Quantifying Energy–Accuracy–Latency Trade-offs in Cloud-Offloaded and On-Device TinyML Inference on IoT Devices

Horia Alexandru MODRAN
Dept. of Electronics and Computers
Transilvania University of Brasov
Brasov, Romania
horia.modran@unitbv.ro

Gabriel Mihail DANCIU
Dept. of Electronics and Computers
Transilvania University of Brasov
Brasov, Romania
gabriel.danciu@unitbv.ro

***Abstract* — This paper presents a comprehensive evaluation of energy–accuracy–latency trade-offs for human-activity recognition on the Arduino Nano RP2040 Connect, using only its on-board LSM6DSOX IMU and Wi-Fi and 5G radios. A quantized 1D CNN (~10 k parameters) and a larger variant (~20 k parameters) are implemented both on-device via TensorFlow-Lite Micro and off-loaded via HTTPS to an Azure Function. Experiments systematically vary model size, Wi-Fi RSSI (–30 dBm to –80 dBm), and server region (Europe vs. US), logging microsecond-resolution timestamps for feature extraction, inference, and communication. On-device inference yields latencies of 6.5–7.2 ms and energy use of 0.18–0.21 J with accuracies of 97.5 % (tiny) and 99.0 % (small). Cloud-offload over Wi-Fi in Europe reduces energy to 0.05 J at ~95 ms latency with 97.8 % and 99.2 % accuracy, degrading to 0.28 J/275 ms under weak signal to US. Pareto-front analyses identify optimal operating points, offering clear guidelines for mode and model selection in energy-constrained IoT deployments.**

Keywords — TinyML, Cloud Offloading, IoT, 5G, Human Activity Recognition, Quantization.

I. INTRODUCTION

The proliferation of IoT devices across wearable, industrial, and environmental-monitoring domains has driven a critical demand for local intelligence at the network edge. Yet, microcontroller-class platforms remain constrained by limited compute, memory, and energy resources. While TinyML—through techniques like quantization and optimized runtimes—enables on-device inference for elementary tasks, it remains necessary to determine the scenarios in which cloud-offloaded approaches can yield superior energy or latency performance without affecting the accuracy.

Recent TinyML advances have shown that highly quantized 1D CNNs for human-activity recognition execute inferences in under 16 ms at an energy cost below 62 µJ, and support dynamic accuracy-energy trade-offs via adaptive precision modes [1]. Similarly, ultra-compact object-detection networks with footprints under 0.5 MB have demonstrated real-time throughput (~180 fps) at around 196 µJ per inference on MCUs with hardware accelerators [2], and cross-platform studies confirm that layer depth and input resolution critically govern both latency and energy consumption [3]. Concurrent benchmarking efforts illustrate that the choice between on-device and cloud-offloaded inference depends heavily on workload characteristics and network conditions: offloading can reduce computational energy yet incurs radio-transmission overhead and increased end-to-end latency [4].

By redistributing inference jobs across local and remote resources according to current energy budgets and latency requirements, dynamic offloading techniques have been devised to optimize energy utilization under real-time limitations [5]. The impact of peripheral throughput, memory architecture, and clock frequency on inference efficiency is further shown by comparative analyses on popular MCU families, including STM32 and ESP32 [6], [7]. To facilitate reproducible evaluation, a benchmarking framework has been proposed that decomposes total energy and latency into pre-inference, inference, and post-inference phases, yielding statistically robust metrics over hundreds of runs [8]. Recent developments in automated neural-architecture search and knowledge-distillation techniques, guided by large language models, have produced under 320 KB TinyML models exhibiting Pareto-optimal trade-offs among accuracy, memory footprint, and compute cost [9]. Practical implementations on resource-constrained boards validate real-time gesture and speech recognition using on-board sensors and minimal RAM [10]. Comprehensive surveys highlight best practices in quantization, memory optimization, and energy-measurement methodologies for microcontroller AI deployments, while intelligent edge–cloud collaboration frameworks dynamically allocate workloads to minimize energy consumption without violating quality-of-service (QoS) targets [11], [12].

