Proceedings of CONF-MLA 2026 Symposium: Learning and Decision Making in Multi Agent Software Systems
DOI: 10.54254/2755-2721/2026.BA37682

Balancing Affordability and Accuracy: A Hardware Selection Guide for Wearable Fall Detection Systems

Zhuoyuan Li

Harrow International School, Hong Kong, China
zhuoyuanli0428@gmail.com

Abstract. Unintentional falls among older adults represent a severe global health concern, necessitating reliable, automated wearable Fall Detection Systems (FDS). However, developing practical wearable solutions requires navigating intricate engineering trade-offs among diagnostic accuracy, power consumption, memory footprint, and software development overhead. This paper presents a structured hardware selection guide evaluating three representative embedded design paradigms: low-cost threshold logic on 8-bit microcontrollers (Profile A), balanced classical machine learning on standard 32-bit platforms (Profile B), and compressed on-chip TinyML neural networks on edge-capable microcontrollers (Profile C). Comparative analysis reveals that while threshold methods achieve multi-week battery longevity at sub-$12 costs, they suffer from high false-alarm rates during dynamic daily activities. Conversely, TinyML architectures deliver superior classification accuracy (98%) on raw kinematic streams, but introduce steep memory quantization barriers, complex toolchains, and shorter battery recharge intervals. Classical machine learning emerges as the optimal design compromise for most developers, achieving robust diagnostic discrimination (95% sensitivity and specificity) with manageable computational demands at $15-$25. Finally, this review evaluates sensor placement ergonomics, addresses pervasive demographic bias in youth-simulated training datasets, and delivers a practical hardware selection framework for student researchers and independent developers.

Keywords: Wearable fall detection systems, Inertial Measurement Units (IMUs), Geriatric Healthcare Technology, Machine Learning Classifiers

1. Introduction

Unintentional falls among older adults represent a critical public health crisis. Globally, falls claim 646,000 to 684,000 lives annually as the second leading cause of accidental injury deaths [1]. Between 40% and 60% of elderly falls cause severe physical trauma, such as fractures or traumatic brain injuries [2], causing $30 billion to $50 billion in direct annual medical costs in the United States alone [3, 4]. Because many elderly individuals are unable to manually activate emergency alarm buttons after falling due to shock or loss of consciousness [5], automated wearable Fall Detection Systems (FDS) are essential for rapid medical intervention [5].

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.BA37682

Designing wearable FDS requires balancing classification accuracy against severe hardware constraints. While Machine Learning (ML) and Deep Learning (DL) architectures consistently achieve higher classification accuracy than simple threshold-based logic [6], their computational complexity means heavy processing and storage demands that strain low-cost microcontrollers and drain battery life [7]. Furthermore, a major data bottleneck exists: most current detection models are trained on simulated falls from young volunteers [6]. Because the biomechanics and reaction dynamics of young subjects differ significantly from older adults, these models frequently experience performance degradation when deployed in real-world clinical settings [6].

This paper evaluates existing literature to establish an optimal, accessible hardware design framework. This review is structured as follows: Section 1 introduces wearable architectures and core evaluation criteria, section 2 compares the classification accuracy of threshold-based logic versus machine learning algorithms, section 3 evaluates physical hardware suitability, focusing on battery draw, chip storage, and form factor trade-offs, and section 4 analyzes implementation complexity and data requirements to deliver a practical hardware selection guide.

2. Wearable system architectures and evaluation framework

A wearable Fall Detection System (FDS) has 4 functional stages when operating: movement sensing, on-chip processing, algorithmic classification, and alert transmission. Inertial Measurement Units (IMUs), specifically tri-axial accelerometers and gyroscopes, acquire physical motion data. This kinematic data is sampled by a Microcontroller Unit (MCU), where an embedded algorithm classifies the movement as either a normal Activity of Daily Living (ADL) or a fall event. Upon detecting a fall, the MCU triggers local buzzers or wireless communication modules (e.g., GSM, GPS, or Ultra-Wideband) to alert caregivers.

To benchmark these technologies, this guide categorizes wearable designs into four representative System Profiles:

• Profile A (Low-Cost Threshold): 3-axis accelerometer + low-power MCU (e.g., Arduino Nano / ATmega328P) + rule-based threshold logic.

• Profile B (Balanced Classical ML): 6-axis IMU + mid-range MCU (e.g., standard ESP32) + feature-based classical machine learning (e.g., SVM, k-NN).

• Profile C (Edge AI/TinyML): 6-axis IMU + edge-capable MCU (e.g., ESP32-S3/ARM Cortex-M4) + compressed on-chip neural network (TinyML).

