AURITUS: An Open-Source Optimization Toolkit for Training and Development of Human Movement Models and Filters Using Earables

SWAPNIL SAYAN SAHA, University of California - Los Angeles, USA

SANDEEP SINGH SANDHA, University of California - Los Angeles, USA

SIYOU PEI, University of California - Los Angeles, USA

VIVEK JAIN, University of California - Los Angeles, USA

ZIQI WANG, University of California - Los Angeles, USA

YUCHEN LI, University of California - Los Angeles, USA

ANKUR SARKER, University of California - Los Angeles, USA

MANI SRIVASTAVA, University of California - Los Angeles, USA

Smart ear-worn devices (called earables) are being equipped with various onboard sensors and algorithms, transforming earphones from simple audio transducers to multi-modal interfaces making rich inferences about human motion and vital signals. However, developing sensory applications using earables is currently quite cumbersome with several barriers in the way. First, time-series data from earable sensors incorporate information about physical phenomena in complex settings, requiring machine-learning (ML) models learned from large-scale labeled data. This is challenging in the context of earables because large-scale open-source datasets are missing. Secondly, the small size and compute constraints of earable devices make on-device integration of many existing algorithms for tasks such as human activity and head-pose estimation difficult. To address these challenges, we introduce AURITUS, an extendable and open-source optimization toolkit designed to enhance and replicate earable applications. AURITUS serves two primary functions. *Firstly*, AURITUS handles data collection, pre-processing, and labeling tasks for creating customized earable datasets using graphical tools. The system includes an open-source dataset with 2.43 million inertial samples related to head and full-body movements, consisting of 34 head poses and 9 activities from 45 volunteers. *Secondly*, AURITUS provides a tightly-integrated hardware-in-the-loop (HIL) optimizer and TinyML interface to develop lightweight and real-time machine-learning (ML) models for activity detection and filters for head-pose tracking. To validate the utlity of AURITUS, we showcase three sample applications, namely fall detection, spatial audio rendering, and augmented reality (AR) interfacing. AURITUS recognizes activities with 91% leave 1-out test accuracy (98% test accuracy) using real-time models as small as 6-13 kB. Our models are 98-740× smaller and 3-6% more accurate over the state-of-the-art. We also estimate head pose with absolute errors as low as 5 degrees using 20kB filters, achieving up to 1.6× precision improvement over existing techniques. We make the entire system open-source so that researchers and developers can contribute to any layer of the system or rapidly prototype their applications using our dataset and algorithms.

CCS Concepts: • **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; • **Computing methodologies** → *Machine learning*; • **Computer systems organization** → Embedded systems.

1 INTRODUCTION

Earables – electronic devices that sit inside one's ears – have undergone rapid transformation in recent years [19, 46]. From being primarily used for wireless audio capture and playback in a convenient form factor and natural placement, earables are now being transformed into devices with rich multimodal sensing capabilities, on-device local processing, and storage [19, 103]. While one could always re-purpose the existing microphone and radio transceivers already in the earables [14, 19, 26, 72], many devices have begun to add other sensors [102]. In-ear headphones and smart earbuds are now equipped with ultra-low-power inertial measurement units (IMU) [46], beamforming microphone arrays [19], temperature sensors [69], and heart-rate monitors [79]. Information from these additional sensing modalities not only enhance the primary audio functionality for purposes such as smarter spatial audio but, as recent research activity shows, can also potentially be harnessed to sense the wearer's physical states and contexts, such as facial activity, head motion, vital signs, etc [45].

Among the current applications of earables, human activity detection [5, 38, 39, 51, 64, 75, 92] and head-pose recognition [28, 76] form the principal engineering research problem for most applications [14, 19, 54, 63, 77, 91, 96]. The natural placement of ear-worn devices can provide key information about the wearer's movement and vital signals [19, 45]. As a result, the ability to track head motion and gait data has spawned a plethora of emerging applications. These include dead-reckoning [3, 72, 74], binaural audio rendering [102], robot control [67], authentication [19, 98], context detection [26, 43], health monitoring [32, 60, 66, 80], mixed-reality (MR), AR and tangible interfaces [26, 65, 93, 102]. These emerging applications have been shown to benefit not only general consumers but parties with special needs and interests [19]. For example, dead-reckoning and 3D spatial audio can be combined to provide localization cues for the visually-impaired [3, 14, 65, 102]. Gesture, posture, and expression monitoring have been used to regulate food intake [63], mental health [45, 54, 96], sleep [60], vital signs [32, 66, 80], and proper posture [92] in patients through mHealth. Fitness enthusiasts can use earables not only to listen to music during exercise but also monitor their performance and respiration rate [75, 77, 80]. Given the opportunities, the global earable market is projected to grow exponentially, and significant commercial ventures are already underway [19, 45, 79, 103].

1.1 Challenges

The bulk of emerging innovations builds upon recent advances in machine learning (ML) for sensor-based activity detection and gesture recognition [2, 11, 52, 68] because classical signal processing techniques do not yield high enough accuracy for complex activity recognition. However, the development of ML classifiers for human activity detection (as well as analytical head-pose filters) on earables face the following challenges:

- **The Absence of Large-Scale Earable Datasets and Software Tools**: The data-hungry nature of ML training demands access to large-scale earable datasets. Although abundant datasets are publicly available for smartphones and non-ear-worn wearables [4, 15, 33, 44, 61, 83, 95, 104], the relative newness of earables means that we are missing open-source and well-curated earable datasets [19]. In addition, the software tools enabling automation in the earable data collection pipeline have not received significant attention.

Hence, in the context of earables, the reproducibility and rigorous benchmarking of the performance of models and algorithms remain challenging.

- **The Compute Constraints of Earables**: Earables have tight memory, power, and compute constraints [19]. For example, an earable has only 56 KB RAM and 16 MB of flash [46], compared to 4 GB of RAM and 64 GB of storage available on a smartphone [57]. As a result, directly transferring existing ML classifiers and filters for activity detection and head-pose estimation from other domains (e.g., smartphones and smartwatches) is not feasible for onboard computation on earable.
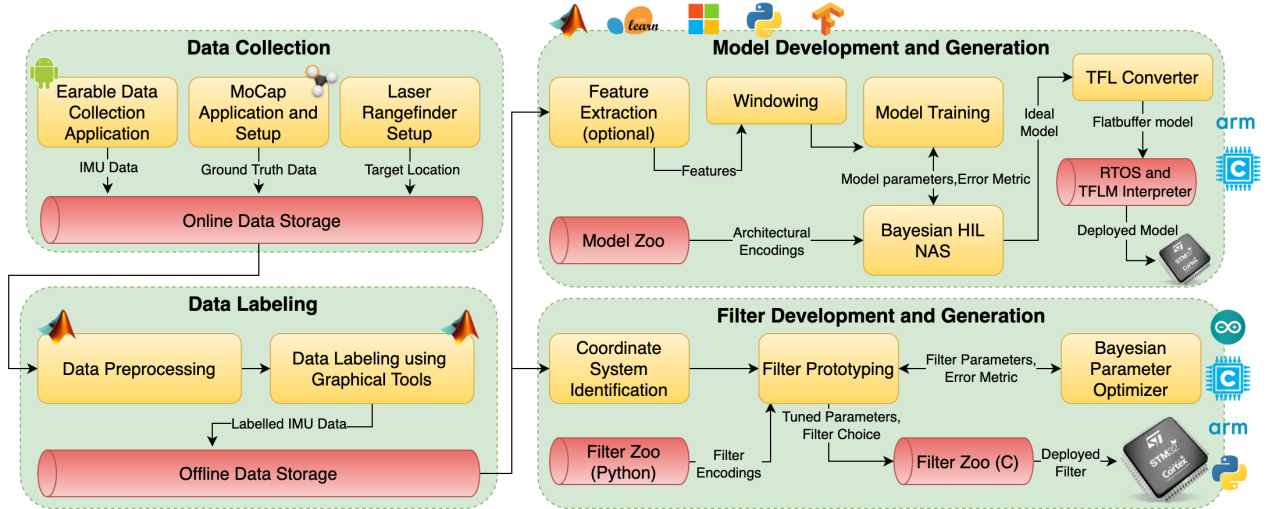

Fig. 1. Architecture of Auritus. The first two modules take care of collecting and labeling high resolution earable data interactively. The development and generation modules allow model and filter optimization through automated HIL Bayesian NAS and optimization, respectively. Yellow boxes signify a process (e.g., transformation, optimization, etc.) and red cylinders signify stored artefacts (e.g., data, models, libraries, etc.).

1.2 Contributions

To address the above challenges hindering earable research, we propose an open-source[1] and extendable optimization toolkit called Auritus that supports an earable research life cycle from data collection to algorithmic development. To tackle earable dataset scarcity, Auritus provides access to an open-source, large-scale, IRB-approved[2] earable dataset from 45 volunteers containing 34 distinct head-poses and 9 classes of simple activities of daily living (ADL) with 2.43 million samples. The dataset is large enough to train high-accuracy activity classifiers and head-pose filters encompassing sufficient statistical diversity. We provide both sliced sequences and continuous trajectories in the world coordinate frame. Further, Auritus is accompanied with tools (data collection and labeling components in Fig. 1) to enable similar dataset collection, pre-processing, and labeling by other researchers and application developers.

To enable the development and training of ML classifiers targeted for resource-constrained earable platforms, Auritus performs ML model training using completely automated hardware-aware neural architecture search (NAS). We achieve this optimization through tight integration of state-of-the-art (SOTA) advances in TinyML

---

[1]https://github.com/nesl/auritus
[2]IRB approved for public release. IRB number: 21-001253

and NAS [78, 85] in the model development and generation stages as shown in Fig. 1. We develop a zoo of well-characterized pre-trained activity classification models and head-pose estimation filters using the optimization workflow in both Python and C targeted towards five different microcontroller devices. The collection contains five lightweight activity models, five conventional activity models, and four lightweight head-pose filters at various points in accuracy - model size space with performance superior to existing models and filters. In our evaluation, we achieve 98% activity detection accuracy (91% leave 1-out accuracy) with 6-13 kB models, which is 98-740× smaller and 3-6% more accurate than the state-of-the-art (SOTA). The included head-pose estimation filters have errors as low as ~5 degrees with a size of 20 kB; providing 1.6× improvement over the SOTA.

Finally, we showcase three representative applications developed using the workflow provided by Auritus, namely fall detection, spatial audio rendering, and AR interfacing. The resulting software, applications, and libraries are completely open-source, explicitly designed to facilitate replication and use by others. Overall, Auritus provides a way to allow others to develop new human movement models and filters on our dataset, collect new data, label data, and compare the results with prior work through a combination of automated scripts and human-in-the-loop. Our target communities include researchers who will benefit from our benchmark dataset by allowing them to compare their newly developed models with existing ones. We also target practitioners looking to deploy onboard human motion inference models and filters by optimizing for specific hardware platforms. Our contributions are summarized as follows:

- **End-to-End Earable Application Development Framework:** To the best of our knowledge, we are the first to provide an end-to-end learning-enabled application development framework for earables. The framework includes the first large-scale open-source human movement dataset, data collection, and labeling tools, a hardware-aware optimization toolkit, and a zoo of well-characterized human movement models and filters.
- **Hardware-in-the-loop Network Architecture Search:** Among all existing NAS frameworks developed for microcontrollers [8][27][56][57], we are the first to provide a gradient-free Bayesian NAS for microcontrollers that supports use of both proxies and communication with real hardware in real-time to receive hardware metrics. We show that existing NAS frameworks fall short due to absence of real hardware or proper search strategy in the NAS phase.
- **A Zoo of Lightweight Models and Filters:** We develop and provide 5 lightweight models, 5 conventional models, and 4 headpose filters with Auritus for activity detection and head-pose estimation. The said lightweight models and filters have not been used in conjunction with NAS for earable activity detection and head-pose tracking before.
- **Pushing the State-of-the-Art:** Our lightweight models and filters significantly reduce inference error and resource consumption than existing earable activity classifiers and head-pose filters. We also showcase three representative applications using Auritus that exploit the optimization pipeline.
- **Challenges and Limitations in TinyML:** Through Auritus, we find several unsolved technical problems characteristic not only for earables, but the overall TinyML pipeline itself.

1.3 Organization

The rest of the paper is organized as follows: Section 2 presents background and related work. Section 3 illustrates the data collection setup, statistical tests on participants and activities, and our semi-automated graphical data labeling tool. Section 4 details both the training and development pipeline for HIL activity detection models and filters. Afterwards, Section 5 presents extensive experimental evaluations of the developed models and filters. Section 6 provides three exemplary applications developed through Auritus. Finally, Section 7 provides concluding remarks and future directions.

2 BACKGROUND AND RELATED WORK

There is a serious push towards embedding sensors in commercial earbuds. Apple, Samsung, Bose, and Microsoft have already embedded directional microphone arrays, touch sensors, and accelerometers in their earbuds for speech, motion detection, and gesture sensing [19, 26]. Bragi earbuds are also capable of performing in-ear fitness tracking, gesture recognition, and passive noise reduction [73]. In this paper, we focus on human activity detection and head-pose recognition using earables that enable multiple interesting and diverse applications as listed below:

Health Monitoring: Gil *et al.* [32] proposed an earable device to measure cardiovascular conditions during physical exercise. The authors fuse ECG, impedance, amperometric, and potentiometric measurements with 3DoF inertial measurements to capture the electric potential around the ear. Roddiger *et al.* [80] showed that filtering, interpolation, and principal component analysis can allow earables to measure respiratory rates. Nirjon *et al.* [66] showed that it is also possible to detect heart rate using ear-worn sensors. Recently, electrodes are being used in earables to monitor sleep quality through in-ear EEG measurements [60].

Context Detection: Emotion regulator conversational agents have been ported to earbuds to dynamically adjust conversation style, tone, and volume in response to the wearer's emotional, environmental, social, and activity context. The context is gathered through speech prosody, motion signals, and ambient sound [43].

AR, MR and Tangible Interfaces with 3D Sound: Yang *et al.* [102] illustrated the fusion of acoustic and inertial sensors from earbuds and smartphones to project 3D binaural audio based on wearer location and gazing orientation. Nasser *et al.* [65] presented an AR concept with earables to provide thermal haptic cues near the ear region, which then provides directions and notifications to visually impaired individuals. Taniguchi *et al.* [93] designed an augmented earable hands-free interface to control devices using tongue movements.

Security, Authentication, and Speech Recognition: Head motion signatures and inertial disturbances generated during speech can be used for standalone authentication purposes [19, 98]. This is useful to counteract voice fingerprinting attacks. In addition, the fusion of IMU and the microphone can be potentially harnessed to identify the voice in noisy environments [19].

Dead-Reckoning: Prakash *et al.* [74] exploited the natural filtering of noise/vibrations (associated with inertial sensor data) as inertial signals propagate from lower-body to the ear canals to develop a subject, and pattern-independent step counting reflex model (called STEAR) using dynamic time warp (DTW) algorithm. Ahuja *et al.* [3] showed that it is possible to perform on-board dead-reckoning on earables using magnetometer-based heading estimation and a belief-based step-counting algorithm from an earable IMU. The directional beamforming microphones in earables can also be used to perform ultrasonic echo-localization for generating indoor maps [72].

2.1 Activity Detection and Head-Tracking Using Earables

To better facilitate earable computing applications, it is necessary to realize more accurate activity detection. Researchers [5, 38, 74, 75] have proposed several ML classifiers for activity detection using earables, summarized in Table 1. The location of earables enables the natural filtering of noise and vibrations by the upper body, providing the potential for high accuracy and multi-granular activity detection with earables [19, 74, 75]. In fact, Atallah *et al.* [5] validated the use of a 3DoF earable sensor for gait monitoring by correlating acceleration features with gait parameters from a piezoelectric force-plate instrumented treadmill. Existing techniques have used convolutional neural network (CNN), k-nearest neighbors (kNN), random forests (RF), and k-means clustering for

multiscale activity detection. These include distinguishing between head-mouth (verbal) and full-body mechanics (non-verbal) [38, 51], exercise activity detection [75, 77], facial expression detection [54, 91, 96], and activity-level detectors [66]. However, existing work completely ignores the compute constraints of earables. As shown in Table 1, these algorithms are not deployable on embedded earable platforms due to their model size requiring high RAM usage or having feature extraction overhead. In contrast, AURITUS uses hardware-aware NAS to train ML classifiers targeted towards microcontroller class processors typically used in earable platforms. Our trained models are directly transferred to the earable platforms to benchmark their performance within the computation limits of earables.

