprocessing or feature extraction kernels. Instead, the input is sent directly to the input layer of a fully-convolutional network (FCN). The proposed FCN architecture has three 1D convolutional layers, each including batch normalization, pooling, and ReLU layers, and two fully connected layers. Most computations are 16-bit FxP multiply-accumulate (MAC) operations due to convolution, as 90 % of the execution is spent in the convolutional layers. Moreover, this application includes a parallel implementation, as the high degree of parallelism is an inherent characteristic of CNNs.

4) Cognitive workload monitor (CognWorkMon): Is designed for real-time monitoring of the cognitive workload state of a subject [8] and is based on EEG input. The EEG signal is sampled by four leads at 256 Hz with 32-bit accuracy. The input signal is processed in 14 batches of 4 s for a total of 56 s. Preprocessing and feature extraction are executed 14× per channel before the classification phase is executed. Preprocessing involves blink removal (BLR) and a band-pass filter (BPF) through infinite impulse response (IIR) filters. Feature extraction contains time-domain features (i.e., skewness/kurtosis, Hjorth activity), frequency-domain features (i.e., power spectral density), and entropy features. A random forest (RF) uses these features to classify the stress condition of the subject.

The extraction of frequency features, which contains the FFT, is the most demanding computational kernel, accounting for more than 80 % of the total computation time. The main operations are 32-bit integer multiplications with a 64-bit intermediate result followed by a shift since we transformed the original application into a FxP implementation with a negligible accuracy drop.

5) Gesture classifier (GestureClass): Aims to classify hand gestures by inspecting signals captured by sEMG of the forearm [9]. The idea is to extract the motor unit action potential train (MUAPT) and identify the motor neuron activity patterns to classify the hand gesture. The signal is sampled from 16 channels at 4 kHz with 24-bit accuracy for only 0.2 s. This application has no signal preprocessing. The authors apply a blind source separation (BSS) method to the input signal, namely independent component analysis (ICA), and classify the gesture using a SVM or a multilayer Perceptron (MLP). We use the MLP for the inference stage to boost the variability of the kernels under test.

GestureClass is implemented in 32-bit FP arithmetic, and the dominant workload is the ICA which features matrix multiplications. Hence, the main operations are 32-bit FP MACs. We include the original parallel implementation of this application and have also converted it to run on a single core to make it available for single-core platforms.

6) Cough detector (CoughDet): Is a novel application [10] using non-invasive chest-worn biosensors to count the number of cough episodes people experience per day, thus providing a quantifiable means of evaluating the efficacy of chronic cough treatment. The device records audio data, sampled at 16 kHz with 32-bit precision, as well as 3-axis accelerometer and 3-axis gyroscope signals from an inertial measurement unit (IMU), each sampled at 100 Hz with 16-bit precision. Biosignals are processed every 0.3 s.

Feature extraction includes computations in the time and frequency domain. Time-domain computations include the extraction of statistical values (such as zero crossing rate, root-means-squared, and kurtosis) of the IMU signals. An FFT is used to extract spectral statistics (including standard deviation and dominant frequency), power spectral density, and mel-frequency cepstral coefficients (MFCC) of the audio signal. Features extracted from audio and IMU signals are forwarded to an RF classifier that computes the probability of a cough event.

The MFCC constitutes the most intensive kernel that requires the iterative computation of FFT and transcendental functions (i.e., the logarithm in the discrete cosine transform (DCT)). The application is implemented in 32-bit FP arithmetic, and the main operations include FP multiplications.

7) Emotion classifier (EmotionClass): Classifies the fear status of patients to prevent gender-based violence [11] based on three physiological signals, namely Galvanic skin response (GSR), PPG, and skin temperature (ST). PPG is sampled at 200 Hz with 32-bit precision, GSR is sampled at 5 Hz with 32-bit precision, and ST is sampled at 1 Hz with 16-bit precision. The acquisition window lasts 10 s and is divided into 10 batches of partial inference before the final classification is performed based on the 10 partial classifications.

EmotionClass has no signal preprocessing step. Feature extraction includes the average (AVG) of the three input signals over 1 s before forwarding them to a k-nearest neighbors (KNN) classifier. The classifier computes the distances of the new 3D tuple from the training points that have already been labeled as fear or no fear. Using n training points, we select the $\sqrt{n}$ closest training points by running $\sqrt{n}$ steps of selection sort before classifying the new tuple based on the percentage of neighboring fear-labeled points. We use 685 training points—a tradeoff between accuracy and complexity [11].

