Acquisition and analysis of data through IEEE 802.15.4 (2.4GHz)

Wei Long
TU Dresden
wei.long@mailbox.tu-dresden.de

Arian Majidi
TU Dresden
arian.majidi@mailbox.tu-dresden.de

Ashvinviknes Saravanaperumal
TU Dresden
ashvinviknes.saravanaperumal@mailbox.tu-dresden.de

ABSTRACT

We investigate whether low-level link-quality telemetry from IEEE 802.15.4 radios—Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI)—is sufficient to automatically recognize deployment environments and, in a second task, to distinguish transmitting nodes in a small wireless-sensor network. Our raw data consist of per-frame RSSI/LQI pairs emitted by RIOT OS packet logs on nRF52840-based nodes running 802.15.4 at 2.4 GHz, recorded as timestamped text streams (e.g., "-71, 88") for each TX/RX session. We convert these logs into fixed-length windows via a reproducible pipeline: (i) per-file differencing (RSSI, LQI), (ii) min–max normalization and gentle IQR clipping for robustness, and (iii) 10s frames with 50% overlap, resampled to a common length by time-axis interpolation. Then, we use two deep learning models to uniquely classify deployment environments and additional configuration types based on link quality metrics. These are a convolutional neural network (CNN) and a ResNet model. CNN is used as a reference point and ResNet is built to overcome some of the limitations (degradation) of the CNN model. Finally, we evaluate the resulted plots and confusion matrices to deduce the outcome of our experiment.

1 INTRODUCTION

Wireless-sensor networks (WSNs) often operate in dynamic, partially observed conditions. Our target platform is an nRF52840-sense node running RIOT OS with IEEE 802.15.4 enabled; the stack provides RSSI and LQI for every received MAC frame, since both metrics are measured by the radio at the PHY/MAC layers. The practical appeal is obvious: we can instrument a deployment without extra sensors, logging only the per-packet "signal-quality" pairs produced during routine communication (e.g., "-79, 56", "-83, 40"). We study two supervised tasks:

(1) **Environment recognition (5 classes).** Sessions are collected in multiple real-world environments. We must learn invariant temporal patterns in RSSI/LQI dynamics that correlate with the environment despite nuisance factors such as TX/RX geometry or short-term fading.
(2) **Transmitting-node identification (A/B/C).** Given sequences recorded at the receivers, we ask whether traffic originated from node A, B, or C—useful for diagnostics and security but more confounded by topology and traffic scheduling.

A consistent, reproducible preprocessing pipeline is critical. We (a) differentiate RSSI/LQI to emphasize local dynamics; (b) normalize per file to remove scale; (c) apply light IQR clipping to reduce outlier leverage; and (d) segment into overlapping 10 s frames that are interpolated onto a fixed grid, yielding uniform tensors suitable for deep models. This design preserves temporal shape while controlling for rate variability and missing timestamps present in the raw logs. For modeling, we adopt a compact multi-scale ResNet1D to capture patterns at multiple temporal receptive fields and compare against a lighter CNN reference.

2 ACQUISITION AND SCENARIOS

We measured wireless link quality in five real-world outdoor settings using three sensor nodes (A, B, C). Each node periodically transmitted IEEE 802.15.4 frames; the receivers logged per-frame RSSI and LQI together with a timestamp. This gave us a time series per link that reflects both path loss and short-term fading.

2.1 Environments

We collected data in five distinct environments, used as the target classes throughout the project:

(1) Garden as can be seen in Figure 2
(2) Lake in Figure 3
(3) Forest in Figure 4
(4) Campus in Figure 5
(5) Bridge in Figure 6

2.2 Session design

For each environment, we recorded 1.5 hours of traffic, split into three 30-minute segments so every node served as the transmitter once:

- 0–30 min: A → (B, C) receive
- 30–60 min: B → (A, C) receive
- 60–90 min: C → (B, A) receive

This rotation yields balanced coverage of all three directed links in every setting while keeping transmitter/receiver roles disjoint within each segment.

2.3 What we log

At the receivers, each successfully decoded frame is logged as a single row with:

- timestamp — receive time
- rssi — received signal strength (dBm)
- lqi — link quality indicator (unitless)

2.4 File naming and organization

Raw logs are organized per environment and link using a consistent standardized naming system (these files can be found in GitHub).

