tinyRadar for Fitness: A Contactless Framework for Edge Computing

Satyapreet Singh Yadav[1⋆], Radha Agarwal[1⋆], Kola Bharath[1], Sandeep Rao[2], Chetan Singh Thakur[1]
{satyapreets, radhaagarwal, kolabharath, csthakur}@iisc.ac.in, s-rao@ti.com
[1]Department of Electronic Systems Engineering, Indian Institute of Science, Bangalore, India, 560012
[2]Texas Instruments, Bangalore, India, 560093

***Abstract*—Healthcare technology is evolving from a conventional hub-based system to a personalized healthcare system accelerated by rapid advancements in smart fitness trackers. Modern fitness trackers are mostly lightweight wearables and can monitor the user's health round the clock, supporting ubiquitous connectivity and real-time tracking. However, prolonged skin contact with wearable trackers can cause discomfort. They are susceptible to false results and breach of privacy due to the exchange of user's personal data over the internet. We propose tinyRadar, a novel on-edge millimeter wave (mmWave) radar-based fitness tracker that solves the issues of discomfortness, and privacy risk in a small form factor, making it an ideal choice for a smart home setting. This work uses the Texas Instruments IWR1843 mmWave radar board to recognize the exercise type and measure its repetition counts, using signal processing and Convolutional Neural Network (CNN) implemented on board. The radar board is interfaced with ESP32 to transfer the results to the user's smartphone over Bluetooth Low Energy (BLE). Our dataset comprises eight exercises collected from fourteen human subjects. Data from ten subjects were used to train an 8-bit quantized CNN model. tinyRadar provides real-time repetition counts with 96% average accuracy and has an overall subject-independent classification accuracy of 97% when evaluated on the rest of the four subjects. CNN has a memory utilization of 11.36 KB, which includes only 1.46 KB for the model parameters (weights and biases) and the remaining for output activations.**

***Index Terms*—tinyRadar, IWR1843, CNN, VT map, tinyML, CMSIS-NN, fitness tracker, edge computing, ESP32**

I. Introduction

Regular exercising is essential for physical and mental well-being [1]. Personalized insights such as step count, calories burnt, and distance travelled motivate people to exercise regularly and set higher fitness goals [2]. Fitness trackers help self-health management in clinical settings [3] for people suffering from lifestyle diseases like obesity and diabetes. Smart fitness trackers require features such as real-time activity recognition with privacy preservation [4], [5], and compact form factor.

Wearable fitness trackers are commonly used, but they are uncomfortable to wear for prolonged use, potentially causing skin irritation and rashes [6]–[8]. Wearable trackers are attached to one part of the body and might track only that location, sometimes providing false results since they cannot capture the whole-body movement, especially when other limbs are moving [9], [10]. Also, false results occur when the device loses contact with the skin when the user begins to sweat during exercise [11].

⋆Equal contribution

We can classify the data collected through modern fitness trackers into three broad categories based on the sensitivity of the information: Personal Data (PD), Activity Data (AD), and Geolocation Data (GD) [12]. PD comprises personal attributes like name, age, and gender, which a user usually fills in the tracker at the time of tracker activation. AD includes health-related details like step counts and calories burnt measured by the tracker. GD contains information about the user's mobile and its location. We understand that modern wearable trackers are susceptible to privacy violations since they collect PD, AD, and GD, which can be leaked through the tracker [13], [14]. Wearable fitness tracker vendors might gain data (PD, AD, and GD) ownership and might also collect more information without the user's awareness [12]–[14], which can later be stored on a cloud server or sold to third-party vendors [12], [15].

Camera-based fitness trackers used to monitor human activity generally pose privacy concerns [16]. Or they require complex algorithms to protect privacy by blurring the face of the user in the video [10], at the cost of additional computation overhead. We understand these trackers require extra hardware for processing, like a PC/cloud server. Thus, limiting the portability and scalability of the system. These trackers might fail in low light conditions (e.g., shadows, foggy conditions).

tinyRadar, a radar-based fitness tracker, can overcome the shortcomings mentioned above through contactless sensing and tracking the whole body motion providing accurate results compared to wearable trackers. It gives compact point cloud data which inherently protects user privacy. Also, it collects only Activity Data (AD), preserving the user's anonymity. Previous works [17]–[19] propose using mmWave radar for fitness or human activity tracking, where they use an external PC for real-time processing. Using an external PC for processing increases power consumption and limits the portability and scalability of the system. [5], [20], [21] highlights the importance of on-edge systems providing real-time, fast, portable, and secure solutions on resource constraint hardware.

We propose tinyRadar as a fitness tracker, with IWR1843 mmWave radar as a sensing and processing modality and ESP32 for wireless data transmission. IWR1843 mmWave radar operates in the frequency range of 76-81 GHz and provides high-resolution point cloud data of the target environment, which helps to differentiate various human activities. It has an on-chip Hardware Accelerator (HWA), a Cortex®-R4F microcontroller which runs at 200 MHz, and a Digital

