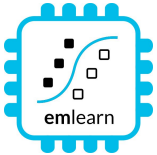
Machine Learning for Microcontrollers

with Zephyr and emlearn

https://github.com/emlearn/emlearn

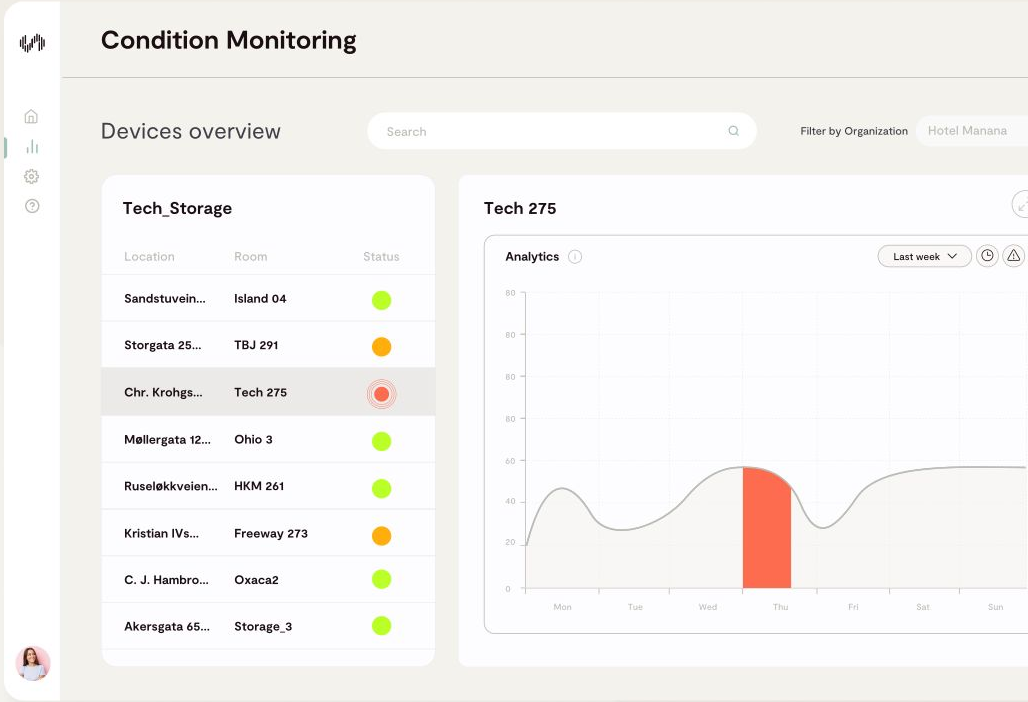
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT EUROPE



Jon Nordby
jononor@gmail.com
August 2025, Amsterdam

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT EUROPE

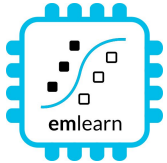
soundsensing



Trusted by Nordic market leaders

anthon eiendom · RAGDE EIENDOM · NORDR · SELVAAG · AF EIENDOM · OBOS · THON EIENDOM · EIENDOMSSPAR · Oslo · WÄRTSILÄ · ASPELIN RAMM · MALLING & CO · STATSBYGG