<TX>_<RX>_<ENV>.txt

Examples:

- A_B_1.txt → A transmitted, B received, Garden
- C_A_4.txt → C transmitted, A received, Campus

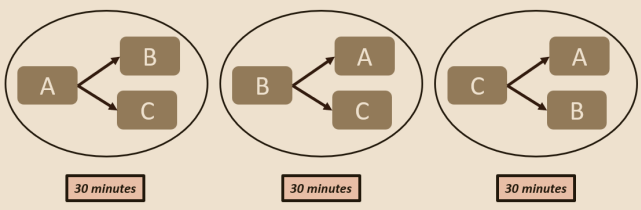

Figure 1: Nodes' formation and timing in each environment

Figure 2: First environment: Garden

Figure 3: Second environment: Lake

Downstream, after TXT → CSV normalization and cleaning, the same stems appear as *_cleaned.csv. We also maintain a simple metadata table that carries the stem (file), TX, RX, and the numeric environment label (Env ∈ {1..5}).

This acquisition plan produces five environment classes × three directed links per segment × three segments per environment, with 7.5 hours of total recording across all scenarios, ready for reproducible preprocessing and modeling.

Figure 4: Third environment: Forest

Figure 5: Fourth environment: Campus

Figure 6: Fifth environment: Bridge

3 DATASET AND PROCESSING

3.1 Cleaning (TXT → tidy time series)

Input format. Each file is named <TX>_<RX>_<ENV>.txt (e.g., A_B_4.txt) and contains timestamped RSSI and LQI readings. Here's how we proceed:

(1) **Robust parsing.** Accept commas, tabs, or variable whitespace; ignore BOMs and comment lines. We auto-detect the three columns of interest:
   - Time (timestamp, time, or ts)
   - RSSI (rssi, rssi_dbm, etc.)
   - LQI (lqi)
(2) **Time normalization.** Convert the time column to UTC milliseconds (timestamp_ms) using a single parser so mixed formats do not break the pipeline.
(3) **Basic sanitization.**
   - Drop rows with missing or non-finite values.
   - Sort by time and deduplicate exact repeats to remove logger hiccups.
   - Apply light sanity checks and discard obviously corrupted values (e.g., nonsensical RSSI/LQI).

Output. One clean CSV per input: *_cleaned.csv with exactly three columns: timestamp, rssi, lqi.

3.2 Window Strategy

Settings used for processing:

- Length: 10 seconds per window
- Overlap: 50% (hop = 5 seconds)
- Uniform grid: 100 samples per window (≈10 Hz rate)
- Interpolation: linear (windows with fewer than 2 points are skipped)

3.3 Data Processing

After producing the cleaned CSV files, we apply several processing steps to make the data suitable for modeling. The aim is to emphasize temporal changes in link quality, remove scale differences, and reduce the impact of outliers.

(1) **Differencing.** To minimize the effect of transmission distance and antenna orientation, we calculate the differences of the raw RSSI and LQI values:

$$y_i = x_{i+1} - x_i \tag{1}$$

(2) **Normalization.** Since RSSI (dBm) and LQI (unitless) have different units and ranges, we scale each differenced series to the range [0,1] using min–max normalization:

$$z_i = \frac{y_i - \min(y)}{\max(y) - \min(y)} \tag{2}$$

(3) **Outlier mitigation.** To reduce the leverage of rare spikes while preserving overall shape, we apply Tukey's IQR rule to normalized sequences. Let Q_1 and Q_3 be the first and third quartiles, we clip the values into (with a constant k = 1.5):

$$\left[\, Q_1 - k(Q_3 - Q_1),\ Q_3 + k(Q_3 - Q_1) \,\right] \tag{3}$$

After these operations, the time series are segmented into overlapping frames as described in **Section 3.2**. Thus, each frame represents a normalized and distance-independent fluctuation pattern of RSSI and LQI, serving as a standardized input to subsequent learning models.

4 MODEL TRAINING

4.1 Model Architectures

We implemented two deep learning architectures suitable for time-series classification of RSSI/LQI frames:

(1) **SimpleCNN1D:** A 3-layer CNN composed of convolutional blocks with ReLU activations and max pooling, followed by a global average pooling layer and a fully connected output layer. It processes each frame (shape: $C \times L$, where C = 1 or 2) using temporal convolutions.
(2) **ResNet1D:** A residual convolutional neural network inspired by ResNet architectures. It includes an initial stem (convolution, batch normalization, and pooling), followed by three residual blocks and a global average pooling layer. Skip connections allow better gradient flow and stabilize deeper training.

Both models are implemented using PyTorch and support input of different channel configurations: RSSI-only, LQI-only, or both.

4.2 Scenario Definitions

The training tasks are defined by two classification scenarios, following the project guidelines:

(1) **Scenario I: Environment Classification** The model learns to distinguish between different deployment environments (e.g., garden, lake, campus, etc.)
Labels: Environment IDs (1–5)
Input: All node pairs are considered together; label is based on env.
(2) **Scenario II: Node Classification** The model learns to classify the receiver node based on RSSI/LQI patterns.
Labels: Receiving Nodes (A, B, C)
Input: All environments are merged; label is based on rx.

The model is trained separately for each scenario with shared architecture choices and hyper-parameters.

