Smart Posture Corrector: An Edge-AI IMU-Based Wearable System with Haptic Feedback on ESP32

Tazkia Nizami*, Bagas Sambega Rosyada*, and Raden Francisco Trianto Bratadiningrat*
* Bandung Institute of Technology (Institut Teknologi Bandung), Indonesia
E-mail: 13522032@std.stei.itb.ac.id, 13522071@std.stei.itb.ac.id, 13522091@std.stei.itb.ac.id

***Abstract*—This paper presents a deployment-aware IMU-based posture classification and haptic feedback system built on an ESP32 wearable node. The system acquires accelerometer and gyroscope signals from an MPU-6050 sensor, applies a lightweight preprocessing pipeline, and classifies posture into six classes using edge-AI machine-learning models. Dataset collection was performed across multiple sessions per label, initially stored as separate per-label CSV files with Indonesian label names, then consolidated into a single unified dataset (`dataset_all.csv`) of 45,774 samples with standardised English class labels. Twelve candidate classifiers—spanning classical machine learning (ML) and compact deep-learning (DL) architectures—are evaluated under deployment-aware criteria that include macro-averaged F1-score, cross-validation stability, and serialised model size. LightGBM, tuned with regularisation constraints, achieves the best balance: a test macro F1-score of 0.9813 and an exported C header of approximately 585 kB via `m2cgen`, which is compatible with ESP32 flash constraints. The embedded firmware delivers local vibratory haptic feedback upon detection of a slouched posture, while an MQTT–InfluxDB–dashboard stack supports optional telemetry and visualisation.**

***Index Terms*—posture classification, inertial measurement unit, edge AI, LightGBM, ESP32, wearable, haptic feedback, IoT.**

I. Introduction

Poor posture during prolonged sitting and screen-centred activity remains a practical concern in work, study, and everyday digital routines. Extended sitting behaviour and sustained non-neutral postures during computer or smartphone use have been associated with posture-related musculoskeletal discomfort, including neck, shoulder, and back discomfort [1]. This has motivated sustained interest in posture-monitoring technologies across ergonomics, rehabilitation, and wearable sensing research. However, in embedded wearable systems, the central challenge is no longer merely to show that posture monitoring is useful, but to implement posture classification and feedback efficiently and reliably on resource-constrained hardware.

Compared with vision-based or smartphone-only approaches, wearable inertial sensing offers a practical compromise for mobile posture monitoring because it does not depend on camera placement, lighting conditions, or fixed device usage assumptions [2]. When combined with on-device inference, IMU-based systems can keep sensing, feature computation, and classification close to the user, thereby reducing network dependence and avoiding unnecessary transmission of raw motion data [3]. This architecture is also well aligned with responsive feedback, since a poor-posture event can be translated directly into a local actuation signal. In this context, haptic feedback is especially suitable because it can be delivered immediately, discreetly, and without requiring the user to attend to a screen or audio cue.

A key issue in developing IMU-based posture monitoring on microcontroller-class hardware is the gap between offline classification performance and practical embedded deployment. Posture-classification studies commonly report model performance using classification metrics such as accuracy, precision, recall, and F1-score [4]. However, for a wearable node, the selected model must also be compatible with limited memory, low computational capacity, real-time sensing, and feedback execution. This consideration is especially important for low-cost microcontrollers such as the ESP32, where inference latency, model size, implementation complexity, and integration with sensor acquisition and actuator control can directly affect system usability. Therefore, this work focuses on a deployment-aware evaluation of feature-based IMU posture classification, where candidate models are compared not only by predictive performance but also by their suitability for local execution on an inexpensive wearable device.

Accordingly, this paper presents a deployment-aware IMU-based posture classification and haptic feedback system built on an ESP32-based wearable node. The system acquires accelerometer and gyroscope signals from an MPU-6050 sensor, applies a deployment-aware preprocessing pipeline, and classifies posture into six classes using lightweight machine-learning models. The contributions of this work are threefold: (1) we design a low-cost ESP32–MPU-6050 wearable node for IMU-based posture monitoring with local haptic feedback; (2) we construct and consolidate a multi-session IMU dataset with standardised English labels covering six posture classes; and (3) we compare twelve candidate models—nine classical ML and three compact DL architectures—under deployment-aware criteria and identify regularised LightGBM as the selected deployment candidate.