Evaluating these setups requires looking beyond simple detection rates through two primary criteria:

2.1. System performance criteria

• Diagnostic Performance (Sensitivity & Specificity): Due to severe class imbalance in daily activity data, relying solely on raw accuracy is often misleading [6]. Instead, systems must be evaluated using Sensitivity (catching true falls) and Specificity (preventing false alarms).

• Energy Efficiency: Gyroscopes and continuous GPS tracking draw significantly higher current than standalone accelerometers, dictating battery lifespan during continuous monitoring.

• Memory Footprint: Embedded microcontrollers feature strict SRAM and Flash storage limits, requiring algorithms to minimize memory allocation.

DOI: 10.54254/2755-2721/2026.BA37682

2.2. Implementation feasibility criteria

• Data Dependency: Machine learning models require large labeled datasets, which frequently suffer from demographic bias because most open benchmarks are collected from young volunteers rather than elderly individuals [6].

• Development Overhead: Simple threshold logic requires basic C++ conditional statements, whereas deploying embedded neural networks requires optimization techniques such as 8-bit model quantization and specialized frameworks like TensorFlow Lite for Microcontrollers [7].

3. Assessment of accuracy across fall detection approaches

Evaluating diagnostic accuracy requires recognizing that a true fall is not an isolated acceleration impulse, but a distinct multi-phase kinematic sequence unfolding in several seconds [6, 7]:

• Free-Fall Phase: Prior to ground impact, the body experiences a brief loss of balance and downward acceleration, causing the Signal Magnitude Vector (SMV) to drop sharply below the gravitational baseline (<0.5-0.8g) [7].

• Impact Spike: The physical collision produces high-magnitude, high-frequency acceleration peaks (typically exceeding 2.5-3g) across multiple axes [7].

• Orientation Change: The body transitions from an upright posture to a horizontal resting plane, resulting in a persistent angular shift relative to the static gravity vector [6, 7].

• Post-Impact Immobility: Following an incapacitating fall, the subject remains motionless for an extended duration, characterized by near-zero signal variance over an observation window of 10–30 seconds [6, 7].

Integrating this temporal sequence directly shapes how different algorithmic paradigms balance sensitivity and specificity:

Threshold-Based Methods (TBM) utilize cascading logical filters across these four phases [7]. While single-threshold systems trigger frequent false alarms on high-acceleration Activities of Daily Living (ADLs), such as jumping, running, or plopping onto a soft couch, incorporating post-impact orientation shifts and a 10-to-30-second immobility check substantially improves specificity without adding computational overhead [6, 7]. If normal dynamic motion resumes immediately following an impact spike, the algorithm safely discards the event as a vigorous ADL rather than an emergency [6].

For complex or borderline movements, classical Machine Learning (ML) segments this sequence into discrete temporal windows (pre-impact, impact, and post-fall recovery) [6]. By extracting multi-dimensional statistical features (variance, kurtosis, and spectral energy) across each phase, classical ML achieves >95-98% sensitivity and specificity [6, 7]. Deep Learning and TinyML models (e.g., 1D-CNNs and LSTMs) further optimize detection (>98-99%) by learning raw temporal dependencies across the full sequence, though performance remains sensitive to sensor placement: waist and chest locations closer to the body's Center of Mass (CoM) capture distinct phase transitions far more reliably than noisy wrist placements [6, 7].

4. Assessment of suitability of different approaches as a wearable device

While algorithmic classification accuracy is critical, practical deployment requires evaluating how different fall detection paradigms translate into physical hardware constraints. Operating on miniature rechargeable batteries dictates strict trade-offs among sensor power draw, microcontroller computational overhead, memory allocation, and physical wearability across Profiles A, B and C.

DOI: 10.54254/2755-2721/2026.BA37682

4.1. Assessment of battery consumption and power draw

System power consumption is governed by physical sensor draw, microcontroller computational duty-cycling, and system architectural design.

At the sensor level, a substantial efficiency gap exists between inertial sensor types. Standalone 3-axis accelerometers (used in Profile A) operate in the ultralow-power domain, drawing between 10 μA and 150 μA during continuous sampling [7]. In contrast, 6-axis IMUs (used in Profiles B and C) integrate a 3-axis gyroscope to capture rotational velocity, which requires active drive circuitry and increases current draw to 2-4 mA—roughly a 20 to 200 times increase in sensor power consumption [7].