4.3 Seen vs. Unseen Data Splits

We evaluate generalization in two settings:

(1) **Seen Split:** Data are randomly split across all available samples (75%/25% train/test). The test set includes samples from the same environments or nodes seen during training.
(2) **Unseen Split:** This setup simulates deployment generalization. The model is trained without access to specific nodes or environments:
   - **Scenario I:** One node (e.g., Node C) is completely excluded during training. The test set consists only of its data.
   - **Scenario II:** For each node, data from one environment (e.g. Env 5) are excluded and used solely for testing.

These configurations are selectable via configuration flags and controlled in code via logic branching (HELD_OUT_NODE, HELD_OUT_ENV).

4.4 Training logic

Our training procedure is based on a hold-out split, where the dataset is randomly divided into training and test sets according to

a fixed ratio (e.g., TEST_SIZE=0.25). This provides a consistent partition for all experiments and ensures comparability across different configurations.

The experiments were organized as a systematic exploration of different configurations, referred to as combinations (combos). Each combo is defined by a scenario (I: environment classification, II: node classification), a split strategy (seen or unseen), the input channel, and the model type (CNN or ResNet). With respect to input channels, we restricted the analysis to RSSI and RSSI+LQI. The use of LQI alone was excluded because in our dataset LQI strongly correlates with RSSI, leading to nearly identical values and therefore offering no additional discriminative information when used in isolation.

In the seen case, training and test data are drawn from the same dataset with a fixed ratio (e.g., 75/25). In the unseen case, training and testing originate from disjoint sources, such as unseen nodes in Scenario I or unseen environments in Scenario II.

Training was conducted with fixed epochs per run and multiple runs per combo. Instead of training for the full epoch budget in a single run, the process was divided into shorter runs that resume from the checkpoint of the previous run. For example, 10 runs of 5 epochs each correspond to 50 epochs of continuous training. This design allows us to monitor accuracy changes across runs, store checkpoints and confusion matrices after each run, and identify the best-performing run for each configuration. In addition, a global best confusion matrix is maintained to reflect the highest test accuracy achieved across all runs.

5 EVALUATION, INSIGHTS AND INFERENCE

5.1 Test Accuracy Curve

In Scenario I (environment classification), the curves plot the best test accuracy achieved in each run. For the seen split, both CNN and ResNet steadily improve across runs, with ResNet surpassing 0.85 and showing greater robustness, particularly with RSSI as input. The unseen splits (A, B, C), however, remain below 0.5 and fluctuate heavily, indicating that the models fail to generalize to previously unobserved nodes even when evaluated by their best test accuracy. These results highlight that environment-specific patterns are well captured in seen data, but unseen cases expose severe limitations in transferability.

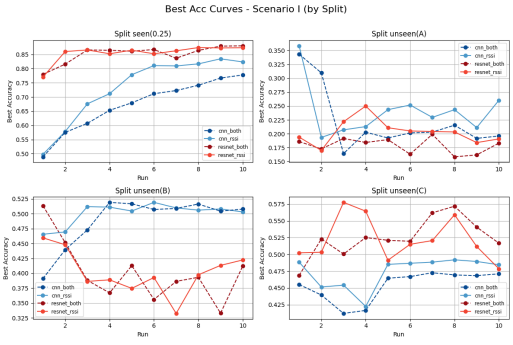

Figure 7: Accuracy curves for Scenario I.

In Scenario II (node classification), the curves likewise represent best test accuracies per run. The seen split converges around 0.75–0.8, again with ResNet performing more consistently than CNN. In contrast, the unseen splits across environments (1–5) are markedly unstable: accuracies often fall below 0.4 and vary irregularly from run to run, with no clear upward trend. This demonstrates that node classification is inherently more difficult than environment classification, and that even the best test results from multiple runs fail to reach robust levels in the unseen setting.

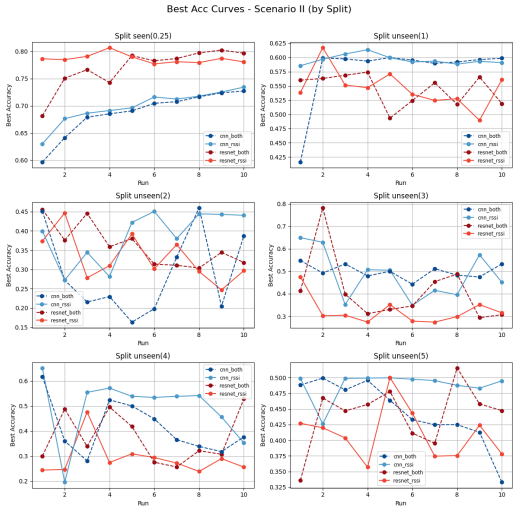

Figure 8: Accuracy curves for Scenario I.