II. System Architecture and Implementation

A. System Architecture

The overall architecture of the smart posture corrector is designed to operate primarily on the edge while maintaining optional cloud connectivity for remote monitoring. The system consists of three main components: the wearable node, the MQTT broker, and the cloud dashboard. At its core, the wearable node acts as an edge computing device that performs real-

time sensor data acquisition, feature extraction, and posture classification entirely on-device. This edge-heavy architecture ensures that haptic feedback can be delivered with minimal latency and without relying on a continuous internet connection. When network connectivity is available, the node transmits telemetry data—including raw sensor readings and predicted posture labels—to a central MQTT broker. A backend server subscribes to these MQTT topics, ingests the incoming data, and stores it in an InfluxDB time-series database. Finally, a web-based dashboard visualises the historical posture data, allowing users to track their posture habits over time.

B. Hardware Implementation

The hardware layer of the wearable node is engineered around the ESP32 DevKit V1 microcontroller, an MPU-6050 Inertial Measurement Unit (IMU), and a 5V coin-type vibration motor for local tactile haptic alerts. The ESP32 was selected for its dual-core processor, sufficient SRAM for running lightweight machine learning models, and built-in Wi-Fi capabilities.

The MPU-6050 contains a 3-axis MEMS accelerometer and a 3-axis MEMS gyroscope encapsulated on a single chip, capturing raw physical acceleration vectors and angular velocities across the anatomical planes of the human body. The MPU-6050 communicates with the ESP32 via the I^2C protocol. Specifically, the Serial Data (SDA) line is tied to ESP32 GPIO pin 21, and the Serial Clock (SCL) line is tied to GPIO pin 22. Power delivery to the IMU is sourced directly from the ESP32 3.3V regulated output rail.

For local actuation, a coreless vibration motor is assigned to GPIO pin 15, driven through a transistor circuit to provide sufficient current. A physical tactile push-button is wired between GPIO pin 14 and the common ground, utilising the microcontroller's internal pull-up resistor to capture hardware interrupt events for sensor calibration. Alternatively, calibration can be triggered remotely by publishing a `CALIBRATE` payload to a designated MQTT topic.

For the present study, the IMU and the complete circuit are housed in a compact enclosure positioned on the user's upper back using a custom strap, with the sensor axes aligned approximately with the wearer's anatomical axes.

C. Cloud Implementation

While the primary posture detection loop and feedback mechanism operate independently on the ESP32, the cloud implementation provides a robust infrastructure for long-term data logging and visualisation. The wearable node connects to a local Wi-Fi network and establishes a connection with an MQTT broker. Telemetry data is serialised into a lightweight JSON payload containing the timestamp, raw IMU values, extracted features, and the predicted posture class. This payload is published to the broker at a configurable interval.

A backend service implemented in Go acts as an MQTT subscriber. It receives the incoming JSON payloads, parses the sensor and classification data, and performs batch writes to an InfluxDB time-series database. A frontend dashboard built with HTML and Chart.js connects to the backend API to retrieve the stored data. The dashboard provides colour-coded visualisations of the user's posture history, clearly distinguishing between upright ("good") and slouched ("bad") states, thereby offering users actionable insights into their daily ergonomic habits.

D. Data Collection

Data collection is performed by a single participant, with each recording session dedicated to a single posture label and preceded by an offset-calibration step to compensate for sensor drift and mounting bias. Raw recordings are initially stored as separate per-label CSV files using Indonesian label names (e.g., `duduk_bungkuk.csv` for sitting slouched). Multiple sessions per label are recorded to increase data volume and intra-class variability.

After all sessions are completed, the per-label files are merged into a single unified dataset file (`dataset_all.csv`). As part of this consolidation step, each Indonesian label name is mapped to its standardised English equivalent according to the scheme in Table I. The consolidated dataset comprises 45,774 samples with 20 columns, including raw accelerometer and gyroscope channels, intermediate tilt estimates, offset terms, derived rotation-speed features, and the ground-truth label column.