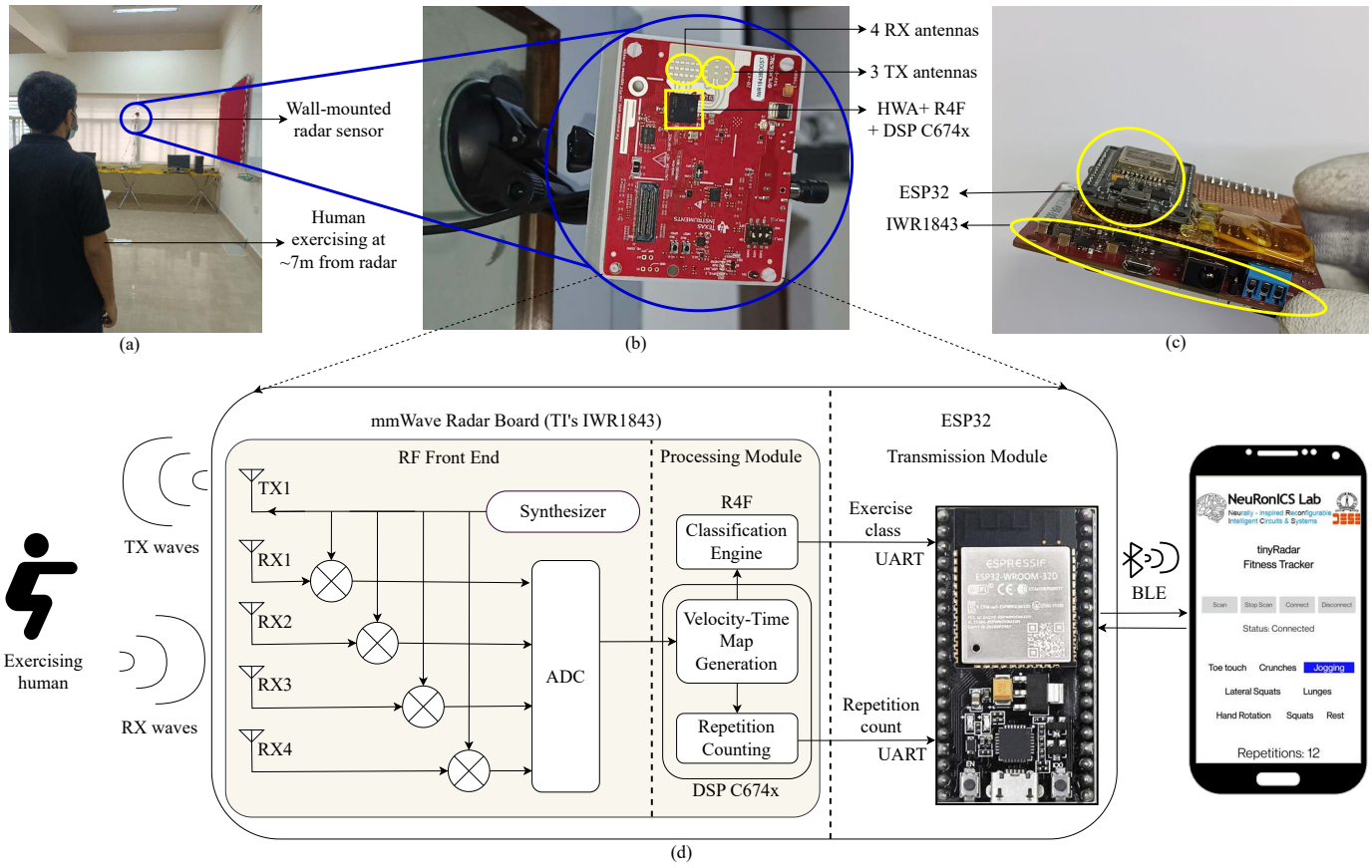

Fig. 1. Top-level block diagram of tinyRadar fitness tracker comprising IWR1843 mmWave radar (we used one TX and four RX antennas) as sensing and processing modality and ESP32 for wireless transmission. (a) An operational scenario of wall-mounted tinyRadar with a human exercising at $\sim$ 7m. (b) Zoomed view of tinyRadar with IWR1843. (c) Opened view of tinyRadar showing the ESP32 integrated with the radar board. (d) Block diagram indicating the signal and implementation flow from sensing the target environment using the RF front-end to VT map generation, classification, repetition counting and transmission of results.

Signal Processor (DSP) C674x, which runs at 600 MHz. The availability of these processors on a single chip enables the on-edge implementation of signal processing algorithms and deep-learning models in real-time.

This work is an extension of our previous work [5], where we showed the applicability of mmWave radar for human activity classification by deploying a quantized CNN model on board for classification on the IWR6843 board. We have extended this work by developing an end-to-end, complete plug-and-play system using mmWave radar IWR1843, ESP32, and a mobile application. We performed hardware integration between IWR1843 and ESP32 boards using a separate daughter board. We performed software integration and synchronization between modules running on different processors, where the CNN-based classification network runs on Cortex-R4F, and the repetition counter runs on DSP C674x. We implemented an onboard repetition counting algorithm which involves heavy signal processing techniques from scratch with efficient utilization of resources to count the number of times the user performs a particular exercise. We have optimized the onboard processing chain using task parallelization and ring buffer implementation to decrease the system's latency and memory footprint, respectively. We extended the number of classes from four in the previous work to eight to showcase the system's scalability. We also generated our own dataset from 14 (10 for training + 4 for testing) human subjects for eight exercises, available at [22].

II. System Description

A Frequency-Modulated Continuous Wave (FMCW) radar continuously transmits chirps in the form of Radio Frequency (RF) waves. A chirp is a signal whose frequency increases linearly with time, and a collection of chirps constitutes a frame. The radar is configured with the desired chirp and frame parameters for the target application.

As shown in Fig. 1, the radar is configured to use one TX and four RX antennas. The processing starts by mixing the transmitted chirp with the received chirp to generate an Intermediate Frequency (IF) signal, which is later digitized using the onboard Analog-Digital Converter (ADC). The range information of the target is directly related to the frequency of the IF signal. Hence, Fast Fourier Transform (FFT) is performed on the digitized data across the samples of each IF signal to extract range information. Range processing is performed in HWA during the acquisition phase. The phase differences among the consecutive IF signals contain the velocity information. Hence, FFT is performed on the Range-Time maps across the IF signals resulting in Range-Velocity

maps. The Range-Velocity maps (for each frame) across the four receive antennas are later converted into a velocity column by performing incoherent addition across the receive antennas and range bins. The velocity columns are stitched across multiple frames to create a Velocity-Time map (VT map). VT maps contain unique activity signatures and are used for classification and repetition count computation. The onboard DSP C674X is used to generate the Range-Velocity and VT maps.

The classification network for exercise recognition and repetition counting algorithm is deployed on Cortex®-R4F and DSP C674x, respectively, with VT map as input. The classification network classifies eight exercises using a lightweight CNN model deployed using the Common Microcontroller Software Interface Standard - Neural Network (CMSIS-NN) framework [23]. The radar communicates the classification result and repetition counts to ESP32 via Universal Asynchronous Receiver Transmitter (UART), which is transferred to a smartphone over BLE for monitoring. The BLE mobile application is built using MIT APP Inventor, an open-source web-based application for building android applications.

III. Dataset Description

The chirp configurations of tinyRadar are shown in TABLE I. For our application, the radar was configured to sample each chirp at a rate of 3000 Ksps with 241 samples to measure activities up to a maximum distance of ~ 8 m. The maximum measurable range of the radar depends on peak power, the gain of the transmit and receive antenna, the effective aperture of the antenna, Radar Cross Section (RCS), which is the effective area of the target from which reflections are received, and the frequency of operation of the radar. We kept the maximum measurable range to ~ 8 m since tinyRadar is a wall-mounted tracker, and the user would perform activity inside his personal space by remaining close to the sensor. The bandwidth of a chirp is the product of frequency slope and ramp end time. The range resolution of radar is inversely proportional to its bandwidth. The bandwidth of the radar was set to 3.969 GHz to achieve a range resolution of ~ 4 cm.

Increasing the number of chirps in a frame or the chirp duration improves the velocity resolution. But, increasing chirp duration also decreases the maximum measurable velocity. Each frame was configured with 124 chirps, each chirp with a time duration (ramp end time + idle time) of 287 μs to achieve a velocity resolution of 6 cm/s and a maximum measurable velocity of 4.23 m/s. The frame duration was set to 70 ms to keep the duty cycle of the radar within 50% to meet hardware constraints. All four receive antennas were utilized for VT map generation to increase the system's Signal-to-Noise Ratio (SNR).

Training data comprising VT maps of eight exercises across 10 people was collected using the mmWave radar board. The radar was placed at the height of ~ 1 m, and exercises were performed by the people in the line of sight of the radar. The measured velocity is close to the target velocity when the activity is performed in the line of sight and gradually decreases as the user moves away from the line of sight. Each exercise was performed in two sessions of 2.5 minutes. Thus, we collected training data of 10 persons x 8 activities x 2 sessions x 2.5 minutes = 400 minutes of data ($\sim 3,43,000$ frames of data). The data was transferred via UART to a local PC using a Python-based script for offline training. After data cleaning, a total of $4,461$ VT maps were generated, each of size 64 x 64.