Computationally, algorithmic complexity directly dictates how often the Microcontroller Unit (MCU) can enter low-power sleep states. In an unoptimized, always-on configuration, Profile A evaluates instantaneous kinematic thresholds with minimal CPU overhead, allowing the MCU to remain in deep sleep for over 95% of operational time [7]. Conversely, always-on execution of Profiles B and C requires continuous sampling (e.g., 50 Hz) and periodic execution of multi-parameter statistical algorithms or convolutional matrix operations. This sustained processing holds the MCU at elevated clock frequencies (160-240 MHz on an ESP32), restricting operational battery life to 24-48 hours on a standard 200 mAh Lithium-Polymer (Li-Po) cell [7].

To resolve this continuous-sampling bottleneck, production-grade wearables implement a hierarchical two-stage duty-cycling architecture [6, 7]. The primary MCU remains in deep sleep (< 10 μA) while the inertial sensor operates independently in an ultralow-power monitoring mode (< 50 μA), continuously logging data into an internal FIFO (First-In, First-Out) hardware buffer [7]. When an instantaneous kinematic threshold (e.g., a > 2.5 g impact spike or < 0.5 g free-fall dip) is breached, the sensor asserts a hardware interrupt pin to wake the MCU [7]. The microcontroller then reads the buffered 2-to-3-second motion window and executes the Stage-2 ML (Profile B) or TinyML (Profile C) classifier to verify the event [6, 7]. This cascaded approach confines power-intensive compute cycles strictly to potential fall events, extending battery longevity from days to multiple weeks while maintaining high specificity [6, 7].

4.2. Assessment of storage and memory footprint

Hardware feasibility also depends heavily on dynamic memory (SRAM) and non-volatile program storage (Flash) limitations within embedded chips.

Profile A imposes minimal memory overhead. Because threshold comparisons process discrete sample points in real time without storing historical sequences, dynamic RAM usage is negligible (<1 KB), and compiled C++ conditional rules occupy less than 5 KB of Flash storage [7]. Consequently, Profile A executes easily on low-end 8-bit microcontrollers, such as the ATmega328P (2 KB SRAM, 32 KB Flash).

In contrast, classical ML (Profile B) and TinyML (Profile C) require dedicated sliding time-series windows (typically 2-4 seconds of 6-axis data at 50 Hz), consuming several kilobytes of allocated SRAM buffers [7]. Furthermore, deploying compressed neural networks in Profile C requires allocating a continuous memory block called a "tensor arena" (50-250 KB of SRAM) to execute intermediate activation layers, alongside 200-500 KB of Flash storage for 8-bit quantized weight matrices [7]. These memory demands make 32-bit platforms with larger SRAM pools, such as the ESP32 or ESP32-S3, mandatory.

DOI: 10.54254/2755-2721/2026.BA37682

4.3. Sensor placement

Sensor placement governs both data fidelity and long-term user compliance. Mounting sensors near the body's Center of Mass (CoM), specifically at the waist or chest, provides stable kinematic signals that reflect whole-body displacement, minimizing false alarms during upper-body movement [6, 7]. However, waist belts and chest straps often suffer from poor user adherence among elderly populations [6]. Conversely, wrist-worn devices achieve the highest daily compliance due to familiarity with watches, but introduce significant kinematic noise from routine hand gestures (e.g., cooking, gesticulating, or washing), requiring higher algorithmic discrimination [6, 7].

5. Implementation feasibility and development complexity

Selecting an appropriate fall detection system requires balancing diagnostic performance against practical engineering barriers. For student researchers, hobbyists, and independent developers, the optimal design depends heavily on training data requirements, software pipeline complexity, and the overall Bill of Materials (BOM) cost.

5.1. Data dependency and dataset bottlenecks

The volume and nature of required motion data vary substantially across algorithmic approaches:

• Profile A (Rule-Based Thresholds): Imposes zero training data requirements. Threshold systems rely purely on established physical kinematics, such as free-fall zero-g drops, high-impact acceleration peaks, and subsequent posture tilt angles. Calibration requires only empirical threshold tuning rather than statistical dataset collection.

• Profile B (Classical Machine Learning): Requires moderate dataset sizes, typically comprising several hundred to thousands of labeled motion windows. Statistical feature extraction (e.g., calculating variance, skewness, and spectral energy) provides sufficient mathematical abstraction to train classifiers like Support Vector Machines (SVM) or Random Forests effectively on open-source repositories [6].

• Profile C (Edge AI/TinyML): Demonstrates high data dependency. Deep learning architectures (such as 1D-CNNs and LSTMs) require tens of thousands of continuous time-series frames to learn hierarchical spatial-temporal representations without severe overfitting [6].

