On-Sensor Online Learning and Classification Under 8 KB Memory

Mahesh Chowdhary
STMicroelectronics
Santa Clara, CA, USA
mahesh.chowdhary@st.com

Swapnil Sayan Saha
University of California, Los Angeles
Los Angeles, CA, USA
swapnilsayan@g.ucla.edu

***Abstract*—Inertial sensors provide a low-power and high-fidelity pathway for state estimation and sensor fusion. Inertial measurement units now feature on-chip processors for ultra-low-power information fusion, signal processing, and neural network-based classification at the extreme edge. However, accounting for domain shifts, personalized inference requirements, and application diversity makes adopting existing learning-enabled on-device training, classification, and fusion frameworks for on-sensor processors difficult. This paper introduces a method for personalized and on-device learning for on-chip classification, inference, and information fusion applications. The proposed framework automatically segments and stores quantized gravity vector image templates and axes variance information of motion artifacts during training. During inference, templates created from the time-series windows are matched against uniform blurred templates using the universal image quality index. An adaptive rep counting module robust to varying motion primitives counts repetitions of matched motion primitives. The framework requires no human-engineered parameters and allows for the personalization and addition of new motion artifacts. Our framework recognizes human activities with 96.7% test accuracy and achieves an average rep count error of 0.44, while reducing the memory usage by 1000-2000× over existing tiny machine learning on-device learning techniques, allowing on-sensor learning and inference under 8 KB of memory.**

***Index Terms*—intelligent sensor processing unit, on-device learning, tinyML, inertial measurement unit, classification, rep counting, inference, template matching, segmentation**

I. Introduction

Inertial measurement units (IMU) featuring a triaxial accelerometer, a triaxial gyroscope, and a triaxial compass on a single chip are ubiquitous in smartphones, wearables, vehicles, robots, home appliances, and industrial machinery [1]. MEMS inertial sensors have a broad usage spectrum thanks to their low footprint, ultra-low-power usage, and high-resolution data streaming [2]. Notable applications include orientation estimation, gesture recognition, gait analysis, workout tracking, fall detection, activity recognition, pose estimation, behavioral analysis, inertial navigation, and anomaly detection [3]–[9].

Advancements in machine learning (ML) have paved the way for performantly realizing most of the aforementioned applications [10]–[15]. ML allows rich, robust, and complex inferences to be made from unstructured data over first principles-based approaches [16]. Applications such as unmanned vehicle navigation, virtual reality head pose estimation, biologging, robot joint state estimation, and exercise monitoring demand intelligent yet lightweight (due to device size, compute, and power constraints) and real-time decision-making capabilities [17]–[22]. To enable artificial intelligence (AI) for time-critical and remote applications, tiny machine learning (tinyML) provides hardware and software paradigms that enable always-on, real-time, low-cost, and ultra-low-power inference on microcontrollers [23]–[26]. To achieve even lower power envelope, lower latency, and smaller footprint, IMU manufacturers now integrate custom processing cores directly in the sensor die, eliminating the need for general-purpose microcontrollers for data analytics [1,2,3,4]. These integrated processing units provide instructions for on-chip sensor fusion (e.g., Kalman filtering and absolute orientation), signal conditioning (e.g., Fourier transform and feature extraction), and making ML-based inference (e.g., decision trees, finite state machines, and neural networks) at the extreme edge [27], [28].

A. Challenges of Deploying Motion Classification

Motion classification algorithms deployed in the wild require the ability to handle domain shifts in incoming data distribution, fit customer inference classes, and adapt to varying application scenarios [23], [36]–[38]. Unfortunately, adopting conventional ML-based motion classifiers for on-sensor motion recognition is challenging due to four shortcomings.

The Compute Constraints of Integrated Processors: A microcontroller used for tinyML applications typically has 1-2 MB of SRAM and flash [23], while an on-sensor processor typically has 8 kB of SRAM and 32 kB of volatile data memory (no flash) [5]. Furthermore, the architecture of on-sensor processors (e.g., STRED) is different from general-purpose microcontrollers (e.g., ARM Cortex M4), making tinyML compiler suites incompatible for use on-sensor.

On-sensor Domain Adaptation: The same motion primitive can have multiple feature distributions across different users [36], [39]–[41]. On-device fine-tuning and personalization are necessary for achieving robust algorithmic performance [23], preventing the adoption of static heuristics. How-

[1]https://www.st.com/en/mems-and-sensors/lsm6dso16is.html
[2]https://www.st.com/en/mems-and-sensors/lsm6dsox.html
[3]https://www.bosch-sensortec.com/products/smart-sensor-systems/bhi380/
[4]https://www.bosch-sensortec.com/products/smart-sensors/bha260ab/
[5]intelligent sensor processing unit (ISPU) from STMicroelectronics [27]

TABLE I
EXISTING TINYML ON-DEVICE LEARNING TECHNIQUES VERSUS PROPOSED ALGORITHM

Method	Framework	Supported Hardware*	Application	Personalizing Output Classes	Automatic Learning
Transfer learning	LITW [29]	TI MSP430 (66 kB)	Image recognition	No (static CNN)	Per-output feature distribution divergence
	TinyOL [30]	ARM Cortex-M (256 kB)	Inertial anomaly detection	No (static autoencoder)	Running mean and variance of streaming input
	TinyTL [31]	ARM Cortex-A (32-66 MB)	Image recognition	No (static MBNet)	None
Incremental training	Train++ [32]	ARM Cortex-M, ARM Cortex-A, Espressif ESP32, Xtensa LX (6 kB - 860 kB)	Image recognition, mHealth	No (static binary classifiers)	Confidence score of prediction
Optimized backpropagation	ML-MCU [33]	ARM Cortex-M, Espressif ESP32 (6kB - 241 kB)	Image recognition, mHealth	No (static binary classifiers)	None
	TTE [34]	ARM Cortex-M (256 kB)	Image recognition	No (static MBNet, MCUNet)	None
Continual learning	QLR-CL [35]	PULP (64 MB)	Image recognition	No (static MBNetV1)	None
Template matching (ours)	This work	Any MCU, ISPU (8 kB)	Inertial motion recognition	Yes (template modification)	Running mean and variance of streaming input

∗ Parenthesis shows working memory (SRAM) usage

ever, constrained learning theories and fixed resource budgets make online learning difficult to deploy on microcontrollers, let alone integrated processors [23], [42].