Goal & structure of this talk



You, as an **embedded software engineer**

become familiar with how to

develop & deploy Machine Learning

on a microcontroller

using Zephyr and emlearn

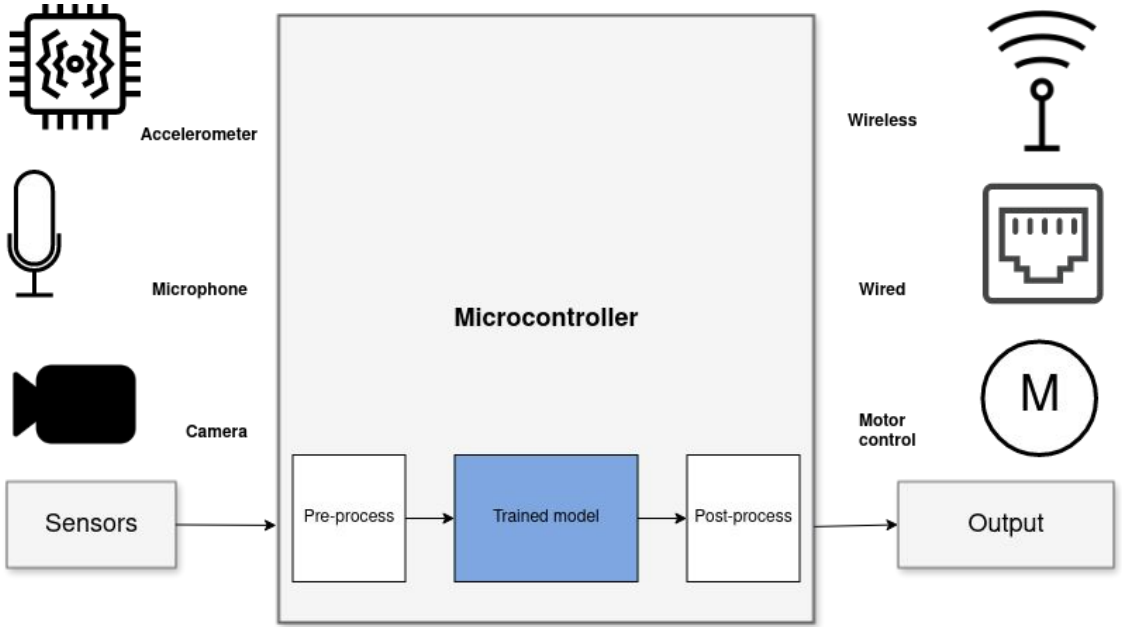
Outline

Case: Activity Recognition

- Reading sensors
- Creating dataset
- Training model
- Deploying model

Bonus: MicroPython

Sensor data analysis with Machine Learning

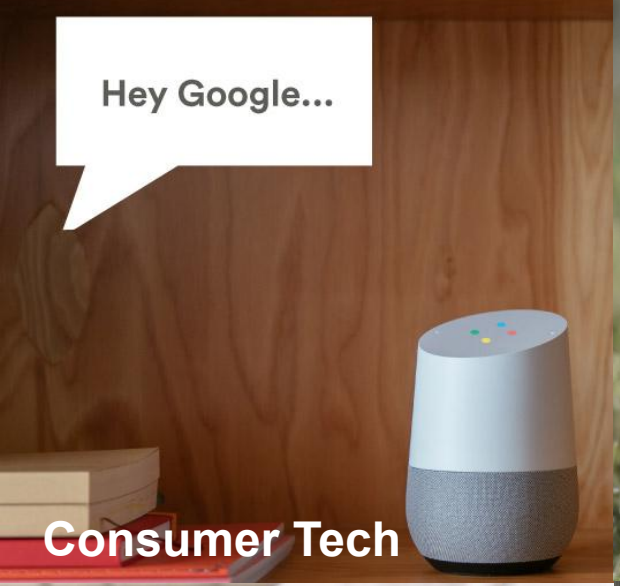


Benefits vs sending data to cloud

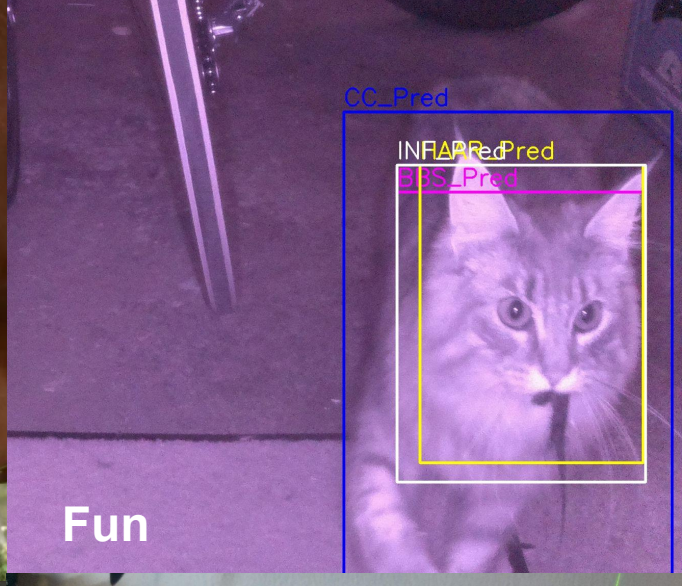
- Stand-alone
- Low latency
- Power efficiency
- Privacy
- Low cost

1) Read sensors –> 2) Process data * –> 3) Transmit/act

* including Digital Signal Processing (DSP) and Machine Learning (ML)


