Federated On-Device Training on Arduino Nano 33 BLE

Haoran Li Jiuen Feng Yuhe Bian Zhehao Chen

December 2025

Outline

Project Context

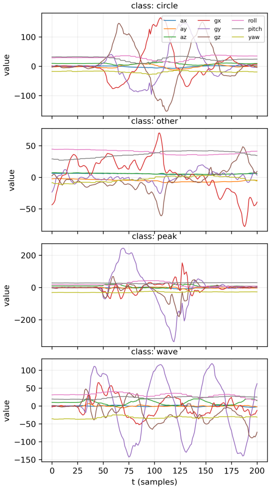
- Hardware: **Arduino Nano 33 BLE** + TinyML Shield.
- Sensors: **IMU** – accelerometer, gyroscope, orientation.
- Target task: classify IMU segments into multiple gestures / movements in “Spell Game”.
- Federated learning on two boards, communication through BLE
- Constraint: training **directly on the board** with very limited RAM and compute. BLE transmitting limitation.

Raw Data from Edge Impulse

- Data collected with Edge Impulse pipeline.
- Stored as `.cbor` messages:
  - Payload includes: `interval_ms`, `sensors`, `values`.
  - Each sample: around **2000 ms**, sampled at **100 Hz**.
- Shape of `values`:

$$(T, 9) \approx (201, 9),$$

where 9 channels are (accel, gyro, orientation).



75-D Feature Extraction (Python)

- For each segment $v \in \mathbb{R}^{T\times 9}$:
  - Global statistics per channel (9 dims):
    - mean, std, min, max $\Rightarrow 9 \times 4 = 36$.
  - Split time into 3 segments:
    - mean in each segment $\Rightarrow 3 \times 9 = 27$.
  - Per-channel energy:

$$\text{energy} = \frac{1}{T}\sum_t v_{t,c}^2 \Rightarrow 9 \text{ dims.}$$

  - Magnitude RMS for 3 groups (accel / gyro / orientation):

$$\text{RMS}_{\text{acc}},\ \text{RMS}_{\text{gyro}},\ \text{RMS}_{\text{ori}} \Rightarrow 3 \text{ dims.}$$

- Total: $36 + 27 + 9 + 3 =$ **75** features per sample.

Network Architecture on Nano 33 BLE

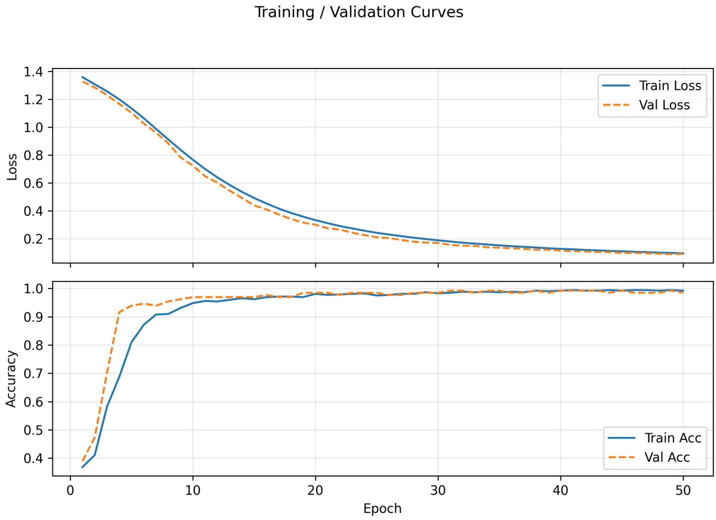
- Simple fully connected network:

$$75 \rightarrow 32 \rightarrow \text{classes\_cnt}.$$

- Activation:
  - Hidden layer: ReLU.
  - Output layer: softmax.
- Loss:
  - Cross-entropy between predicted probabilities and one-hot labels.
- Implemented in C with:
  - Manually allocated layers and neurons.
  - Forward and backward propagation using SGD.

PC-Side Reference Training (PyTorch)

- Use exactly the same 75-D feature vectors as on the Arduino:
- Results:
  - After about 50 epochs, validation accuracy reaches $\approx 95\%$.
  - This PC model is used as an **upper-bound reference** for on-device learning performance.



PC-side centralized training reaches ∼95% val. accuracy in ∼50 epochs.

From Single Board to Federated Setup

- We have two Nano 33 BLE boards:
  - Each board has its own local dataset from different users' gesture recordings.
- Idea: In each round
  - Each board trains locally for several epochs.
  - Boards use BLE to **exchange network weights**.
  - Then **average** the weights as a simple federated aggregation.
- **Only parameters are sent over BLE**.No raw sensor data is transmitted.
- Our federated on-device training protocol is based on an open-source GitHub repository.

Reference:
`https://github.com/niil87/Machine-Learning-for-IOT--Fall-2022-Batch-Lun`

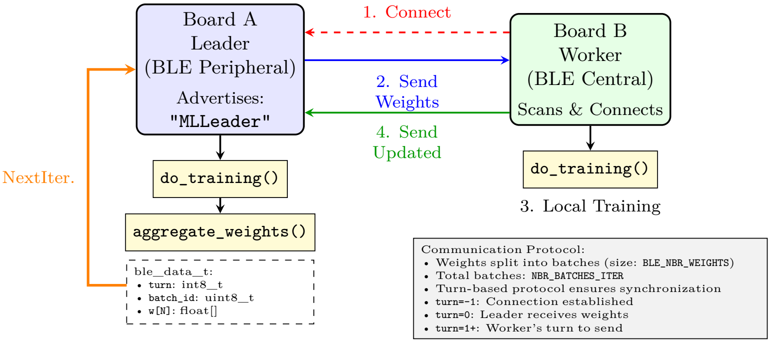
BLE Communication

Board A: Leader (BLE Peripheral)

- Advertises BLE service `MLLeader` and accepts a connection.
- **Sends** the current **global weights** over BLE (split into batches).
- **Receives** updated weights from Board B over BLE.
- Aggregates parameters and starts the next round.