EmotionClass uses both 16-bit and 32-bit FP arithmetic, as it uses different representations for the three different signals. The dominant kernel is the KNN inference, which includes the 32-bit FP calculation and sorting of the Euclidean distances in 3D. Sorting includes multiple minimum search iterations over the array of distances.

8) Biological back-propagation-free (Bio-BPfree): Is the only benchmark that performs on-device training. We use the Bio-BPfree training algorithm presented in [12]. The main notion of Bio-BPfree is to perform per-layer training by maximizing the distance between the intermediate outputs of different classes and minimizing the distance between the intermediate outputs of the same class. Bio-BPfree avoids the prohibitive memory cost of backpropagation, thus opening possibilities for on-device training in low-power devices.

We used Bio-BPfree to fine-tune the DNN presented in [7] for seizure detection. The model was originally trained on the server using a leave-one-out-patient on the CHB-MIT database. Later, we retrain the model on the device with Bio-BPfree by exploiting the personalized samples acquired from the patient under test. The on-device training yields a significant improvement in the F1 score up to 25% thanks to the personalized samples available on the device while ensuring data privacy.

The implementation of Bio-BPfree is based on the computation of the gradients of the loss function with respect to the trainable parameters. We define a custom loss function per layer [12] and then compute the gradients using the chain rule to account for the intermediate layers (i.e., ReLU, batch normalization, max pooling). The main operations are 32-bit FP MACs because of the convolution in the forward passes and the vector-matrix multiplications involved in the chain rule of the gradient computation. There is no acquisition phase. We assume that four pre-recorded input samples are already stored in the FLASH and expect the retraining to occur during the device charging phase. One epoch is executed for benchmark purposes.

IV. Experimental Setup

In this section, we show the deployment and evaluation process of BiomedBench on a set of representative SoA low-power boards.

A. Considered low-power boards

We target low-power platforms with low-end MCUs, with clock frequencies in the range of MHz, and RAM in the range of some hundreds of KiB, as the application characterization in Table I suggests. Typically, such platforms feature simple processor architectures with in-order execution, no instruction-level parallelism, simple memory hierarchies, deep-sleep modes for long idle periods, and SPI support for signal acquisition. Such platforms often meet the application requirements in the domain [5], [6], [8], [9]. Finally, we explicitly target variability in processor architectures (i.e., ARM, RISC-V).

We have selected five popular commercial low-power boards featuring five different MCUs and four different processors for our experiments. The selected boards are: Nucleo-L4R5ZI[1] from ST Microelectronics, Ambiq Apollo 3 Blue[2], Raspberry

[1]https://www.st.com/en/evaluation-tools/nucleo-l4r5zi.html

[2]https://ambiq.com/apollo3-blue/

Board	**Manufacturer**	**MCU**	**Cores**	**FPU**	**RAM** (KiB)	**FLASH** (MB)
Raspberry Pi Pico	Raspberry	RP2040	2x ARM Cortex-M0+	No	264	2 (off-chip)
Nucleo L4R5	STMicroelectronics	STM32L4R5ZI	1x ARM Cortex-M4	Yes	640	2 (on-chip)
Ambiq Apollo 3	Ambiq	Apollo 3 Blue	1x ARM Cortex-M4	Yes	384	1 (on-chip)
Gapuino	GreenWaves	GAP8	1x CV32E40P (FC)	No	512	2 (off-chip)
	Technologies		8x CV32E40P (Cluster)	Yes	64	
GAP9EVK	GreenWaves	GAP9	1x CV32E40P (FC)	Yes	1564	2 (off-chip)
	Technologies		9x CV32E40P (Cluster)	Yes	128	

TABLE II: Selected boards - Summary of basic features

Pi Pico[3], Gapuino v1.1[4] and GAP9EVK[5] from GreenWaves Technologies. We summarize the architecture and storage specifications of each MCU in Table II.

B. Sensor emulation and sleep modes

For signal acquisition, we emulate the sensor and ADC functionality using an external board that artificially produces data. We assume an external ADC with 768 bytes of RAM buffer[6] since, typically, MCUs do not feature embedded ADCs or feature ADCs with insufficient bit precision. We perform per-batch acquisition by transferring data to the MCU when the buffer is full. We employ an SPI acquisition scheme using the DMA while the core is in sleep mode. Moreover, we assume that the sensors have no processing ability and that all the computations take part in the MCU.