With the advent of virtual reality (VR) and AR applications, the innate challenges of head-tracking using inertial sensors have also been explored. Existing works usually perform simple gyroscope integration [102], often fusing data from two earables with tilt/yaw correction and predictive/positional tracking [28] from tertiary head-tracking devices (e.g., Oculus Rift). However, gyroscopes suffer from time-varying bias due to bias instability and angular random walk, leading the integral to drift over time [48]. To enable long-term head motion tracking, AURITUS provides lightweight yet accurate head-tracking filters that can account for sensor errors statistically or heuristically. The filter parameters are optimized using Bayesian optimization (BO).

2.2 Neural Architecture Search

Several NAS frameworks have been proposed for microcontroller-class devices. SpArSe [27] treats NAS as a gradient-driven multi-objective BO problem, treating hardware attributes via proxies and coupling pruning with NAS. MicroNets [8] uses a quantization-aware gradient-driven approach to optimize task-aware DNN backbones. MCUNet [57] tailors Once-for-All (OFA) NAS [12] for microcontrollers, using a two-stage evolutionary NAS to train a single OFA network in an optimized search space for a broad spectrum of target hardware. Adopting MCUNet is a challenge as it uses a custom inference engine and its latency/resource measurements rely on a closed-source software stack. In AURITUS, we perform hardware-aware NAS using multi-objective BO, where the acquisition function is optimized using Monte Carlo sampling.

We adopt BO due to the following reasons: (i) BO provides a state-of-the-art approach to optimize expensive objective functions in a few evaluations [87], (ii) BO allows explicit inclusion of non-gradient-friendly constraints of the model size and accuracy tradeoffs during the training process [27]. The choice of Monte Carlo sampling instead of the gradient-driven approach of SpArSe [27] is based on the fact that neural architecture search space consists of categorical variables where the sampling approach evaluates the acquisition function only at valid configurations only [31, 86]. AURITUS includes the hardware-aware training where the resource utilization of a model is computed at runtime by its real deployment on the target hardware, instead of just using proxies as done

Table 1. Performance metrics of existing activity detectors and head-pose filters for earables

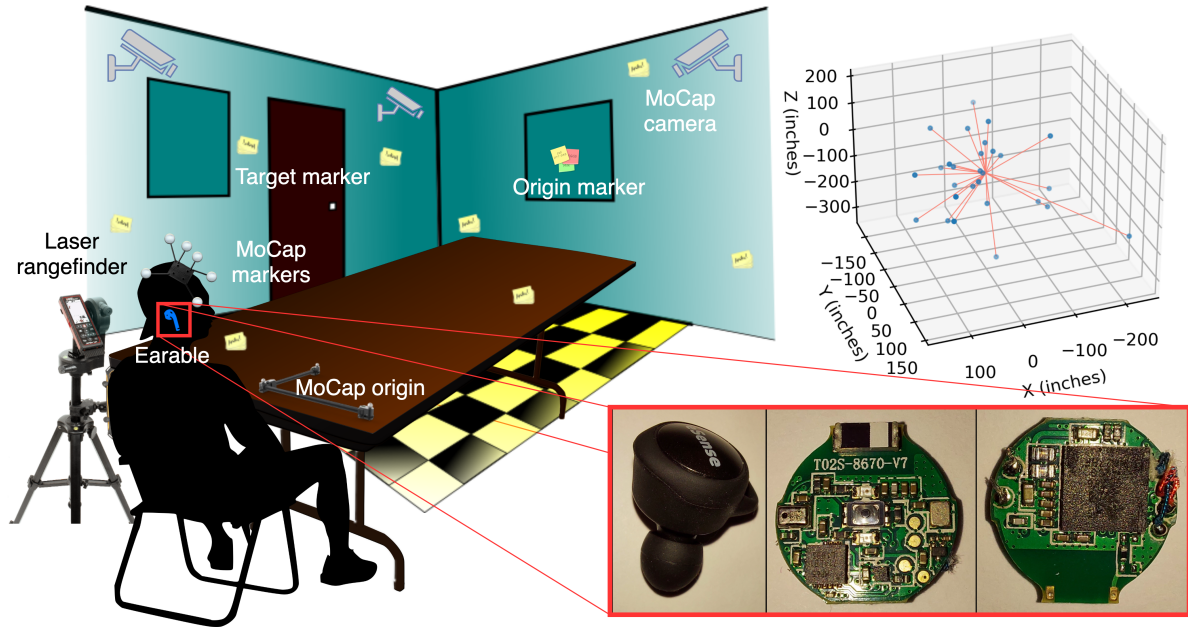| Activity Detectors (Full Body)			
Classifier	*Accuracy* (%)	*Feature Extraction*	*Model Size* (kB)
CNN [38]	88.3	No	9758
kNN [38]	81.2	Yes	381
Random Forest (RF) [75]	92.94	Yes	593
K-means Clustering [66]	96.8	Yes	-
Head-Pose Filters			
Filters		*Error characteristics*	
Ferlini *et al.* [28]		5.4 degrees (short) and 18.7 degrees (long)	
Yang *et al.* [102]		~8 degrees (after 3 rounds of rotations)	

Fig. 2. Experimental setup for head-pose and ground truth data collection, with the positions (in inches) of target markers characterized in Cartesian coordinates w.r.t. origin marker.

by SpArSe [27]. Our evaluation shows that proxies are only approximations of the real hardware constraints, which are noisy for extremely resource-constrained devices. In addition, none of the NAS frameworks can optimize energy and Tensorflow Lite Micro arena size in real-time, as they do not use real-hardware during the NAS.

3 DATA COLLECTION AND LABELING PIPELINE

To address the challenge of open source data scarcity for earable, Auritus provides a pipeline specifically designed to ease data collection, pre-processing, and labeling of earables. In this section, we introduce the experimental testbed used to collect human movement data using earables (Section 3.1) and describe statistical tests on the collected data, participants, and activities (Section 3.2) to verify data quality. To ease the data labeling effort, Auritus also incorporates a graphical data-labeling tool (Section 3.3).

3.1 Data Collection Setup

For data logging, we used the popular eSense[3] earable device from Nokia Bell Labs [45, 46], shown in Fig. 2. We used the built-in Butterworth filter with the cutoff set to 5 Hz to reject high-frequency noise beyond motion parameters. We used BLE advertisement and connection intervals of 45-55 mS and 20-30 mS, respectively. The 6-channel inertial data was broadcasted at ∼100 Hz to an in-house smartphone application we developed using eSense Android middleware backend [45].

---

[3]https://www.esense.io/

Table 2. Summary of executed activities.

	ADL	Description
1	Walking (W)	The participant is asked to walk forward in a straight line, make a turn (clockwise or anti-clockwise) at the end of a corridor and repeat (average speed: 1.5-3.5 mph).
2	Jogging (R)	Similar to walking, but the participant runs slowly at uniform pace instead of normal walking pace (average speed: 3.0-5.0 mph).
3	Jumping (J)	Each participant jumps at a particular spot without translational motion.
4	Standing (St)	The participant stands freely with true-to-life head and limb movements allowed.
5	Turning Left (Tl)	The participant walks forward in a counter-clockwise circle.
6	Turning Right (Tr)	The participant walks forward in a clockwise circle.
7	Sitting (Si)	Each participant is asked to sit on an office chair freely, allowing for natural limb and head movements.
8	Lying (L)	Each participants lay on a sofa in various common sleeping positions.
9	Falling (F)	The participant falls freely on a padded sofa from a standing position.

The experimental testbed for head-pose data collection is also illustrated in Fig. 2. For sub-mm resolution ground truth collection, the participants wore a hat with OptiTrack Prime 17W[4] MoCap infrared markers [29] mounted in a rigid body configuration. The motion data of the head and the marker visual cues were tracked using Motive:Tracker [5] [6] and screen recorder applications, respectively. To synchronize the discrete data management elements, we harmonized the local system clocks to the Network Time Protocol (NTP) [62] and graphically identifiable calibration nods performed before the data collection phase. The testbed contained 27 target markers, one of which is an origin marker. A participant is supposed to move the head to and from an origin marker to target markers. To characterize the position of each marked point in the 3D testbed, we used a Leica Disto X3 laser rangefinder[6] and a digital compass to obtain the distance, azimuth, and elevation angles of the targets from the participant's position. To log activity data, we asked participants to perform nine common ADL [83] after providing them sample cues. Each participant performed calibration nods in between each activity to signify the start and endpoints for each ADL. We ensured that there were no path obstructions or distractions during activity data collection and restarted the process when the earable became loose or any unforeseen circumstances arose.

3.2 Participants and Activities

For the head-pose dataset, we collected 34 distinct head-poses from the 27 targets per participant, obtaining ~ 2.43 million inertial samples in total. We considered two common types of head movements. The simple class included movements of the head from the origin marker to a target marker and back to the origin marker. We collected 411,103 such samples (after preprocessing). The more complex class involved movements of the head from the origin marker to a target marker 1 (phase A), target marker 1 to a target marker 2 (phase B), and target marker 2 to the origin marker (phase C). We collected 1,068,211 samples for the complex head movements (after preprocessing). For the activity dataset, nine classes of actions were recorded, illustrated in Table 2. The length of time for each activity varied for each participant, but on average, each activity IMU trace for each participant was around 23-25 seconds. A total of 958,182 inertial samples were recorded (after preprocessing). We collected data from 45 participants (29M, 16F) in total. The sampling rate at which the accelerometer and gyroscope data was collected was set to 100 Hz. For each participant, we calibrated the gyroscope and the accelerometer three times (once before collecting simple head movements, once before collecting complex head movements, and once before collecting activity recognition data) to remove gyroscope static bias drift and set up the accelerometer gain factors and accelerometer biases using static calibration techniques described in [20] and [100]. For gyroscope

---

[4]https://optitrack.com/cameras/prime-17w/
[5]https://optitrack.com/software/motive/
[6]https://shop.leica-geosystems.com/buy/package/x3

calibration, we placed the earable on a flat table in a static position, sampled the gyroscope for 10 seconds, and averaged the readings to get the static bias value which we subtracted from subsequent gyroscope readings. For accelerometer calibration, we placed the earable in six different stationary tilt angles and used the iterative method in [100] to estimate the gain factors, G_i and biases, B_i for each axis. The true acceleration in each axis is then given by: $A_i = (S_i - B_i)/G_i$, where S_i is the raw accelerometer readings for each axis.

Table 3. Results of Kolmogorov-Smirnov test on dataset participant statistics.

Parameter	DF	Statistic	P > D	Inference
Age (yrs)	45	0.1271	0.37217	Can't reject normality at 0.05 significance level
Height (m)	45	0.13309	0.30585	Can't reject normality at 0.05 significance level
Weight (kg)	45	0.14081	0.43016	Can't reject normality at 0.05 significance level
Ear height from origin (inches)	45	0.11569	0.55641	Can't reject normality at 0.05 significance level

To ensure sufficient statistical diversity in physiological parameters without bias, the Kolmogorov-Smirnov test [40] was performed on age, weight, height, and ear height from the origin. Table 3 summarizes the normality test results. No bias was observed in age, height, weight, and ear height from the origin, and at the 0.05 level, all parameters were significantly drawn from a normally distributed population.

Fig. 3 illustrates the heatmap of the average dynamic time warping (DTW) distances among the motion traces of the same class versus different classes. The DTW distance provides a measure for the similarity between two temporal sequences with different speeds [9]. Intuitively, inertial traces belonging to the same class should have a small DTW distance, and signals belonging to different classes should have a large DTW distance. We applied Kruskal-Wallis ANOVA on the average DTW distance across 10 random accelerometer vector sum snapshots ($\sqrt{A_x^2 + A_y^2 + A_z^2}$) from different participants, with each snapshot being 400 samples in length. The results are illustrated in Table 4. At the 0.05 level, the distributions pertaining to the DTW distances from the same class and different classes are significantly different, indicating the presence of well-separated clusters for each ADL in low-dimensional latent space learnable by ML algorithms. The same test was applied to selected simple head movements (varying sample count) from different participants belonging to eight random target markers. The
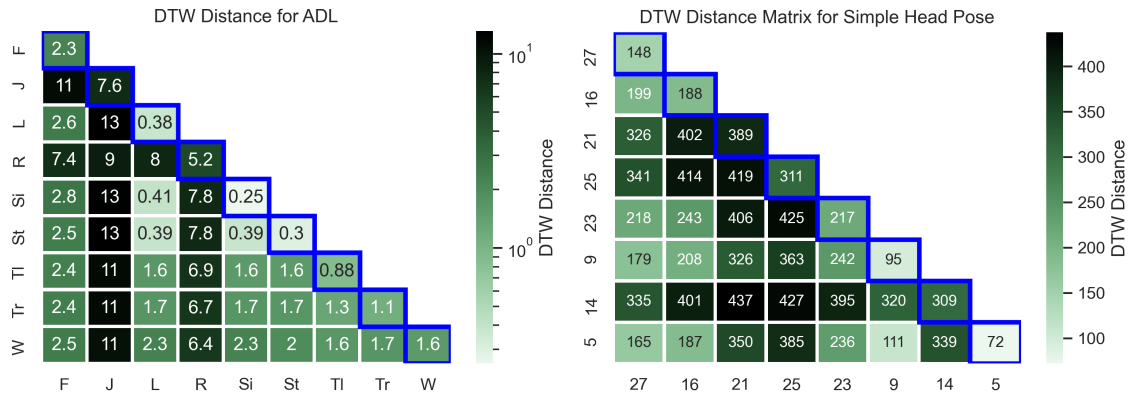

Fig. 3. Heatmap of average DTW distance across all activity classes and selected simple head movements. The distance should be small among inertial traces of same class (marked with blue bounding box) and large for different classes.

Table 4. Results of Kruskal-Wallis ANOVA between DTW distances of human motion from same class and different classes.

Motion Type	χ^2	$P > \chi^2$	Inference
Activity	6.96337	0.00832	At the 0.05 level, the populations are significantly different
Head-Pose	5.56564	0.01832	At the 0.05 level, the populations are significantly different

inertial traces consisted of gyroscope sum snapshots ($\omega_x + \omega_y + \omega_z$), and the same statistical inference was observed for head movements.

3.3 Graphical Data Labeling Tool

To ease labeling time-series data collected in continuous chunks, we designed a graphical inertial data labeling tool to allow head-pose and activity data annotation using a graphical-user-interface (GUI). Using the aid of the ground truth videos from Motive:Tracker, the application developer selects points directly on the plot signifying the start and endpoints of calibration nods and head movements. A single head rotation on the gyroscope-time plot essentially consists of a triangular/bell-curved shape peak, with the rate of change of the angular velocity proportional to the head motion velocity (faster = thinner and taller peak, slower = thicker and smaller peak). After specifying all the endpoints and making any numerical adjustments to the data, the developer exports the endpoints to the GUI workspace and runs a script to perform automatic segmentation and labeling based on the endpoints. Three such scripts are provided, one each for activity, simple head-pose, and complex head-pose. The developer only needs to input the volunteer number and labels to the scripts.