Board B: Worker (BLE Central)

- Scans for `MLLeader`, connects, and subscribes to updates.
- **Receives** global weights over BLE and loads them into the NN.
- Runs **local on-device training** on its own IMU dataset.
- **Uploads** its updated weights back to Board A over BLE.

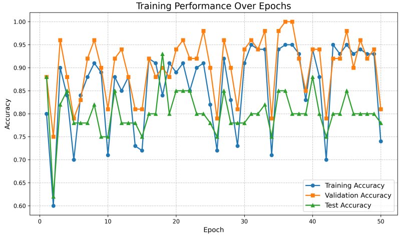

Board A
Leader
(BLE Peripheral)
Advertises:
"MLLeader"
1. Connect
2. Send Weights
4. Send Updated
Board B
Worker
(BLE Central)
Scans & Connects
do_training()
3. Local Training
NextIter.
do_training()
aggregate_weights()
ble__data__t:
• turn: int8__t
• batch_id: uint8__t
• w[N]: float[]
Communication Protocol:
• Weights split into batches (size: BLE_NBR_WEIGHTS)
• Total batches: NBR_BATCHES_ITER
• Turn-based protocol ensures synchronization
• turn=-1: Connection established
• turn=0: Leader receives weights
• turn=1+: Worker's turn to send

Simple Federated Averaging

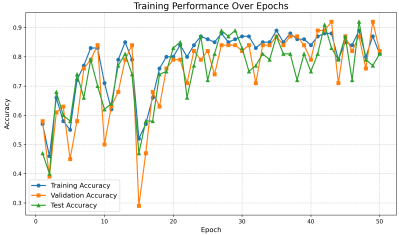
- In our prototype with two boards: Use equal weighting as an approximation:

$$w_{\text{new}} = \frac{w_A + w_B}{2}.$$

On-Device Training Curves



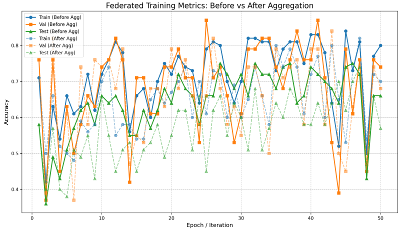
Local Train with data_byh



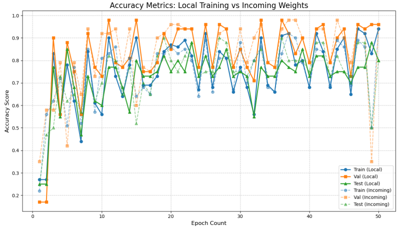
Local Train with data_fje

- Federated training improves **cross-user generalization** without sharing raw IMU data.

On-Device Training Curves



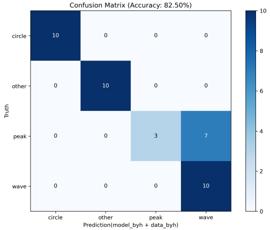
TODO



TODO

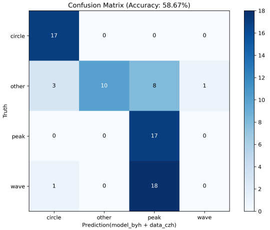
- Federated training improves **cross-user generalization** without sharing raw IMU data.

Confusion Matrices: Local Training one Board



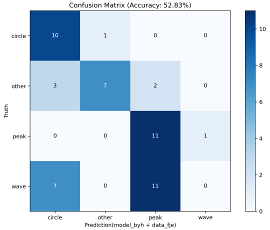
Train byh → Test byh

acc = 82.50%



Train byh → Test czh

acc = 58.67%

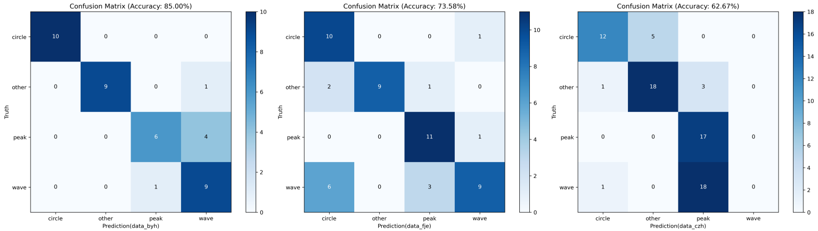


Train byh → Test fje

acc =52.83%

- Best performance on **in-domain** test (A→A).
- Accuracy drops on B/C due to **non-IID user motion patterns**.

Confusion Matrices: Federated Training (fje+byh, FedAvg)



FedAvg → Test byh	FedAvg → Test fje	FedAvg → Test czh
acc = 85.00%	acc = 73.58%	acc = 62.67%

- Federated model reduces cross-user confusion compared to local models.
- Gains come with communication cost (BLE weight exchange each round).

Demo Video

Click to watch demo video

Summary

- Built a **full pipeline**:
  - Raw IMU segments ⇒ 75-D features.
  - Python preprocessing and PC training.
  - Exported `data.h` for on-device training.
- Implemented a small **on-device neural network**:
  - 75–64–`classes_cnt`, ReLU + softmax, SGD training.
- Designed and tested a **federated learning prototype**:
  - Two Nano 33 BLE boards exchanging weights via BLE.
  - Simple parameter averaging after local training epochs.
- Demonstrated that even with tight resource constraints, federated ideas can be prototyped on microcontrollers.

Thank you for listening!