Personalizing Output Classes: Each customer may have specific inference requirements depending on the application. For example, customer A may want to perform motion recognition for full-body fitness monitoring. Customer B may want a solution for anomaly detection of fan blades. Customer C may want to detect hand gestures. The diversity and variability in required motion primitives among users prevent the adoption of static ML classifiers, which can only detect the classes present in the training dataset.

Automatic Segmentation: During fine-tuning, the incoming data stream must be properly segmented to store only the portion of data concerning the event of interest. The customer should not be burdened with the task of selecting the start and end times of the event of interest. In addition, the segmentation algorithm should fit within the compute budget of the on-sensor processor [43]–[45].

B. Contributions

In this article, we introduce an ultra-lightweight, application-agnostic, and on-device learning and inference algorithm for on-sensor motion recognition. During the *learning* phase, the user performs the target event of interest with the IMU mounted on the target device. Our pipeline uses an inexpensive event segmentation method that can automatically identify the start and end points of the target motion primitive during the training phase. The segmented data stream is then converted to an image template. The template and axes variance information is stored on the sensor. During *inference*, image templates are created in real-time from IMU data windows. These templates are matched against stored templates using image similarity and axes variance metrics to provide the detected class label. A rep counter with an adaptive hyperparameter accurately counts the number of repetitions of matched classes. Customers can replace or remove stored classes with new motion primitives on-the-fly, allowing personalization and tuning. The pipeline is fully automated and requires no user-supplied parameters. The whole algorithm fits within the tight memory bounds of an intelligent sensor processing unit (ISPU) from STMicroelectronics [27], consuming less than 8 kB of memory, achieving 1000-2000× memory savings over competing techniques designed for microcontrollers. For 6-class exercise activity recognition, our algorithm achieves 96.7% test accuracy and a mean rep count error of 0.44. The algorithm generalizes to a wide variety of motion recognition applications, including gesture detection, full-body human activity recognition, transportation mode recognition, fall detection, head pose estimation, and anomaly detection.

II. BACKGROUND AND RELATED WORK

The proliferation of IMU in the consumer, industrial, and military domains for pervasive sensing has motivated manufacturers to embed AI and functional programs directly on the sensor circuit board, converting inertial IoT platforms from simple data streamers to frugal smart objects [23], [36], [50]. Executing smart algorithms at the extreme edge allows inertial sensors to be deployed for time-critical, power-constrained, and remote applications, where reliable communication with the cloud may not be available [55]. However, due to the constrained nature of inertial IoT platforms, on-device programs and ML models for motion recognition raises challenges of online domain adaptation, personalization, and automatic segmentation. We highlight the state-of-the-art approaches for on-device learning (Section II-A) and motion recognition (Section II-B) on IoT platforms.

A. Online Learning on IoT Platforms

Table I summarizes online learning techniques developed for low-end IoT platforms. Existing on-device learning paradigms (**how to learn**) for ML models can be classified into four classes. *Transfer learning* fine-tunes the last few layers of a frozen network graph [56] by reusing feedforward propagation representation maps for backpropagation [29], sample-wise stochastic gradient descent [30], and performing bias updates via lite residual learning [31]. Transfer learning suffers from catastrophic forgetting and limited capacity of unfrozen layers [31]. *Incremental training* performs sample-wise and gradient-free weight updates using constrained optimization [32], suffering from limited supported model types. *Optimized backpropagation* algorithms reduce training memory usage through sparse updates of the most important network layers [34], compile-time gradient calculation [34], and combining the stability of gradient descent with the efficiency of stochastic gradient descent [33]. On-device auto differentiation has a higher learning capacity than transfer learning, but limits online learning to either limited model types or processors. *Continual learning* uses slow learning and a latent replay layer to store feature maps from previous training samples, preventing catastrophic forgetting [35]. However, continual learning

TABLE II
EXISTING IOT INERTIAL MOTION RECOGNITION FRAMEWORKS VERSUS PROPOSED ALGORITHM

Framework	SRAM Usage	Online Learning	Segmentation	Personalized Output Classes	Application	Rep Counting	No Tuning Needed
Machine Learning-Based Approaches							
MiLift [43]	$<$ 512 MB on Moto 360	SVM confidence scores	Hierarchical ML pipeline	No (static CRF, DT+HMM, and SVM)	Workout tracking	Revisit-Based	No
GesturePod [46]	$\sim$2 kB on Cortex-M0+	None	None	No (static ProtoNN)	Gesture recognition	None	No
AURITUS [36]	$\sim$2 kB on Cortex MCU	None	None	No (static FastGRNN, Bonsai, ProtoNN)	Activity recognition	None	No
FastGRNN [47]	$\sim$2 kB on AVR RISC	None	None	No (static FastGRNN)	Activity recognition	None	No
TinyOL [30]	$<$ 256 kB on Cortex-M4	Transfer Learning	Running mean and variance	No (static autoencoder)	Anomaly detection	None	No
Coelho *et al.* DT [48]	$<$ 128 kB on Cortex-M4	None	None	No (static DT)	Activity recognition	None	No
Active Learning [49]	$<$ 512 MB on Gear Live	Active learning (querying)	None	No (static RF, DT, NB, SVM)	Activity recognition	None	No
Elsts *et al.* CNN [50]	$<$ 96 kB on Cortex-M3	None	None	No (static CNN)	Activity recognition	None	No
T'Jonck *et al.* CNN [51]	$<$ 256 kB on Cortex-M4	None	None	No (static CNN)	Bed activity, anomaly	None	No
Finite State Machines (FSM) and Functional Programs							
FSM Sequence [52]	$<$ 3 GB on Core 2 Duo	Sequence matching	Smoothing	Yes (Fast learning FSM)	Gesture recognition	None	Yes
FnSM [53]	$<$ 512 MB on Sony Watch 3	None	None	No (static functional FSM)	Gesture recognition	None	No
Bosch GDL [54]	$<$ 44 kB on integrated processor	GDL evaluation and regex	(Supported)	Yes (GDL + Non-deterministic FSM)	Gesture recognition	(Supported)	Yes
Template Matching							
Ours	$<$8 kB on ISPU	Template-based	Running mean and variance	Yes (template modification)	All of the above	Adaptive	Yes

has high resource usage. To detect feature distribution shift in the streaming data and start the training process automatically (**when to learn**), the frameworks monitor the moments in the input data window [30], the model prediction confidence scores [32], or the divergence in the covariate distribution of principal components in the input data window [29].