4 DEVELOPMENT OF MODELS AND FILTERS

Although ML classifiers for activity detection are well explored in the domain of smartphones and smart-watches [37, 81, 99], their direct transfer to the resource-constrained domain of earable platforms is not feasible. Auritus incorporates the ML model training in combination with completely automated hardware-aware NAS. The NAS is designed to ensure classifier inference is directly possible on edge in real-time within the available flash and RAM constraints of the target device. Given different inputs and different requirements from the application developer, the optimization workflow produce different model implementations automatically. The system can either select to optimize a specific model or give a model based on developer requirements. Further, to provide superior head-pose tracking, Auritus includes a set of filters with different computation complexity. We provide the models and filters in Python and C for real-time application on embedded hardware via Mbed real-time operating system (RTOS) and TensorFlow Lite Micro (TFLM) backend. In this section, we outline the pipeline for training and developing activity classifiers from the model and filter zoo on real hardware using Bayesian HIL NAS and lightweight model architectures (Section 4.1). Next, we delineate the activity classifier implementation details by discussing feature extraction, windowing, dataset splits, design space, and hardware/software details (Section 4.2). Last, we discuss the generation of head-pose estimation filters (Section 4.3)

4.1 Hardware-Aware Lightweight Model Generation

The memory and compute capability of TinyML devices are significantly smaller than cloud or even mobile devices. For example, an Arduino BLE33 has only 320 KB of SRAM and 1 MB of flash, compared to 4 GB of RAM and 64 GB of storage on a smartphone. A GPU can have 16 GB of memory on a workstation with secondary storage in the order of terabytes. Thus. optimizing larger models for smaller devices directly using techniques such as dimension reduction, pruning, quantization, and model compression alone are insufficient to mitigate the loss of accuracy [12, 57]. Moreover, the type of ML operators supported by such devices is limited by the processor architecture and the runtime interpreter. For example, vanilla recurrent neural network (RNN) operators are not

widely supported by off-the-shelf TinyML software frameworks [8]. As a result, the design goals of models and the ML operator space should be optimized through the integration of novel lightweight model design paradigms and target hardware specifications in order to strike an equilibrium between accuracy and efficiency [27, 49, 57].

4.1.1 Hardware-Aware Bayesian NAS. To find the ideal activity detection model candidate from a backbone deep neural network (DNN) search space for limited flash, RAM, and latency requirements, we model the search as a parallelizable black-box BO problem. The search space Ω consists of neural network weights w, hyperparameters θ, network structure denoted as a directed acyclic graph (DAG) g with edges E and vertices V representing activation maps, and common ML operations v (e.g., convolution, batch normalization, pooling, etc.) which act on V. The goal is to find a DNN that maximizes the hardware SRAM and flash usage within the device capabilities while minimizing latency and classification error on the validation set.

$$f_{\text{opt}} = \lambda_1 f_{\text{error}}(\Omega) + \lambda_2 f_{\text{flash}}(\Omega) + \lambda_3 f_{\text{SRAM}}(\Omega) + \lambda_4 f_{\text{latency}}(\Omega) \tag{1}$$

where

$$f_{\text{error}}(\Omega) = \mathcal{L}_{\text{validation}}(\Omega), \Omega = \{\{V, E\}, w, \theta, v\} \tag{2}$$

$$f_{\text{flash}}(\Omega) = \begin{cases} -\frac{||h_{\text{FB}}(w,\{V,E\})||_0}{\text{flash}_{\text{max}}} \vee -\frac{\text{HIL information}}{\text{flash}_{\text{max}}} \\ \infty, f_{\text{flash}}(\Omega) > \text{flash}_{\text{max}} \end{cases} \tag{3}$$

$$f_{\text{SRAM}}(\Omega) = \begin{cases} -\frac{\max_{l\in[1,L]}\{||x_l||_0+||a_l||_0\}}{\text{SRAM}_{\text{max}}} \vee -\frac{\text{HIL information}}{\text{SRAM}_{\text{max}}} \\ \infty, f_{\text{SRAM}}(\Omega) > \text{SRAM}_{\text{max}} \end{cases} \tag{4}$$

$$f_{\text{latency}}(\Omega) = \frac{\text{FLOPS}}{\text{FLOPS}_{\text{target FLOPS}}} \vee \frac{\text{HIL information}}{\text{Latency}_{\text{target latency}}} \tag{5}$$

$$a = w \vee y, \qquad y = \sum_{k=1}^{K} v_k g_k(x, w_k)$$

The objective function f_{opt} can be thought of as seeking a pareto-optimal configuration of parameters Ω^* under competing objectives [27], such that:

$$f_k(\Omega^*) <= f_k(\Omega) \ \ \forall k, \Omega \ \ \wedge \exists j : f_j(\Omega^*) < f_j(\Omega) \ \ \forall \Omega \neq \Omega^* \tag{6}$$

First, validation accuracy serves as a proxy for the error characteristics $f_{\text{error}}(\Omega)$ of the model. Secondly, the size of the serialized flatbuffer model schema $h_{\text{FB}}(\cdot)$ [21] generated by TFL acts as a proxy for flash usage when real-hardware is absent. Thirdly, off-the-shelf tools such as TFLM store network weights, quantization parameters, and network graphs on flash. These tools use a predefined portion of the SRAM called the arena to store intermediate activation maps and tensors, persistent buffer, and TFLM runtime interpreter parameters. We use this standard RAM usage model as a proxy for SRAM usage $f_{\text{SRAM}}(\Omega)$ [27]. Lastly, since model latency is linearly proportional to the OPS count for a variety of convolutional models for TinyML devices, we use FLOPS or OPS as a proxy for runtime latency [8]. When real hardware is available, we obtain the SRAM, flash, and latency parameters directly via the serial interface from the target compiler and RTOS, illustrated in Fig. 4. We normalize all the hardware parameters by device capacity or target metrics.

We use Gaussian process $\mathcal{GP}$ as the surrogate model to approximate f_{opt}, which allows priors on the distribution of moments to propagate forward as the search progresses. In addition, the domain of random scalarizations λ can be specified by the developer to guide the parallel search acquisition functions (hallucination or K-means clustering) into the promising Pareto-optimal regions of the gradient plane. The acquisition function decides the next set of Ω_n to sample from the design space using Bayesian Upper-Confidence Bounds (UCB), which balances exploration and exploitation [86]. Apart from speeding up the NAS, parallel search ensures that NAS is not being
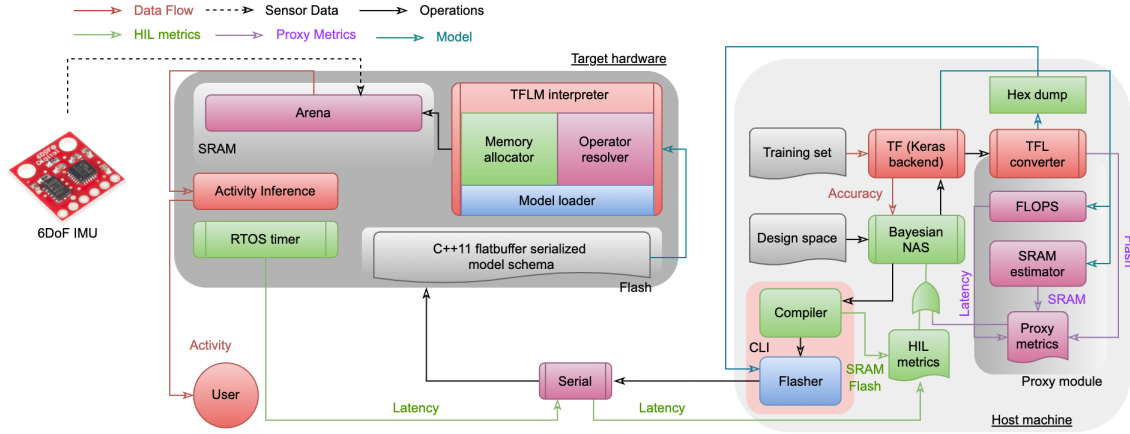
Fig. 4. HIL model optimization for earable activity detection using Bayesian NAS. The system supports both the use of proxy and real hardware to get hardware constraint estimates.

performed on network morphs early on (exploitation) and information gain is maximized in the search process (exploration), yielding a stage-wise "coarse-to-fine" search space.

$$\hat{f}(\Omega) \sim \mathcal{GP}(\mu(\Omega), k(\Omega, \Omega')) \tag{7}$$

$$\Omega_t = \arg\max_{\Omega}(\mu_{t-1}(\Omega) + \beta^{0.5}\sigma_{t-1}(\Omega)) \tag{8}$$

Note that while minimizing the latency and classification error within the hardware SRAM and flash bounds of the device should generate classifiers that perform reasonably well in theory, we observed that without $f_{\text{flash}}(\Omega)$ and $f_{\text{SRAM}}(\Omega)$ in f_{opt}, the NAS program generates models that do not fully exploit the device capabilities and produces small models that may be $2 - 5\%$ less accurate than larger models. $f_{\text{flash}}(\Omega)$ and $f_{\text{SRAM}}(\Omega)$ act as regularizers in f_{opt}, penalizing the NAS program for picking small models, while also promoting the generation of a fine-grained surrogate model. Note that all SOTA NAS frameworks for microcontrollers [8][27][56][57] use a formulation similar to f_{opt}.

4.1.2 Conventional Activity Classifiers. We included five conventional ML activity classifiers from literature, namely bagged trees [83], AdaBoost [52], coarse decision tree (DT) [2], support vector machine (SVM) [4][83], and multilayer perceptron (MLP) [82] in the model zoo to compare against lightweight activity classifiers.

- **Bagged Trees**: Bootstrap aggregation combines several decision trees trained on bootstrap samples to form an ensemble classifier, using majority voting to provide the final label [83].
- **AdaBoost**: Combines weak decision stumps to form an ensemble classifier in a weighted form depending on misclassified points [52].
- **Coarse DT**: Graph of decisions where each node makes binary decisions based on values of the input activation and predefined rules, optimized through splitting and pruning [2].
- **SVM**: An SVM finds a linear decision hyperplane in the feature space whose margin is maximum from the support vectors (cleanly split examples) of two classes, using kernels to project data into a linearly separable manifold [4][83].
- **MLP**: A 2-layer fully-connected feedforward neural network with sigmoid hidden neurons, trained using scaled conjugate gradient backpropagation with cross entropy loss [82].

4.1.3 Lightweight Activity Classifiers. To enable real-time activity classification on resource-constrained devices through our HIL model optimization tool, we designed and included several lightweight classifiers suitable for onboard activity inference in the model and filter zoo. We implemented temporal convolutional network (TCN) [53, 94], fast gated RNN [50], fast RNN [50], Bonsai [49] and ProtoNN [35] models for lightweight yet accurate activity detection. These models use several design techniques to reduce model size and latency while maintaining performance on-par with conventional ML and deep learning (DL) algorithms for time-series processing:

- **Temporal Convolution**: Without explosion of parameter, memory footprint, layer count, or overfitting, TCN kernels allow the network to discover the global context in long inertial sequences while maintaining input resolution and coverage. In TCN, the convolution operation has three desirable properties:
  - *Causality*: The output of the operator at the current timestep t depends only on the current and past inputs but not future inputs. This ensures temporal ordering of the input sequences without requiring recurrent connections. The ordering is maintained via weight sharing among the input chunks.
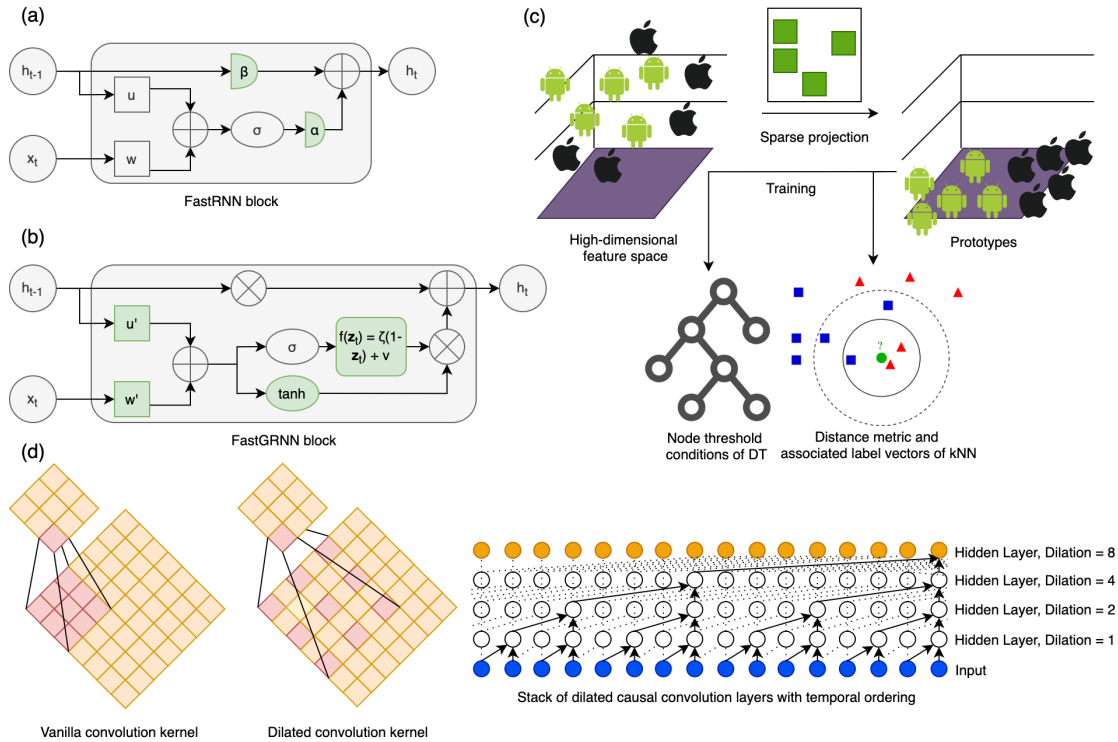

Fig. 5. Illustration of lightweight model architectures geared towards TinyML devices. (a) The addition of a residual connection with two scalars (α, β) stabilizes vanilla RNN training while taking advantage of the relative lightweightness of vanilla RNN against gated RNN. (b) Converting the residual connection to a gate while enforcing $\mathbf{U}$ and $\mathbf{W}$ to be LSQ yields lightweight yet accurate gated RNN. (c) Sparsely projecting input features to a low-dimensional space allows DT and kNN to be computationally efficient. (d) Enforcing causal convolution and dilated kernels allows spatial and temporal feature extraction in long time-series sequences without requiring recurrent connections or significant compute.

– *Dilated Convolution*: The receptive field F_i of each unit in the ith layer in a TCN dilated causal kernel of size $k \times k$ with dilation factor l is given by:

$$F_{i,\text{TCN}} = F_{i-1} + (k_l - 1) \times l, F_0 = 1 \tag{9}$$

$F_{i,\text{TCN}}$ is larger than $F_{i,\text{CNN}}$, which is $i \times (k - 1) + k$. When dilated CNN are stacked on top of each other, the dilation factor increases exponentially, increasing model capacity and receptive field size with fewer layers and parameter count over vanilla CNN or RNN

– *Residual Blocks:* Two stacks of dilated causal convolution layers, f and g, are fused through gated residual blocks $\mathbf{z}$ for expressive yet bounded non-linearity, complex interactions and temporal correlation modeling in the input sequence:

$$\mathbf{z} = \tanh(\mathbf{W}_{f,k} * \mathbf{x}) \odot \sigma(\mathbf{W}_{g,k} * \mathbf{x}) \tag{10}$$

where $\mathbf{W}$ are the weights in each layer, σ is the sigmoid function and $\mathbf{x}$ is the input.

• **Stabilized RNN with LSQ Matrices**: Vanilla RNN, albeit lightweight, suffer from exploding and vanishing gradient problem (EVGP) for long temporal sequences. Existing solutions to EVGP (e.g., gated RNN (long short-term memory (LSTM) and unitary RNN) come at the cost of either accuracy loss or increased memory and latency overhead. Fast RNN [50] solves EVGP by adding a weighted residual connection with two scalars (α, β) to generate well-conditioned gradients:

$$\tilde{\mathbf{h}}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}), \quad \mathbf{h}_t = \alpha\tilde{\mathbf{h}}_t + \beta\mathbf{h}_{t-1} \tag{11}$$

where $0 \leq \alpha \ll 1, \beta \approx 1 - \alpha, \beta \leq 1$, σ is a non-linear activation function, $\mathbf{W}$ and $\mathbf{U}$ are RNN matrices, $\mathbf{b}$ is bias vector, $\mathbf{h}$ is the hidden state and $\mathbf{x}$ is the input. By varying α and β, we can control the update extent of $\mathbf{h}_t$ based on $\mathbf{x}_t$. Fast GRNN [50] then converts this residual connection to a gate while enforcing $\mathbf{W}$ and $\mathbf{U}$ to be low-rank, sparse and quantized (LSQ):

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}'\mathbf{x}_t + \mathbf{U}'\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{12}$$

$$\mathbf{h}_t = (\zeta(\mathbf{1} - \mathbf{z}_t) + v) \odot \tilde{\mathbf{h}}_t + \mathbf{z}_t \odot \mathbf{h}_{t-1}, \quad \mathbf{z}_t = \sigma(\mathbf{W}'\mathbf{x}_t + \mathbf{U}'\mathbf{h}_{t-1} + \mathbf{b}_z) \tag{13}$$

$$\mathbf{W}' = \mathbf{W}^1(\mathbf{W}^2)^\top, \; \mathbf{U}' = \mathbf{U}^1(\mathbf{U}^2)^\top \tag{14}$$

where, $\zeta \geq 0, v \leq 1$. Fast GRNN, thus, is able to provide the capabilities of gated RNN without the associated compute overhead.