Documentation of data collection process and usage of the system can be seen in Figure 3.

TABLE I
LABEL MAPPING FROM INDONESIAN TO ENGLISH

Indonesian label	English label
`duduk_bungkuk`	`sitting_slouched`
`duduk_tegak`	`sitting_upright`
`berdiri_bungkuk`	`standing_slouched`
`berdiri_tegak`	`standing_upright`
`jalan_bungkuk`	`walking_slouched`
`jalan_tegak`	`walking_upright`

III. METHODOLOGY

A. Dataset and Preprocessing

The consolidated dataset (`dataset_all.csv`) contains 45,774 samples and 20 columns. The full column schema of the raw data is: `timestamp`, `rawAccX`, `rawAccY`, `rawAccZ`, `rawGyroX`, `rawGyroY`, `rawGyroZ`, `rawKemiringanX`, `rawKemiringanY`, `offsetKemiringanX`, `offsetKemiringanY`, `kemiringanX`, `kemiringanY`, `gyroOffsetX`, `gyroOffsetY`, `gyroOffsetZ`, `kecepatanRotasiX`, `kecepatanRotasiY`, `kecepatanRotasiZ`, and `label`.

To reduce model complexity and avoid overfitting to session-specific artifacts, a deployment-aware preprocessing pipeline is applied. First, the `timestamp` column is discarded because it is session-dependent and does not carry posture-specific information. Second, offset-related columns (`offsetKemiringanX`, `offsetKemiringanY`,

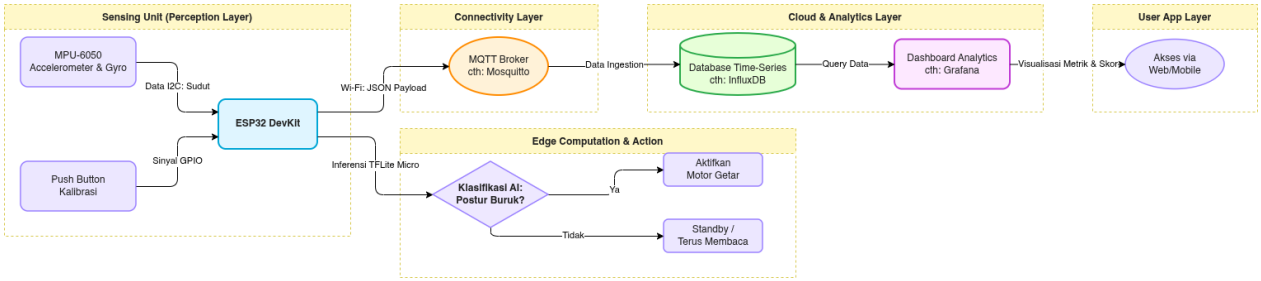


Fig. 1. System architecture encompassing the wearable node, MQTT broker, and cloud dashboard.

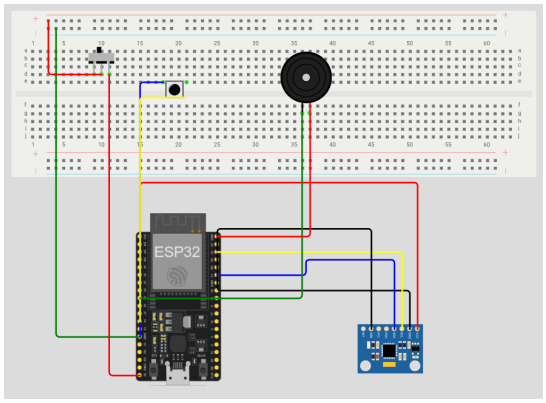


Fig. 2. Circuit board and hardware components wiring diagram.



Fig. 3. Demonstration of Data Collection and System Usage Process

`gyroOffsetX/Y/Z`) are removed, since they exhibit negligible variability within sessions and risk acting as spurious identifiers of calibration episodes rather than posture cues. Third, intermediate composite features (`rawKemiringanX/Y`, raw gyroscope channels) are pruned so that the model relies on final tilt and rotation-speed estimates rather than multiple redundant representations of the same underlying signal.