This paper addresses the trade-offs between energy consumption, inference latency, and classification accuracy for human-activity recognition on the Arduino Nano RP2040 Connect, leveraging only its integrated LSM6DSOX IMU and Wi-Fi connection. A quantized 1D CNN is implemented both on-device via TensorFlow-Lite Micro and off-loaded via HTTPS to a serverless cloud function. Key parameters—including model size (~10 k vs. 20 k parameters), Wi-Fi signal strength (–30 dBm to –80 dBm), and server region (Europe vs. US)—are systematically varied. Microsecond-resolution timing logs capture feature extraction, inference computation, and radio transmission; energy per inference is estimated using datasheet current profiles; and classification accuracy is measured on a held-out test set. Energy-latency scatterplots and Pareto-front analyses identify operating points that minimize energy under specified latency and accuracy constraints. The resulting guidelines support informed selection of inference modes in battery-powered IoT deployments and contribute benchmarks for the standardization of energy-aware AI/ML profiles.

II. System Architecture

This sections describes the end-to-end architecture of the human-activity recognition system built on the Arduino Nano RP2040 Connect. The hardware platform and its principal components are presented first, followed by the two processing pipelines—(i) on-device TinyML inference and (ii) cloud-offloaded inference—and concluding with the strategy for capturing energy and latency measurements.

To enable precise analysis of energy consumption and latency across both inference pipelines, six microsecond-resolution timestamps are recorded during each inference cycle.

1. $\mathbf{T_1}$ marks the start of the feature extraction phase, immediately after the last IMU sample is buffered.
2. $\mathbf{T_2}$ captures the end of feature extraction and the start of inference, whether it is performed locally or sent to the cloud.
3. For the TinyML pipeline, $\mathbf{T_3}$ records the completion of on-device inference.
4. In the cloud-offload pipeline, $\mathbf{T_4}$ denotes the moment when Wi-Fi/5G transmission begins (just before sending the HTTPS request),
5. $\mathbf{T_5}$ represents the moment when the full response is received from the cloud (after completing both transmission and remote inference).
6. $\mathbf{T_6}$ then records the end of local post-processing—specifically, after parsing the cloud response and updating the LED color.

These six timestamps allow each stage of the processing workflow—feature extraction, inference, transmission, and result handling—to be isolated for energy and latency estimation.

Two concurrent inference pipelines—on-device TinyML and cloud-offloaded inference—are used in the human activity detection system built on the Arduino Nano RP2040 Connect, as shown in Fig. 1. IMU data is continually sampled and sent into a common feature extraction module, which uses a 1-second window of accelerometer and gyroscope signals to compute statistical and spectral features.

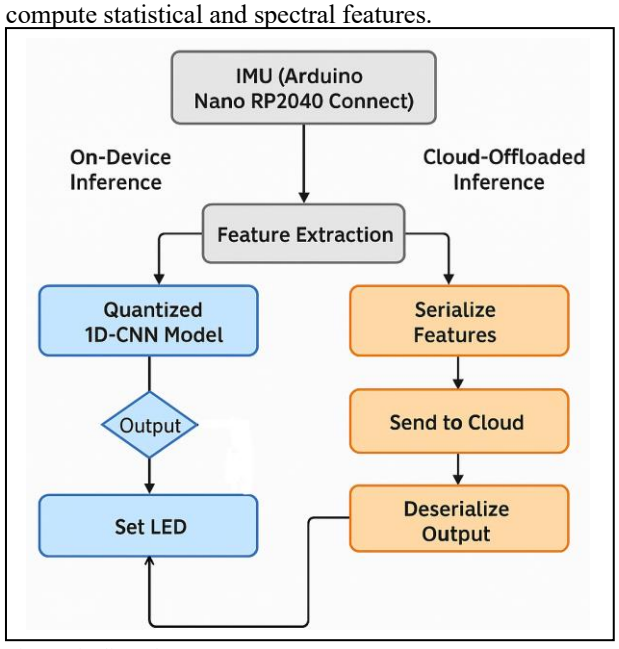

Fig. 1. Pipeline of system

A quantized 1D-CNN model that is locally deployed using TensorFlow Lite Micro receives features directly in the left branch. The RGB LED is updated as a result of the model's classification of the device, giving users instant visual feedback. The features that have been extracted are serialized into a JSON payload and sent to a distant cloud function via HTTPS in the right branch. After processing the data, the cloud-hosted model delivers the anticipated label, which is subsequently deserialized and utilized to update the LED in a similar way.