TABLE I
Chirp Configurations

Symbol	Parameter	Value
S	Frequency slope	45.624 MHz/μs
T_{idle}	Idle time	199.8 μs
N_r	ADC samples	241
F_s	Sample rate	3000 Ksps
T_{end}	Ramp end time	87 μs
N_c	Chirps per frame	124
T_{fr}	Frame periodicity	70 ms
N_{tx}	TX antennas	1
N_a	RX antennas	4
R_{max}	Maximum range	7.89 m
R_{res}	Range resolution	4.09 cm
V_{max}	Maximum velocity	4.23 m/s
V_{res}	Velocity resolution	6 cm/s

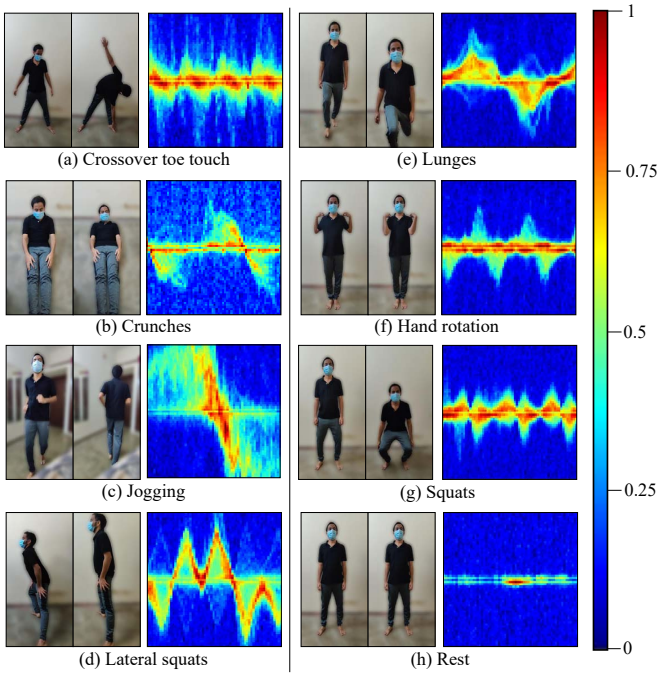

Fig. 2. Exercise types and their VT maps: Snapshots of exercise performed by the user (left) and corresponding normalized VT map (right). In VT maps, the x-axis represents time ranging from 0 to 4.5 seconds, and the y-axis represents velocity ranging from -2 to $+2$ m/s.

Fig. 2 highlights the eight different exercises: Crossover toe touch, Crunches, Jogging, Lateral squats, Lunges, Hand rotation, Squats, and Rest, with corresponding normalized VT maps. The eighth activity, 'Rest', corresponds to when the person is idle or is not present in front of the radar.

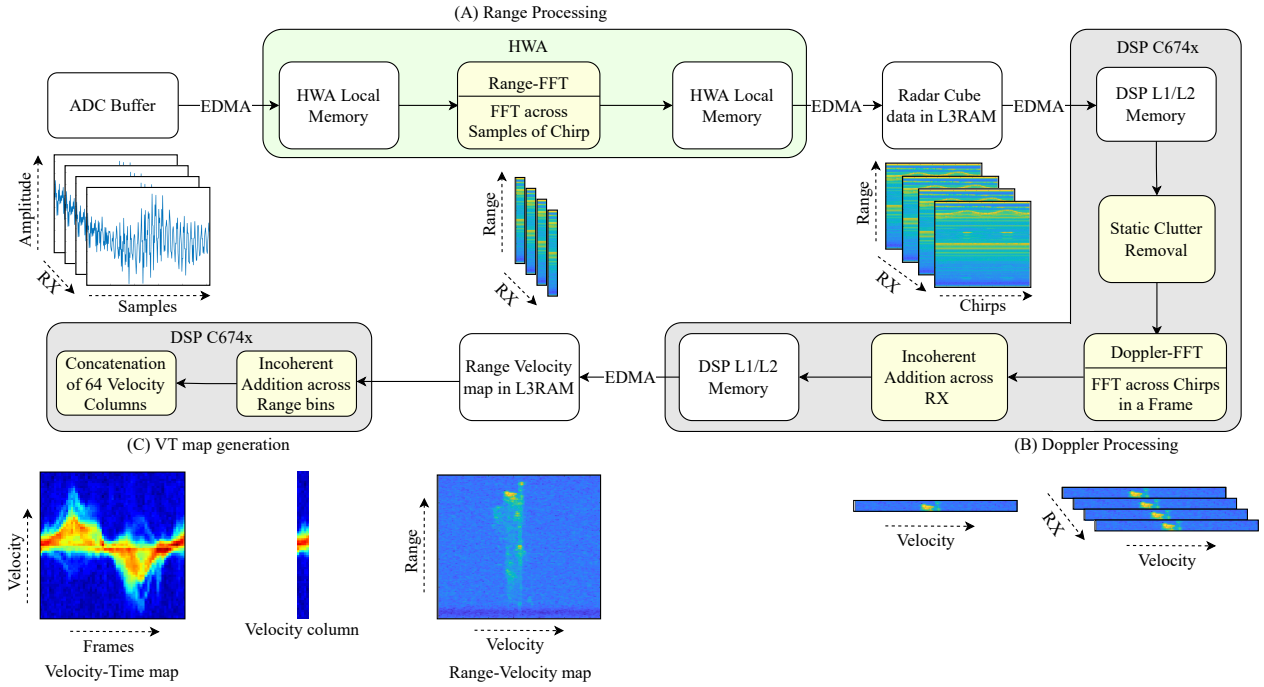

Fig. 3. Flow chart showing radar signal processing steps, the intermediate outputs, and the data flow among different cores of radar board for generation of the VT map. The white blocks symbolize the different memory elements of the radar board. (A) Range processing on the ADC samples of a single chirp for all the RX at a time. (B) Doppler processing on the chirps of a single range bin for all the RX at a time. (C) Velocity-Time map generation.

IV. Radar Signal Processing

The radar signal processing is performed onboard to generate VT maps. It starts with range processing, followed by static clutter removal, doppler processing and incoherent addition of the Range-Velocity maps across the receive antennas and range bins to provide a velocity column for each frame. Concatenation of velocity columns across multiple frames creates a VT map. A detailed explanation of each processing step is provided in the following sub-sections.

A. Range Processing

As shown in Fig. 3, the samples of a chirp are stored in the ADC buffer after digitization for all the RX antennas. Enhanced Direct Memory Access (EDMA) transfers the chirp samples to HWA's local memory from the ADC buffer. HWA applies $blackman$ window and performs FFT across the samples of the chirp in a 16-bit fixed-point format as described in (1). This operation is called Range-FFT.

$$X_r(a,r) = \sum_{s=0}^{N_r-1} x(a,s) \cdot W_b(s) \cdot exp\left(\frac{-i2\pi rs}{N_r}\right) \quad (1)$$

Here $x(a,s)$ denotes the ADC data with a^{th} receiver and s^{th} sample of the chirp, where $s \in [0, N_r)$, and $a \in [0, N_a)$. $X_r(a,r)$ denotes range processed data of a^{th} receiver, and r^{th} range bin, where $r \in [0, N_r)$. W_b denotes $blackman$ window of length N_r as described in (2).

$$W_b(s) = 0.42 - 0.5\,cos\left(2\pi\frac{s}{N_r-1}\right) + 0.08\,cos\left(4\pi\frac{s}{N_r-1}\right) \quad (2)$$

EDMA transfers the processed data from HWA's local memory to L3RAM after the completion of Range-FFT of each chirp. A radar data cube of dimension N_c x N_a x N_r is formed in the L3RAM after the completion of range processing for all the chirps in a frame.