After pruning, the feature set is restricted to eight variables: `rawAccX`, `rawAccY`, `rawAccZ`, `kemiringanX`, `kemiringanY`, `kecepatanRotasiX`, `kecepatanRotasiY`, and `kecepatanRotasiZ`. The label column is then encoded into an integer `label_encoded` variable using a standard label-encoding scheme, with the following mapping: `sitting_slouched`→0, `sitting_upright`→1, `standing_slouched`→2, `standing_upright`→3, `walking_slouched`→4, `walking_upright`→5. The resulting preprocessed dataset comprises 45,774 samples with eight numerical features and one encoded class label.

1) Label Distribution: Figure 4 shows the distribution of labels in the consolidated dataset. The class distribution is reasonably balanced, with `sitting_slouched` being the most frequent class (20.61%, 9,421 samples) and `standing_upright` the least frequent (12.52%, 5,730 samples). This near-balanced distribution supports the use of macro-averaged F1-score as the primary evaluation metric.

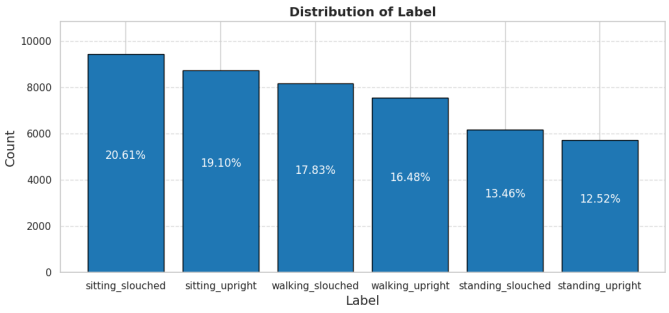


Fig. 4. Distribution of the six posture classes in the consolidated dataset (`dataset_all.csv`, $n = 45{,}774$).

B. Train–Test Split

The preprocessed dataset is split into an 80% training set (36,619 samples) and a 20% test set (9,155 samples) using a stratified random split to preserve label proportions across both

partitions. The resulting class proportions in the training and test sets are shown in Table II, confirming that stratification is successful.

TABLE II
CLASS PROPORTIONS AFTER STRATIFIED 80/20 SPLIT

Label (encoded)	Train proportion	Test proportion
`sitting_slouched` (0)	0.2061	0.2061
`sitting_upright` (1)	0.1910	0.1909
`standing_slouched` (2)	0.1346	0.1346
`standing_upright` (3)	0.1252	0.1252
`walking_slouched` (4)	0.1783	0.1784
`walking_upright` (5)	0.1648	0.1648
Total samples	36,619	9,155

C. Candidate Models and Evaluation Protocol

To compare candidate models under both predictive and deployment-related criteria, twelve classifiers are trained and evaluated. The suite comprises nine classical ML models—Logistic Regression, SVM with RBF kernel, AdaBoost, Gradient Boosting, Random Forest, Extra Trees, XGBoost, LightGBM, and CatBoost—and three compact deep-learning architectures: a small feed-forward neural network (FFNN small, hidden layers 32–16), a deeper FFNN (FFNN deep, layers 64–32–16), and a simple RNN (RNN simple, recurrent layer of 32 units followed by a dense layer of 16 units). All neural networks use ReLU activations, L2 regularisation, and dropout on hidden layers, and are trained on standardised features.

On the training set, 5-fold stratified cross-validation is used to estimate the mean and standard deviation of the macro-averaged F1-score for each model. Each model is then retrained on the full training set and evaluated on the held-out test set. The serialised model size (in kB) is recorded as a proxy for memory footprint on the target device. Together, these metrics form the deployment-aware evaluation framework.