Hey Google...
Consumer Tech


Industrial

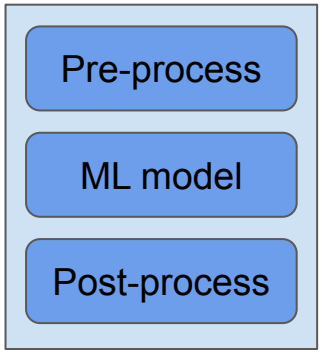

CC_Pred
INFRARed_Pred
BBS_Pred
Fun


15:34
Heart Rate
52
62
06:45
00:45 AM
75%

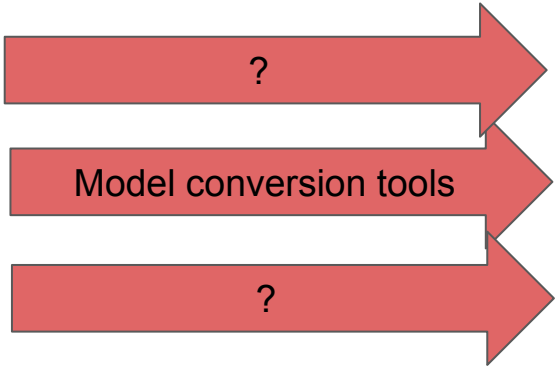
TinyML challenge: **PC/host** vs **embedded/device** pipeline