B. Doppler Processing

EDMA transfers one range bin at a time from L3RAM to DSP local L1/L2 memory, on which the DSP C674x performs the rest of the signal processing during the inter-frame period, which begins with static clutter removal.

1) Static Clutter Removal: Static clutter removal is explained in (3), where averaging is performed on all the chirps of a range bin for each receive antenna, followed by subtraction of the obtained average from the corresponding range bin of radar cube data.

$$X_{rm}(a,r) = \frac{1}{N_c}\sum_{c=0}^{N_c-1} X_r(c,a,r)$$
$$X_{scr}(c,a,r) = X_r(c,a,r) - X_{rm}(a,r) \quad (3)$$

Here X_r, X_{rm}, X_{scr} represent the radar cube data, chirp averaged data, and radar cube data with static clutter removed, respectively, where $a \in [0, N_a)$, and $c \in [0, N_c)$. X_{scr} contains information on moving objects.

2) Doppler FFT: After static clutter removal, DSP C674x applies a $hanning$ window and performs 16-bit fixed-point FFT across the chirps of a range bin as described in (4). This operation is called Doppler-FFT.

$$X_d(v,a,r) = \sum_{c=0}^{N_c-1} X_{scr}(c,a,r) \cdot W_h(c) \cdot exp\left(\frac{-i2\pi vc}{N_c}\right) \quad (4)$$

Here $X_{scr}(c,a,r)$ denotes the radar cube data after static clutter removal with c^{th} chirp, a^{th} receiver, and r^{th} range bin, where $a \in [0, N_a)$, and $c \in [0, N_c)$. X_d denotes the doppler processed data, and W_h denotes $hanning$ window of length N_c as described in (5).

$$W_h(c) = \frac{1}{2}\Big(1 - cos\,(2\pi c N_c)\Big) \quad (5)$$

3) Incoherent addition across RX: After completing Doppler-FFT for all the receive antennas of range bin r, incoherent addition across RX is performed as described in (6).

$$Y_d(v,r) = \frac{1}{N_a} \sum_{a=0}^{N_a-1} \big|X_d(v,a,r)\big| \quad (6)$$

Here $X_d(v,a,r)$ denotes the doppler processed data with v^{th} velocity bin, a^{th} receiver, and r^{th} range bin, where $a \in [0, N_a)$, and $v \in [0, N_c)$. Y_d denotes the Range-Velocity map, and $|.|$ denotes the log magnitude operation. EDMA then transfers $Y_d(v,r)$ from DSP's local memory (L1/L2) to L3RAM for range bin r. A Range-Velocity map of dimension N_r x N_d is formed in L3RAM after completing doppler processing for all the range bins.

C. Velocity-Time Map Generation

The incoherent addition of range bins in the Range-Velocity map creates a velocity column on DSP C674x as described in (7) and is stored in L3RAM.

$$Y_d(v) = \frac{1}{N_r} \sum_{r=0}^{N_r-1} \big|Y_d(v,r)\big| \quad (7)$$

Here $Y_d(v,r)$ denotes the Range-Velocity map with v^{th} velocity bin, and r^{th} range bin, where $r \in [0, N_r)$. Y_d denotes the velocity column, and $|.|$ denotes the log magnitude operation.

Each velocity column is subjected to FFT-shift followed by a cropping operation. The FFT-shift operation brings the zero velocity bin to the centre. We observed that the dominant signatures were present $\frac{N_c}{4}$ bins above and below the zero velocity bin. Cropping operation as described in (8) is performed to reduce the memory footprint by half for the CNN network.

$$Y_{dr} = Y_d\left(\frac{N_c}{4} - 1 : 3\frac{N_c}{4}\right) \quad (8)$$

Here Y_d, Y_{dr} denote the original and reduced velocity columns, respectively.

The velocity column generated per frame is stored in the ring buffer, allocated on L3RAM. Concatenation of Y_{dr} velocity columns across multiple frames generates a 16-bit fixed-point VT map in the ring buffer. The ring buffer is shared between DSP C674x and Cortex®-R4F for easy access to data, using which Cortex®-R4F accesses the VT map to perform the classification task.

V. CNN Based Activity Classification

CNN is a data-driven classification network that learns the features through the training process, unlike traditional algorithms, such as Support Vector Machine (SVM), requiring hand-engineered feature selection. VT maps contain unique signatures for different human activities. With a VT map as input, a CNN can be trained to understand the representations for identifying various human activities and provide excellent classification performance with relatively few trainable parameters. The details of the proposed CNN architecture and training parameters are listed in the following sub-sections.

A. Data Normalization

The 16-bit fixed-point input VT map is normalized to 8-bit fixed-point format using (9).

$$x(m,n) = 255\left(\frac{x(m,n) - x_{min}}{x_{max} - x_{min}}\right) - 128 \quad (9)$$

Here x, x_{min} and x_{max} represent the VT map, minimum and maximum values of the VT map respectively, where $m, n \in [0, 64)$.

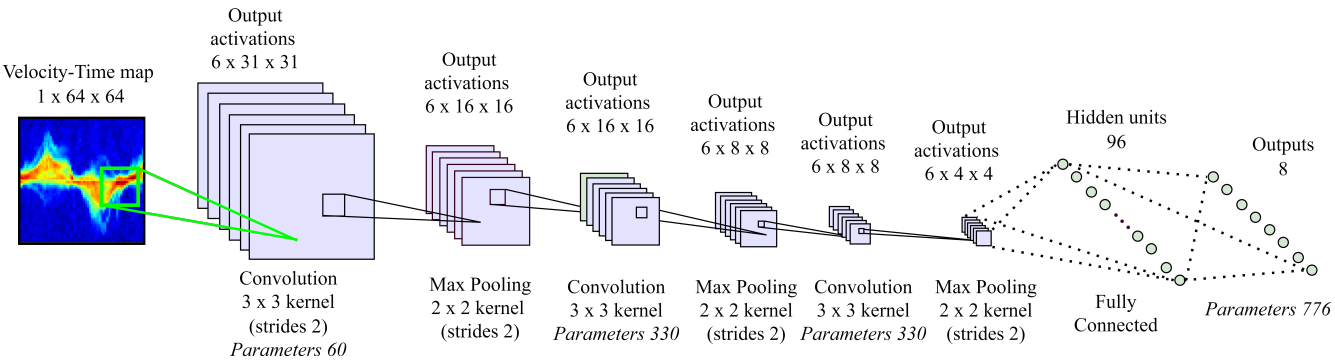

Fig. 4. Proposed CNN architecture

B. Training, Quantization and Deployment

Fig. 4 shows the network architecture of the onboard CNN model to recognise the eight exercise types mentioned in Section III. Training data collection across ten human subjects for the eight activities resulted in 4,461 VT maps. The CNN is trained on 5000 epochs for multinomial logistic loss function using Adam optimizer with L2 regularisation, a learning rate of 0.001, a momentum of 0.9, and a batch size of 64. The trained CNN network contains only 1,496 32-bit parameters (weights and biases).