D. Final Model: Regularised LightGBM

Based on the deployment-aware evaluation, LightGBM is selected as the primary deployment candidate. To further improve generalisation and reduce the serialised model footprint, a regularised configuration with constrained tree depth and reduced estimator count is applied:

```
LGBMClassifier(
    n_estimators=50,
    max_depth=4,
    learning_rate=0.1,
    random_state=42
)
```

This regularised variant trades a marginal reduction in training F1 for better generalisation and a smaller exported C header size, making it more suitable for deployment on the ESP32.

IV. RESULTS AND DISCUSSION

A. Comparative Model Evaluation

Table III presents the cross-validation macro F1-score (mean and standard deviation), test macro F1-score, and serialised model size for all twelve evaluated models, ranked by test F1-score. LightGBM achieves the highest test macro F1 of 0.9917, followed closely by XGBoost (0.9913), CatBoost (0.9876), and Extra Trees (0.9876). Among the deep-learning baselines, RNN simple attains 0.9822 with only 50.9 kB, demonstrating that compact neural networks are also competitive on this dataset. AdaBoost is the weakest performer, with a test F1 of 0.7677, indicating poor suitability for this feature representation and class distribution.

Regarding model size, Logistic Regression is the most compact ML model at 2.3 kB but severely underfits (test F1 = 0.8574). The compact DL models (38.0–72.7 kB) offer an attractive size-performance trade-off but require a different deployment toolchain (e.g., TensorFlow Lite for Microcontrollers). Among tree-based models, LightGBM (2,028.9 kB) and XGBoost (1,232.0 kB) are substantially smaller than Extra Trees (61,391.1 kB), which makes them more practical for embedded deployment via C-code export.

B. Final LightGBM Model Performance

The regularised LightGBM model (50 estimators, max_depth = 4) achieves a training macro F1-score of 0.9842 and a test macro F1-score of 0.9813, confirming that the regularisation constraints slightly reduce overfitting compared to the unregularised baseline while maintaining high predictive performance. Table IV shows the per-class precision, recall, and F1-scores on the test set.

The model performs best on the sitting classes, which exhibit near-perfect precision and recall, likely because these postures produce highly distinctive, quasi-static IMU signatures. The walking classes show slightly lower recall, particularly `walking_slouched` (0.95), which is consistent with the expectation that dynamic transitions between upright and slouched walking can produce intermediate orientations resembling static postures in the feature space.

C. Confusion Matrix Analysis

Figure V shows the confusion matrix for the regularised LightGBM model on the test set. The matrix confirms strong diagonal dominance across all six classes. The most notable off-diagonal entries occur in the walking classes: 56 `walking_slouched` samples are misclassified as `walking_upright`, and 28 `walking_slouched` samples are misclassified as `standing_slouched`. Similarly, 17 `walking_upright` samples are misclassified as `walking_slouched`.

Most misclassifications occur between classes that share the same posture category (upright vs. slouched) but differ in activity, particularly between `Wk_S` (walking slouched) and `St_S` (standing slouched). From the perspective of the embedded system, this type of error is less critical because

TABLE III
COMPARATIVE EVALUATION OF ALL TWELVE CANDIDATE MODELS (RANKED BY TEST MACRO F1-SCORE)

Rank	Model	Type	Train F1 mean (± std)	Test F1	Size (kB)
1	LightGBM	ML	0.9885 ± 0.0005	0.9917	2,028.9
2	XGBoost	ML	0.9883 ± 0.0012	0.9913	1,232.0
3	CatBoost	ML	0.9855 ± 0.0003	0.9876	3,601.2
4	Extra Trees	ML	0.9858 ± 0.0011	0.9876	61,391.1
5	Random Forest	ML	0.9808 ± 0.0013	0.9841	21,773.9
6	RNN simple	DL	0.9766 ± 0.0035	0.9822	50.9
7	Gradient Boosting	ML	0.9786 ± 0.0011	0.9815	755.2
8	FFNN deep	DL	0.9750 ± 0.0021	0.9799	72.7
9	FFNN small	DL	0.9745 ± 0.0014	0.9793	38.0
10	SVM	ML	0.9738 ± 0.0018	0.9766	648.7
11	Logistic Regression	ML	0.8507 ± 0.0032	0.8574	2.3
12	AdaBoost	ML	0.6692 ± 0.0578	0.7677	39.3