During the idle period, we set the MCU to deep-sleep mode with RAM retention to preserve the data needed for the next processing cycle. The MCUs support a wake-up interrupt mechanism to switch to active mode when the data are ready. For the processing phase, we select the lowest operating voltage that allows the processing frequency to meet the real-time constraints of each application. For the selected voltage, we configure the highest available frequency for maximum energy efficiency as validated experimentally.

C. Energy measurements

We use the evaluation boards provided by the manufacturers to measure the energy consumption of each MCU executing BiomedBench applications. We do not consider the energy of the sensor and ADC in our experiments, as it is common for all platforms. We have measured the energy consumption of all boards at the power supply entry point of the integrated circuit (IC)[7] as we target a fair comparison across all platforms. However, we highlight that the reported energy numbers include the energy drawn by the input-to-core step-down voltage converter embodied in the integrated circuit. Future platform energy measurements must comply with this procedure for the results to be considered valid.

We have selected the Otii Arc provided by Qoitech, which samples at $4\,\mathrm{kHz}$, to obtain an energy profile over time and extract the energy and execution time per phase. However, due to the limited $\pm 10\,\mu\mathrm{A}$ precision of the Otii device, we used the Fluke 8846A multimeter to measure the average current of the Nucleo and Apollo boards in deep-sleep mode. This device can achieve a precision of $0.03\,\mu\mathrm{A}$.

[3]https://www.raspberrypi.com/products/raspberry-pi-pico/

[4]https://greenwaves-technologies.com/product/gapuino/

[5]https://greenwaves-technologies.com/gap9-store/

[6]AD4130-8, https://www.analog.com/en/products/ad4130-8.html

[7]We measure the total board energy for Raspberry Pi Pico with all peripherals disabled — no test points for the MCU provided

D. Portability and software support

We use the boards' toolchain and software development kit (SDK) to compile and load the C/C++ program on each board. For the runtime, we use the portable FreeRTOS API, which all boards support, for dynamic memory management. Finally, we utilize the hardware abstraction layer (HAL) of each board's SDK to program the SPI peripheral communication, to configure the power management unit (PMU) for the sleep modes, and to orchestrate multicore execution.

V. Experimental Results

In this section, we evaluate SoA platforms running BiomedBench in terms of energy efficiency and processing capability. We showcase that BiomedBench stresses different architectural aspects of the platforms, hence making it an effective hardware evaluation tool. Table III reports the processing cycles and energy per application and board.

A. Processing cycles

The amount of processing cycles required to execute the computational phase of the applications is a critical metric for evaluating wearable devices. Fewer processing cycles translate to shorter active phases for the MCU and energy efficiency. Analyzing in depth the processing performance discrepancies of the SoA platforms is the key to comprehending the exact microarchitectural challenges in the domain and, hence, facilitating domain-specific hardware design.

We summarize our observations stemming from Table III in four key points. First, CV32E40P GAP9 consistently scores the highest in all applications. Second, the relative performance of CV32E40P GAP8 and Arm Cortex-M4 varies significantly depending on the application type. In some applications, Arm Cortex-M4 outperforms CV32E40P GAP8 and matches CV32E40P GAP9, while in other applications, CV32E40P GAP8 outperforms Arm Cortex-M4 and matches CV32E40P GAP9. Third, Arm Cortex-M0+ cannot handle computations as efficiently as the other processors. Finally, Arm Cortex-M0+ and CV32E40P GAP8 lack a floating-point unit (FPU) and suffer in FP applications.

B. Energy

Understanding why some platforms are more energy-efficient than others and how the energy profile changes among