Deploying existing online learning frameworks in the wild raises two issues. *Firstly*, these techniques operate on a static ML classifier pre-trained on an application-specific dataset. Thus, the customer cannot add, replace, or remove additional output classes from the model, lacking personalization capabilities. These techniques also fail to adapt to significantly different tasks and data distributions. In contrast, our technique assumes no prior information about the customer application or data distribution, allowing the user to create motion templates in the field with only 10 seconds of IMU data per class. Training data segmentation is performed automatically by observing the input stream moments. *Secondly*, existing online learning techniques are not suitable for execution on on-sensor processors, usually requiring around 256 kB of SRAM. These frameworks also support limited processor architectures depending upon the tinyML compiler suite and vector accelerators. In comparison, our method can run on any commodity microcontroller and even on-sensor processors supporting C instructions, requiring less than 8 kB of SRAM.

B. Motion Recognition on IoT Platforms

Table II showcases inertial motion recognition frameworks designed to be run in real-time on IoT platforms. These frameworks can be divided into three categories.

Machine Learning-Based Approaches: These frameworks run static ML classifiers to make application-specific inferences. Specially-designed classifiers with low-rank, sparse, shallow, and quantized parameters (e.g., Bonsai, ProtoNN, and FastGRNN) [47], [57], [58] allow motion recognition under 2 kB of SRAM [36], [46], [47] at the cost of domain generalizability [36]. Vanilla convolutional neural networks (CNN), decision trees (DT), and autoencoders can either be hand-tuned or optimized using neural architecture search for making inferences under 256 kB SRAM [30], [48], [50], [51]. CNN and autoencoders also exploit post-training quantization and weight pruning to lower latency and memory usage even further by 6-9$\times$ and 9-14$\times$ [23], [59]. These large sparse models are more robust to domain shifts than small dense models [36]. IoT devices with more relaxed memory constraints (e.g., smartwatches) feature conventional ML classifiers such as conditional random fields (CRF), random forests (RF), support vector machines (SVM), naive Bayes (NB), and DT [43], [49]. These devices feature online learning in the form of binary classification (in-class or not) through the use of SVM confidence scores or ensemble voting [43], [49]. Unfortunately, all ML-based approaches are task-specific, lack the ability to add, remove or modify output classes, and require user supervision (data processing) during training.

State Machines and Functional Programs: Finite state machines (FSM) are computational models that can take one of a finite number of states at a time and simulate sequential logic, regex pattern matching, and directed graphs [60]. A functional program is composed of a sequence of subroutines (expression trees) [61]. These approaches construct motion expressions from an ordered sequence of atoms, which are repetitions of symbols (predicates over raw data), described using a context-free grammar [54]. Regex matching detects new motion primitives by computing similarity scores with sample sequences. The onboard FSM is updated to add the new motion primitive using the optimal event sequence [52], [54]. Smoothing removes noise and unwanted motion artifacts [52]. While these techniques need no manual parameter tuning, allow modification of classes, are task-agnostic, and support online learning, the accuracy of these frameworks is around 75% [52], [53], while ML-based approaches easily exceed 95% [36], [43], [46]–[48], [51], [62].

Template Matching: Template matching converts time-series data into image templates and stores them onboard. Images are created on-the-fly from the IMU data stream and matched against stored templates using a similarity metric. This approach enjoys the accuracy of ML-based approaches while allowing on-device learning and personalization of output classes similar to state machines and functional programs. While the analysis of time-series data using image representation and template matching is well-explored [63]–[68], the end-to-end integration of inertial template matching with on-device learning, automatic segmentation, rep counting, and personalizability on on-sensor processors is unexplored.

III. Intelligent Sensor Processing Unit

The ISPU is an ultra-low-power (μW envelope) 32-bit RISC processor integrated in the LSM6DSO16IS and ISM330IS [6] 6DoF sensor ASIC. We describe the ISPU hardware architecture (Section III-A) and software organization (Section III-B).

A. ISPU Architecture

Fig. 1 shows the hardware architecture of the smart sensor which hosts the ISPU core. The *sensor core* features an accelerometer, a gyroscope, and a temperature sensor. The sensor core talks to an external microcontroller via I2C or SPI interfaces and controls the sensitivity and output data rate (ODR) of the inertial sensors. The sensor hub is responsible for data collection from additional sensors. The interface registers in the *processing core* receive sensor data from the sensor core and commands from the microcontroller. It contains registers for ISPU configuration, sending signals to and from the ISPU, and writing processed output. The ISPU core is an 8-kilogate Harvard RISC STRED processor running at 5 MHz or 10 MHz. The core has a floating point unit, a binary convolutional accelerator, 4 pipelining stages, 2 branch shadows, and a CoreMark®score of 70μW/MHz. The processing core features 8 kB of data memory and 32 kB of volatile program memory. The IMU package consumes 0.6-1.2 mA @ 1.8V, 5-10 MHz during normal operation. In contrast, a Cortex-M0 core consumes 1.6-2.6 mA @ 1.8V, 4-8 MHz. The Cortex-M0 core also consumes 5$\times$ more current than the ISPU for sensor fusion. The IMU is packaged in LGA-14.

B. ISPU Software Organization

The STRED core is an extended RISC architecture with support for vectorization, digital signal processing instructions, and the use of operands directly from memory. Programs are written in standard C code to allow code reuse and flexibility and compiled to machine code using the ISPU toolchain [7] or X-Cube-ISPU [8]. NanoEdge AI Studio [9] allows porting ML models from popular ML frameworks such as TensorFlow [69], Keras [70], QKeras [71], and Scikit-Learn [72] to the ISPU. 28 instructions containing move, arithmetic, floating, shift, logical, branch, and special operations are available. Since the program memory is non-volatile, the on-sensor program is loaded on the ISPU during power-up by the external microcontroller. The ISPU can only use the 8 kB data memory for storing variable data. Each processing cycle consists of the ISPU waking up when new data is available, running the onboard program within the ODR or IQR, and going back to sleep. Through the use of interrupt requests and an interrupt vector table, the ISPU can execute 30 separate algorithms running concurrently. The ISPU can wake up the external microcontroller when processing is completed through hardware interrupt pins, saving energy by preventing polling of the ISPU output registers.