TABLE IV
PER-CLASS TEST PERFORMANCE FOR REGULARISED LIGHTGBM (TEST SET, $n = 9{,}155$)

Class	Prec.	Recall	F1	Support
`sitting_slouched`	1.00	1.00	1.00	1,887
`sitting_upright`	0.99	1.00	0.99	1,748
`standing_slouched`	0.98	0.99	0.98	1,232
`standing_upright`	0.99	0.98	0.98	1,146
`walking_slouched`	0.98	0.95	0.96	1,633
`walking_upright`	0.95	0.98	0.97	1,509
Accuracy			0.98	
Macro avg	0.98	0.98	0.98	9,155
Weighted avg	0.98	0.98	0.98	9,155

TABLE V
CONFUSION MATRIX FOR LIGHTGBM ON THE TEST SET (ROWS: ACTUAL, COLUMNS: PREDICTED)

	St_S	St_U	Si_S	Si_U	Wk_S	Wk_U
St_S	1882	0	2	0	3	0
St_U	2	1740	0	5	0	1
Si_S	1	0	1223	0	8	0
Si_U	0	13	0	1118	0	15
Wk_S	0	0	28	2	1547	56
Wk_U	0	1	0	10	17	1481

the ESP32 firmware triggers the vibration motor whenever a slouched posture class is detected, irrespective of whether the model labels it as standing slouched or walking slouched. In contrast, false negatives in which a slouched posture is misclassified as an upright posture would be more problematic, but the confusion matrix shows that such cases are rare in the present dataset.

D. Feature Importance Analysis

Figure 5 shows the split-based feature importance scores for the regularised LightGBM model. `rawAccZ` achieves the highest importance (1,047), followed by `kemiringanY` (931) and `kemiringanX` (655). The horizontal accelerometer components `rawAccX` (470) and `rawAccY` (411) rank fourth and fifth, while the rotational-speed features `kecepatanRotasiX` (268), `kecepatanRotasiZ` (254), and `kecepatanRotasiY` (162) have lower but non-negligible importance.

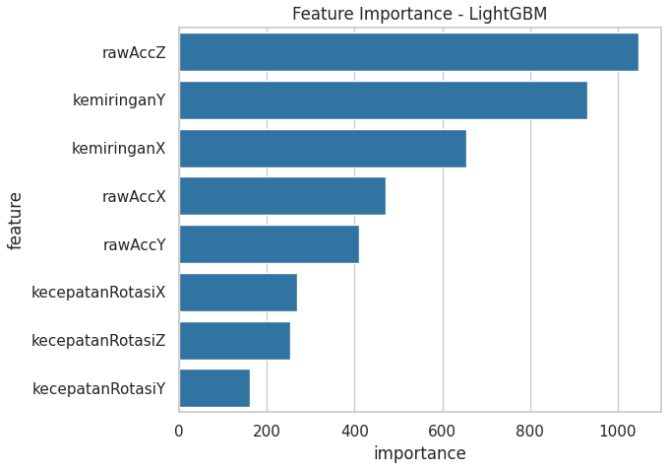


Fig. 5. LightGBM feature importance scores (split-based count) for the regularised model.

TABLE VI
LIGHTGBM FEATURE IMPORTANCE SCORES (REGULARISED MODEL)

Feature	Importance
`rawAccZ`	1,047
`kemiringanY`	931
`kemiringanX`	655
`rawAccX`	470
`rawAccY`	411
`kecepatanRotasiX`	268
`kecepatanRotasiZ`	254
`kecepatanRotasiY`	162

The dominance of `rawAccZ` (vertical acceleration) and the tilt angles (`kemiringanX`, `kemiringanY`) confirms that posture differences are primarily captured by quasi-static orientation and gravitational loading along the vertical axis, while dynamic rotational cues (`kecepatanRotasiX/Y/Z`) contribute more to distinguishing walking from static sitting or standing. From a system-design perspective, this confirms that the selected eight-feature subset is sufficient to capture salient posture differences and can be computed efficiently on the ESP32.