MCU	Processor	Application	Cycles (M)	Energy (mJ)				Application	Cycles (M)	Energy (mJ)			
				Idle	Acq.	Proc.	Total			Idle	Acq.	Proc.	Total
RP2040	Arm Cortex-M0+		11.6	29.647	3.519	6.532	39.698		149.7	0	2.972	87.543	90.515
STM32L4R5ZI	Arm Cortex-M4		7.4	0.118	0.002	2.604	2.724		9.9	0.002	0.001	6.649	6.652
Apollo 3 Blue	Arm Cortex-M4	HeartBeatClass	7.4	0.073	0.061	0.226	0.360	CoughDet	9.9	0.001	0.198	0.444	0.642
GAP8	CV32E40P GAP8		5.1	9.386	0.042	0.416	9.844		-	-	-	-	-
GAP9	CV32E40P GAP9		5.1	9.833	0.154	0.411	10.398		9.1	0.081	0.046	0.352	0.479
RP2040	Arm Cortex-M0+		4.3	118.805	2.402	2.420	123.627		571.6	0	8.008	347.104	355.112
STM32L4R5ZI	Arm Cortex-M4		2.8	0.476	0.001	1.313	1.790		23.0	0	0.004	13.425	13.429
Apollo 3 Blue	Arm Cortex-M4	SeizureDetSVM	2.8	0.294	0.042	0.137	0.473	GestureClass	23.0	0	0.525	2.500	3.025
GAP8	CV32E40P GAP8		3.2	37.724	0.029	0.353	38.106		635.8	0	0.096	220.933	221.029
GAP9	CV32E40P GAP9		2.8	39.50	0.037	0.090	39.627		20.2	0.027	0.124	0.604	0.755
RP2040	Arm Cortex-M0+		283.0	3.514	7.528	167.87	178.912		15.3	19.651	1.259	8.760	29.670
STM32L4R5ZI	Arm Cortex-M4		240.0	0.015	0.004	112.049	112.068		2.5	0.002	0.001	1.462	1.465
Apollo 3 Blue	Arm Cortex-M4	SeizureDetCNN	240.0	0.010	0.494	18.262	18.766	EmotionClass	2.5	0.061	0.083	0.110	0.254
GAP8	CV32E40P GAP8		160.0	0.464	0.090	31.987	32.541		14.3	6.224	0.015	1.052	7.291
GAP9	CV32E40P GAP9		160.0	2.234	0.117	5.101	7.452		1.6	6.572	0.019	0.061	6.652
RP2040	Arm Cortex-M0+		346.0	104.902	35.876	195.910	336.688		16758.0	-	-	9374.500	9374.500
STM32L4R5ZI	Arm Cortex-M4		141.0	0.432	0.017	70.629	71.078		662.0	-	-	432.227	432.227
Apollo 3 Blue	Arm Cortex-M4	CognWorkMon	141.0	0.325	0.620	3.887	4.832	Bio-BPfree	662.0	-	-	32.222	32.222
GAP8	CV32E40P GAP8		205.0	33.930	0.431	16.008	50.369		18450.0	-	-	1453.368	1453.368
GAP9	CV32E40P GAP9		145.0	36.303	0.557	3.711	40.571		633.0	-	-	24.970	24.970

TABLE III: Energy breakdown and processing cycles per application.

different applications and their phases is vital to boosting low-power platform design. Table III reports the total energy and the energy per phase for each application and platform. Interestingly, the impact of the phases on the total energy footprint fluctuates with the application.

1) Idle: Low-duty-cycle applications highlight the importance of a well-designed deep-sleep mode. SeizureDetSVM, featuring the lowest duty cycle of all applications, illustrates that STM32L4R5ZI and Apollo 3 Blue have excellent deep-sleep modes and dominate their rivals in total energy. Similar observations apply to HeartBeatClass and EmotionClass. In contrast, idle energy is much less impactful in high-duty-cycle applications such as SeizureDetCNN, GestureClass, and CoughDet.

2) Acquisition: Applications with a high input bandwidth highlight the need for an energy-efficient acquisition mode. CoughDet and GestureClass, featuring the highest input bandwidth, stress the importance of an energy-efficient acquisition mode (i.e., a sleep mode that allows DMA operation). Apart from Apollo 3, all platforms employ the DMA and spend negligible energy during acquisition in CoughDet and GestureClass. The acquisition energy is very low for all other applications.

3) Processing: High-duty-cycle applications, like SeizureDetCNN, CoughDet, and GestureClass, necessitate an energy-efficient processing mode. Each application features a different computational workload that impacts the duration of the processing phase per platform. The platforms' relative processing efficiency fluctuates per application, depending on their processor's performance across different workloads.

In general, Apollo 3 and GAP9 have the lowest processing energy. However, GAP9 manages to outperform Apollo 3 in applications that it can complete in significantly fewer cycles (i.e., SeizureDetCNN, EmotionClass). GAP8 spends more processing energy than GAP9 even when it matches GAP9's processing cycles (i.e., HeartBeatClass, SeizureDetCNN). Despite featuring the same processor, STM32L4R5ZI consumes approximately an order of magnitude more processing energy than Apollo 3 Blue in all applications. RP2040 spends, on average, two orders of magnitude more processing energy than the most efficient MCU.