1) Train model
on PC

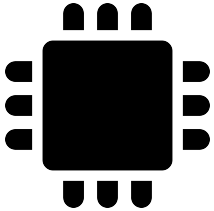
RAM (min)
10 GB+

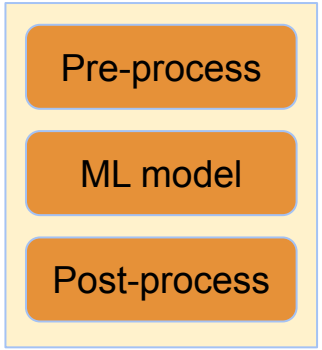


Deep-learning pipeline



2) Run
on Device





RAM (max):
1 MB

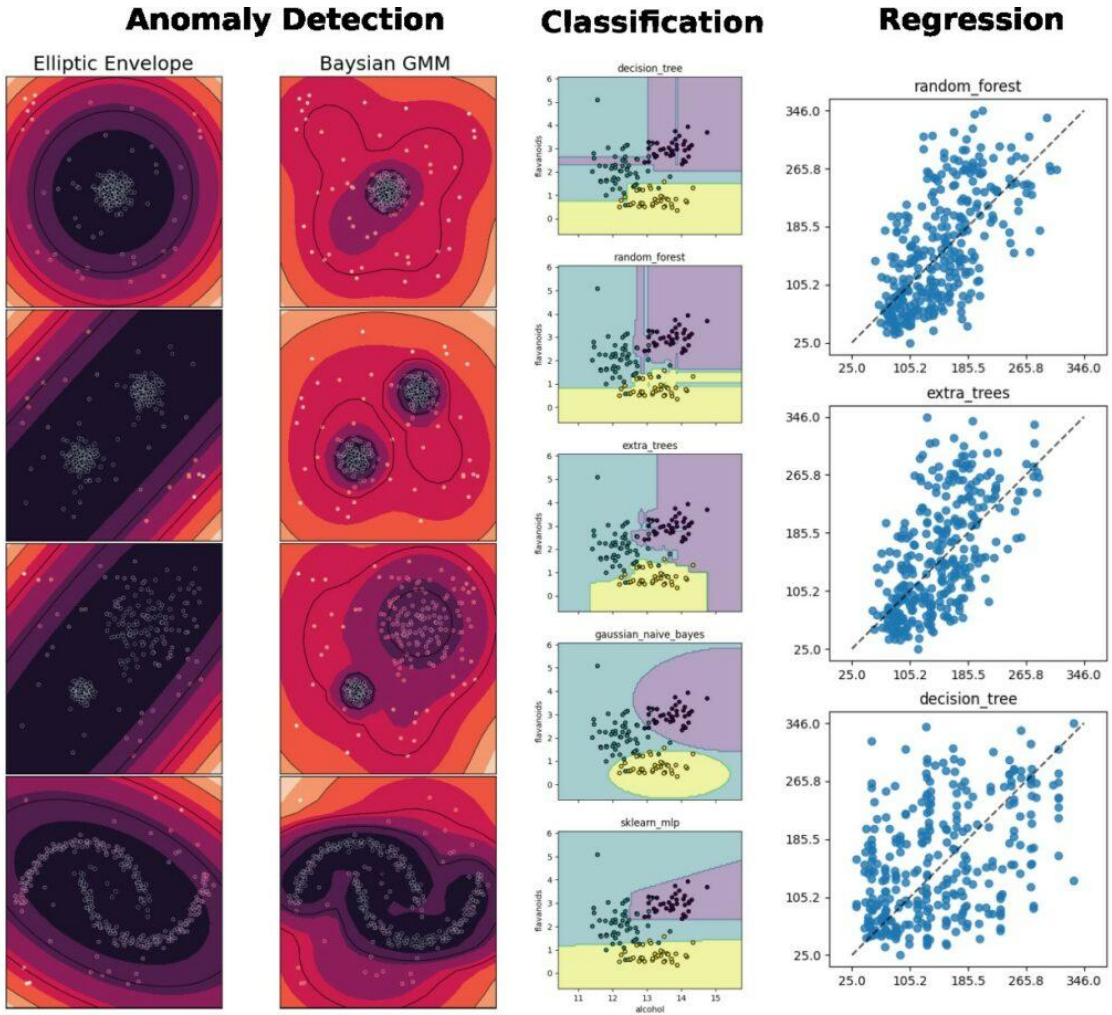
Embedded optimized pipeline

Challenges

- Very different constraints in the two environments
- The two pipelines must be 100% compatible
- Pre- and post-processing often application-specific, Turing complete
- Different languages, tools, libraries, cultures

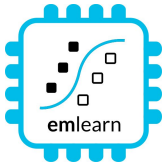
"Data Scientist land"

"Firmware Engineer land"



Most common tasks
for embedded
& sensor data use cases.