[6]https://www.st.com/en/mems-and-sensors/ism330is.html
[7]https://www.st.com/en/development-tools/ispu-toolchain.html

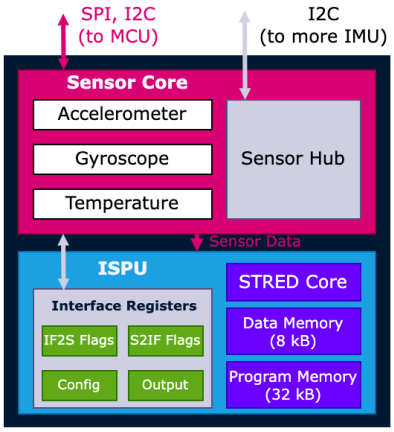

Fig. 1. Hardware architecture of the LSM6DSO16IS and ISM330IS.

IV. On-Sensor Learning and Classification

Keeping the ISPU characteristics and the shortcomings of existing IoT inertial motion recognition frameworks (Table II), we have designed an on-device inertial motion learning and classification framework for execution on-sensor. We discuss the automatic segmentation and learning process (Section IV-A), the inference and personalization pipeline (Section IV-B), and the adaptive rep counting module (Section IV-C).

A. Unsupervised Segmentation and Online Learning

The training process follows data segmentation, template creation, and storage.

Automatic Segmentation: When the user starts the training phase, accelerometer samples a_t are stored in a buffer a of size N. The user is asked to remain static for n seconds, monitored using the sum of the rolling mean ${}_R\bar{a}_t$ and the rolling variance ${}_R\sigma^2_{a_t}$ for all three accelerometer axes. ${}_R\sigma^2_{a_t}$ is as follows:

$$ {}_R\sigma^2_{a_t} = (1-\alpha)\cdot(\sigma^2_{a_t} + \alpha \cdot d_t^2) \tag{1} $$

where,

$$ d_t = \sqrt{a_{x,t}^2 + a_{y,t}^2 + a_{z,t}^2} - {}_R\,\bar{a}_t \tag{2} $$

$$ {}_R\bar{a}_t = {}_R\,\bar{a}_t + (\alpha \cdot d_t),\ {}_R\bar{a}_0 = \sqrt{a_{x,0}^2 + a_{y,0}^2 + a_{z,0}^2} \tag{3} $$

α is a tunable parameter controlling the smoothness of the running moments. Static condition is detected when the sum of rolling mean and variances is less than a pre-defined threshold β. To tune α and β, we used Bayesian optimization [73], with the objective function minimizing the difference between start and end times predicted by the segmentation algorithm versus human-picked points for a task-specific dataset. If a stationary state of n seconds is detected, then the data collection for the motion primitive to be detected starts, otherwise, the training phase is canceled. The user performs the desired motion primitive for p seconds (until the buffer is full). Running

[8]https://www.st.com/en/embedded-software/x-cube-ispu.html
[9]https://www.st.com/en/development-tools/nanoedgeaistudio.html

moments are used to discard the small number of samples that elapsed between the time the user was asked to perform the motion primitive and when the user actually started the motion primitive. If the number of static samples during this phase exceeds $z\%$ of the buffer length, the training is canceled. This "static-motion" based segmentation algorithm is computationally efficient over autocorrelation, time-warping, revisit-based, or ML-based segmentation approaches [43], [74].

Template Creation: First, the roll ϕ and pitch θ are calculated from the stored buffer a:

$$\begin{cases} \phi_t = \arctan 2(a_{y,t}, a_{z,t}) \\ \theta_t = \arcsin \frac{a_{x,t}}{\sqrt{a_{x,t}^2 + a_{y,t}^2 + a_{z,t}^2}} \end{cases} \tag{4}$$

Next, the gravity vector swing g in each axis is calculated:

$$\begin{cases} g_{x,t} = \sin \theta_t \\ g_{y,t} = \cos \theta_t \sin \phi_t \\ g_{z,t} = \cos \theta_t \cos \phi_t \end{cases} \tag{5}$$

The variance of g is calculated for each axis:

$$\sigma_g^2 = \frac{1}{N} \sum_t (g_t - \bar{g})^2 \tag{6}$$

The two axes with the highest variance, g_a and g_b, are noted to retain maximal information (principal components). Afterward, an $m \times m$ byte grid is created. The numerical values in g_a and g_b are quantized into buckets as follows:

$$\begin{aligned} g_{u,t}^{\text{quantized}} &= 0.5\Big(\Delta \cdot i + \Delta \cdot (i-1)\Big) \\ \Leftrightarrow \Big(g_{u,t} < \Delta \cdot i\Big) &\wedge \Big(g_{u,t} > \Delta \cdot i - 1\Big) \end{aligned} \tag{7}$$

where, $i \in [1, m]$, $u = \{a, b\}$. Each element in the grid takes a value of $g_{u,t}^{\text{quantized}}$. Δ is the quantization resolution, which can be calculated from either of the two axes a or b:

$$\Delta = \frac{\max(g_v) - \min(g_v)}{m - 1}, \; v = \{a, b\} \tag{8}$$

The quantized values are used to fill up the elements of the byte grid, with the maximum value of a grid element equal to 255. The result is an $m \times m$ quantized image of g_a and g_b plotted against each other. Fig. 2 shows 20×20 templates created for three different exercise activities in real-time, namely bicep curl, lateral, and jack.

The motivation to convert accelerometer time series to 2D image templates are twofold. *Firstly,* the gravity vector creates a spatial view of the motion primitive by overlapping the accelerometer samples with the same gravity vector swing. It gets rid of the computational expense and algorithmic challenge to time align and then overlap consecutive motion primitive sequences for a certain motion primitive. *Secondly*, 2D images provide more information (variance) corresponding to real-world applications than 1D principal components. Most real-world motion sensing applications (e.g., fitness monitoring, activity recognition, and gesture detection) usually span two gravity vector axes.