• **Sparse Low-Dimensional Projection**: Bonsai [49] is a shallow and sparse DT with non-linear activations, making inferences on data projected in low-dimensional space called prototypes. Similarly, ProtoNN [35] is a lightweight k-nearest neighbor (kNN) classifier designed to operate on prototypes. The sparse projection matrix is learned using stochastic gradient descent and iterative hard thresholding. Sparsely projecting high-dimensional feature space onto a low-dimensional linear manifold reduces parameter count for Bonsai and ProtoNN, allowing them to be computationally efficient.

4.2 Activity Classifier Implementation Specifics

In this sub-section, we provide details on the implementation of the activity classification training pipeline, including feature extraction, windowing, dataset splits, design space optimization, and specifications of target hardware and host machine for training.

Table 5. Features extracted for Bonsai, ProtoNN, and the five conventional activity classifiers. Only the shaded features were used for Bonsai and ProtoNN.

Dominant Sign	Entropy	Integration	Interquartile Range
Kurtosis	Mean Abs. Dev.	Maximum	Minimum
Mean	Avg. Mov. Mean	Avg. Mov. Med.	Avg. Mov. Max.
Avg. Mov. Min.	Avg. Mov. SD	Avg. Mov. Var.	Avg. Mov. MAD
Autocorrelation	Avg. Vec. Norm	Avg. Z Score	Median
Norm	Pearson CC	Range	Skewness
Slope Sign Change	Signal Mag. Area	Standard Deviation	Variance
Variation	Zero Crossing	Time Window	

4.2.1 Feature Computation and Windowing. For activity classification using conventional ML algorithms, 241 spatial features were extracted from our dataset with varying sliding window sizes (1, 3, 5, and 10 seconds) and stride of 0.5 seconds, shown in Table 5. Each feature (except the time window) was applied separately to 3 accelerometer and 3 gyroscope channels (180 features). Each feature (except the time window) was also applied to the vector sum of accelerometer and gyroscope channels (60 features). We included time window as a feature to account for sampling rate jitter and missing data in the dataset [84]. For Bonsai and ProtoNN, we apply five lightweight features from the 241 features on the accelerometer and gyroscope vector sums, namely maxima, minima, range, variance, and standard deviation, totaling 10 features. A sliding window of varying size (1,2,3 and 5 seconds) with a stride length of 0.5 seconds were chosen for Bonsai, ProtoNN, Fast RNN, Fast GRNN, and TCN. We do not extract any features for Fast RNN, Fast GRNN, and TCN and feed raw windowed inertial samples to the three classifiers. For all classifiers, no normalization or standardization was applied to the raw data.

4.2.2 Hardware and Software Specifications. All models were trained on a host machine with 256 GB RAM, 2× 24 GB Nvidia GeForce RTX 3090, and 3.7 GHz AMD Ryzen Threadripper 3970X 32-core CPU. For benchmarking HIL NAS, we use three real ARM Cortex-M target boards and two virtual hardware models (proxies) with varying resource constraints. The processors run Mbed RTOS and TFLM interpreter on-board. To communicate with the target hardware via system commands from the host machine, we used the Mbed command-line interface (CLI). The target hardware specifications are outlined in Table 6:

Table 6. List of hardware evaluated for NAS.

Hardware	SRAM (kB)	Flash (kB)	Proxy/HIL
Qualcomm CSR8670 (eSense platform)	128	16000	Proxy
STM32F446RE	128	512	HIL, Proxy
STM32F407VET6	192	512	Proxy
STM32L476RG	128	1024	HIL, Proxy
STM32F746ZG	320	1024	HIL, Proxy

All the conventional ML models were implemented in MATLAB and later converted to C optimized for Cortex-M processors. All lightweight models were implemented in Jupyter notebook (Python), using Keras and Microsoft EdgeML via a Tensorflow and TFLM [21] backend. The TCN, ProtoNN, and Bonsai models were converted to flatbuffer model schema [21] using TFL and the other models were converted to C-compatible formats for further benchmarking. All models used cross-entropy loss except Bonsai, which used multi-class hinge loss.

4.2.3 Dataset Splits. We split the dataset in three different ways for our evaluation:

- **Split with no unseen participants:** In this split, there are no unseen participants, i.e. data from all participants are present in the training set. This was used to report **test accuracy**. We used holdout validation (train: validation: test) ratios of 80:10:10 and 70:15:15 for classical models and MLP, respectively. For the lightweight models, we used holdout validation of 80:10:10 for the TCN and 90:0:10 for Bonsai, ProtoNN, FastRNN, and FastGRNN. Except for MLP, we ensured that the dataset splits are the same for all classifiers for a fair comparison.[7]
- **Split with leave-1 out:** The data is split per user, such that the models are trained on data from 44 participants, and tested on data from a participant not present in the training set. This was used to report **leave-1 out test accuracy**. We performed a 10-way cross-validation, choosing a random participant each time to be left out of the training set while the model is trained on other 44 participants. We then average the leave-1 out accuracy of the 10 models. The train: validation ratio was 90:10 for the data from the 44 participants.
- **Split with leave-n out:** The data is again split per user, however, the number of participants left out now varies. This was used to perform **leave-n out cross-validation studies**. The train: validation ratio was 90:10 for the data from the participants present in the training set.

4.2.4 Design Space Optimization. For the conventional classifiers, we used a variation of the BO pipeline we showed earlier for hyperparameter tuning. We used 80 iterations for the candidate models in the search space with the expected improvement per epoch as the acquisition function to select the most optimal hyperparameter for each model. We did not include hardware constraints in f_{opt} for conventional ML models but aimed to maximize the test accuracy of the conventional activity detection models. For the five lightweight classifiers, we used 50 iterations for the candidate models in the search space. We incorporate hardware constraints only for the TCN in f_{opt}, as we observed Bonsai, ProtoNN, Fast RNN, and Fast GRNN to be resource-efficient without requiring explicit hardware-aware optimization by design. Table 7 lists the architectural search space for all 10 models in the model and filter zoo, as well as support for HIL optimization. We fixed some of the parameters of each model to default or well-known values and excluded them from the search space.

4.3 Head-pose Filters

For real-time head-pose estimation, we fed the raw, unprocessed head-pose streams to analytical orientation estimation algorithms. We include four filters in the model and filter zoo for head tracking. To optimize the filter parameters for root-mean-squared error (RMSE) minimization, we use BO.

- **Complementary Filter**: The complementary filter [48] acts as a low-pass filter for accelerometers to mitigate high-frequency Gaussian noise, and a high-pass filter for gyroscopes to counteract time-varying drift, thereby amplifying the strengths and attenuating the weakness of each sensor in the IMU for attitude estimation. The filter simply integrates the gyroscope readings to get the 3D attitude from the gyroscope and takes linear accelerometer components in the appropriate direction to get the 2D attitude. The only tunable parameter in the filter is β, which weighs the contribution of accelerometer and gyroscope attitude. While the filter is simple and lightweight, it does not account for statistical treatment of drift and noise, leading to quick orientation drift. The filter also suffers from gimbal lock due to operation in the Euler domain and hence does not perform optimally for fast movements.
- **Mahony Filter**: The Mahony filter [24] solves the gimbal lock problem by operating in the quaternion domain. It also reduces the attitude drift caused by gyroscope bias by adjusting gyroscope error using

---

[7]Bonsai, ProtoNN, FastRNN, and FastGRNN do not have any validation step, so we transferred the validation data to the training set.

Table 7. Design space for all 10 models in the model and filter zoo. The classifiers marked (F) require feature extraction overhead.

Candidate Model	Design Space (Ω)	HIL	Other Parameters (Fixed)
Bagged Trees (F)	Number of learners: 10-500 Maximum number of splits: 1-23401 Number of predictors to sample: 1-241	✗	✗
AdaBoost (F)	Number of learners: 10-500 Maximum number of splits: 1-23401 Learning rate: 0.001-1	✗	✗
Coarse DT (F)	Maximum number of splits: 1-23401 Split criterion: {Gini, Twoing, Deviance Reduction}	✗	✗
SVM (F)	Kernel: {Quadratic, Cubic, Linear} Multiclass method: One-vs-All, One-vs-One Box constraint level: 0.001-1000 Standardize data: true, false	✗	✗
MLP (F)	Number of hidden units: {15, 20, 50, 100}	✗	Number of layers: 2
TCN	Number of filters: 2-64 Kernel size: 2-16 Use residual: true, false Number of layers: 3-8 Dilation factors: [1,2,4,8,16,32,64,128,256]	✓	Number of stacks: 1 Dropout: 0.0 Activation: ReLU Normalization (weight, batch, layer): False Learning Rate: 0.001 (Adam)
Fast GRNN Fast RNN	Hidden Units: 20-60	✗	Learning Rate: 0.01 Decay Step and Rate: 200, 0.1 Sparsity (U, W): (1.0, 1.0) Nonlinearity (update, gate): (tanh, sigmoid) Rank (U, W): (Full, Full)
Bonsai (F)	Sigmoid Parameter: 1-4 Depth: 1-6 Projection dimension: 10-70	✗	Regularization (Z, W, V, T): (0.0001, 0.001, 0.001, 0.001) Sparsity (Z, W, V, T): (0.2,0.3,0.3,0.62) Learning rate: 0.01
ProtoNN (F)	Projection dimension: 10-70 Number of prototypes: 10-70 γ: 0.0015-0.05	✗	Regularization (W, B, Z): (0.000005, 0, 0.00005) Sparsity (W, B, Z): (0.8,1.0,1.0) Learning rate: 0.03

accelerometer readings through proportional-integral compensation without significant markup in computation time. The two tunable parameters are K_p and K_I. K_I mitigates the steady-state error in orientation estimation, while K_p reduces the rise time to the actual orientation estimate produced from accelerometer readings.

- **Madgwick Filter**: The Madgwick filter [59] improves attitude estimation error upon the Mahony filter by incorporating accelerometer attitude increment in the orientation estimation formula. The filter performs one-step gradient descent to get the optimal attitude increment from accelerometer readings. The filter is computationally inexpensive (109 scalar operations), works well for low-sampling rate IMU, and includes pre-calibration steps. The only tunable parameter in the filter is β, which serves the same purpose as β in the complementary filter.

- **Indirect Extended Kalman Filter** (IEKF): The KF is an iterative optimal state estimation algorithm (from fusing consecutive samples of single or multiple noisy indirect modalities) under Gaussian variations [41]. It is composed of prediction (process or transition or time update) and correction (measurement update). KF is a subset of Bayes filter with Gaussian prior, linear process and measurement model with Gaussian noise and satisfying Markov property, with the goal of maximizing posterior probability EKF can deal with

globally non-linear system dynamics via Taylor series and Jacobians. It linearizes the non-linear process model locally about the running state mean [10]. Instead of modeling the attitude directly, IEKF models the error in attitude estimate. We use the gravity estimation from the gyroscope and accelerometer orientation as the error model, and update the actual attitude by multiplying the errors with the head pose. While IEKF yields the most accurate head-pose estimate, it is the most computationally expensive among the four filters.

5 ALGORITHMIC EVALUATION, COMPARISON AND DISCUSSION

In this section, we illustrate the experimental results related to the performance of our hardware-aware optimization framework, trained models, and filter zoo on our dataset. We also compare proposed models and filters with the SOTA in earable activity detection and head pose estimation. For our proposed activity detection models, we carried out activity detection on all the 9 activities reported in Table 2.

5.1 Activity Detection Model Size and Accuracy

Table 8 showcases the best performance of conventional ML activity classifiers in terms of test accuracy (no unseen participants), average leave 1-out test accuracy, and model size on the entire dataset. The hyperparameters stated were the most optimal found by BO. From Table 8, we can see that the test accuracy of classifiers ranges from 98.5-100%, while the leave-1 out test accuracy ranges from 81.3-91%. Even though bagged trees had the highest test accuracy among all classifiers, SVM generalized the best overall on unseen participants. However, the SVM model was also 4× larger than the bagged ensemble model. MLP had the lowest model size of 418 kB among all conventional classifiers, while AdaBoost was 195× larger (largest model among all) but ∼ 1% less accurate than MLP in terms of leave-1 out test accuracy.

Table 8. Best performance of conventional ML activity classifiers on our dataset. The optimal window size was 10 seconds for all classifiers.

Classifier	Optimal Hyperparameters	Test Accuracy (%)	Leave 1-out Test Accuracy (%)	Model Size (kB)
DT ensemble	Learners: 237, Splits: 23019, Mode: Bagging	**100**	90.0±8.5	5700
	Learners: 344, Splits: 715, Learning Rate: 0.44 Mode: AdaBoost	98.7	86.7±9.5	81600
Coarse DT	Splits: 736, Criterion: Dev. Red., S.gate Dec. Splits: All	98.5	81.3±11.4	1100
SVM (1-1)	Kernel: Cubic, Penalty Level: 26.5, Normalization: Yes	99.9	**91.0±5.4**	22500
MLP	Hidden Layer: 2, Hidden Unit: 50	99.8	87.7±8.2	**418**
	Hidden Layer: 2, Hidden Unit: 100	99.5	86.7±8.6	

Table 9 illustrates the accuracy (test accuracy (no unseen participants) and average leave-1 out test accuracy), RAM usage, flash usage, FLOPS, and energy consumption of lightweight classifiers on the entire dataset. Excluding Bonsai and ProtoNN, none of the classifiers require feature extraction. For the TCN, we showcase five models targeted towards five different hardware classes (specified in parenthesis in Table 9), optimized via our HIL Bayesian NAS pipeline. To showcase energy usage for lightweight classifiers, we ran the industry-standard EEMBC EnergyRunner benchmark [7] for TCN, FastGRNN, and FastRNN running on ARM Cortex M4 processors, while using a widely used power monitor[8] to log power usage of Bonsai and ProtoNN running on ARM Cortex-A processors. Our HIL NAS adapts the TCN model to achieve better accuracy with an increase in computing resources. The highest test accuracy of 98.3% was obtained by FastRNN, which also had the smallest model size of 6.04 kB among all models. However, FastGRNN achieved the highest leave-1 out accuracy of 91%, requiring only 7.08 kB more flash than FastRNN while being 4.3% more accurate. Observe that the largest model in Table 9 is 5.6× smaller than the smallest model in Table 8. Furthermore, FastGRNN and SVM provide the same leave-1 out

---

[8]https://www.msoon.com/lvpm-software-download

Table 9. Best performance of lightweight ML activity classifiers on our dataset. The optimal window size was 5 seconds for all classifiers. Note that the results shown for TCN are using proxies.