E. Model Export and Embedded Deployment

Beyond offline evaluation, the final model is assessed for its suitability for deployment on the ESP32 as standalone C code. Three Python-to-C conversion libraries are explored: `emlearn`, `micromlgen`, and `m2cgen`. The former two are primarily oriented towards tree-based ensembles and scikit-learn classifiers, whereas `m2cgen` supports a broader range of gradient-boosting models by emitting explicit prediction logic in C.

For the regularised LightGBM model, `m2cgen` is used to generate a self-contained C header implementing the decision logic without requiring the LightGBM runtime on the device. The export result is summarised in Table VII.

TABLE VII
MODEL EXPORT RESULTS VIA M2CGEN

Package	Model	Exported	Header size (kB)	Header size (MB)
m2cgen	LightGBM	Yes	584.97	0.5713

The generated C header (`output/m2cgen/lightgbm_model.h`) has a size of approximately 585 kB (0.57 MB), which is compatible with the flash and RAM constraints of the ESP32 used in this prototype. This export step confirms that the chosen LightGBM model can be integrated into the ESP32 firmware as a plain C module, with the inference function invoked directly from the sensor-processing loop. The experience also highlights that model and toolchain selection for TinyML deployment must consider not only predictive performance but also conversion support and the resulting code footprint on the target hardware.

V. CONCLUSION

This paper presented a deployment-aware IMU-based posture classification and haptic feedback system on an ESP32 wearable node. A multi-session dataset of 45,774 samples covering six posture classes was collected, consolidated from per-label CSV files with Indonesian labels into a unified dataset with standardised English labels, and preprocessed into an eight-feature representation. Twelve candidate classifiers were evaluated under a deployment-aware protocol combining macro-averaged F1-score, cross-validation stability, and serialised model size. Regularised LightGBM achieved the best balance, with a test macro F1-score of 0.9813 and an exported C header of approximately 585 kB via `m2cgen`, making it suitable for direct integration into ESP32 firmware as a plain C module. Feature importance analysis confirmed that vertical acceleration and tilt angles are the primary discriminative signals, with rotational-speed features providing supplementary contributions for walking-class separation. The embedded firmware delivers local vibratory haptic feedback upon detection of any slouched posture class, and an MQTT–InfluxDB–dashboard stack supports optional cloud telemetry and visualisation.

Future work should address multi-subject generalisation, real-time latency measurement on the ESP32, and further model compression to reduce the C-header footprint while maintaining classification performance.

ACKNOWLEDGMENT

The authors thank the Bandung Institute of Technology for providing access to laboratory infrastructure used in hardware development and data collection.

REFERENCES

[1] N. de Langen and K. Peereboom. "Musculoskeletal disorders and prolonged static sitting." European Agency for Safety and Health at Work, OSHwiki, Accessed: Jun. 10, 2026. [Online]. Available: https://oshwiki.osha.europa.eu/en/themes/musculoskeletal-disorders-and-prolonged-static-sitting.

[2] K. Adhikari, "Computer vision based posture estimation and fall detection," Ph.D. dissertation, Bournemouth University, Jul. 2019. Accessed: Jun. 10, 2026. [Online]. Available: https://eprints.bournemouth.ac.uk/33227/1/ADHIKARI%2C%20Kripesh_Ph.D._2019.pdf.

[3] X. Huang, Y. Xue, S. Ren, and F. Wang, "Sensor-based wearable systems for monitoring human motion and posture: A review," *Sensors*, vol. 23, no. 22, p. 9047, 2023. DOI: 10.3390/s23229047. Accessed: Jun. 10, 2026. [Online]. Available: https://www.mdpi.com/1424-8220/23/22/9047.

[4] M. Nadeem, E. Elbasi, A. I. Zreikat, and M. Sharsheer, "Sitting posture recognition systems: Comprehensive literature review and analysis," *Applied Sciences*, vol. 14, no. 18, p. 8557, 2024. DOI: 10.3390/app14188557. Accessed: Jun. 10, 2026. [Online]. Available: https://www.mdpi.com/2076-3417/14/18/8557.