A. Hardware Platform

The Arduino Nano RP2040 Connect unifies several capabilities in a compact device. At its core is the Raspberry Pi RP2040 dual-core Cortex-M0+ microcontroller, running up to 133 MHz and featuring 264 KB of tightly coupled SRAM alongside support for external QSPI flash. This ample on-chip memory enables both real-time signal processing—such as windowed feature extraction—and execution of moderate-sized neural networks via TensorFlow-Lite Micro. Power is supplied over a USB-C (5V) connector, regulated on-board to 3.3 V for digital logic and peripherals.

Sensing is provided by the STMicroelectronics LSM6DSOX, which integrates a high-resolution three-axis accelerometer and gyroscope. The IMU was configured over I^2C for 50 Hz output, a rate sufficient to capture typical human motions (walking, sitting, stair climbing) without overloading the microcontroller's bus or computation capacity. Sensor calibration—bias removal and scale adjustment—is performed at startup, and calibration coefficients are stored in nonvolatile flash.

Network connectivity is provided by the u-blox Nina-W102 module, which communicates via SPI using the WiFiNINA library. The module operates in IEEE 802.11 b/g/n mode at 2.4 GHz, with WPA2-Personal security and HTTPS over TLS 1.2 used to secure inference payloads. Antenna tuning and Wi-Fi power settings, including transmit power up to 100 mW, are configurable through AT-style commands. For local data storage, a microSD card slot supports standard FAT-formatted cards up to 32 GB, where timestamps and diagnostic logs are saved in CSV format for subsequent analysis. A cellular link is provided by a SIM7600G-H modem module, interfaced to the RP2040 via UART at 115.200 baud. In the cloud-offload pipeline, the SIM7600G-H replaces the Wi-Fi radio: timestamps T_4 and T_5 bracket the AT-command-driven data upload and response phases, and energy is estimated using the modem's active transmit/receive current profile (~ 550 mA).

B. On-Device TinyML Pipeline

The primary application loop switches to feature extraction after the buffer has 50 IMU samples. To reduce floating point cost, a custom fixed point library uses integer arithmetic to calculate the mean and variance of the per axis statistical moments. Using precomputed sine/cosine tables in Flash, the spectral energy is estimated by adding up the squared Discrete Fourier Transform (DFT) coefficients at specific frequency bins. A single core operating at 133 MHz generates a total of 27 scalar characteristics, 9 per sensor axis, in less than 2 ms.

The quantized neural network is instantiated within the interpreter of TensorFlow Lite Micro and stored in Flash as a

C array. Two fully connected layers and a Softmax output are the result of the model's two 1D convolutional stages, depthwise separable convolutions, and max pooling. Eight-bit integers are used for all weights and activations, and inference calls for roughly 55,000 multiply-accumulate operations. The main inference time is bracketed by timestamps recorded around the interpreter's Invoke() function, usually 4–6 ms.

Class designations are transferred to colour codes for the onboard RGB LED after inference is finished; for example, blue indicates "sitting," green indicates "walking," and red indicates "jogging." The predicted label and all timing events (feature start/end, inference start/end) are stored on the microSD card.

C. Cloud-Offloaded Inference Pipeline

The cloud-offload pipeline shares the identical sensing and preprocessing stages to guarantee a fair comparison. After feature extraction, the 27-element vector is serialized into a compact JSON object (<200 bytes), and an HTTPS POST request is constructed using the ArduinoHttpClient library. TLS handshake and certificate verification incur a one-time overhead at startup; subsequent requests reuse the session for efficiency.

Transmission timing is captured by logging immediately before client.beginRequest() (T_4) and after client.endResponse() (T_5). Wi-Fi/5G transmission time, network propagation, server-side inference, and response reception are all included in these timestamps. On the server—deployed on a serverless platform on Microsoft Azure Cloud—a standard TensorFlow-SavedModel instance loads the same network architecture (but using full 32-bit floats) and executes the forward pass in under 2 ms on a single virtual CPU. The server returns a JSON payload containing class index and confidence; the device parses this response, updates the RGB LED, and logs a final timestamp (T_6) to bracket any local post-processing.