Classifier	Optimal Hyperparameters	RAM (kB)	Flash (kB)	FLOPS (M)	Test Accuracy (%)	Leave 1-out Test Accuracy (%)	Energy (mW)
TCN	(*eSense*) Filters: 15, Kernel Size: 2, Dilations: [1, 2, 4, 8, 32, 128, 256], Skip Connections: No	39.3	52.8	7.52	94.6	80.0±9.4	-
	(*STM32F407VET6*) Filters: 17, Kernel Size: 3, Dilations: [2, 4, 32, 128, 256], Skip Connections: No	47.6	54.6	10.3	94.0	83.0±10.3	-
	(*STM32F446RE*) Filters: 18, Kernel Size: 2, Dilations: [2, 4, 8, 16, 32, 64, 128, 256], Skip Connections: Yes	55.4	73.4	12.3	95.3	83.2±9.7	116$^{\wedge}$
	(*STM32L476RG*) Filters: 13, Kernel Size: 7, Dilations: [1, 4, 16, 32], Skip Connections: No	49.9	53.3	10.1	95.0	82.0±14.4	50$^{\wedge}$
	(*STM32F746ZG*) Filters: 21, Kernel Size: 2, Dilations: [2, 8, 16, 64, 128, 256], Skip Connections: Yes	55.6	66.4	10.1	96.7	79.0±9.9	418$^{\wedge}$
Fast GRNN	Hidden Unit: 50	∼ 2	13.12	-	97.6	**91.0±5.0**	**41**-133[#]
Fast RNN	Hidden Unit: 32	∼ 2	**6.04**	-	**98.3**	86.7±3.10	**41**-133[#]
Bonsai	Depth: 3, Sigmoid Parameter: 1.0, Projection Dimension: 22	∼ 2	14.8	0.0136	80.3	78.7±5.9	250$^{\vee}$
ProtoNN	Projection Dimension: 70, γ: 0.004, Prototypes: 70	∼ 2	24.9	0.0174	74.0	66.7±8.4	275$^{\vee}$

$^{\wedge}$ EEMBC EnergyRunner™ benchmark [7], RTOS: Mbed, Interpreter: TFLM
[#] EEMBC EnergyRunner™ benchmark, RTOS: Arduino, Compiler: SeeDot [34], Hardware: STM32L476RG and STM32F446RE
$^{\vee}$ Monsoon Power Monitor, OS: Raspbian, Interpreter: TFL, Hardware: Broadcom BCM2711

test accuracy, but the former is 1700× smaller than the latter without requiring any feature extraction overhead. FastGRNN also has the lowest energy usage of 41 mW among all lightweight classifiers, which is 9mW less than the industry standard recommended power consumption[9] for TinyML classifiers [7] We can make several high-level inferences from Table 8 and Table 9:

- The relationship between model accuracy and model size is non-linear, i.e., models with more parameters necessarily do not yield higher accuracies. With appropriate architectural encodings, it is possible to achieve better accuracy with smaller models. Han *et al.* [36] showed that only a small number of weights/parameters contribute to the model performance. Thus, we can further reduce the model size shown in Table 8 without losing accuracy significantly. The improvement is reflected through intelligent and lightweight model architectural formulations shown in Table 9.
- Lightweight classifiers are less robust to domain shifts than conventional classifiers, as evident from the leave 1-out test accuracies. This is because the lightweight classifiers do not have enough redundant weights or parameters to model globally significant attributes that may be common across all users, but instead overfit on the participant-specific characteristics in the temporal sequences, sacrificing generalizability over accuracy.
- Energy usage of the lightweight classifiers depend on the underlying hardware on which the energy benchmarks are being run, as well as runtime interpreter and RTOS being used. For example, the energy consumption of TCN ranges from 50-418 mW depending on the hardware platform. The L-series STM32 boards are branded as ultra-low-power, while the F-series STM32 boards are high-performance [10]. Thus, the same classifier implemented on different hardware can yield different energy consumption, evident from FastRNN and FastGRNN's implementation on two STM32 boards. In addition, Raspbian RTOS and TensorFlow Lite interpreter consume more power to run the same model over Mbed/Arduino RTOS and Tensorflow Lite Micro interpreter.

Fig. 6 showcases the accuracy and model size of our earable activity detection models (colored black) versus proposed models in literature (colored red), namely CNN [38], RF [75], and kNN [38]. For activity detection

---

[9]https://github.com/mlcommons/tiny
[10]https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html
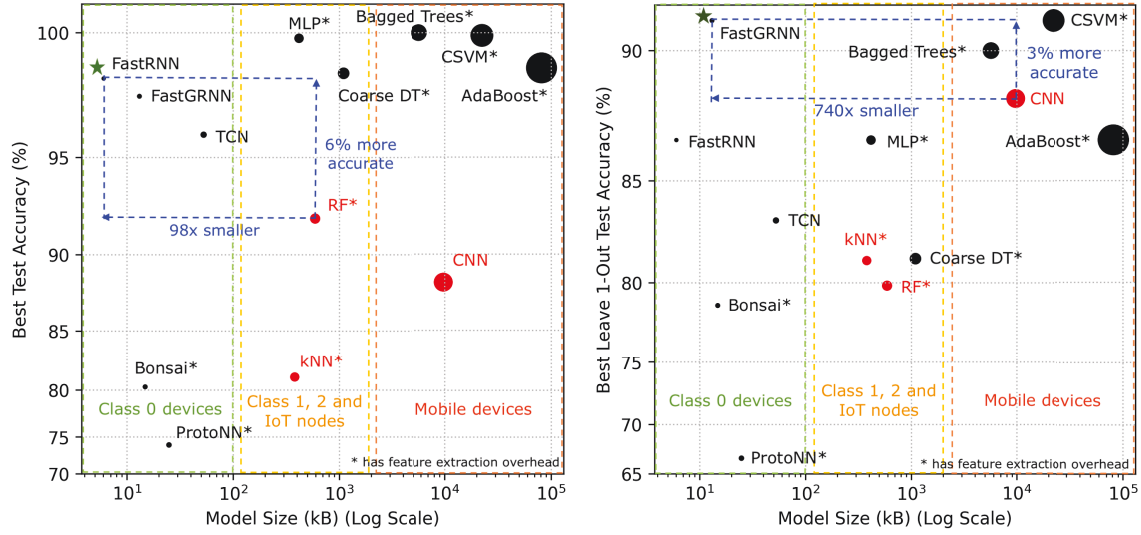
Fig. 6. Performance comparison of our earable activity detection models (colored black) versus proposed models in literature (colored red). For activity detection on seen participants, FastRNN provides 6% accuracy improvement over the SOTA, while being 98× smaller. For activity detection on unseen participants, FastGRNN provides 3% accuracy improvement over the SOTA, while being 740× smaller. Both classifiers are suitable for deployment on ultra-resource-constrained devices.

on seen participants, compared to the SOTA RF model, FastRNN is 98× smaller and 6% more accurate without needing additional feature extraction overhead. For activity detection on unseen participants, compared to the SOTA CNN model, FastGRNN is 740× smaller and 3% more accurate without needing additional feature extraction overhead. In addition, our lightweight models are suitable for implementation on devices on Class 0 devices (Internet-of-Things (IoT) devices with < 100 kB flash [47]), while the SOTA, as well as our conventional classifiers, can only be run on mobile devices or Class 1 (IoT devices with ∼ 100 kB flash [47]) and Class 2 (IoT devices with ∼ 250 kB flash [47]) devices. Using our model zoo, it is possible to generate activity detection models suitable for a broad spectrum of hardware classes with different compute constraints while maintaining superior accuracy.

5.2 Activity Detection Multiclass Metrics and Effect of Window Size

Fig. 7 outlines the leave-1 out class-dependent errors (precision, recall, and F1 score) for all 10 activity classifiers. The multiclass metrics were obtained for different window sizes (1, 3, 5, and 10 seconds for conventional classifiers; 1, 2, 3 and 5 seconds for lightweight classifiers). From Fig. 7 (left), we can see that the median precision and recall of each classifier are roughly similar. We can also observe that SVM, TCN, Bonsai, and ProtoNN are the most stable in terms of multiclass classification quality across different window sizes. The SVM classifier achieves the highest median precision, recall, and F1 score, indicating a high degree of completeness and exactness and a low number of false positives and false negatives across classes. On the other hand, AdaBoost and MLP have the largest range of class-dependent error for different window sizes, indicating a significant dependence of accuracy on window size.

Fig. 8 shows the normalized leave-1 out test accuracies for all 10 classifiers with varying window sizes. The accuracy of all classifiers improves with larger window sizes. This is because, with larger time windows, the classifier has access to more spatial and temporal information. Small windows may not always have enough
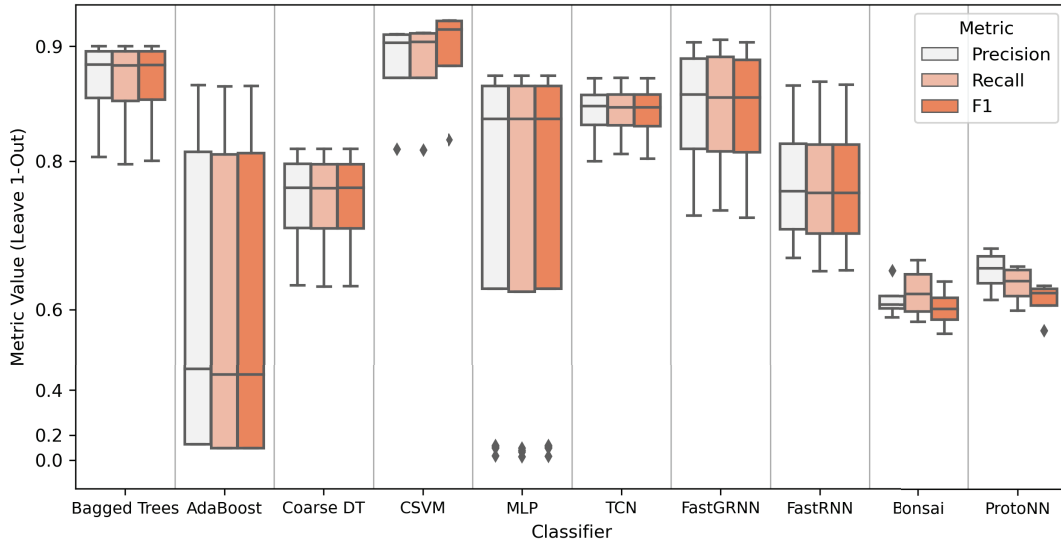
Fig. 7. Boxplot showing leave-1 out multiclass error metrics for our proposed classifiers for varying window sizes.
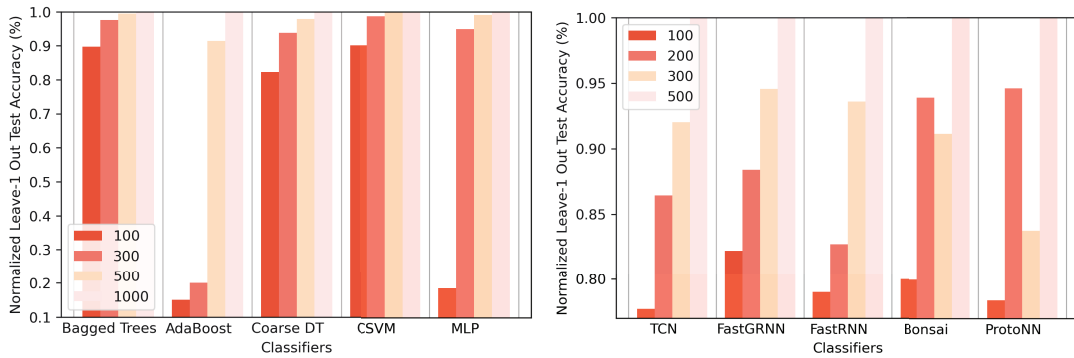

Fig. 8. Effect of window size on normalized leave-1 out test accuracy on conventional and lightweight activity classifiers.

differential features to classify each activity separately. However, time windows longer than 2.5-3.5 seconds [97] may not be helpful when rapid changes in activities occur or when a macro-activity can be decomposed into transient micro-activities. Furthermore, longer time windows can reduce inference speed [97]. Thus, for practical deployment, it is recommended to keep window size around 2.5-3.5 seconds [97].

5.3 Activity Detection Cross-Validation Studies (Leave-n Out)

To test the generalization capability of all 10 classifiers with a varying number of unseen participants in the training set, we performed a leave-n out study where we tested the accuracy of all the classifiers with a varying number of participants left out of the training set. Fig. 9 showcases summarizes the results of the study. While the test accuracy on unseen participants drops with an increase in the number of participants left out of the
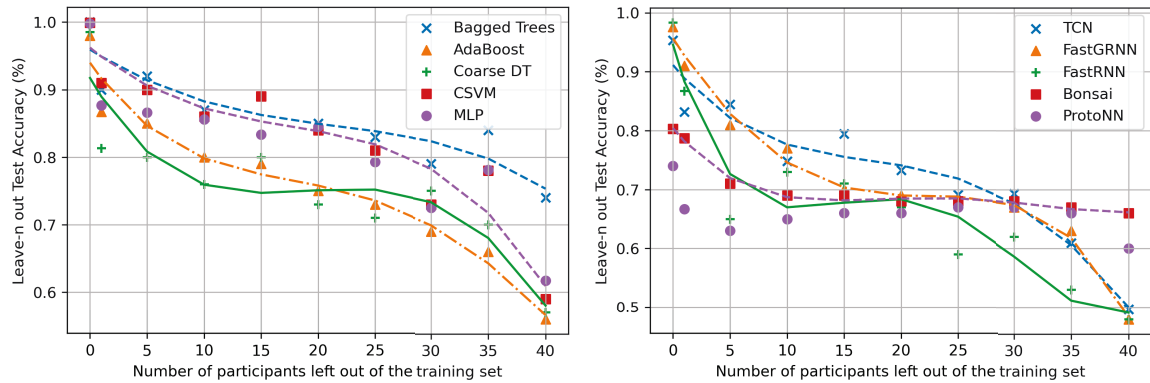
Fig. 9. Generalization performance (leave-n out test accuracy) of our earable activity classifiers with increase in the number of participants left out of the training set. The window size was 10 seconds for conventional classifiers and 5 seconds for lightweight classifiers.

training set, the accuracy of lightweight classifiers drops by around 11.8% more on average over conventional classifiers for the same value of n. As discussed in Section 5.1, lightweight models suffer from generalizability due to a low number of redundant weights to model global features. The problem is particularly worse for FastRNN, FastGRNN, and TCN, which attempt to make inferences on raw data, compared to Bonsai and ProtoNN, which make inferences on features. FastRNN, FastGRNN, and TCN require the injection of domain adaption, possibly via domain adversarial training to make these NN robust across domains [16, 30] if they are to work without feature extraction.

5.4 Performance of Hardware-in-the-loop Bayesian Neural Architecture Search

To showcase how our NAS helps adapt the same model for different hardware, we optimized the TCN model for five different hardware with different compute capabilities. Fig. 10 illustrates how our hardware-aware NAS tunes the TCN architecture for three of those hardware to improve model accuracy by maximizing the available compute resources of the device. As the SRAM capacity of the device increases, the NAS framework increases the number of layers and filters in the TCN model. To prevent EVGP, NAS also adds skip connections as the number of layers increases. Another interesting observation is how our NAS pipeline assigns the dilation factor to each layer. To capture both local and global dynamics within a limited computing budget, NAS assigns a small dilation factor to the lower layer to capture short-term local context, and a large dilation factor in higher layers to capture long-term global inter-dependencies in the temporal sequence. Classically, a human designer would assign dilation factors that increase by a constant factor with each successive layer instead of the complex dilation factor assignment that the NAS performs. This observation further strengthens the need for intelligent AutoML frameworks for deployable ML model development.

We also performed an ablation study to see how proxyless (with real-hardware) and proxied versions (with proxy to simulate hardware metric) of our NAS framework differ in performance with three real hardware devices. The results are shown in Fig. 11 (left). From Fig. 11 (left), we can observe that as the resource budget of target hardware loosens, the difference in accuracy between the best performing model found by proxyless NAS and proxied NAS reduces. In both cases, the accuracy of the model improves with more capable hardware. The difference in accuracy, albeit within ±6%, arises from the runtime interpreter and RTOS overhead which the proxy for SRAM and flash fails to account for. As a result, some well-performing model candidates found by proxied
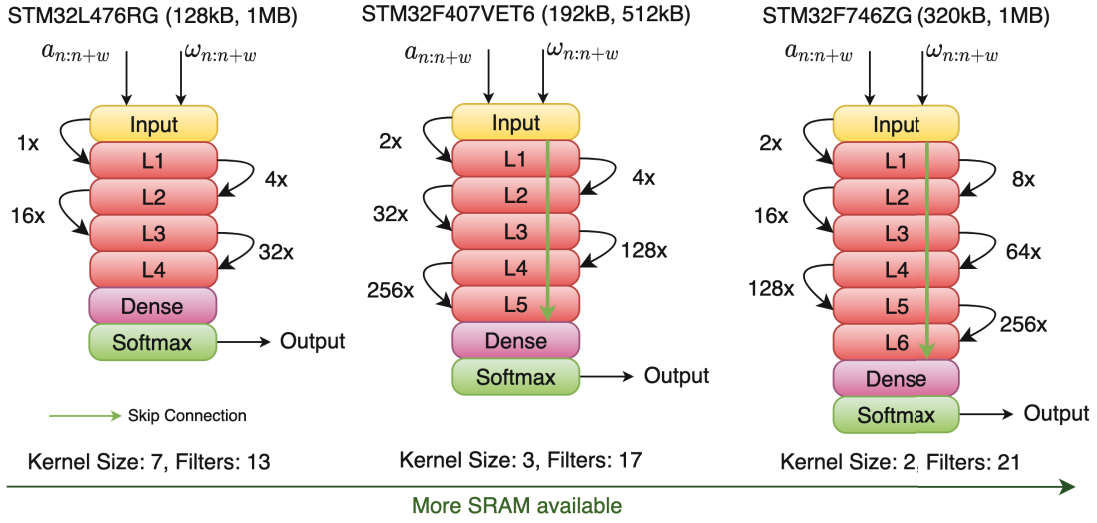
Fig. 10. Example architectural adaption and device capability exploitation by Bayesian NAS based on resource usage for TCN activity classifier. The RAM and flash constraints of the device are written inside parenthesis. L_i refers to i^{th} layer of TCN.