The offline training and post-training quantization are performed using Convolutional Architecture for Fast Feature Embedding (Caffe) [24] framework. 8-bit quantization of CNN parameters provides a four-fold advantage in terms of memory saving and speed. The 8-bit quantized CNN is deployed on the Cortex®-R4F board using CMSIS-NN custom Application Programming Interfaces (APIs) and is tested on four subjects who were not a part of the training session to evaluate the subject-independent accuracy of the trained model.

VI. Repetition Counts Algorithm

A VT map shows periodicity related to the repetition counts of the exercise. Intuitively, performing 2D FFT on the VT map and finding the index of the maximum peak can give the repetition counts. However, computing 2D FFT on the board can be expensive in resource utilization. The algorithm shown in Fig. 5 computes the repetition counts by deriving a one-dimensional envelope from the VT maps. Four consecutive VT maps of size 64 x 64 are concatenated to form an observation window of size 64 x 256. The observation window is converted to a one-dimensional envelope, and counts are computed from its magnitude spectrum. The observation window of 256 frames helps improve the FFT resolution compared to a window of 64 frames. A detailed explanation of each step implemented on DSP C674x is provided in the following sub-sections.

A. Normalization

Normalization as a first step provides full-scale contrast to the observation window realized using (10) to get an unsigned 8-bit fixed-point representation.

$$x(m,n) = 255\left(\frac{x(m,n) - x_{min}}{x_{max} - x_{min}}\right) \tag{10}$$

Here x, x_{min} and x_{max} represent the observation window, the minimum and maximum value of the observation window, respectively, where $m \in [0, 64)$, and $n \in [0, 256)$.

B. Splitting of the Observation Window

Spurious signatures in the observation window can often lead to miscalculating repetition counts since the calculation is based on the FFT method. We observed that the spurious signatures were not simultaneously present in positive and negative velocity bins. Thus, the observation window is split into two halves across the zero velocity axis as described in (11). The count with the maximum confidence level out of the two is chosen as the repetition count described in section VI-H.

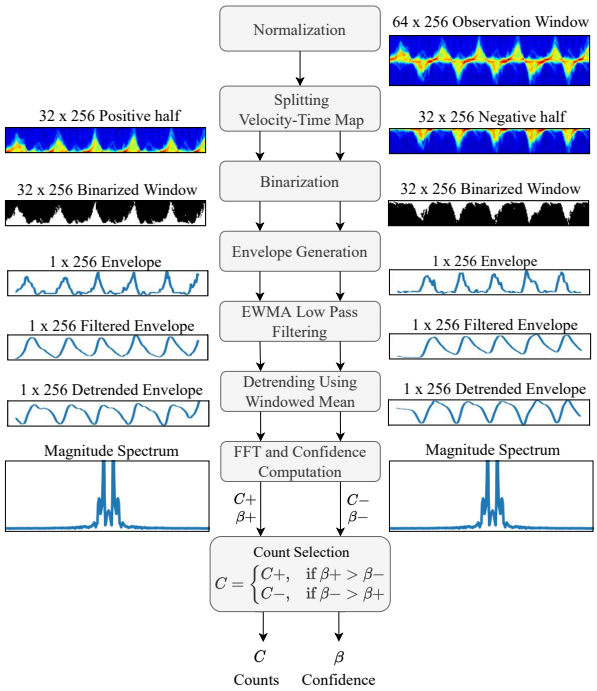

Fig. 5. Repetition counts algorithm flowchart showing each step and the corresponding output.

$$\begin{aligned} x_P(m,n) &= x(m,n) \\ x_N(m,n) &= x(m+32,n) \end{aligned} \tag{11}$$

Here x, x_P and x_N represent the observation window, its positive and negative half, respectively, where $m \in [0, 32)$, and $n \in [0, 256)$. Both x_P and x_N will be represented as y in the subsequent sub-sections.

C. Binarization

Binarization assigns equal value to the effective pixels in the observation window, which helps to provide a smooth one-dimensional envelope. Each observation window is binarized to unsigned 8-bit fixed-point format by thresholding it against its mean value as described in (12).

$$y(m,n) = \begin{cases} 255, & \text{if } y \geq y_{mean} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

where y_{mean} is the average value of the observation window, y.

D. Envelope Generation

Computing the average velocity across each frame of the binarized observation window generates the one-dimensional envelope as described in (13).

$$y_e(n) = \frac{1}{M}\sum_{m=0}^{M-1} y(m,n) \tag{13}$$

Here y and y_e represent the observation window and its envelope, respectively, where $M = 32$ and $n \in [0, 256)$.

E. Low Pass Filtering

The envelope is low-pass filtered using an Exponentially Weighted Moving Average (EWMA) filter. EWMA filter eliminates the high-frequency contents by taking a weighted average as described in (14).

$$\hat{y}_e(n) = \alpha\, y_e(n) + (1-\alpha)\, \hat{y}_e(n-1) \tag{14}$$

Here y_e and $\hat{y}_e$ are the original and low pass filtered envelope, respectively, where $n \in [0, 256)$ and α is the smoothing parameter between zero and unity.

F. Detrending

The filtered envelope might have a trend due to the varying rate at which the user exercises. Subtracting the local mean computed over a sliding window of the length of 16 samples removes the trend from the filtered envelope as described in (15).

$$y_d(m) = \hat{y}_e(m) - \hat{y}_{e_{mean}} \tag{15}$$

Here $\hat{y}_e$, $\hat{y}_{e_{mean}}$, and y_d represent the filtered envelope, its mean computed within the window length and detrended envelope, respectively, where $m \in [i, i+16)$, and $i \in [0, 242)$.

G. FFT and Confidence Computation

Performing FFT on the detrended envelope and finding the index corresponding to the maximum peak in the magnitude spectrum determines the repetition counts in the observation window. The confidence β in the detected peak is computed as described in (16).

$$\beta = \frac{E_p}{E_s} \tag{16}$$

where,

$$\begin{aligned} E_p &= \sum_{n=i-M}^{i+M} |Y_d(n)|^2 \\ E_s &= \sum_{n=0}^{N-1} |Y_d(n)|^2 \end{aligned} \tag{17}$$

Here Y_d represents the magnitude spectrum of the detrended envelope y_d of length $N(=256)$, E_p represents the energy in the peak with index i within a window of length $2M+1$, and E_s represents the energy in the entire spectrum.

H. Repetition Counts Computation

Each half of the observation window provides a count and its confidence value. The count with maximum confidence is chosen as the repetition count of the exercise in the current observation window. Repetition counts from multiple observation windows of the same exercise with a sliding window of length 16 are stored in a buffer $cnts$ to compute the actual repetition counts $Reps$ as described in (18).

$$Reps = \frac{\sum cnts}{N_w} \frac{L_d}{L_w} \tag{18}$$

where $cnts$, N_w, L_d, and L_w denote the counts' buffer, the number of observation windows, total data length, and observation window length ($L_w = 256$) of the exercise respectively.