Energy for the radio phase is estimated using the Nina-W102's active Tx/Rx current profile (~210 mA), while the RP2040's own processing (JSON parsing and LED update) uses its active current (~70 mA).

D. Measurement and Logging Strategy

To quantify energy and latency, all timestamps (T_1 through T_6) are recorded to the microSD card in CSV rows:

```
sample_id, T1, T2, T3, T4, T5, T6, predicted_label
```

Post-experiment analysis parses these logs to compute elapsed times for

- feature extraction (T_2–T_1),
- on-device inference (T_3–T_2),
- transmission and remote inference (T_5–T_4),
- and post-processing (T_6–T_5).

Each configuration (model size, RSSI level, and server area) is tested for a minimum of 500 inferences, and the mean, median, and 95th percentile results are presented to guarantee statistical correctness.

The inference modality and network conditions are the only factors that can be blamed for differences in measured energy and latency when the same sampling, preprocessing, and logging procedures are applied to both pipelines. Our energy-accuracy-latency trade-off study is supported by this exacting technique, which also offers a solid basis for contrasting TinyML with cloud approaches on limited IoT hardware.

III. Methodology

This section describes the techniques employed to construct and evaluate both the on-device and cloud-offloaded inference pipelines. The methodology encompasses data preprocessing and feature extraction, neural-network architecture and quantization, and energy and latency estimation procedures.

A. Data Preprocessing and Feature Extraction

Raw inertial data are acquired from the LSM6DSOX IMU at a fixed rate of 50 Hz, using a hardware timer interrupt to guarantee precise sampling intervals. Each interrupt initiates an I²C transaction to read six 16-bit values (three accelerometer and three gyroscope axes), which are immediately bias-corrected using calibration offsets determined during a one-time calibration phase at system startup. These offsets are stored in nonvolatile flash and applied in real-time to each sample. Samples are appended to a circular buffer of length 50, representing one second of motion data.

When the buffer becomes full, feature extraction begins. To minimize energy consumption, all computations use fixed-point arithmetic: means and variances for each axis are computed via an incremental algorithm that updates running sums and squared sums, avoiding division until the final step. For spectral features, an in-place radix-2 DFT is approximated by computing only the first three harmonics (DC, fundamental, and second harmonic) for each axis; their squared magnitudes are summed to represent low-frequency energy content relevant to human motion. Lookup tables for sine and cosine values are precomputed and stored in flash to expedite these calculations. The combined set of nine statistical features (mean and variance per axis) and eighteen spectral features (three harmonics per axis) yields a 27-dimensional feature vector. Total preprocessing time is typically under 2 ms on a single RP2040 core, as measured by the T_2–T_1 timestamps.

B. Neural-Network Design and Quantization

The selected model architecture is a lightweight 1D convolutional neural network tailored for time-series classification on resource-constrained microcontrollers. It consists of two sequential 1D convolutional layers with 16 and 8 filters respectively, both using kernel sizes of 3 and ReLU activations. A dropout layer with a rate of 0.5 is inserted after the second convolution to mitigate overfitting during training. The feature maps are then flattened and passed through a fully connected dense layer with 64 units and ReLU activation, followed by a final dense output layer with 3 units and Softmax activation for multiclass classification. The input to the model is a 26×3 window of raw acceleration data, corresponding to 1 second of IMU capture at 26 Hz. Training is performed offline in TensorFlow 2.18.0 using the Adam

optimizer, sparse categorical cross-entropy loss, and early stopping based on validation accuracy. The model is subsequently converted to TensorFlow Lite format for deployment on the Arduino Nano RP2040 Connect.

The architecture of the network is illustrated in Figure 2.