NAS may not fit on the real hardware when one takes We can infer this from the offset observed between SRAM usage reported by proxyless and proxied NAS. Overall, HIL becomes important for ultra resource-constrained devices, where all overheads need to be accounted for.

Besides quantifying the difference between proxyless and proxied NAS for memory and accuracy modeling, we also studied the relationship between FLOPS, model latency (from real hardware), and model accuracy, summarized in Fig. 11 (right). We observed that there is a strong positive correlation (Pearson Coefficient, $\rho = 0.998$) between FLOPS and model latency, indicating that it is possible to develop an analytical model
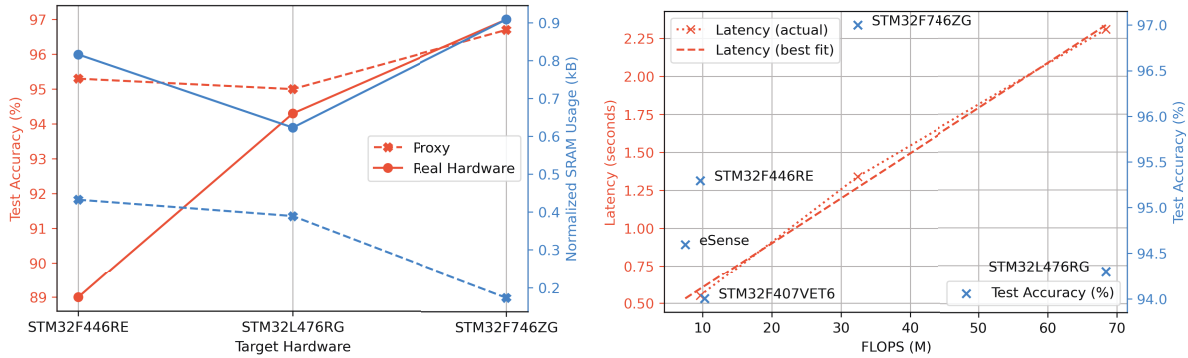

Fig. 11. (Left) Accuracy and SRAM usage estimation comparison between proxyless Bayesian NAS and proxied Bayesian NAS for different devices. The SRAM usage is normalized by maximum RAM capacity of each device. (Right) Relationship between FLOPS, model latency and accuracy for TCN earable activity classifiers geared towards different devices. FLOPS and latency have a strong linear correlation.

correlating FLOPS and model latency without requiring HIL. The same observation was made by Banbury *et al.* [8] for models geared towards microcontrollers. Analytical models for latency will be much faster over getting the latency metric directly from real hardware [78]. However, we found that more FLOPS do not always translate to higher model accuracy. We did not observe a significant correlation ($\rho = 0.0107$) between FLOPS and model accuracy, which was also observed in Table 9. As discussed earlier, only a small portion of model parameters are responsible for contributing towards model accuracy, resulting in no correlation between FLOPS and classification performance.

5.5 Head-Pose Filter Size and Error Characteristics

Fig. 12 summarizes the error characteristics and resource usage of the head-pose filters in filter zoo. From Fig. 12 (a), we can observe that the IEKF provided the lowest mean absolute error (MAE) of 6.49° and 3.53° for head-tracking in the azimuth and elevation plane, respectively. This is expected, as the IEKF can minimize the variance in the attitude estimate optimally through the innovation and estimate covariance matrices. Compared to Yang *et al.* [102], the IEKF provides 1.6× improvement in error characteristics as shown in Fig. 12 (b) using a single earable IMU. Note that the average MAE of the IEKF increases by $\sim 2°$ when translational motion artifacts (e.g., walking) are introduced along with head movements. Although the IEKF provides superior error characteristics over other filters, it is also the most resource-intensive as shown in Fig. 12 (c). Compared to Madgwick or Mahony filter, the IEKF requires 17% more flash and 3.3× more SRAM when implemented for Mbed RTOS, while providing 1.8× lower MAE. Given the resource usage, IEKF cannot be implemented on AVR RISC processors running Arduino. A trade-off would be to choose the Mahony filter, which requires an average of 1.5 kB of SRAM while providing an MAE of 8.62°. The filter requires 19.8 kB and 40.4 kB of flash when implemented on Cortex-M4 and AVR RISC microcontrollers, respectively. The lightest of all filters is the complementary filter, which, unfortunately, also suffers from the largest MAE, as it cannot account for drift and noise as elegantly as other filters. Note that since these filters are analytical, they are wearer-independent, unlike the activity detection models.

Fig. 12 (d) showcases the importance of tuning filter parameters. Without optimization, the average MAE of the filters increases anywhere between 8.1° and 72.9°. As a result, before using the filters in the filter zoo for head-pose estimation, it is recommended to calibrate the filters on some samples of the wearer's head movements. We suggest the following orientation filter calibration program:

- First, the accelerometer and the gyroscope within the earable must be calibrated to remove static gyroscope bias drift and estimate the accelerometer gains and biases using the techniques described in Section 3.2.
- The user immediately wears the earable after calibration, then faces roughly straight ahead (azimuth angle of 0°) and starts IMU data logging, all while not making any significant head movements. The wearer then moves the head slowly from 0° to any angle larger than 10° but less than 90° to the right, and back. Note that the user does not have to move the head exactly back to the initial azimuth angle of 0°.
- The wearer repeats the process but for the elevation plane (vertical head movement). Note that we suggest the user complete the two data logging steps within a minute of performing accelerometer and gyroscope calibration.
- Since the gyroscope and the accelerometer has just been calibrated, we can obtain the 3D ground truth orientation (roll (ϕ_t), pitch (θ_t), and yaw (ψ_t)) for the user's head trajectory at timestep t directly from the IMU readings using the following equations:

$$\begin{bmatrix} \phi_t \\ \theta_t \end{bmatrix}_a = \begin{bmatrix} \arctan\left(\frac{A_{y,t}}{\sqrt{A_{x,t}^2+A_{z,t}^2}} \cdot \frac{180}{\pi}\right) \\ \arctan\left(\frac{A_{x,t}}{\sqrt{A_{y,t}^2+A_{z,t}^2}} \cdot \frac{180}{\pi}\right) \end{bmatrix}, \qquad \begin{bmatrix} \phi_t \\ \theta_t \\ \psi_t \end{bmatrix}_\omega = \begin{bmatrix} \phi_{t-1} \\ \theta_{t-1} \\ \psi_{t-1} \end{bmatrix} + \begin{bmatrix} \omega_{x,t} \\ \omega_{y,t} \\ \omega_{z,t} \end{bmatrix} \begin{bmatrix} f_s^{-1} \\ f_s^{-1} \\ f_s^{-1} \end{bmatrix} \qquad (15)$$

$$\begin{bmatrix} \phi_t \\ \theta_t \\ \psi_t \end{bmatrix}_{\text{GT}} = \begin{bmatrix} 0.5(\phi_{t,a} + \phi_{t,\omega}) \\ 0.5(\theta_{t,a} + \phi_{t,\omega}) \\ \psi_{t,\omega} \end{bmatrix} \tag{16}$$

where, $A_{x,y,z}$ refers to accelerometer readings, $\omega_{x,y,z}$ refers to gyroscope readings, and f_s refers to IMU sampling rate.
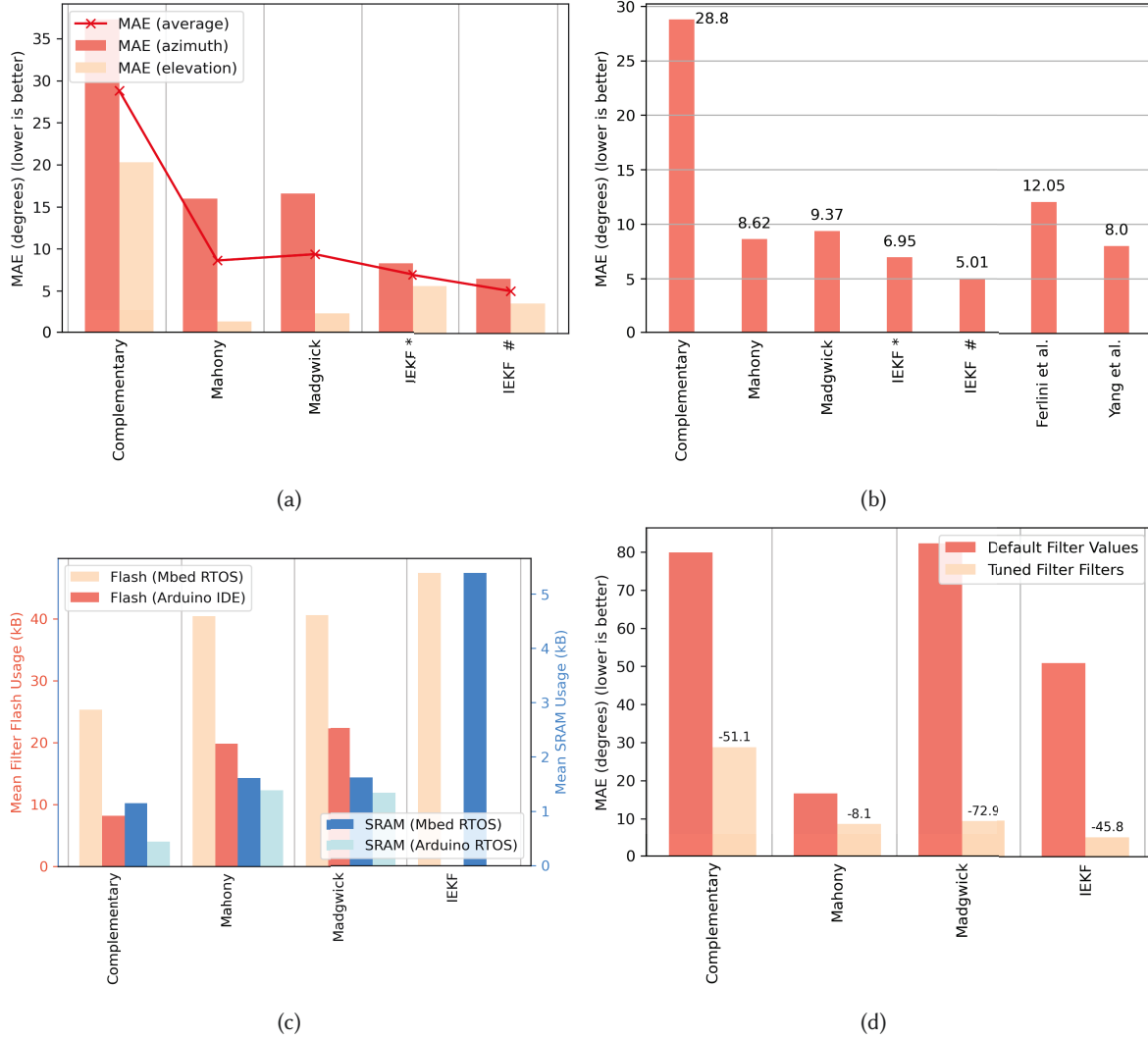

Fig. 12. (a) Error characteristics (planar and average) of the head-pose filters (tuned) in filter zoo . IEKF* refers to IEKF with translational movements and IEKF# refers to IEKF with only head movements. (b) Comparison of error characteristics of proposed head-pose filters against SOTA (c) SRAM and flash usage of proposed filters for ARM Cortex M4 (running Mbed) and AVR RISC (running Arduino) processor architectures (d) Error reduction via BO of filter parameters.

- The filter parameters can then be tuned by plugging in the recorded IMU data and the ground truth orientation and performing an exhaustive search over the possible range of filter parameters to minimize filter MAE.

The orientation program does not cause any hassle to the user as the user does not have to follow any strictly bounded head motion trajectory. In theory, the number of data points required for calibration equals the number of tunable and initial filter parameters (e.g., Madgwick and complementary filters have only a single tunable parameter, Mahony filter has two tunable parameters, while the IEKF has 4 tunable parameters and a 9×9 initial process noise covariance matrix). More data points can help provide a global and over-parametrized notion to the exhaustive search, possibly yielding better optimal values of the filter parameters. However, the search will be slow if the number of points is too large. Furthermore, if the user attempts to collect the orientation filter calibration data over a long time period, then the initial static IMU calibration parameters will become invalid. Therefore, more data points will not essentially lead to better estimates of filter parameters.

6 APPLICATIONS AND CASE STUDIES

To highlight the utility of AURITUS, we showcase three canonical applications developed by using the tools provided in AURITUS. These include fall detection (Section 6.1), spatial audio generation (Section 6.2) and interacting with objects in an AR digital twin (Section 6.3).

6.1 Fall Detection

Falls cause frequent injuries and death among the elderly population, with ~684,000 fatal cases occurring annually [1, 42, 101]. With an increasing number of elderly people living alone [90], there is a strong association between living alone and suffering from a fall ($\chi^2 = 0.005$) [23] among senior citizens, with 37 million cases requiring medical attention [1]. Thus, it is necessary to develop an accurate yet lightweight and real-time fall detection system that can reduce the lead time between detection of a fall and receiving medical attention [90, 101]. Thereby, we designed ultra-lightweight ML models using AURITUS that can distinguish between falls and non-falls through earables. Fig. 13 illustrates the performance of fall detection in terms of model size and leave-1 out test accuracy for various window sizes. In general, the leave-1 out accuracy of fall detection models improved with larger window sizes. Among all the models, Bonsai and ProtoNN had the highest average leave-1 out fall
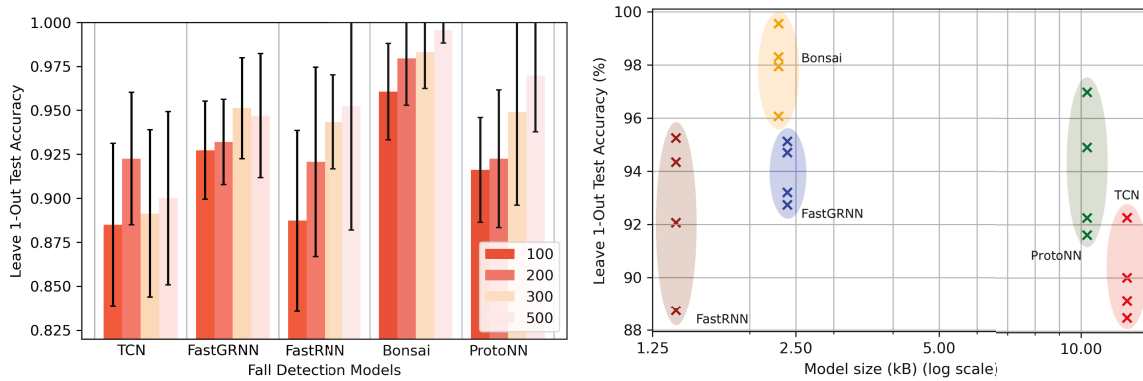

Fig. 13. (Left) Leave-1 out fall detection accuracy of lightweight ML models for various window sizes trained to distinguish between falls and non-falls. (Right) Model size versus leave-1 out fall detection accuracy of lightweight ML models trained to distinguish between falls and non-falls.

detection accuracy of 99% and 98%, respectively for 5-second windows. We managed to squeeze the model size to only 2.3 kB for Bonsai using BO. The model size is so small that Bonsai requires negligible resources to be run in real-time on microcontroller-class devices. Bonsai can also maintain its accuracy within ±1% for window sizes smaller than 5 seconds (e.g., 2 and 3 seconds). Small window sizes are important for fall detection as the essential part of the fall event typically lasts around 2 seconds [58].