VII. Hardware Implementation

The real-time hardware implementation of tinyRadar works by proper integration and synchronization of signal processing chain running on HWA and DSP C674x and CNN-based inference engine running on Cortex-R4F.

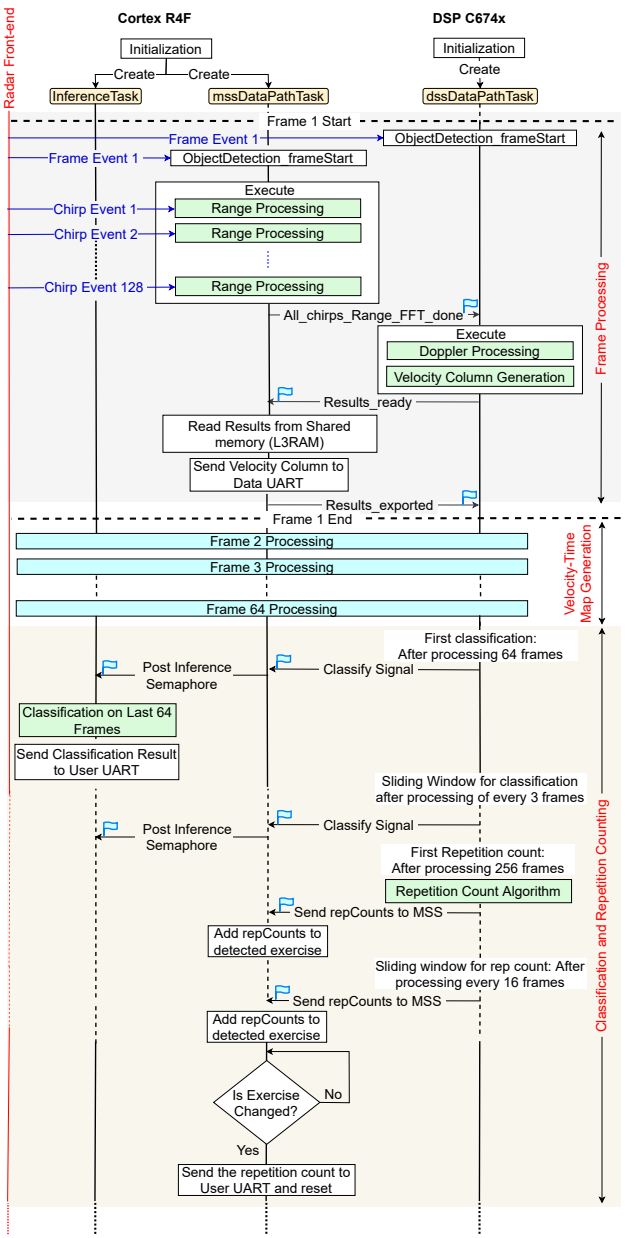

Fig. 6. Hardware implementation flowchart showing the sequential flow from initialization, frame processing, VT map generation to classification & repetition counting. Synchronization between different tasks is achieved through flags (shown in cyan).

A. Initialization

RF front-end, UART, ESP32, and other drivers are configured and initialized when the board is powered ON. After all the drivers are initialized, the highest priority `mssDatapathTask` is created on Cortex®-R4F to manage the datapath and handle the signals sent to and received from DSP C674x. Similar to `mssDatapathTask`,

dssDatapathTask is created on DSP C674x to manage the datapath and handle the signals sent to and received from Cortex®-R4F. InferenceTask is created on Cortex®-R4F for classification inference implementation.

B. Frame Processing

Once the radar front-end has been configured and all the tasks have been created and synchronized, mssDatapathTask calls the Execute API on every frame event, where HWA performs Range-FFT on the chirp data during the acquisition period. dssDatapathTask remains in a pending state during range processing. After the range processing is completed, upon reception of *All_chirps_Range_FFT_done* flag from mssDatapathTask, dssDatapathTask calls the Execute API to perform doppler processing and velocity column generation on the radar cube.

dssDatapathTask stores the velocity column in ring buffer and sends *Results_ready* flag to mssDatapathTask after completion of signal processing chain on DSP C674x. mssDatapathTask sends the velocity column to the DATA UART port after reception of *Results_ready* signal and acknowledges dssDatapathTask with a *Results_exported* flag. DATA UART port is used for logging velocity column data. As seen in Fig. 6, this part of the execution phase is called frame processing and repeats for every frame event.

C. Classification and Repetition Counting

A ring buffer is implemented to store the velocity columns for every frame sequentially. After processing the first 64 frames, dssDatapathTask sends the address of the latest VT map and *Classify_signal* to mssDatapathTask, which then posts the inference semaphore to the InferenceTask for classification. The above process repeats itself with every three new frames of sliding window implemented on the ring buffer. InferenceTask writes the exercise label on the USER UART port when the inference engine detects a change in exercise.

The repetition Counting algorithm runs on DSP C674x in the dssDatapathTask. It starts after processing the first 256 frames and repeats after every 16 frames. The calculated counts are sent to mssDatapathTask and accumulated for the current exercise reported by CNN. The total repetition counts for a particular exercise are calculated from the accumulated sum using (18) and are written on the USER UART port when the user changes the exercise. The ESP32, which is interfaced with the radar board, receives the exercise type and repetition count through the USER UART port and sends the same to a mobile application via BLE.

VIII. Results and Discussion

The radar signal processing, repetition counting and classification pipeline are implemented on IWR1843. The memory consumption, processing latency, implementation results, comparative study and limitations of the system are described in the subsequent subsections.

A. Memory Utilization and Processing Latency

TABLE II contains the memory required to store the CNN's parameters and output activations, output buffers of each signal processing stage and the corresponding processing latency. The processing latency includes the time required for memory read/write, data transfer and signal processing. The velocity column generation and repetition counting chains have a total memory footprint of 560.13 KB and 40 KB on L3RAM, respectively. The CNN-based inference engine implemented on Cortex®-R4F uses 11.36 KB of memory space, utilizing $\sim$ 5.76% of total available data RAM.

TABLE II
Memory Consumption and Processing Latency

Processing Stage		**Core**	**Memory Consumption**	**Processing Latency** (ms)
Velocity Column Generation	Range processing	HWA	496 KB	$\sim$ 6.66
	Doppler processing	C674x	64 KB	$\sim$ 4.73
	Incoherent addition of range bins		128 B	$\sim$ 2.97
Repetition Counting Chain			40 KB	$\sim$ 3.69
CNN	Parameters	R4F	1.46 KB	$\sim$ 69
	Output activation		9.9 KB	

B. Power Consumption

The Cortex®-R4F, DSP C674x, and HWA consume 80 mW, 350 mW, and 100 mW of power, respectively. tinyRadar at present consumes a total of $\sim$1W power with the RF front end. The CNN-based inference engine uses 60.8% of CPU resources, resulting in 48.64 mW (0.608 x 80 mW) power consumption out of 80 mW.