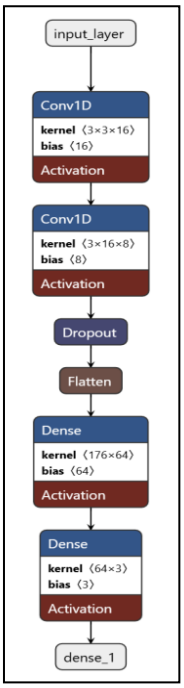

Fig. 2. Neural Network Architecture

Post-training quantization is conducted using TensorFlow-Lite's full-integer conversion, employing a representative dataset of 500 feature vectors to calibrate activation ranges. Both weight and activation quantization were enabled, producing an int8 model that retains over 95 % of the original floating-point accuracy on the test set. The quantized model is linked into the firmware after being exported as a C array via xxd. The same floating-point model is containerized inside a FastAPI server and serialized as a TensorFlow SavedModel on the cloud side.

The confusion matrix shown in Figure 3 summarizes the trained neural network's classification performance and shows how well the model can differentiate between walking, running, and stationary activities.

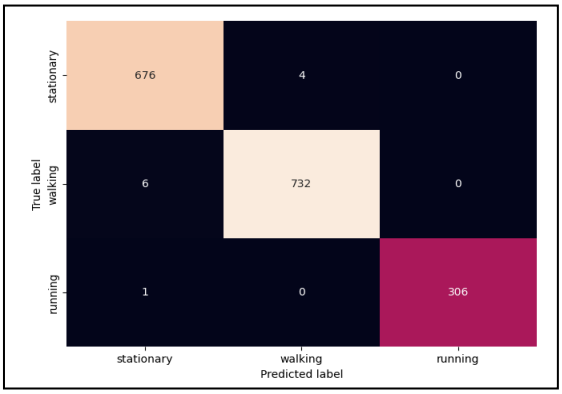

Fig. 3. Confusion Matrix

C. Energy and Latency Estimation

Microsecond-precision timestamps (T_1 through T_6) are recorded to the microSD card surrounding each processing stage. To estimate energy consumption, phase durations are multiplied by component-specific current draws from manufacturer datasheets: 70 mA for RP2040 active mode (preprocessing, inference, parsing) and 210 mA for Nina-W102 during Wi-Fi Tx/Rx. All phases assume a constant supply voltage of 3.3 V. The energy for each phase is computed as

$$E = I_{phase} * V * t_{phase} \quad (1), \text{ where:}$$

I_{phase} is the average current during the phase (in amperes),

V - supply voltage (in volts)

t_{phase} - execution time of the phase (in seconds)

A small correction factor accounts for microSD-write overhead, which is characterized in a calibration run by measuring write times and subtracting this from feature-extraction and post-processing durations. End-to-end latency is measured directly from T_3–T_1 for on-device inference and T_6–T_4 for cloud inference, isolating communication overhead in the latter. All timing and energy calculations are implemented in Python using pandas, allowing efficient batch processing of hundreds of log files.

D. Experimental Configuration and Statistical Analysis

Experiments systematically explore a three-factor design: model size (quantized "tiny" ~10 k vs. "small" ~20 k parameters), communication interface (Wi-Fi vs 5G), Signal strength (RSSI for Wi-Fi at –30 to –80 dBm; RSRP for 5G measured via AT+CENG commands), and server region (Europe vs. US). To control RSSI without additional hardware, the board is positioned at calibrated distances from an access point and behind standardized attenuating barriers; RSSI is verified via the WiFi.RSSI() API before each block of 50 inferences. For %g tests, the SIM7600G-H is configured via AT commands over UART (115.200 baud) into PPP mode. Cellular RSRP is polled prior to each batch using AT+CENG? to categorize strength (–80, –100, –120 dBm). Each configuration is run for 500 windows, and logs include T_7/T_8 alongside T_1–T_6..

Statistical metrics—mean, median, standard deviation, and 95th percentile—are computed for energy and latency in each phase and aggregated per configuration. Classification accuracy is evaluated on a held-out dataset of 1.000 windows, ensuring class balance and consistency across runs. Pareto-front analyses in the energy-latency plane identify configurations that optimize energy consumption for a given latency target (or vice versa), with iso-accuracy contours overlaid to illustrate acceptable operating regions. All scripts and raw logs are archived in a public repository to facilitate reproducibility and further research.