Template Storage: The image and the name of the two axes with maximum variance are stored as an element in an array of structures in the data memory of the ISPU. Each element in the array of structures has 3 members: the quantized $m \times m$ byte template, a character specifying the axis with the highest variance, and a character specifying the axis with the second highest variance. Additionally, a counter maintains how many of the elements in the data structure have been used and is incremented by 1 when a new template is added.

B. Real-time Classification and Personalization

During inference, the buffer size is M, such that $M < N$. To promote variable reuse, we use the same buffer used during training. When the buffer has M elements, templates are created using the formulation described in Section IV-A. For each element in the array of structures, the two stored maximal axis variance names are matched with the two maximal axis variance names for the inference-time gravity vector swing. If a match is found, then the universal image quality index (UQI) [75] Q is calculated between the selected stored candidate image template e and the inference image f:

$$Q = \frac{4 \cdot \sigma_{ef} \cdot \bar{e} \cdot \bar{f}}{(\sigma_e^2 + \sigma_f^2) \cdot (\bar{e}^2 + \bar{f}^2)} \tag{9}$$

where

$$\bar{e} = \frac{1}{m^2} \sum_{i=1}^{m^2} e_i, \; \bar{f} = \frac{1}{m^2} \sum_{i=1}^{m^2} f_i \tag{10}$$

$$\sigma_e^2 = \frac{1}{m^2 - 1} \sum_{i=1}^{m^2} (e_i - \bar{e}), \; \sigma_f^2 = \frac{1}{m^2 - 1} \sum_{i=1}^{m^2} (f_i - \bar{f}) \tag{11}$$

$$\sigma_{e,f} = \frac{1}{m^2 - 1} \sum_{i=1}^{m^2} (e_i - \bar{e}) \cdot (f_i - \bar{f}) \tag{12}$$

Q varies between -1 and +1, with +1 signifying the two templates are identical, while -1 signifies the highest dissimilarity between the two templates. We perform the axis match first to save valuable computation time needed to calculate the UQI by eliminating stored image candidates whose axis does not align with the inference template. To handle domain shifts caused by noise, sensor placement offset, and inertial disturbances, we downsample e and f by applying uniform filtering:

$$w_{\text{blurred}} = w \; * \; J_k, \; w = \{e, f\} \tag{13}$$

$*$ is the convolution operator, performed only in the valid regions. J is the square unit matrix of length k, also known as the blur kernel. The idea is inspired by how a CNN successively downsamples an input image layer-by-layer by applying a convolution kernel to extract an abstract yet generalizable representation map [16].

For personalization, the onboard software provides the customer the option to replace or erase an existing template. The user can choose to replace an existing template with a new one, append new templates at the end of the array of structures (provided the maximum number of allowable templates has not been reached), or erase the contents of the whole array of structures. The training process described in Section IV-A is followed during this phase.

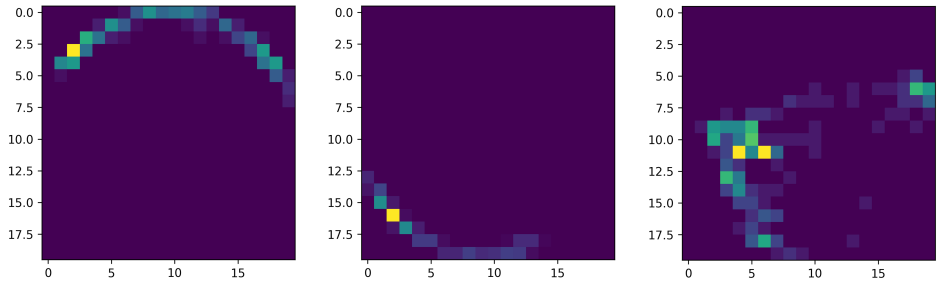

Fig. 2. Example templates generated by our algorithm for (Left) bicep curl (Center) lateral (Right) jack exercises.

C. Adaptive Repetition Counter

While the preceding algorithms can detect event labels, they cannot count the number of instances of the detected event in the current window. This may be necessary for applications such as fitness monitoring or gym activity detection. Thus, we developed a peak detector to count repetitions of detected events that can adapt to varying motion primitives. The peak detector identifies a point as a peak if it has the maximal value and was preceded to the left by a value lower by ∇. The peak detector operates on the accelerometer axis q which has the maximum variance:

$$\nabla = \frac{\text{percentile}(q, 0.95) - \min(q)}{2\sigma_q^2} \tag{14}$$

∇ is the adaptive hyperparameter, adapting the peak detection algorithm with different motion types. The number of peaks detected equals the number of repetitions of the detected activity in the current window. If the motion label in the current window matches the label in the previous window, the rep counter is incremented by the number of peaks detected in the current window. Otherwise, the rep counter is reset.

V. Evaluation

We evaluate the algorithm for workout activity recognition (Section V-A) and rep counting (Section V-B), quantifying resource usage for commodity microcontrollers and the ISPU (Section V-C). We use an internally collected IMU dataset containing 30 workout sessions from multiple volunteers spanning 6 classes, namely bicep curl, jack, lateral, overhead, push up, and squat exercises. During data collection, the participants wore the IMU on their wrist using a wristband containing a ST SensorTile.Box, and the device could be oriented in any orientation. The data was stored on an onboard SD card. For an initial evaluation, the test dataset was created by partitioning this dataset. However, we also did a real-world test with a different user set later. The sensor core ODR was 26 Hz. n was 3, α was 0.01, z was 50, m was 20, M was 52 (2 seconds), N was 260 (10 seconds), and k was 8. The array of structures was limited to 6 templates.