C. Classification Results

The 8-bit quantized CNN engine correctly classified 97% of the exercises in real-time when tested on the rest of the four human subjects who were not a part of the training session. The confusion matrix described in TABLE III highlights the accuracy achieved per class by the onboard CNN network. Jogging and Lunges were classified with 100% accuracy, whereas Crunches was classified with an accuracy of 90.6%.

TABLE III
Confusion Matrix in Percentage

Predicted $\rightarrow$ Actual $\downarrow$	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
(A) Crossover toe-touch	98	0	0	0	0	2	0	0
(B) Crunches	1.2	90.6	1.2	0	0	7	0	0
(C) Jogging	0	0	100	0	0	0	0	0
(D) Lateral squats	0	0	0	97.2	2.9	0	0	0
(E) Lunges	0	0	0	0	100	0	0	0
(F) Hand rotation	1	0	0	0	0	95.2	3.8	0
(G) Squats	1	0	0	0	0	0	99	0
(H) Rest	0	1	0	0	0	0	0	99

D. Repetition Counts Results

Using the repetition counting algorithm, we obtained an average counting accuracy of 96% on 80 test cases where users performed exercises with repetition counts ranging from 20 to 120. As shown in Fig. 7, we measured a maximum median error of 2.2% in the case of Crunches and a minimum of 0% in cases of Jogging, Lateral squats, and Squats.

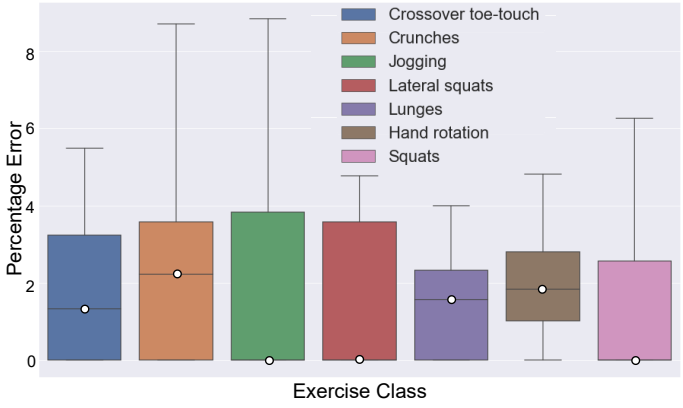

Fig. 7. Box plot representing the error percentage of repetition counts against the exercise classes. The white dot represents the median error percentage.

E. Comparison

We measured the system performance of different fitness trackers using metrics like primary sensors present, features measured, on-edge model presence and accuracy. To the best of our knowledge, a qualitative comparison to evaluate tinyRadar against other similar fitness-tracking technologies is presented in TABLE IV. Wearable fitness trackers contain many sensors and provide rich information with moderate accuracy. Other works in non-contact technology give a result with high accuracy but at the cost of computational overhead by performing inference on a PC. Though the types of exercise classified by each work are different, tinyradar performs better using a compact onboard classification model with high accuracy.

F. Limitations

tinyRadar is a wall-mounted system designed for personalised spaces such as rooms and gardens. At present, it can detect the activity of a single person up to a distance of ~ 8 m and provides the best classification accuracy when exercise is performed in the line of sight of radar. Large deviations from the line of sight could decrease the system's classification accuracy.

IX. Conclusion

We have presented tinyRadar as a contactless, edge-enabled real-time mmWave radar-based fitness tracker. It offers several advantages over wearable fitness trackers, such as contactless sensing, privacy preservation, and provides accurate data by tracking whole-body motion. It gives excellent real-time classification accuracy of 97% and average counting accuracy of 96%. tinyRadar, compared to other non-contact technologies like camera-based trackers, offers significant advantages like robustness to low light conditions. Also, it processes sparse and compact point cloud data which inherently preserves users' privacy which is not the case with conventional camera-based trackers.

The proposed architecture can be scaled to detect multiple people, and classify more activities, such as fall detection, to remotely monitor patients suffering from diseases such as dementia and Alzheimer's in a healthcare setting. To the best of our knowledge, our work on human activity classification and the present work on tinyRadar as a fitness tracker are the first to demonstrate tinyML-based CNN implementation on mmWave radar boards using VT maps.

TABLE IV
Comparison of tinyRadar With Other Existing Technologies

Device	Primary Sensors	Features Measured	On-edge Models Present	Accuracy
Fitbit* [25]	Accelerometer Heart meter Altimeter Gyroscope	Steps count	Yes	Moderate
		Heart rate		
		Calories burnt		
		Sleep cycle		
mmFit [19]	Radar	Exercise classification	No**	High
		User identification		
mmFiT [17]	Radar	Exercise classification and Repetition counting	No**	High
GymCam [10]	Camera		No**	High
tinyRadar (this work)	Radar		Yes	High

* Fitbit symbolises the different wearable trackers with most common features and sensors
** indicates inference is performed on a PC

References

[1] Lawton, E., Brymer, E., Clough, P., amp; Denovan, A. (2017). The relationship between the physical activity environment, nature relatedness, anxiety, and the psychological well-being benefits of regular exercisers. Frontiers in Psychology, 8. https://doi.org/10.3389/fpsyg.2017.01058

[2] Shin, G., Jarrahi, M. H., Fei, Y., Karami, A., Gafinowitz, N., Byun, A., Lu, X. (2019). Wearable activity trackers, accuracy, adoption, acceptance and health impact: A systematic literature review. Journal of biomedical informatics, 93, 103153. https://doi.org/10.1016/j.jbi.2019.103153

[3] Thomas, J. G., Raynor, H. A., Bond, D. S., Luke, A. K., Cardoso, C. C., Foster, G. D., Wing, R. R. (2017). Weight loss in Weight Watchers Online with and without an activity tracking device compared to control: A randomized trial. Obesity (Silver Spring, Md.), 25(6), 1014–1021. https://doi.org/10.1002/oby.21846

[4] B. R. Pradhan, Y. Bethi, S. Narayanan, A. Chakraborty and C. S. Thakur, "N-HAR: A Neuromorphic Event-Based Human Activity Recognition System using Memory Surfaces," 2019 IEEE International Symposium on Circuits and Systems (ISCAS), 2019, pp. 1-5, doi: 10.1109/ISCAS.2019.8702581.

[5] S. S. Yadav, R. Agarwal, K. Bharath, S. Rao and C. S. Thakur, "tinyRadar: mmWave Radar based Human Activity Classification for Edge Computing," 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 2022, pp. 2414-2417, doi: 10.1109/ISCAS48785.2022.9937293.