A fundamental challenge across both Profile B and Profile C is the demographic data bottleneck. Most open benchmark repositories (such as *SisFall* and *MobiFall*) rely almost exclusively on simulated falls performed by young, healthy volunteers onto cushioned mats [6]. Because older adults exhibit distinct gait alterations, slower protective reflexes, and varied fall mechanics, models trained solely on simulated youthful movements frequently suffer performance degradation when deployed in real-world geriatric environments [6].

5.2. Algorithmic and software stack complexity

The technical barrier to entry is directly tied to the development toolchain required for each profile:

• Profile A (Low Complexity): Accessible directly to beginners. Threshold logic is written entirely in basic embedded C/C++ using straightforward conditional statements (if/else) within standard environments like the Arduino IDE, requiring minimal external libraries.

• Profile B (Moderate Complexity): Introduces a dual-language workflow. Feature extraction algorithms, dimensionality reduction, and model training are typically executed in Python (e.g.,

DOI: 10.54254/2755-2721/2026.BA37682

using Scikit-Learn). Once trained, decision tree boundaries or support vector coefficients must be manually exported or translated into embedded C++ arrays for microcontroller execution.

• Profile C (High Complexity): Involves a steep technical learning curve. Developers must construct neural network topologies in Python (TensorFlow or PyTorch), apply post-training 8-bit integer quantization (INT8) to compress floating-point weights, convert the binary into flat C source arrays, and manage dynamic tensor arena memory allocations using specialized frameworks like TensorFlow Lite for Microcontrollers [7].

• Hierarchical System Firmware Overhead: Implementing a two-stage cascaded architecture introduces firmware-level complexity across all profiles. Developers must configure on-chip sensor registers via I2C/SPI buses—including motion interrupt thresholds, high-g triggers, and circular FIFO buffer watermarks—to wake the MCU only when an impact candidate is registered [7].

5.3. Bill of Materials (BOM) and cost analysis

Component expenses scale alongside computational and sensing requirements:

• Profile A ($8–$12 total): Utilizes an entry-level 3-axis accelerometer (e.g., ADXL345 at $2-$3), an 8-bit microcontroller (e.g., Arduino Nano/ATmega328P at $3-$4), and a basic coin-cell or small battery casing.

• Profile B ($15–$25 total): Integrates a standard 6-axis IMU (e.g., MPU6050 at $2-$4), a dual-core 32-bit MCU (e.g., ESP32 at $4-$6), and a dedicated rechargeable Lithium-Polymer (Li-Po) battery with charging circuitry.

• Profile C ($25–$40 total): Requires a low-noise 6-axis IMU (e.g., BMI270 at $5-$8), an edge-optimized MCU with vector instruction acceleration and expanded PSRAM (e.g., ESP32-S3 or ARM Cortex-M4 at $8-$12), and a higher-capacity Li-Po battery to sustain continuous active CPU cycles.

5.4. Hardware selection synthesis matrix

To provide an integrated overview of the trade-offs evaluated across diagnostic performance, embedded constraints, software overhead and economic feasibility, Table 1 synthesizes the core metrics across all evaluated architectures.

Table 1. Comprehensive hardware and algorithmic selection matrix

Evaluation Dimension	Profile A (Low-Cost Threshold)	Profile B (Balanced Classical ML)	Profile C (Edge AI/ TinyML)	Hierarchical Two-Stage Pipeline
Core Hardware	8-bit MCU (ATmega328P) + 3-Axis Accel (ADXL345)	32-bit MCU (ESP32) + 6-Axis IMU (MPU6050)	Edge MCU (ESP32-S3) + 6-Axis IMU (BMI270)	32-bit MCU (ESP32/S3) + Interrupt/FIFO IMU
Algorithmic Paradigm	Deterministic Kinematic Rules (SMV, Tilt Angle)	Statistical Feature Extraction + SVM / Random Forest	Quantized Neural Network (1D-CNN/ LSTM)	Stage 1: Hardware Threshold Stage 2: ML/ TinyML Verification

DOI: 10.54254/2755-2721/2026.BA37682

Table 1. (continued)