4) Total: We summarize our observations on the total energy footprint in two key points. First, no single platform is the most energy-efficient for every benchmark. GAP9 is the most energy-efficient in computationally intensive, high-duty-cycle applications but features an uncompetitive deep-sleep mode. Apollo 3 and STM have the best deep-sleep modes and perform the best in low-duty-cycle applications. Second, the energy comparison among platforms varies significantly depending on the application. For instance, STM is $22\times$ more energy-efficient than GAP9 in the SeizureDetSVM but has $23.5\times$ more energy consumption in SeizureDetCNN. Consequently, we demonstrate that BiomedBench features applications with varying requirements and can serve as a benchmark tool to evaluate architectural designs in the biomedical domain.

VI. Conclusion

In this paper, we have introduced BiomedBench, a new benchmark suite aiming to systematize the evaluation of low-power platforms in the domain of patient monitoring wearables. BiomedBench features end-to-end applications with diverse computational pipelines, active-to-idle ratios, and acquisition profiles. BiomedBench, boosted by a systematic application characterization, unveils the key hardware challenges of deploying modern biomedical applications during idle, acquisition, and processing phases on wearable platforms. By evaluating BiomedBench's impact on energy and performance for five SoA low-power platforms, we have shown that no single MCU can efficiently handle the varying challenges of different benchmarks. To this end, BiomedBench will be released as an open-source suite to systemize platform evaluation, accelerate hardware design, and enable further advances in bioengineering systems and TinyML application design.

References

[1] Jafari, R., Dehzangi, O., Zong, C. & Nathan, V. BCIBench: A benchmarking suite for EEG-based brain-computer interface. *Proceedings of the 11th Workshop on Optimizations for DSP and Embedded Systems.* pp. 19-24 (2014), https://doi.org/10.1145/2568326.2568330

[2] Limaye, A. & Adegbija, T. HERMIT: A Benchmark Suite for the Internet of Medical Things. *IEEE Internet Of Things Journal*. **5**, 4212-4222 (2018)

[3] Strydis, C., Kachris, C. & Gaydadjiev, G. ImpBench: A novel benchmark suite for biomedical, microelectronic implants. *2008 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. pp. 82-91 (2008)

[4] Braojos, R., Ansaloni, G. & Atienza, D. A methodology for the embedded classification of heartbeats using random projections. *DATE*. pp. 899-904 (2013)

[5] De Giovanni, E., Montagna, F., Denkinger, B., Machetti, S., Peón-Quirós, M., Benatti, S., Rossi, D., Benini, L. & Atienza, D. Modular Design and Optimization of Biomedical Applications for Ultra-Low Power Heterogeneous Platforms. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*. **39**, 3821-3832 (2020)

[6] Forooghifar, F., Aminifar, A. & Atienza Alonso, D. Self-Aware Wearable Systems in Epileptic Seizure Detection. *DSD 2018*. pp. 426-432 (2018)

[7] Gomez, C., Arbelaez, P., Navarrete, M., Alvarado-Rojas, C., Le Van Quyen, M. & Valderrama, M. Automatic seizure detection based on imaged-EEG signals through fully convolutional networks. *Scientific Reports*. **10** (2020,12)

[8] Zanetti, R., Arza, A., Aminifar, A. & Atienza, D. Real-Time EEG-Based Cognitive Workload Monitoring on Wearable Devices. *IEEE Trans. Biomed. Eng.*. **69**, 265-277 (2022)

[9] Orlandi, M., Zanghieri, M., Morinigo, V., Conti, F., Schiavone, D., Benini, L. & Benatti, S. sEMG Neural Spikes Reconstruction for Gesture Recognition on a Low-Power Multicore Processor. *2022 IEEE Biomedical Circuits And Systems Conference (BioCAS)*. pp. 704-708 (2022)

[10] Orlandic, L., Thevenot, J., Teijeiro, T. & Atienza, D. A Multimodal Dataset for Automatic Edge-AI Cough Detection. (Zenodo,2023,1), https://zenodo.org/record/7562332, Type: dataset

[11] Miranda Calero, J., Marino, R., Lanza-Gutierrez, J., Riesgo, T., Garcia-Valderas, M. & Lopez-Ongil, C. Embedded Emotion Recognition within Cyber-Physical Systems using Physiological Signals. *DCIS*. pp. 1-6 (2018)

[12] Baghersalimi, S., Amirshahi, A., Teijeiro, T., Aminifar, A. & Atienza, D. Layer-Wise Learning Framework for Efficient DNN Deployment in Biomedical Wearable Systems. *2023 IEEE 19th International Conference On Body Sensor Networks (BSN)*. pp. 1-4 (2023)