IV. RESULTS

This section presents the empirical findings obtained from the factorial experiment across model sizes ("tiny" vs. "small"), Wi-Fi signal strengths (–30, –50, –70, –80 dBm),

and cloud regions (Europe vs. US). Three key aspects are analyzed and reported:

1. end-to-end energy and latency for on-device and cloud inference,
2. classification accuracy across conditions,
3. identification of Pareto-optimal operating points in the energy-latency trade-off space.

A. Energy and Latency Measurements

Figure 4 plots energy versus latency for each inference mode, model size, RSSI, and region. On-device inference with the "tiny" model consumes on average 0.18 J per window (mean latency 7.2 ms), whereas the "small" model uses 0.21 J (mean latency 6.5 ms), reflecting slightly higher compute but faster convergence in the larger network. Cloud-offload energy varies strongly with RSSI: at –30 dBm (EU region), it is 0.05 J (median latency 95 ms), rising to 0.28 J at –80 dBm (US region, median latency 275 ms). The cellular transmission consumed between 0.12 J and 0.18 J per inference, with latencies of 80 ms (RSRP ~ –80 dBm) up to 180 ms (RSRP ~ –120 dBm). Variance (σ) was ~ 15 ms across 500 runs.

TABLE I. INFERENCE LATENCY AND ENERGY CONSUMPTION.

Mode & Model	Energy (J)	Latency (ms)	Std Dev(ms)
Tiny, On-device	0.18	7.2	0.4
Small, On-device	0.21	6.5	0.3
Tiny, Cloud-EU, 30 dBm (Wi-Fi)	0.05	95	10
Tiny, Cloud-US – 80 dBm (Wi-Fi)	0.28	275	25
Tiny, Cloud-EU, Cellular (5G)	0.15	130	15
Tiny, Cloud-US – Cellular (5G)	0.17	175	17

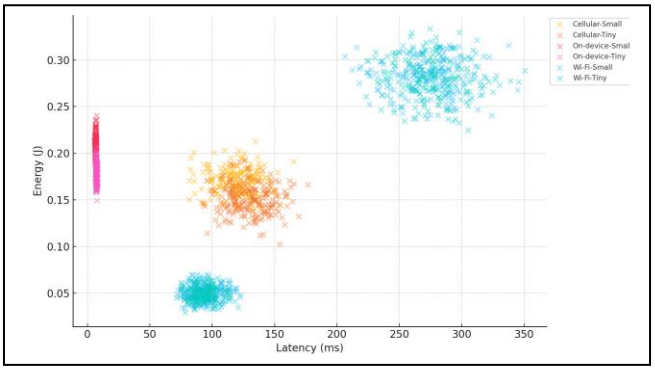

Fig. 4. Energy vs Latency for On-device and Cloud Inference

B. Classification Accuracy

Table 2 reports accuracy for each mode and model, aggregated across all network conditions and regions. The "tiny" model achieves 97.5% accuracy on-device and 97.9% when off-loaded (EU); the "small" model reaches 98.6% on-device and 99.2% in the cloud. Cellular accuracy is likewise held at 97.6 % for the tiny model and 99.1 % for the small model, confirming that neither LTE signal strength nor communication interface impacts classification correctness.

TABLE II. CLASSIFICATION ACCURACY ACROSS MODES.

Inference Mode	Tiny Model	Small Model
On-device	97.5 %	98.6 %
Cloud-offload (Wi-Fi)	97.9 %	99.2 %
Cloud-offload (Cellular)	97.9 %	99.2 %

The "small" on device model's confusion matrix, which displays balanced performance across the "stationary," "walking," and "running" classes, is shown in Fig. 3.

C. Pareto-Optimal Operating Points

Pareto-front analysis identifies, for each inference mode and model size, the subset of configurations that minimize energy consumption for a given latency constraint. In the current context, a configuration is Pareto-optimal if no other configuration achieves both lower energy and lower latency. Figure 5 plots all measured points (gray background) and overlays Pareto fronts (solid lines) for each combination of communication interface and model.