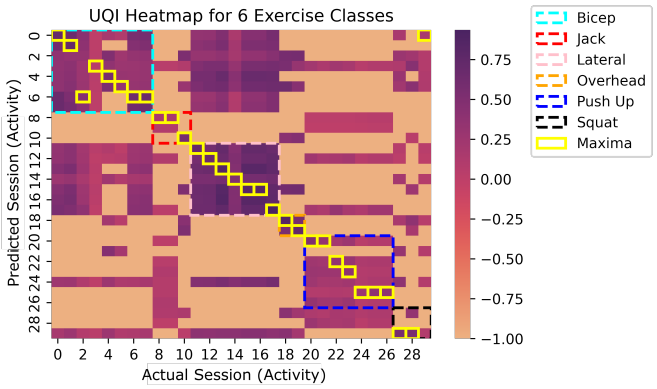

Fig. 3. UQI heatmap and confusion matrix for workout tracking

A. Workout Tracking Accuracy

Fig. 3 shows a heatmap showing detected UQI values and the confusion matrix for workout tracking for the 30 sessions across the 6 classes. Note that inference time aims to match the workout label among the 6 classes, not necessarily the instance or session. For example, if the algorithm detects an activity as 'squat' but the match stems from a 'squat' activity from a different workout session, then the inference is still valid. Our algorithm achieves a 96.7% workout tracking accuracy, only misclassifying an instance of squat with bicep curls. In other cases, the predicted class lies either on the diagonal (matched instance) or the in-class bounds (matched workout label). For the sessions corresponding to the same workout class (e.g., 0-7 corresponds to bicep curl and 8-10 corresponds to jack) the highest UQI is always within the in-class bounds, signifying robustness to in-class domain shifts of our algorithm. Without the uniform filter, our algorithm still achieves 90% + accuracy. Other similarity metrics such as the mean squared error, structural similarity index [76], spatial correlation coefficient [77], peak signal-to-noise ratio [76], and spectral distortion index [78] achieve 30-95% accuracy.

To ensure that the whole algorithm generalizes in real-world settings, we also performed a real-world study where the algorithm was ported to an ISPU within the ST SensorTile.Box. We asked ten volunteers not present in the original dataset to test the training and inference pipelines using their own choice of wristband orientation and activities. The algorithm successfully segmented the region of interest during training and accurately detected the user-defined activities in real time.

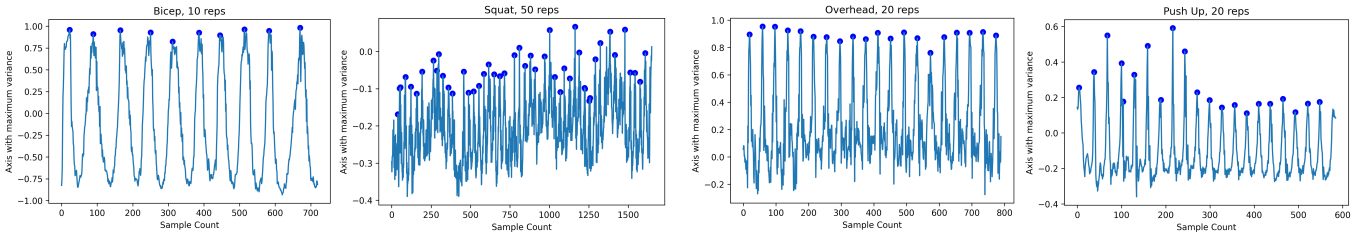

Fig. 4. Adaptive rep counter detecting peaks for various workout classes.

TABLE III
LATENCY AND RESOURCE USAGE OF OUR ALGORITHM

Algorithm Component	Latency (mS)	
	Cortex-M4 (84 MHz)	ISPU (10 MHz)
Template creation	4-6	32-48
Template storage	65	130
Template retrieval	0.19	1.55
Template matching	0.27	2.11
Rep counting	0.25	2

Memory Component	Memory Usage (kB)	
	Cortex-M4 (84 MHz)	ISPU (10 MHz)
Data	17.7	7.2
Program	45.0	24.0

B. Workout Rep Counting

Fig. 4 shows the rep counting performance of the adaptive rep counting for varying motion primitives. The blue dots signify detected rep positions. The rep counter automatically identifies the accelerometer axis with the highest variance and counts reps even when the amplitude or duration of the peaks vary. Moreover, the rep counter is robust to gaps between exercise events as observed in the squat exercise. Across the 30 sequences, the rep counter has an error of 0.44 reps, which is half the maximum tolerable rep count of 0.8 [43].

C. Resource Usage

Table III outlines the resource usage and latency of various components of our algorithm on a general-purpose Cortex-M4 microcontroller and the ISPU. Template retrieval, matching, and rep counting have negligible latency on both devices. Template creation takes 40 mS, which is roughly within the ISPU processing cycle for 26 Hz ODR. The algorithm consumes less than 8 kB memory on the ISPU, which is 2.5× lower than the microcontroller implementation, and 1000-2000× lower than existing online learning frameworks thanks to the optimized instruction set.

VI. CONCLUSION

In this article, we described an on-sensor inertial on-device learning and classification framework that consumes less than 8 kB memory. The framework is application-agnostic, allows personalization of output classes, supports online learning, performs automatic segmentation, and counts repetitions adaptively, all without any user supervision. The framework brings ultra-low-power intelligence closer to the extreme edge across a wide application spectrum. The solution (patent pending) was demoed at the Consumer Electronics Show 2023 [10]. Future directions include quantifying in-field performance for other applications, automated hyperparameter tuning, and optimizing complex mathematical operations.

[10]https://www.st.com/ces-2023

REFERENCES

[1] M. Nazarahari *et al.*, "40 years of sensor fusion for orientation tracking via magnetic and inertial measurement units: Methods, lessons learned, and future challenges," *Info. Fusion*, vol. 68, 2021.

[2] Y. Li *et al.*, "An in situ hand calibration method using a pseudo-observation scheme for low-end inertial measurement units," *Measurement Science and tech.*, vol. 23, no. 10, 2012.

[3] O. D. Lara *et al.*, "A survey on human activity recognition using wearable sensors," *IEEE comm. surveys & tutorials*, vol. 15, no. 3, 2012.

[4] M. Kok *et al.*, "Using inertial sensors for position and orientation estimation," *Foun. and Trends in Signal proc.*, vol. 11, no. 1-2, 2017.

[5] E. E. Cust *et al.*, "Machine and deep learning for sport-specific movement recognition: a systematic review of model development and performance," *J. of sports sciences*, vol. 37, no. 5, 2019.

[6] R. Harle, "A survey of indoor inertial positioning systems for pedestrians," *IEEE comm. Surveys & tut.*, vol. 15, no. 3, 2013.