[6] Balbim, G. M., Marques, I. G., Marquez, D. X., Patel, D., Sharp, L. K., Kitsiou, S., Nyenhuis, S. M. (2021). Using Fitbit as an mHealth Intervention Tool to Promote Physical Activity: Potential Challenges and Solutions. JMIR mHealth and uHealth, 9(3). https://doi.org/10.2196/25289

[7] Kasparian, A. M., Badawy, S. M. (2022). Utility of Fitbit devices among children and adolescents with chronic health conditions: a scoping review. mHealth, 8, 26. https://doi.org/10.21037/mhealth-21-28

[8] Khatsenko, K., Khin, Y., Maibach, H. (2020). Allergic Contact Dermatitis to Components of Wearable Adhesive Health Devices. Dermatitis : contact, atopic, occupational, drug, 31(5), 283–286. https://doi.org/10.1097/DER.0000000000000575

[9] Boudreaux, B. D., Hebert, E. P., Hollander, D. B., Williams, B. M., Cormier, C. L., Naquin, M. R., Gillan, W. W., Gusew, E. E., Kraemer, R. R. (2018). Validity of Wearable Activity Monitors during Cycling and Resistance Exercise. Medicine and science in sports and exercise, 50(3), 624–633. https://doi.org/10.1249/MSS.0000000000001471

[10] Rushil Khurana, Karan Ahuja, Zac Yu, Jennifer Mankoff, Chris Harrison, and Mayank Goel. 2018. GymCam: Detecting, Recognizing and Tracking Simultaneous Exercises in Unconstrained Scenes. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2, 4, Article 185 (December 2018), 17 pages. https://doi.org/10.1145/3287063.

[11] Fitbit ECG App Instructions for Use, Fitbit, San Francisco, California, United States, Version AM. Accessed: January 24, 2023. [Online]. Available: https://help.fitbit.com/manuals/manual_ecg_en_US.pdf

[12] Mendoza, F., Alonso, L., López, A., & And Patricia Arias Cabarcos, D. (2018). Assessment of Fitness Tracker Security: A case of study. UCAmI 2018. https://doi.org/10.3390/proceedings2191235

[13] Greig Paul and James Irvine. 2014. Privacy Implications of Wearable Health Devices. In Proceedings of the 7th International Conference on Security of Information and Networks (SIN '14). Association for Computing Machinery, New York, NY, USA, 117–121. https://doi.org/10.1145/2659651.2659683

[14] Olabenjo, B., Makaroff, D. (2019). Information Leakage in Wearable Applications. In: Wang, G., Feng, J., Bhuiyan, M., Lu, R. (eds) Security, Privacy, and Anonymity in Computation, Communication, and Storage. SpaCCS 2019. Lecture Notes in Computer Science(), vol 11611. Springer, Cham. https://doi.org/10.1007/978-3-030-24907-6_17

[15] P. Harvey, O. Toutsop, K. Kornegay, E. Alale and D. Reaves, "Security and Privacy of Medical Internet of Things Devices for Smart Homes," 2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS), 2020, pp. 1-6, doi: 10.1109/IOTSMS52051.2020.9340231.

[16] J. Bugeja, D. Jönsson and A. Jacobsson, "An Investigation of Vulnerabilities in Smart Connected Cameras," 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2018, pp. 537-542, doi: 10.1109/PERCOMW.2018.8480184.

[17] Tiwari, Girish; Bajaj, Parveen; Gupta, Shalabh (2021): mmFiT: Contactless Fitness Tracker Using mmWave Radar and Edge Computing Enabled Deep Learning. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.16574588.v1

[18] Tiwari, Girish & Gupta, Shalabh. (2021). An mmWave Radar Based Real-Time Contactless Fitness Tracker Using Deep CNNs. IEEE Sensors Journal. PP. 1-1. 10.1109/JSEN.2021.3077511.

[19] Y. Xie, R. Jiang, X. Guo, Y. Wang, J. Cheng and Y. Chen, "mmFit: Low-Effort Personalized Fitness Monitoring Using Millimeter Wave," 2022 International Conference on Computer Communications and Networks (ICCCN), 2022, pp. 1-10, doi: 10.1109/ICCCN54977.2022.9868878.

[20] H. R. Sabbella, A. R. Nair, V. Gumme, S. S. Yadav, S. Chakrabartty and C. S. Thakur, "An Always-On tinyML Acoustic Classifier for Ecological Applications," 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 2022, pp. 2393-2396, doi: 10.1109/ISCAS48785.2022.9937827.

[21] P. Goswami, S. Rao, S. Bharadwaj and A. Nguyen, "Real-Time Multi-Gesture Recognition using 77 GHz FMCW MIMO Single Chip Radar," 2019 IEEE International Conference on Consumer Electronics (ICCE), 2019, pp. 1-4, doi: 10.1109/ICCE.2019.8662006.

[22] "mmWave Radar Human Exercise Dataset v1," NeuRonICS Lab, Indian Institute of Science. [Online]. Available: https://labs.dese.iisc.ac.in/neuronics/datasets/mmwave-radar/. [Accessed: 13-Oct-2022].

[23] Lai, L., Suda, N., Chandra, V. (2018). CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. ArXiv. https://doi.org/10.48550/arXiv.1801.06601

[24] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv. https://doi.org/10.48550/ARXIV.1408.5093

[25] Fitbit Versa 3 User Manual, Fitbit, San Francisco, California, United States, Version 1.13. Accessed: January 24, 2023. [Online]. Available: https://help.fitbit.com/manuals/manual_versa_3_en_US.pdf

Satyapreet Singh Yadav is a PhD student in Brain and Artificial Intelligence program at Indian Institute of Science, Bangalore, India. She has a Master's degree from the same institute and a Bachelor's degree in Electronics and Communication from the National Institute of Technology Calicut, Kerala, India. She has 8 years of experience in the aerospace and space sector, having worked as a scientist at Indian Space Research Organization (ISRO). Her research interests include brain-inspired computing, machine learning, embedded systems, and radar signal processing.

Radha Agarwal is a radar software developer at Texas Instruments in Bangalore, India. She has a Master's degree from the Department of Electronic Systems Engineering at the Indian Institute of Science, Bangalore, India and a Bachelor's degree in Electronics and Communication from the Institute of Engineering and Technology, Lucknow, India. Her research interests include embedded systems, hardware implementation of algorithms, and edge-enabled deep learning architectures.

Kola Bharath is a GPU Architect at NVIDIA in Bangalore, India. He holds a Master's degree in Electronic Systems Engineering from the Indian Institute of Science, Bangalore, India and a Bachelor's degree in Electronics and Communication Engineering from Rajiv Gandhi University of Knowledge Technologies, Nuzvid, India. He has worked for 3 years in mobile communication at Bharat Sanchar Nigam Limited (BSNL). His research interests include computer architecture for high-performance computing and deep learning applications.

Sandeep Rao is with Texas Instruments, where he leads the mmWave sensing Algorithm group. His current research interests are in the area of Radar Signal Processing including Automotive radar, Interference mitigation strategies and classification. Prior to TI, Sandeep was with Hughes Network Systems where he worked on modems for Satellite Communication. He has more than 30 patents in the area of mmWave Radar and GNSS positioning. He has a masters from the University of Maryland and a bachelor's from IIT Madras.

Chetan Singh Thakur joined the Indian Institute of Science, Bangalore, India as an Assistant Professor in 2017. He received his PhD in neuromorphic engineering at the MARCS Research Institute, Western Sydney University and did his postdoc at the Johns Hopkins University, USA. He also worked for a few years with Texas Instruments Singapore as a senior Integrated Circuit Design Engineer. His research interest is to understand the computing principles of the brain and apply those to build novel intelligent VLSI systems. He has received several awards, such as Pratiksha Trust young investigator award and Inspire Faculty Award from DST for brain-inspired computing.