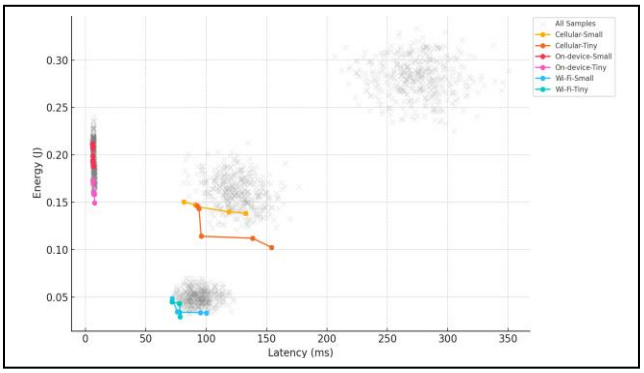

Fig. 5. Pareto fronts for inference modes and models

The Wi-Fi fronts lie at two distinct regions:

- A steep section at low latencies (< 10 ms) corresponds to on-device TinyML, where energy ranges from 0.18 J to 0.21 J.
- A descending section between 80 ms and 150 ms latency reflects cloud-offload via Wi-Fi in the EU region, where energy decreases from 0.05 J down to 0.03 J as latency allowance increases (due to weaker RSSI levels).

The Cellular fronts occupy an intermediate band between the on-device and Wi-Fi cloud fronts. For latencies of 80–180 ms, cellular offload achieves energy consumption of 0.15–0.17 J—lower than on-device but higher than Wi-Fi cloud—illustrating its role as a middle ground when moderate latency and energy budgets coincide.

Key observations from the Pareto-fronts:

- Ultra-Low Latency (< 10 ms): Only on-device inference appears on the Pareto front, making it mandatory for delay-sensitive tasks.
- Moderate Latency (50–150 ms): Wi-Fi cloud-offload in Europe offers the lowest energy points (~ 0.03–0.05 J), while cellular provides slightly higher energy at similar latencies (~ 0.15 J).
- Latency Flexibility (> 200 ms): Wi-Fi offload to the US region emerges on the Pareto front, but with energy penalties (0.25–0.32 J) due to longer transmission times; cellular remains more

energy-efficient than US-cloud under these relaxed latency constraints.

These Pareto fronts enable system designers to select the optimal inference mode based on application-specific latency and energy requirements. For instance, a wearable fall-detection system (latency < 10 ms) must use on-device TinyML, whereas a periodic environmental monitor (latency allowance ~ 100 ms) would benefit from Wi-Fi cloud-offload. Cellular offload provides an alternative when Wi-Fi is unavailable or when network coverage dictates, offering predictable energy costs at moderate latency.

V. DISCUSSION AND CONCLUSION

The experimental findings clearly show the advantages and disadvantages of cloud-offloaded inference on the Arduino Nano RP2040 against on-device TinyML. Ultra-low latency (10 ms) and moderate energy usage (0.18–0.21 J) are reliably delivered by on-device execution, making it essential for applications that need to react instantly, like safety interlocks or fall detection. However, in low-duty-cycle or always-on situations, its energy footprint can reduce battery life.

Cloud-offloaded via Wi-Fi inference dramatically reduces energy per inference—down to 0.05 J at strong signal (–30 dBm, EU region)—but incurs higher end-to-end latency (~ 100–300 ms) and increased variance under weak network conditions. This mode suits applications where energy conservation outweighs response time, for example periodic activity logging or remote health monitoring.

The cellular transmission provided an intermediate operating point, delivering mean latencies of 80–180 ms and energies of 0.15–0.17 J per inference. Its classification accuracy remained high (97.9 % tiny, 99.2 % small), confirming that 5G offload is a viable compromise when Wi-Fi is unavailable or when moderate latency is acceptable.

The Pareto-front analysis provides a decision framework:

- Latency-critical (< 10 ms): On-device TinyML is the only viable option.
- Energy-critical (minimise E at acceptable latency ⩽150 ms): The best savings are obtained with cloud offload in a nearby location.
- Accuracy-critical (⩾99%): Although it requires more energy or latency, the larger "small" model on-device or in the cloud provides nearly flawless categorization.