[7] T. de Quadros *et al.*, "A movement decomposition and machine learning-based fall detection system using wrist wearable device," *IEEE Sensors J.*, vol. 18, no. 12, 2018.

[8] A. Avci *et al.*, "Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey," in *23rd Intl. Conf. on architecture of comp. sys. 2010*. VDE, 2010.

[9] M. Georgi *et al.*, "Recognizing hand and finger gestures with imu based motion and emg based muscle activity sensing." in *Biosignals*, 2015.

[10] N. Y. Hammerla *et al.*, "Deep, convolutional, and recurrent models for human activity recognition using wearables," in *Proc. of the Twenty-Fifth Intl. Joint Conf. on Artificial intel.*, 2016.

[11] Y. Li *et al.*, "Inertial sensing meets machine learning: opportunity or challenge?" *IEEE Tran. on Intelligent Trans. sys.*, 2021.

[12] D. Ravi *et al.*, "A deep learning approach to on-node sensor data analytics for mobile or wearable devices," *IEEE J. of biomed. and health informatics*, vol. 21, no. 1, 2016.

[13] A. Saboor *et al.*, "Latest research trends in gait analysis using wearable sensors and machine learning: A systematic review," *Ieee Access*, vol. 8, 2020.

[14] J. Wang *et al.*, "Deep learning for sensor-based activity recognition: A survey," *Pattern recognition let.*, vol. 119, 2019.

[15] C. Chen *et al.*, "Ionet: Learning to cure the curse of drift in inertial odometry," in *Proc. of the AAAI Conf. on Artificial intel.*, vol. 32, no. 1, 2018.

[16] I. Goodfellow *et al.*, *Deep learning*. MIT press, 2016.

[17] H. J. Williams *et al.*, "Optimizing the use of biologgers for movement ecology research," *J. of Animal Ecology*, vol. 89, no. 1, 2020.

[18] J. D. Hol *et al.*, "Sensor fusion for augmented reality," in *2006 9th Intl. Conf. on Info. Fusion*. IEEE, 2006.

[19] J. C. Kinsey *et al.*, "A survey of underwater vehicle navigation: Recent advances and new challenges," in *IFAC Conf. of manoeuvering and control of marine craft*, vol. 88. Lisbon, 2006.

[20] F. Kendoul, "Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems," *J. of Field Robo.*, vol. 29, no. 2, 2012.

[21] I. Skog *et al.*, "In-car positioning and navigation technologies—a survey," *IEEE Tran. on Intelligent Trans. sys.*, vol. 10, no. 1, 2009.

[22] A. Henriksen *et al.*, "Using fitness trackers and smartwatches to measure physical activity in research: analysis of consumer wrist-worn wearables," *J. of medical Inet. research*, vol. 20, no. 3, 2018.

[23] S. S. Saha *et al.*, "Machine learning for microcontroller-class hardware-a review," *IEEE Sensors J.*, 2022.

[24] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *J. of King Saud University-Computer and Info. Sciences*, 2021.

[25] L. Dutta *et al.*, "Tinyml meets iot: A comprehensive survey," *Inet. of Things*, vol. 16, 2021.

[26] R. Sanchez-Iborra *et al.*, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and sys. Magazine*, vol. 20, no. 3, 2020.

[27] A. Ronco *et al.*, "Machine learning in-sensors: Computation-enabled intelligent sensors for next generation of iot." in *2022 IEEE Sensors*. IEEE, 2022.

[28] M. Chowdhary *et al.*, "Reconfigurable sensor unit for electronic device," Nov. 27 2018, US Patent 10,142,789.

[29] S. Lee *et al.*, "Learning in the wild: When, how, and what to learn for on-device dataset adaptation," in *Proc. of the 2nd Intl. Workshop on Challenges in Artificial intel. and mach. learn. for Inet. of Things*, 2020.

[30] H. Ren *et al.*, "Tinyol: Tinyml with online-learning on microcontrollers," in *2021 Intl. joint Conf. on neural networks (IJCNN)*. IEEE, 2021.

[31] H. Cai *et al.*, "Tinytl: Reduce memory, not parameters for efficient on-device learning," *Advances in Neural Info. proc. sys.*, vol. 33, 2020.

[32] B. Sudharsan *et al.*, "Train++: An incremental ml model training algorithm to create self-learning iot devices," in *2021 IEEE SmartWorld, Ubi. intel. & comp., Advanced & Trusted comp., Scalable comp. & comm., Inet. of People and Smart City Innovation*. IEEE, 2021.

[33] B. Sudharsan *et al.*, "Ml-mcu: A framework to train ml classifiers on mcu-based iot edge devices," *IEEE Inet. of Things J.*, vol. 9, 2021.

[34] J. Lin *et al.*, "On-device training under 256kb memory," *Advances in Neural Info. proc. sys.*, vol. 35, 2022.

[35] L. Ravaglia *et al.*, "A tinyml platform for on-device continual learning with quantized latent replays," *IEEE J. on Emerging and Selected Topics in Circuits and sys.*, vol. 11, no. 4, 2021.

[36] S. S. Saha *et al.*, "Auritus: An open-source optimization toolkit for training and development of human movement models and filters using earables," *Proc. of the ACM on Interactive, Mobile, Wearable and Ubi. tech.*, vol. 6, no. 2, 2022.

[37] Y. Chang *et al.*, "A systematic study of unsupervised domain adaptation for robust human-activity recognition," *Proc. of the ACM on Interactive, Mobile, Wearable and Ubi. tech.*, vol. 4, no. 1, 2020.

[38] V. Bianchi *et al.*, "Iot wearable sensor and deep learning: An integrated approach for personalized human activity recognition in a smart home environment," *IEEE Inet. of Things J.*, vol. 6, no. 5, 2019.

[39] K. Chen *et al.*, "Deep learning for sensor-based human activity recognition: Overview, challenges, and opportunities," *ACM comp. Surveys (CSUR)*, vol. 54, no. 4, 2021.

[40] T. Gong *et al.*, "Metasense: few-shot adaptation to untrained conditions in deep mobile sensing," in *Proc. of the 17th Conf. on embedded networked sensor sys.*, 2019.

[41] A. Ferrari *et al.*, "On the personalization of classification models for human activity recognition," *IEEE Access*, vol. 8, 2020.