These trends are consistent with the summary statistics: ResNet systematically outperforms CNN with higher test accuracy and lower variance, while RSSI alone proves more effective than combining it with LQI. Together, the tables and curves confirm that Scenario I is easier than Scenario II, seen splits outperform unseen ones, and the most reliable setup is ResNet with RSSI input.

5.2 Train/Test Comparison Table

The comparison between CNN and ResNet demonstrates a clear advantage of the deeper architecture across almost all settings. While CNN exhibits lower training accuracy (≈70% on average) and correspondingly modest test results, ResNet reaches above 90% training accuracy and achieves considerably higher test accuracy in the seen split (77.1% vs. 67.7%). The improvement is especially notable in the stability indicators: ResNet yields higher minimum accuracies and lower variance, both in training and testing, suggesting more consistent convergence and generalization. Although test accuracies for both architectures drop substantially in unseen splits, ResNet remains more robust, with test values fluctuating less severely than CNN. These results highlight ResNet as the more reliable choice for this classification task.

When comparing the use of RSSI alone with RSSI combined with LQI, the results indicate that RSSI is the more informative feature. Across scenarios and splits, average test accuracies with RSSI are consistently higher than those with both features, for example 74.1% vs. 70.7% in the seen case, and 45.0% vs. 42.4% in Scenario I. Variance

	CNN(S)	ResNet(S)	CNN(U)	ResNet(U)	CNN(I)	ResNet(I)	CNN(II)	ResNet(II)
Average Train Accuracy	67.55%	90.17%	70.93%	92.24%	70.59%	92.43%	70.03%	91.42%
Average Test Accuracy	67.66%	77.14%	38.14%	31.83%	43.71%	43.72%	44.27%	39.01%
Min Train Accuracy	33.58%	63.96%	35.72%	69.18%	29.34%	67.62%	39.26%	68.48%
Min Test Accuracy	37.39%	59.36%	17.49%	16.89%	20.30%	26.52%	22.25%	24.64%
Variance in Train Accuracy	0.93%	0.63%	0.99%	0.44%	1.66%	0.41%	0.53%	0.53%
Variance in Test Accuracy	0.88%	0.33%	0.71%	0.49%	0.67%	0.40%	0.80%	0.50%

Table 1: Summary of Training and Testing Accuracies by Model and Scenario

	RSSI(S)	BOTH(S)	RSSI(U)	BOTH(U)	RSSI(I)	BOTH(I)	RSSI(II)	BOTH(II)
Average Train Accuracy	79.94%	77.77%	82.28%	80.89%	82.69%	80.33%	81.23%	80.22%
Average Test Accuracy	74.11%	70.69%	35.75%	34.23%	44.99%	42.44%	42.37%	40.91%
Min Train Accuracy	49.54%	48.00%	53.12%	51.78%	49.35%	47.61%	54.44%	53.30%
Min Test Accuracy	52.15%	44.60%	17.81%	16.57%	26.44%	20.38%	23.50%	23.38%
Variance in Train Accuracy	0.79%	0.77%	0.68%	0.75%	1.02%	1.05%	0.49%	0.56%
Variance in Test Accuracy	0.56%	0.65%	0.57%	0.64%	0.44%	0.63%	0.65%	0.64%

Table 2: Summary of Training and Testing Accuracies by Channel and Scenario

measures also suggest that RSSI produces slightly more stable results, particularly in the unseen settings. The inclusion of LQI does not improve accuracy and in some cases reduces the minimum test performance, reflecting the strong correlation between RSSI and LQI observed in the dataset. Thus, RSSI alone provides a sufficient and more robust signal representation for model training.

6 CONCLUSION

In this work, we designed and evaluated a machine learning pipeline for wireless sensor network data, focusing on classification tasks under two scenarios: environment discrimination (Scenario I) and node discrimination (Scenario II). Our systematic experiments across multiple splits, input features, and model architectures revealed clear trends. In the seen setting, both CNN and ResNet achieved strong performance, with ResNet consistently demonstrating higher accuracy and lower variance. However, in the unseen setting particularly for node discrimination performance degraded substantially, indicating limited generalization when the models encounter data from previously unobserved sources.

Feature analysis further showed that RSSI alone is a more reliable input than its combination with LQI, reflecting the redundancy between the two metrics. The use of fixed epochs and multiple resumed runs allowed us to trace performance evolution over time, while final evaluations highlighted the trade-off between stability and generalization.

Overall, our study demonstrates the feasibility of applying deep learning models to sensor data classification but also emphasizes the challenges of generalization in unseen deployment conditions. These findings suggest that while CNN and ResNet provide effective solutions for controlled scenarios, future work should explore strategies such as advanced regularization, domain adaptation, or data augmentation to bridge the gap between seen and unseen cases.