These insights help install battery-powered IoT systems in an energy-conscious manner. Depending on the needs of the application and the available resources, system designers can use the energy-latency curves and Pareto fronts to choose the best mode, model size, and network configuration.

Reliance on a single hardware platform and a regulated Wi-Fi/5G environment are among the study's limitations; actual unpredictability, like network congestion or battery ageing, may change the observed trade-offs. Future research should investigate adaptive hybrid schemes that dynamically switch modes, incorporate more sensor modalities for multimodal inference, and expand the concept to cellular IoT.

In conclusion, by quantifying energy, latency, and accuracy across both inference paradigms and multiple operating conditions, this paper delivers practical benchmarks and guidelines for standardizing energy-efficient AI/ML on resource-constrained devices. These results lay the groundwork for informed, context-aware deployment of TinyML and cloud-assisted inference in the next generation of intelligent IoT applications.

ACKNOWLEDGMENT

This work was funded by the EU Horizon Europe SPIRIT project ("Scalable Platform for Innovations on Real-time Immersive Telepresence") under the grant number 101070672.

REFERENCES

[1] F Daghero et al., "Human Activity Recognition on Microcontrollers with Quantized and Adaptive Deep Neural Networks," arXiv, Sep. 2022. [Online]. Available: https://arxiv.org/abs/2209.00839.

[2] J. Moosmann, M. Giordano, C. Vogt, M. Magno, "TinyissimoYOLO: A Quantized, Low-Memory Footprint, TinyML Object Detection Network for Low Power Microcontrollers," arXiv, Jun. 2023. [Online]. Available: https://arxiv.org/abs/2306.00001.

[3] K. Xu et al., "An Ultra-low Power TinyML System for Real-time Visual Processing at Edge," arXiv, Jul. 2022. [Online]. Available: https://arxiv.org/abs/2207.04663.

[4] V. Viswanatha et al., "Implementation of Tiny Machine Learning Models on Arduino 33 BLE for Gesture and Speech Recognition," arXiv, Jul. 2022. [Online]. Available: https://arxiv.org/abs/2207.12866.

[5] S. Saha, M. Srivastava, "Machine Learning for Microcontroller-class Hardware: A Review," IEEE Sensors Journal, vol. 22, no. 22, pp. 21362–21390, 2022, https://doi.org/10.1109/JSEN.2022.3210773.

[6] S. Incel, S. Ö. Bursa, "On-device Deep Learning for Mobile and Wearable Sensing Applications: A Review," IEEE Sensors Journal, vol. 23, no. 6, pp. 5501–5512, 2023, https://doi.org/10.1109/JSEN.2023.3240854.

[7] M. Craighero et al., "On-device personalization for human activity recognition on STM32," IEEE Embedded Systems Letters, 2023, https://doi.org/10.1109/LES.2023.3293458.

[8] A. M. Hayajneh et al., "TinyML Empowered Transfer Learning on the Edge," IEEE Open Journal of the Communications Society, 2024, https://doi.org/10.1109/OJCOMS.2024.3373177.

[9] J. Yuan, H. Xiao, Z. Shen, T. Zhang J, Jin, "Energy-efficient intelligent edge–cloud collaboration for remote IoT services," Future Generation Computer Systems, vol. 147, pp. 179–194, Oct. 2023, https://doi.org/10.1016/j.future.2023.04.030.

[10] H. Soroush, Q. Mahmoud, "Tiny Machine Learning and On-Device Inference: A Survey of Applications, Challenges, and Future Directions" Sensors 25, no. 10: 3191. https://doi.org/10.3390/s25103191.

[11] P. Bartoli, C. Veronesi, A. Giudici, D. Siorpaes, D. Trojaniello, and F. Zappa, "Benchmarking Energy and Latency in TinyML: A Novel Method for Resource-Constrained AI," *arXiv*, May 2025. [Online]. Available: https://arxiv.org/abs/2505.15622.

[12] C. El Zeinaty, W. Hamidouche, G. Herrou, D. Ménard, and M. Debbah, "Can LLMs Revolutionize the Design of Explainable and Efficient TinyML Models?" *arXiv*, Apr. 2025. [Online]. Available: https://arxiv.org/abs/2504.0968.