6.2 Binaural Audio Rendering

Spatial audio refers to the process of generating audio that provides the listener with a perception about the direction, distance, size, and type of object [25]. Also known as binaural audio, 3D audio is useful for indoor acoustic AR [102], providing directional localization cues [19], exercise feedback [77], and interacting with virtual objects [71]. Given the small size and portability of earables coupled with the presence of both head-tracking sensors and stereo speakers, it is possible to generate a lightweight perception-processing-feedback setup by combining head-tracking with binaural audio. Generally, a head-pose filter supplies azimuth and elevation angles to a pair of two head-related transfer functions (HRTF) [17, 102]. The HRTF is the response of how the human ear perceives the location of the sound. Mono audio is convolved with the HRTF finite impulse response filters to generate binaural audio, with the interpolated HRTF calculated at the position of the head-tracked.
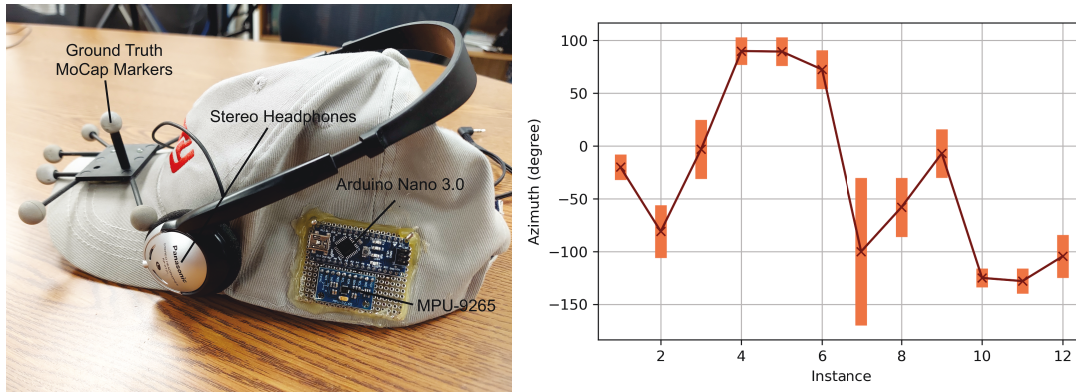

Fig. 14. (Left) Hardware setup for binaural audio rendering experiment. (Right) Snapshot of sound source localization test with real participants to quantify head-pose filter performance.

We implemented a 3D spatial audio framework in MATLAB using Auritus to showcase the utility of our head-tracking filters. We used the ARI HRTF database [17]. The database has data points for 1550 positions for over 200 subjects, with an azimuthal resolution of 2.5° (-45° to +45°) and elevational resolution of 5° (-30° to +80°) [17]. The HRTF for a point outside the angular range in the dataset is found using interpolation. For head tracking, we used the IEKF. Since eSense is non-programmable, we created our own hardware setup to stream head-tracking data in real-time to the HRTF, shown in Fig 14 (left). The setup consists of an Arduino Nano 3.0 connected to an MPU-9265 9DoF IMU. We only used the accelerometer and gyroscope data from the 9DoF IMU as the original earable does not have a magnetometer. We also calibrated the accelerometer and gyroscope of the MPU-9265 using calibration techniques described in Section 3.2. The audio is streamed through headphones from the HRTF kernel.

Fig 14 (right) shows a snapshot for the binaural audio sound source localization test. In the test, we asked a participant to roughly locate which azimuth direction a virtual sound source is situated at by listening to spatial audio using our hardware setup. It is generally assumed that the human sound source localization resolution
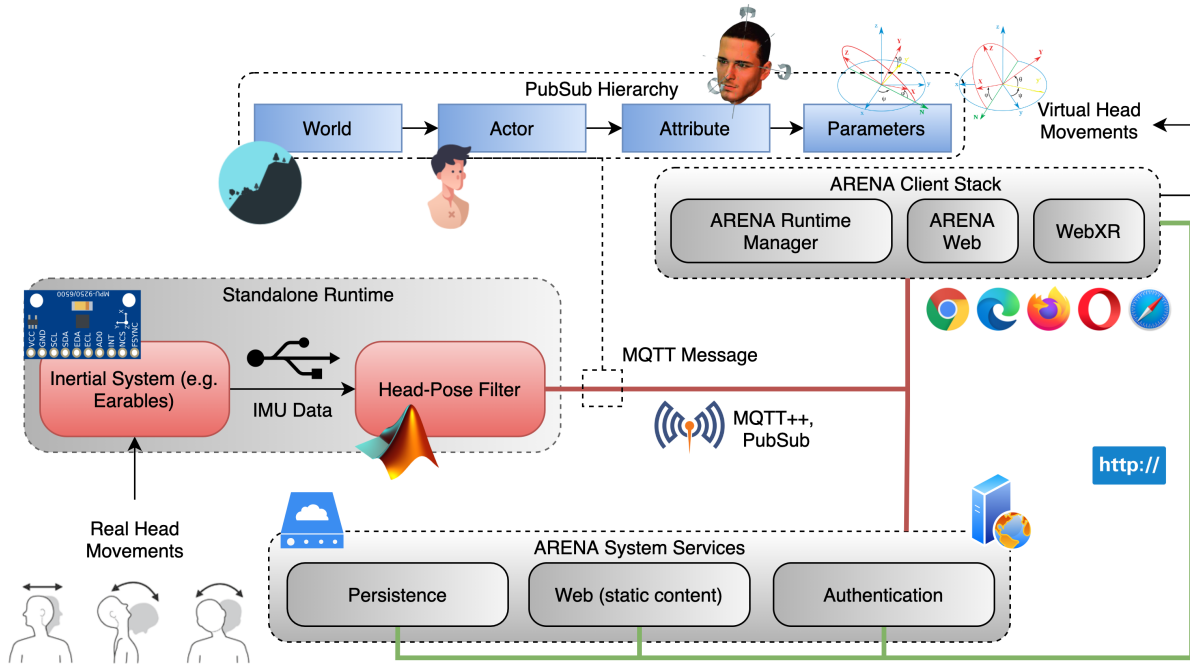
Fig. 15. Controlling head-orientation of virtual actors in an AR space (CONIX ARENA) using real head-pose. The head-pose filter application communicates with the AR framework through Pub/Sub topic hierarchy, which in turn alters atrribute parameters (in this case, head pose) of target actor in the AR world.

using binaural cues varies widely around $\pm 6 - \pm 20°$ [22, 70], and a head-pose filter must be able to provide the direction of the head with an MAE less than $20°$. From our sound source localization test, we obtained a localization error of $\pm 22.7°$. The error is on the higher end of the localization spectrum not due to IEKF errors, but partly due to the use of non-personalized HRTF database [102]. In addition, some of the binaural cues were outside the $-45°$ to $+45°$ azimuth range provided by the dataset, causing errors to be introduced in spatial sound generation due to interpolation of HRTF kernel. The participant we selected for the test may have had an aural localization resolution around $\pm 20°$, which added to the cumulative errors from the HRTF database.

6.3   Interacting with AR Frameworks

One of the most promising applications of earables is the ability to control virtual objects an AR digital twin [19]. We used the same hardware setup developed in Section 6.2 to control the orientation and motion of actors in a virtual world wirelessly using head movements. Fig. 15 showcases the interaction between the AR framework and the head-pose application. We used the open-source CONIX Arena [71] AR architecture to showcase this application. The head-pose application communicates with the virtual world using MQTT (Pub/Sub) messages. The Pub/Sub message specifies the ARENA server to connect to, the realm (world), an object ID within the realm specifying the actor, the attribute (e.g., head orientation) to alter, and the values of the parameters of that attribute (e.g., Euler angles). We observed negligible delays in updating the parameters through MQTT, with the head-pose filter (an IEKF) running at 100 Hz. The framework thus allows one to control the parameters of virtual actors using real sensor values in the real world in near real-time without significant latency. Although we developed

the application to control the head-pose of virtual humans, it is also possible, for example, to control real drones, cars, and appliances using head-movements through their digital twin in the ARENA realm.

7 CONCLUSION, LIMITATIONS, AND FUTURE WORK

Given the commercialization potential of earables, an open-source end-to-end toolkit can enable the accelerated development of future ventures, catalyzing the adoption of new technologies and sensing modalities in smart earbuds [19]. Auritus provides a tightly-coupled collection of open-source and extendable libraries, datasets, and tools that allow application developers and researchers in earable computing to collect human movement data, label time-series data interactively, and develop new human movement models and filters. The model and filters in Auritus are designed to meet hardware constraints without sacrificing accuracy. In the process, Auritus advances the SOTA activity classification models and head-pose filters in their accuracy even with lightweight models and filters. Moreover, our experience with Auritus demonstrates that the toolkit is capable of supporting a variety of different applications and research needs. Several lessons, limitations and directions of future work for our framework are as follows:

- Since the activities in our dataset are scripted and of short periods, the evaluation has some limitations. *Firstly*, the activities are not completely natural and continuous due to missing context and context change. *Secondly*, due to short activity periods, the effects of earable placement and displacement are not significant. Both of these effects can provide erroneous classification results in the wild and lead to an upper bound in accuracy our system can reach.
- While our work shows that Auritus is capable of supporting diverse applications, our framework is currently limited to work with inertial sensor data only. Smart earbuds also include other modalities (e.g., audio, BLE, temperature, etc.), which can be disruptive for a number of physiological applications [19]. The largest change for such an expansion would be in the data collection and labeling pipeline. The smartphone application needs to be expanded to collect data for other modalities, while modality-specific ground truth data collection hardware (e.g., microphones, binaural audio generators, and bio-electrical and physiological signal measurement devices) need to be invested in. While the lightweight model generation pipeline generalizes to any modality without changes, any optional feature extraction and windowing need to be domain-specific (e.g., log-Mel spectrogram for audio, received signal strength indicator for BLE, and fast Fourier transform coefficients for heart rate).
- Our findings indicate that over-the-air model adaption is important to handle cross-user variations and domain variance for lightweight models, which our framework currently does not handle. This would require the earable devices to be capable of collecting human movement data and adapting the decision boundaries of the baseline model onboard on the fly to reduce performance drop [13, 55]. However, since earables are constrained in SRAM, more work needs to be done to allow efficient on-device training. Domain adversarial training may also be able to generate models robust to ambient disturbances and cross-user variations [16, 30].
- Although most commercial earables (including eSense) do not yet support firmware changes and only allow access to data, we speculate that future earables will allow onboard programming with apps specifically developed for onboard inference. For example, early wrist-worn devices were fixed worn devices with applications running on smartphones. Modern smartwatches now allow programming and on-device processing [18]. When such devices emerge in earable computing, Auritus would provide developers with the necessary model training and development framework.
- We found out that sensor data from earable devices suffer from missing data, cross-channel timestamp misalignment, and window jitter, due to packet drops and the absence of on-chip clocks. This can reduce the performance of ML models when training for complex event processing [84]. The solution can be to

either inject ML models with uncertainty awareness via uncertainty-injected training pipeline [84, 89] or use onboard clocks and hardware enhancements for precise time-synchronization [88] and handling packet drops.

ACKNOWLEDGMENTS

REFERENCES

[1] 2021. Falls - World Health Organization. Retrieved from: https://www.who.int/en/news-room/fact-sheets/detail/falls, Accessed: 29th Oct. 2021.

[2] Md Atiqur Rahman Ahad, Anindya Das Antar, and Masud Ahmed. 2020. *IoT sensor-based activity recognition*.

[3] Ashwin Ahuja, Andrea Ferlini, and Cecilia Mascolo. 2021. PilotEar: Enabling In-ear Inertial Navigation. In *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*. 139–145.

[4] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra Perez, and Jorge Luis Reyes Ortiz. 2013. A public domain dataset for human activity recognition using smartphones. In *Proceedings of the 21th International European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 437–442.

[5] Louis Atallah, Anatole Wiik, Gareth G Jones, Benny Lo, Justin P Cobb, Andrew Amis, and Guang-Zhong Yang. 2012. Validation of an ear-worn sensor for gait monitoring using a force-plate instrumented treadmill. *Gait & posture* 35, 4 (2012), 674–676.

[6] Alexander M Aurand, Jonathan S Dufour, and William S Marras. 2017. Accuracy map of an optical motion capture system with 42 or 21 cameras in a large measurement volume. *Journal of biomechanics* 58 (2017), 237–240.

[7] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. MLPerf Tiny Benchmark. *Advances in Neural Information Processing Systems* (2021).

[8] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. 2021. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems* 3 (2021).

[9] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series.. In *KDD workshop*, Vol. 10. Seattle, WA, USA:, 359–370.

[10] Silvere Bonnabel. 2007. Left-invariant extended Kalman filter and attitude estimation. In *2007 46th IEEE Conference on Decision and Control*. IEEE, 1027–1032.

[11] Andreas Bulling, Ulf Blanke, and Bernt Schiele. 2014. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)* 46, 3 (2014), 1–33.

[12] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations*.

[13] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. *Advances in Neural Information Processing Systems* 33 (2020).

[14] Gaoshuai Cao, Kuang Yuan, Jie Xiong, Panlong Yang, Yubo Yan, Hao Zhou, and Xiang-Yang Li. 2020. Earphonetrack: involving earphones into the ecosystem of acoustic motion tracking. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 95–108.

[15] Eduardo Casilari, Jose A Santoyo-Ramón, and Jose M Cano-García. 2017. Umafall: A multisensor dataset for the research on automatic fall detection. *Procedia Computer Science* 110 (2017), 32–39.

[16] Changhao Chen, Yishu Miao, Chris Xiaoxuan Lu, Linhai Xie, Phil Blunsom, Andrew Markham, and Niki Trigoni. 2019. Motiontransformer: Transferring neural inertial tracking between domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 8009–8016.

[17] Wei Chen, Ruimin Hu, Xiaochen Wang, and Dengshi Li. 2020. HRTF Representation with Convolutional Auto-encoder. In *International Conference on Multimedia Modeling*. Springer, 605–616.

[18] Xiao Chen, Wanli Chen, Kui Liu, Chunyang Chen, and Li Li. 2021. A comparative study of smartphone and smartwatch apps. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 1484–1493.

[19] Romit Roy Choudhury. 2021. Earable computing: A new area to think about. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*. 147–153.

[20] Hakyoung Chung, Lauro Ojeda, and Johann Borenstein. 2001. Accurate mobile robot dead-reckoning with a precision-calibrated fiber-optic gyroscope. *IEEE transactions on robotics and automation* 17, 1 (2001), 80–84.

[21] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. 2021. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. *Proceedings of MLSys* 3 (2021).

[22] Michael F Dorman, Louise H Loiselle, Sarah J Cook, William A Yost, and René H Gifford. 2016. Sound source localization by normal-hearing listeners, hearing-impaired listeners and cochlear implant listeners. *Audiology and Neurotology* 21, 3 (2016), 127–131.

[23] Sharon Elliott, Jane Painter, and Suzanne Hudson. 2009. Living alone and fall risk factors in community-dwelling middle age and older adults. *Journal of community health* 34, 4 (2009), 301–310.

[24] Mark Euston, Paul Coote, Robert Mahony, Jonghyuk Kim, and Tarek Hamel. 2008. A complementary filter for attitude estimation of a fixed-wing UAV. In *2008 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 340–345.

[25] Christof Faller. 2004. *Parametric coding of spatial audio*. Technical Report. EPFL.

[26] Xiaoran Fan, Longfei Shangguan, Siddharth Rupavatharam, Yanyong Zhang, Jie Xiong, Yunfei Ma, and Richard Howard. 2021. HeadFi: bringing intelligence to all headphones. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 147–159.

[27] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul N Whatmough. 2019. SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers. *Advances in Neural Information Processing Systems* 32 (2019).

[28] Andrea Ferlini, Alessandro Montanari, Cecilia Mascolo, and Robert Harle. 2019. Head motion tracking through in-ear wearables. In *Proceedings of the 1st International Workshop on Earable Computing*. 8–13.