Diagnostic Accuracy	Sensitivity: 88-92% Specificity: <80% (High false alarms)	Sensitivity: > 95% Specificity: > 95%	Sensitivity: > 98% Specificity: > 98%	Sensitivity: > 97% Specificity: > 97%
Continuous Battery Life (200 mAh Cell)	Multi-week (>95% MCU sleep)	24-48 Hours (Continuous 50 Hz active compute)	18-36 Hours (Continuous matrix inference)	2–4+ Weeks (MCU sleeps until impact interrupt)
Memory Footprint	SRAM: < 1 KB Flash: < 5 KB	SRAM: 4-16 KB Flash: 30–100 KB	SRAM: 50-250 KB (Tensor Arena) Flash: 200–500 KB	SRAM: 8-64 KB Flash: 100–350 KB
Training Data Requirement	None (Physics-based calibration)	Moderate (Hundreds of segmented motion windows)	High (Tens of thousands of time-series frames)	Moderate to High (Matches Stage 2 classifier)
Software Toolchain	Arduino IDE (C/C++ conditional logic)	Python (Scikit-Learn) $\rightarrow$ C++ Array Export	TensorFlow/PyTorch $\rightarrow$ INT8 $\rightarrow$ TFLite Micro	Sensor Register Config (I2C/SPI) + TFLite/ML Pipeline
Estimated BOM Cost	$8-$12	$15-$25	$25-$40	$20-$35
Primary Suitability	Absolute beginners, strict sub-$12 budgets	Optimal balance for most standalone prototypes	Research/commercial systems requiring maximum precision	Recommended industry standard for wearable deployment

6. Conclusion

Selecting an effective hardware architecture for a wearable Fall Detection System is fundamentally a multi-variable optimization problem. As demonstrated across this guide, no single design universally satisfies all clinical, technical and budgetary constraints; rather, the optimal system depends on balancing classification accuracy against battery longevity, memory allocation, development overhead, and component costs.

To guide prospective builders and researchers, hardware selection can be aligned with specific project priorities: (1) Profile A (Low-Cost Thresholds): Recommended for absolute beginners, extreme sub-$12 cost constraints, or applications requiring multi-week battery longevity. While its sensitivity is sufficient for high-impact falls, developers must accept higher false-alarm rates during rapid daily activities. (2) Profile B (Balanced Classical ML): Represents the optimal standalone design compromise for most functional prototypes and intermediate developers. Leveraging an accessible ESP32 platform with statistical classifiers (e.g., SVM or Random Forest) provides robust diagnostic discrimination (>95% sensitivity and specificity) for $15-$25 while avoiding heavy neural network compilation toolchains. (3) Profile C (Edge AI / TinyML): Recommended for

DOI: 10.54254/2755-2721/2026.BA37682

advanced projects where maximum classification accuracy (>98%) on raw motion data is required. This performance comes at the expense of a steeper software learning curve, strict memory quantization requirements, and shorter battery recharge intervals. (4) Hierarchical Two-Stage Integration (Production Standard): For real-world implementation, the most effective engineering strategy is a hybrid pipeline: using Profile A's low-power hardware threshold interrupts to maintain deep MCU sleep, and triggering Profile B (Classical ML) or Profile C (TinyML) inference only on buffered FIFO inertial windows when an impact candidate occurs.

Ultimately, bridging the gap between laboratory prototypes and real-world clinical adoption requires addressing the demographic data bottleneck. Future work must focus on evaluating wearable architectures against real-world geriatric motion datasets and developing lightweight, on-chip adaptive calibration to ensure vulnerable individuals receive reliable, life-saving protection.

References

[1] World Health Organization. (2021). Falls [Internet]. *World Health Organization*. [Aug. 12]. Available from: https://www.who.int/news-room/fact-sheets/detail/falls, Geneva.

[2] Vieira, E. R., Palmer, R. C., & Chaves, P. H. M. (2016). Prevention of falls in older people living in the community. *BMJ, 353*:i1419.

[3] Stevens, J. A., Corso, P. S., Finkelstein, E. A., & Miller, T. R. (2006). The costs of fatal and nonfatal falls among older adults. *Inj Prev, 12*(5), 290–5.

[4] Florence, C. S., Bergen, G., Atherly, A., Burns, E., Stevens, J., & Drake, C. (2018). Medical costs of fatal and nonfatal falls in older adults. *J Am Geriatr Soc, 66*(4), 693–8.

[5] Solbach, M. D., & Tsotsos, J. K. (2017). Vision-based fallen person detection for the elderly. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (pp. 1433–1442).

[6] Gorce, P., & Jacquier-Bret, J. (2025). Fall detection in elderly people: A systematic review of ambient assisted living and smart home-related technology performance. *Sensors, 25*(21), 6540.

[7] Ajerla, D. (2025). A survey on fall detection systems using wearable devices. *Int J Inf Res Multidiscip Stud (IJIRMPS), 13*(3), 1–7.