[42] S. Dhar *et al.*, "A survey of on-device machine learning: An algorithms and learning theory perspective," *ACM Tran. on Inet. of Things*, vol. 2, no. 3, 2021.

[43] C. Shen *et al.*, "Milift: Efficient smartwatch-based workout tracking using automatic segmentation," *IEEE Tran. on Mobile comp.*, vol. 17, no. 7, 2017.

[44] J. Wu *et al.*, "A wearable system for recognizing american sign language in real-time using imu and surface emg sensors," *IEEE J. of biomed. and health informatics*, vol. 20, no. 5, 2016.

[45] S. Kratz *et al.*, "Towards accurate automatic segmentation of imu-tracked motion gestures," in *Proc. of the 33rd Annual ACM Conf. Extended Abstracts on Human Factors in comp. sys.*, 2015.

[46] S. G. Patil *et al.*, "Gesturepod: Enabling on-device gesture-based interaction for white cane users," in *Proc. of the 32nd Annual ACM symp. on User Interface Software and tech.*, 2019.

[47] A. Kusupati *et al.*, "Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network," *Advances in neural Info. proc. sys.*, vol. 31, 2018.

[48] Y. L. Coelho *et al.*, "A lightweight framework for human activity recognition on wearable devices," *IEEE Sensors J.*, vol. 21, no. 21, 2021.

[49] F. Shahmohammadi *et al.*, "Smartwatch based activity recognition using active learning," in *2017 IEEE/ACM Intl. Conf. on Connected Health: app., sys. and eng. tech. (CHASE)*. IEEE, 2017.

[50] A. Elsts *et al.*, "Are microcontrollers ready for deep learning-based human activity recognition?" *Electronics*, vol. 10, no. 21, 2021.

[51] K. T'Jonck *et al.*, "Real-time activity tracking using tinyml to support elderly care," in *2021 XXX Intl. Scientific Conf. Elec.* IEEE, 2021.

[52] N. Ç. Kılıboz *et al.*, "A hand gesture recognition technique for human–computer interaction," *J. of Visual comm. and Image Rep.*, vol. 28, 2015.

[53] F. Waris *et al.*, "Using cultural algorithms to improve wearable device gesture recognition performance," in *2015 IEEE symp. series on computational intel.* IEEE, 2015.

[54] J.-P. Wolff *et al.*, "Towards automated prototyping of gesture recognition systems for wearable devices using inertial sensors," in *Smart sys. Integration; 13th Intl. Conf. and Exhibition on Integration Issues of Miniaturized sys.* VDE, 2019.

[55] C. Banbury *et al.*, "Mlperf tiny benchmark," *Advances in Neural Info. proc. sys.*, 2021.

[56] S. J. Pan *et al.*, "A survey on transfer learning," *IEEE Tran. on knowledge and data eng.*, vol. 22, no. 10, 2010.

[57] C. Gupta *et al.*, "Protonn: Compressed and accurate knn for resource-scarce devices," in *Intl. Conf. on mach. learn.* PMLR, 2017.

[58] A. Kumar *et al.*, "Resource-efficient machine learning in 2 kb ram for the internet of things," in *Intl. Conf. on mach. learn.*, 2017.

[59] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *Intl. Conf. on Learning Rep. (ICLR)*, 2016.

[60] E. Rich *et al.*, *Automata, computability and complexity: theory and app.* Pearson Prentice Hall Upper Saddle River, 2008.

[61] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM comp. Surveys (CSUR)*, vol. 21, no. 3, 1989.

[62] D. Morris *et al.*, "Recofit: using a wearable sensor to find, recognize, and count repetitive exercises," in *Proc. of the SIGCHI Conf. on Human Factors in comp. sys.*, 2014.

[63] S. Barra *et al.*, "Deep learning and time series-to-image encoding for financial forecasting," *IEEE/CAA J. of Auto. Sinica*, vol. 7, no. 3, 2020.

[64] N. Hatami *et al.*, "Classification of time-series images using deep convolutional neural networks," in *Tenth Intl. Conf. on mach. vision*, vol. 10696. SPIE, 2018.

[65] L. Ye *et al.*, "Time series shapelets: a new primitive for data mining," in *Proc. of the 15th ACM SIGKDD Intl. Conf. on Knowledge discovery and data mining*, 2009.

[66] B. Hartmann *et al.*, "Gesture recognition with inertial sensors and optimized dtw prototypes," in *2010 IEEE Intl. Conf. on sys., Man and Cybernetics*. IEEE, 2010.

[67] S. Jiang *et al.*, "Feasibility of wrist-worn, real-time hand, and surface gesture recognition via semg and imu sensing," *IEEE Tran. on Industrial Informatics*, vol. 14, no. 8, 2017.

[68] T.-c. Fu, "A review on time series data mining," *eng. app. of Artificial intel.*, vol. 24, no. 1, 2011.

[69] M. Abadi *et al.*, "Tensorflow: a system for large-scale machine learning," in *Proc. of the 12th USENIX Conf. on Operating sys. Design and Implementation*, 2016.

[70] A. Gulli *et al.*, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[71] C. N. Coelho *et al.*, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature mach. intel.*, vol. 3, no. 8, 2021.

[72] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *the J. of mach. learn. research*, vol. 12, 2011.

[73] S. S. Sandha *et al.*, "Enabling hyperparameter tuning of machine learning classifiers in production," in *2021 IEEE Third International Conference on Cognitive Machine Intelligence (CogMI)*. IEEE, 2021.

[74] P. Esling *et al.*, "Time-series data mining," *ACM comp. Surveys (CSUR)*, vol. 45, no. 1, 2012.

[75] Z. Wang *et al.*, "A universal image quality index," *IEEE signal proc. let.*, vol. 9, no. 3, 2002.

[76] Z. Wang *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE Tran. on image proc.*, vol. 13, no. 4, 2004.

[77] J. Zhou *et al.*, "A wavelet transform method to merge landsat tm and spot panchromatic data," *Intl. J. of remote sensing*, vol. 19, no. 4, 1998.

[78] L. Alparone *et al.*, "Multispectral and panchromatic data fusion assessment without reference," *Photogrammetric eng. & Remote Sensing*, vol. 74, no. 2, 2008.