[29] Joshua S Furtado, Hugh HT Liu, Gilbert Lai, Herve Lacheray, and Jason Desouza-Coelho. 2019. Comparative analysis of optitrack motion capture systems. In *Advances in Motion Sensing and Control for Robotic Applications*. Springer, 15–31.

[30] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *The journal of machine learning research* 17, 1 (2016), 2096–2030.

[31] Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. 2020. Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. *Neurocomputing* 380 (2020), 20–35.

[32] Bruno Gil, Salzitsa Anastasova, and Guang Z Yang. 2019. A smart wireless Ear-Worn device for cardiovascular and sweat parameter monitoring during physical exercise: design and performance results. *Sensors* 19, 7 (2019), 1616.

[33] Hristijan Gjoreski, Mathias Ciliberto, Lin Wang, Francisco Javier Ordonez Morales, Sami Mekki, Stefan Valentin, and Daniel Roggen. 2018. The university of sussex-huawei locomotion and transportation dataset for multimodal analytics with mobile devices. *IEEE Access* 6 (2018), 42592–42604.

[34] Sridhar Gopinath, Nikhil Ghanathe, Vivek Seshadri, and Rahul Sharma. 2019. Compiling KB-sized machine learning models to tiny IoT devices. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 79–95.

[35] Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. 2017. Protonn: Compressed and accurate knn for resource-scarce devices. In *International Conference on Machine Learning*. PMLR, 1331–1340.

[36] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)* (2016).

[37] Mohammed Mehedi Hassan, Md Zia Uddin, Amr Mohamed, and Ahmad Almogren. 2018. A robust human activity recognition system using smartphone sensors and deep learning. *Future Generation Computer Systems* 81 (2018), 307–313.

[38] Tahera Hossain, Md Shafiqul Islam, Md Atiqur Rahman Ahad, and Sozo Inoue. 2019. Human activity recognition using earable device. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. 81–84.

[39] Md Shafiqul Islam, Tahera Hossain, Md Atiqur Rahman Ahad, and Sozo Inoue. 2021. Exploring Human Activities Using eSense Earable Device. In *Activity and Behavior Computing*. Springer, 169–185.

[40] Richard A Johnson, Irwin Miller, and John E Freund. 2000. *Probability and statistics for engineers*. Vol. 2000.

[41] R. E. Kalman. 1960. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering* 82, 1 (1960), 35–45.

[42] Pekka Kannus, Jari Parkkari, Seppo Koskinen, Seppo Niemi, Mika Palvanen, Markku Järvinen, and Ilkka Vuori. 1999. Fall-induced injuries and deaths among older adults. *Jama* 281, 20 (1999), 1895–1899.

[43] Shin Katayama, Akhil Mathur, Marc Van den Broeck, Tadashi Okoshi, Jin Nakazawa, and Fahim Kawsar. 2019. Situation-Aware Emotion Regulation of Conversational Agents with Kinetic Earables. In *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE, 725–731.

[44] Nobuo Kawaguchi, Nobuhiro Ogawa, Yohei Iwasaki, Katsuhiko Kaji, Tsutomu Terada, Kazuya Murao, Sozo Inoue, Yoshihiro Kawahara, Yasuyuki Sumi, and Nobuhiko Nishio. 2011. HASC Challenge: gathering large scale human activity corpus for the real-world activity understandings. In *Proceedings of the 2nd augmented human international conference*. 1–5.

[45] Fahim Kawsar, Chulhong Min, Akhil Mathur, and Allesandro Montanari. 2018. Earables for personal-scale behavior analytics. *IEEE Pervasive Computing* 17, 3 (2018), 83–89.

[46] Fahim Kawsar, Chulhong Min, Akhil Mathur, Alessandro Montanari, Utku Günay Acer, and Marc Van den Broeck. 2018. eSense: Open Earable Platform for Human Sensing. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. 371–372.

[47] James King and Ali Ismail Awad. 2016. A distributed security mechanism for resource-constrained IoT devices. *Informatica* 40, 1 (2016).

[48] Manon Kok, Jeroen D Hol, and Thomas B Schön. 2017. Using Inertial Sensors for Position and Orientation Estimation. *Foundations and Trends in Signal Processing* 11, 1-2 (2017), 1–153.

[49] Ashish Kumar, Saurabh Goyal, and Manik Varma. 2017. Resource-efficient machine learning in 2 kb ram for the internet of things. In *International Conference on Machine Learning*. PMLR, 1935–1944.

[50] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. 2018. FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network. In *NeurIPS*.

[51] Matias Laporte, Preety Baglat, Shkurta Gashi, Martin Gjoreski, Silvia Santini, and Marc Langheinrich. 2021. Detecting Verbal and Non-Verbal Gestures Using Earables. In *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*. 165–170.

[52] Oscar D Lara and Miguel A Labrador. 2012. A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials* 15, 3 (2012), 1192–1209.

[53] Colin Lea, Rene Vidal, Austin Reiter, and Gregory D Hager. 2016. Temporal convolutional networks: A unified approach to action segmentation. In *European Conference on Computer Vision*. Springer, 47–54.

[54] Seungchul Lee, Chulhong Min, Alessandro Montanari, Akhil Mathur, Youngjae Chang, Junehwa Song, and Fahim Kawsar. 2019. Automatic Smile and Frown Recognition with Kinetic Earables. In *Proceedings of the 10th Augmented Human International Conference 2019*. 1–4.

[55] Seulki Lee and Shahriar Nirjon. 2020. Learning in the Wild: When, How, and What to Learn for On-Device Dataset Adaptation. In *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*. 34–40.

[56] Edgar Liberis, Łukasz Dudziak, and Nicholas D Lane. 2021. μNAS: Constrained Neural Architecture Search for Microcontrollers. In *Proc. of the 1st WKSH on Machine Learn. and Sys.* 70–79.

[57] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. 2020. MCUNet: Tiny Deep Learning on IoT Devices. *Advances in Neural Information Processing Systems* 33 (2020), 11711–11722.

[58] Kai-Chun Liu, Chia-Yeh Hsieh, Hsiang-Yun Huang, Steen Jun-Ping Hsu, and Chia-Tai Chan. 2019. An analysis of segmentation approaches and window sizes in wearable-based critical fall detection systems with machine learning models. *IEEE Sensors Journal* 20, 6 (2019), 3303–3313.

[59] Sebastian OH Madgwick, Andrew JL Harrison, and Ravi Vaidyanathan. 2011. Estimation of IMU and MARG orientation using a gradient descent algorithm. In *2011 IEEE international conference on rehabilitation robotics*. IEEE, 1–7.

[60] Swati Mandekar, Lina Jentsch, Dr Kai Lutz, Dr Mehdi Behbahani, and Mark Melnykowycz. 2021. Earable Design Analysis for Sleep EEG Measurements. In *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*. 171–175.

[61] Daniela Micucci, Marco Mobilio, and Paolo Napoletano. 2017. Unimib shar: A dataset for human activity recognition using acceleration data from smartphones. *Applied Sciences* 7, 10 (2017), 1101.

[62] David L Mills. 1991. Internet time synchronization: the network time protocol. *IEEE Transactions on communications* 39, 10 (1991), 1482–1493.

[63] Chulhong Min, Akhil Mathur, and Fahim Kawsar. 2018. Audio-Kinetic Model for Automatic Dietary Monitoring with Earable Devices. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 517–517.

[64] Chulhong Min, Akhil Mathur, and Fahim Kawsar. 2018. Exploring audio and kinetic sensing on earable devices. In *Proceedings of the 4th ACM Workshop on Wearable Systems and Applications*. 5–10.

[65] Arshad Nasser, Kening Zhu, and Sarah Wiseman. 2019. Thermo-haptic earable display for the hearing and visually impaired. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*. 630–632.

[66] Shahriar Nirjon, Robert F Dickerson, Qiang Li, Philip Asare, John A Stankovic, Dezhi Hong, Ben Zhang, Xiaofan Jiang, Guobin Shen, and Feng Zhao. 2012. Musicalheart: A hearty way of listening to music. In *Proceedings of the 10th ACM Conference on Embedded*

Network Sensor Systems. 43–56.

[67] Henry Odoemelem, Alexander Hölzemann, and Kristof Van Laerhoven. 2019. Using the eSense wearable earbud as a light-weight robot arm controller. In *Proceedings of the 1st International Workshop on Earable Computing.* 26–29.

[68] Francisco Javier Ordóñez and Daniel Roggen. 2016. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors* 16, 1 (2016), 115.

[69] Hiroki Ota, Minghan Chao, Yuji Gao, Eric Wu, Li-Chia Tai, Kevin Chen, Yasutomo Matsuoka, Kosuke Iwai, Hossain M Fahad, Wei Gao, et al. 2017. 3d Printed "Earable" Smart Devices for Real-time Detection of Core Body Temperature. *ACS sensors* 2, 7 (2017), 990–997.

[70] Parvaneh Parhizkari. 2008. *Binaural Hearing Human Ability of Sound Source Localization.* Master's thesis. Blekinge Institute of Technology.

[71] Nuno Pereira, Anthony Rowe, Michael Farb, Ivan Liang, Edward Lu, and Eric Riebling. 2021. ARENA: The Augmented Reality Edge Networking Architecture. In *2021 IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR'21)* (Bari, Italy). IEEE/ACM.

[72] Felix Pfreundtner, Jing Yang, and Gábor Sörös. 2021. (W) Earable Microphone Array and Ultrasonic Echo Localization for Coarse Indoor Environment Mapping. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 4475–4479.

[73] Joseph Plazak and Marta Kersten-Oertel. 2018. A Survey on the Affordances of "Hearables". *Inventions* 3, 3 (2018), 48.

[74] Jay Prakash, Zhijian Yang, Yu-Lin Wei, and Romit Roy Choudhury. 2019. Stear: Robust step counting from earables. In *Proceedings of the 1st International Workshop on Earable Computing.* 36–41.

[75] Meera Radhakrishnan and Archan Misra. 2019. Can earables support effective user engagement during weight-based gym exercises?. In *Proceedings of the 1st International Workshop on Earable Computing.* 42–47.

[76] Meera Radhakrishnan, Kushaan Misra, and V Ravichandran. 2021. Applying "Earable" Inertial Sensing for Real-time Head Posture Detection. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops).* IEEE, 176–181.

[77] Meera Radhakrishnan, Darshana Rathnayake, Ong Koon Han, Inseok Hwang, and Archan Misra. 2020. ERICA: enabling real-time mistake detection & corrective feedback for free-weights exercises. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems.* 558–571.

[78] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2021. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–34.

[79] Tobias Röddiger, Michael Beigl, and Anja Exler. 2020. Design space and usability of earable prototyping. In *Proceedings of the 2020 International Symposium on Wearable Computers.* 73–78.

[80] Tobias Röddiger, Daniel Wolffram, David Laubenstein, Matthias Budde, and Michael Beigl. 2019. Towards respiration rate monitoring using an in-ear headphone inertial measurement unit. In *Proceedings of the 1st International Workshop on Earable Computing.* 48–53.

[81] Charissa Ann Ronao and Sung-Bae Cho. 2016. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert systems with applications* 59 (2016), 235–244.

[82] Swapnil Sayan Saha, Shafizur Rahman, Miftahul Jannat Rasna, Tahera Hossain, Sozo Inoue, and Md Atiqur Rahman Ahad. 2018. Supervised and neural classifiers for locomotion analysis. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers.* 1563–1570.

[83] Swapnil Sayan Saha, Shafizur Rahman, Miftahul Jannat Rasna, AKM Mahfuzul Islam, and Md Atiqur Rahman Ahad. 2018. DU-MD: An open-source human action dataset for ubiquitous wearable sensors. In *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR).* IEEE, 567–572.

[84] Swapnil Sayan Saha, Sandeep Singh Sandha, and Mani Srivastava. 2021. Deep Convolutional Bidirectional LSTM for Complex Activity Recognition with Missing Data. In *Human Activity Recognition Challenge.* Springer, 39–53.

[85] Ramon Sanchez-Iborra and Antonio F Skarmeta. 2020. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine* 20, 3 (2020), 4–18.

[86] Sandeep Singh Sandha, Mohit Aggarwal, Igor Fedorov, and Mani Srivastava. 2020. Mango: A Python Library for Parallel Hyperparameter Tuning. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 3987–3991.

[87] Sandeep Singh Sandha, Mohit Aggarwal, Swapnil Sayan Saha, and Mani Srivastava. 2021. Enabling Hyperparameter Tuning of Machine Learning Classifiers in Production. In *2021 IEEE Third Intl' Conf. on Cognitive Machine Intelligence (CogMI).* IEEE, 1–10.

[88] Sandeep Singh Sandha, Joseph Noor, Fatima M Anwar, and Mani Srivastava. 2019. Exploiting smartphone peripherals for precise time synchronization. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP).* IEEE, 1–6.

[89] Sandeep Singh Sandha, Joseph Noor, Fatima M Anwar, and Mani Srivastava. 2020. Time awareness in deep learning-based multimodal fusion across smartphone platforms. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI).* IEEE, 149–156.

[90] Angela Sucerquia, José David López, and Jesús Francisco Vargas-Bonilla. 2017. SisFall: A fall and movement dataset. *Sensors* 17, 1 (2017), 198.

[91] Wei Sun, Franklin Mingzhe Li, Benjamin Steeper, Songlin Xu, Feng Tian, and Cheng Zhang. 2021. TeethTap: Recognizing Discrete Teeth Gestures Using Motion and Acoustic Sensing on an Earpiece. In *26th International Conference on Intelligent User Interfaces*. 161–169.

[92] Yushi Takayama, Shun Ishii, Anna Yokokubo, and Guillaume Lopez. 2021. Detecting forward leaning posture using eSense and developing a posture improvement promoting system. In *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*. 178–179.

[93] Kazuhiro Taniguchi, Hisashi Kondo, Mami Kurosawa, and Atsushi Nishikawa. 2018. Earable TEMPO: a novel, hands-free input device that uses the movement of the tongue measured with a wearable ear sensor. *Sensors* 18, 3 (2018), 733.

[94] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. In *9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*.

[95] George Vavoulas, Charikleia Chatzaki, Thodoris Malliotakis, Matthew Pediaditis, and Manolis Tsiknakis. 2016. The mobiact dataset: Recognition of activities of daily living using smartphones. In *International Conference on Information and Communication Technologies for Ageing Well and e-Health*, Vol. 2. SCITEPRESS, 143–151.

[96] Dhruv Verma, Sejal Bhalla, Dhruv Sahnan, Jainendra Shukla, and Aman Parnami. 2021. ExpressEar: Sensing Fine-Grained Facial Expressions with Earables. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 3 (2021), 1–28.

[97] Gaojing Wang, Qingquan Li, Lei Wang, Wei Wang, Mengqi Wu, and Tao Liu. 2018. Impact of sliding window length in indoor human motion modes and pose pattern recognition based on smartphone sensors. *Sensors* 18, 6 (2018), 1965.

[98] Xue Wang and Yang Zhang. 2021. Nod to Auth: Fluent AR/VR Authentication with User Head-Neck Modeling. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–7.

[99] Gary M Weiss, Jessica L Timko, Catherine M Gallagher, Kenichi Yoneda, and Andrew J Schreiber. 2016. Smartwatch-based activity recognition: A machine learning approach. In *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*. IEEE, 426–429.

[100] Seong-hoon Peter Won and Farid Golnaraghi. 2009. A triaxial accelerometer calibration method using a mathematical model. *IEEE transactions on instrumentation and measurement* 59, 8 (2009), 2144–2153.

[101] Tao Xu, Yun Zhou, and Jing Zhu. 2018. New advances and challenges of fall detection systems: A survey. *Applied Sciences* 8, 3 (2018), 418.

[102] Zhijian Yang, Yu-Lin Wei, Sheng Shen, and Romit Roy Choudhury. 2020. Ear-ar: indoor acoustic augmented reality on earphones. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[103] Clint Zeagler. 2017. Where to wear it: functional, technical, and social considerations in on-body location for wearable technology 20 years of designing for wearability. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers*. 150–157.

[104] Mi Zhang and Alexander A Sawchuk. 2012. USC-HAD: a daily activity dataset for ubiquitous activity recognition using wearable sensors. In *Proceedings of the 2012 ACM conference on ubiquitous computing*. 1036–1043.