Detailed Summary: *Reliable ECG Anomaly Detection on Edge Devices for Internet of Medical Things Applications*

Mohamed Charfeddine
Summary prepared from: Hizem, M., Bousbia, L., Ben Dhiab, Y., Aoueileyine, M.O.-E., Bouallegue, R.

August 12, 2025

Abstract

This document summarizes the methods, experiments, and findings of "Reliable ECG Anomaly Detection on Edge Devices for Internet of Medical Things Applications." The original work proposes a TinyML edge-AI pipeline for real-time ECG anomaly detection using a convolutional neural network (CNN) and advanced model optimizations (pruning + quantization) to enable deployment on low-power embedded platforms (Raspberry Pi, Arduino) for continuous monitoring. Key results reported by the authors (dataset, main model metrics, and device-level measurements) are reproduced and discussed below.

1 Motivation and contributions

Continuous ECG monitoring is crucial for early detection of cardiac anomalies (arrhythmias, PVCs, ST changes). Centralized/cloud processing raises privacy, latency, and power concerns; TinyML on edge devices is a promising alternative. The paper's principal contributions:

- A pipeline (preprocessing, feature extraction, CNN model) trained on the MIT-BIH Arrhythmia dataset and tailored for TinyML deployment.
- Application of pruning and 8-bit quantization to fit the model on constrained devices and reduce power/latency while keeping competitive accuracy.
- A prototype using AD8232 + Arduino Nano for signal acquisition and a Raspberry Pi 4 for edge inference, with measured latency, throughput, and power estimates.

2 Dataset and preprocessing

Dataset: MIT-BIH Arrhythmia dataset (annotated ECG beats with multiple classes). The authors grouped beats into five classes used in training/validation.

Preprocessing highlights:

- Class imbalance correction with SMOTE (Synthetic Minority Over-Sampling Technique).
- Denoising via band-pass filtering (baseline wander removal, HF noise reduction).
- Min-max normalization to $[0, 1]$ and segmentation into overlapping windows: 10-second windows with stride = 1 s.

These steps are intended to improve signal quality and class balance prior to model training.

3 Feature engineering & model selection

- Features: time-domain and spectral features (e.g., HRV metrics, QRS duration), and time-frequency spectrograms (STFT) used to convert 1D ECG to 2D input for the CNN.
- Algorithms evaluated: CNN, Decision Tree, Random Forest, XGBoost. CNN showed the best classification metrics on the dataset and was selected for optimization.

3.1 CNN architecture and training details

(Authors' reported configuration)

- Input: spectrogram-like 2D representation from STFT of 1D ECG windows.
- Convolution layers with kernel size 3, max-pooling, dropout for regularization, followed by fully-connected layers and a softmax output for multi-class classification.
- Training hyperparameters: Adam optimizer, learning rate = 0.001, batch size = 32, epochs = 50.

4 Optimization for TinyML: pruning and quantization

Pruning: Magnitude-based weight pruning (post-training) to remove low-magnitude weights and reduce model size/compute.

Quantization: Post-training integer quantization (32-bit $\rightarrow$ 8-bit) via TensorFlow Lite to reduce memory footprint and accelerate inference on low-power CPUs.

Combined: Pruning followed by quantization produced the "optimized" model used for deployment. The authors report trade-offs between size, inference time, and accuracy.

5 Deployment setup (prototype)

- Acquisition: AD8232 analog front-end connected to Arduino Nano for ECG sampling.
- Transmission: Arduino $\rightarrow$ Raspberry Pi via UART (baud 9600 bps). Authors note a transmission per-byte effective delay ($\approx$ 1.04 ms/byte including start/stop), requiring buffering/optimized protocol to meet real-time constraints.
- Inference: Raspberry Pi 4 runs TensorFlow Lite interpreter for real-time anomaly detection and display/alert logic. The authors compare Raspberry Pi vs NodeMCU (ESP8266) characteristics in the paper.

6 Results (reproduced summaries)

Below we present the paper's key reported metrics (accuracy, model sizes, latency, energy). Note: the paper reports several performance figures for different experiment contexts (host vs target device); we reproduce them as reported.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
CNN	99	98	98	98
Decision Tree	78	73	76	93
Random Forest (RF)	98	97	98	98
XGBoost	97	97	97	97

Table 1: Algorithm performance comparison as reported in the paper (dataset: MIT-BIH).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Model size (KB)
Original	96.09	96.76	96.09	96.33	739.945
Pruned	92.90	95.86	92.90	93.97	739.945
Quantized	96.01	96.70	96.01	96.26	192.914
Optimized	92.89	95.83	92.89	93.96	193.359

Table 2: Host device metrics: accuracy vs model size after pruning/quantization (reported).

6.1 Model algorithm comparison (reported)

6.2 Host-device optimization metrics (reported)

6.3 Target device (Raspberry Pi 4) performance (reported)

7 Discussion and critical observations

- **Strong edge-suitability:** The reported quantized/optimized models reach very low inference latencies (sub-millisecond region) and micro-watt-level power estimates, which is promising for continuous monitoring.
- **Reported accuracy variations:** The paper reports different accuracy figures in different places (e.g., CNN 99% in some comparative tables but 92.89% for the optimized model on the Raspberry Pi). This reflects the common trade-off: compression reduces size/power but can reduce accuracy. The paper reports both host-device and target-device metrics—interpret results in that context.
- **Power reporting:** Power numbers appear to be estimated from CPU-utilization scaling factors rather than direct hardware energy-profiling (authors also mention future plans for hardware-based profiling). So treat the micro-watt figures as indicative rather than hardware-measured ground truth.
- **Limitations discussed by authors:** single dataset (MIT-BIH) and single primary target platform (Raspberry Pi). Authors propose future evaluation on more datasets and devices (Arduino, ESP32) and exploration of NAS and knowledge distillation.

8 Concise takeaways

1. TinyML (pruning + quantization + TFLite) can enable real-time ECG anomaly detection on accessible edge hardware with attractive latency/power trade-offs.
2. The optimized model reported here attains sub-millisecond inference and low CPU usage on a Raspberry Pi 4 while remaining performant; however, accuracy may drop relative to the uncompressed host model—this trade-off must be tuned for clinical use.
3. For rigorous energy claims, hardware profiling is needed (authors acknowledge this).

Model	Avg inf. time (ms)	Throughput (inf/s)	Avg CPU (%)
Original	0.333	3004.76	72.4
Pruned	0.333	3004.89	47.5
Quantized	0.192	5204.03	31.5
Optimized	0.200	5008.04	15.4

Table 3: Selected target-device metrics (Raspberry Pi 4) reproduced from the paper.

9 References

- Original paper summarized: M. Hizem, L. Bousbia, Y. Ben Dhiab, M.O.-E. Aoueileyine, R. Bouallegue, *Reliable ECG Anomaly Detection on Edge Devices for Internet of Medical Things Applications*, Sensors, 2025.
- MIT-BIH Arrhythmia Database (used by authors): Moody & Mark, PhysioNet.