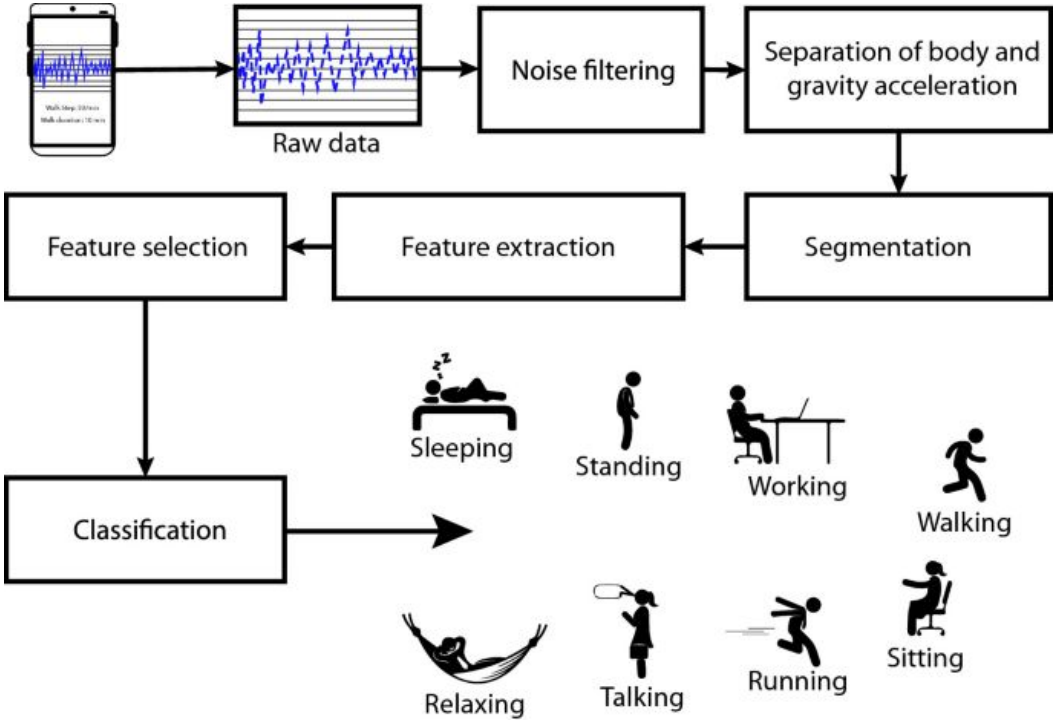
Supported by emlearn

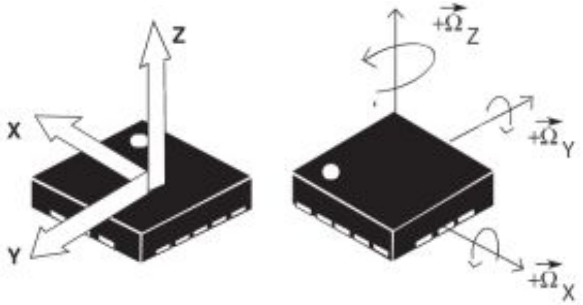


Activity Recognition

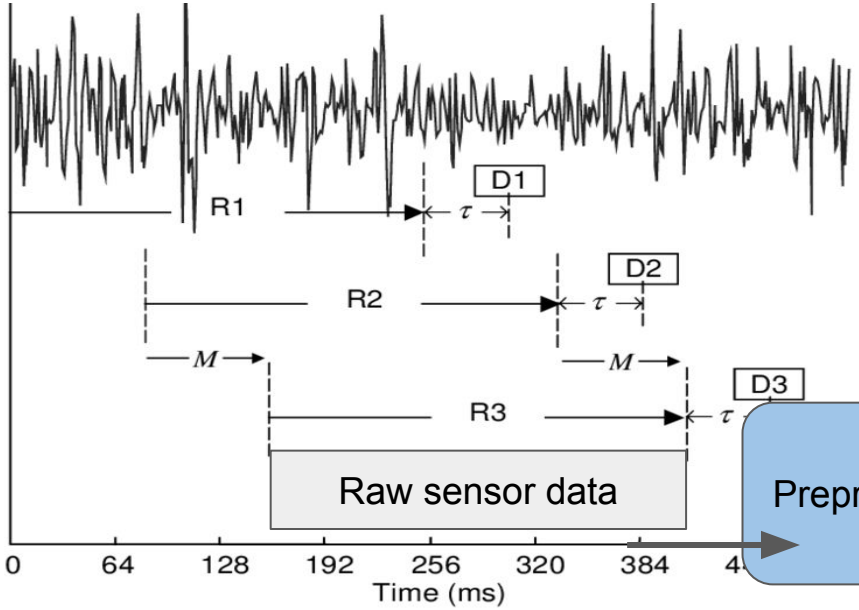
Using IMU accelerometer/gyro

Activity tracker - concept

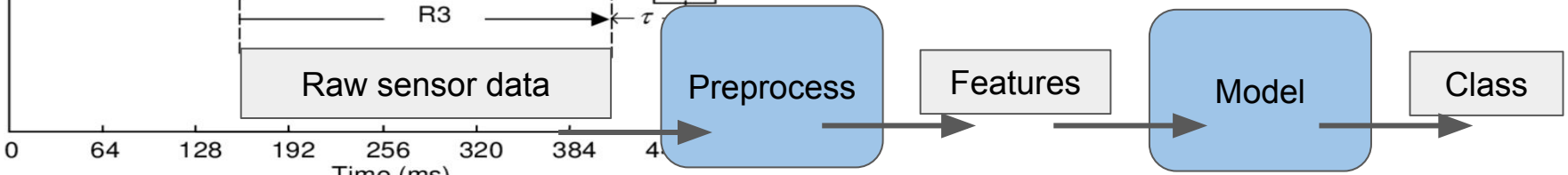




ML on streams: Continuous classification



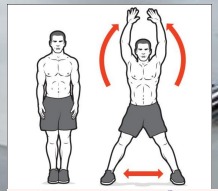
The sensor data stream
is sliced into **overlapping windows**.
Each window **processed independently**

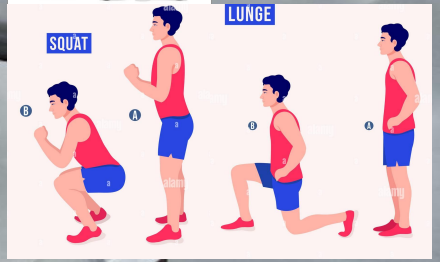


[42, 4002, … , 329]

“Jumping Jacks”

Exercise tracker





Toothbrush timer

Brushing time
00:31 / 02:00
26 % done

https://github.com/jonnor/toothbrush/

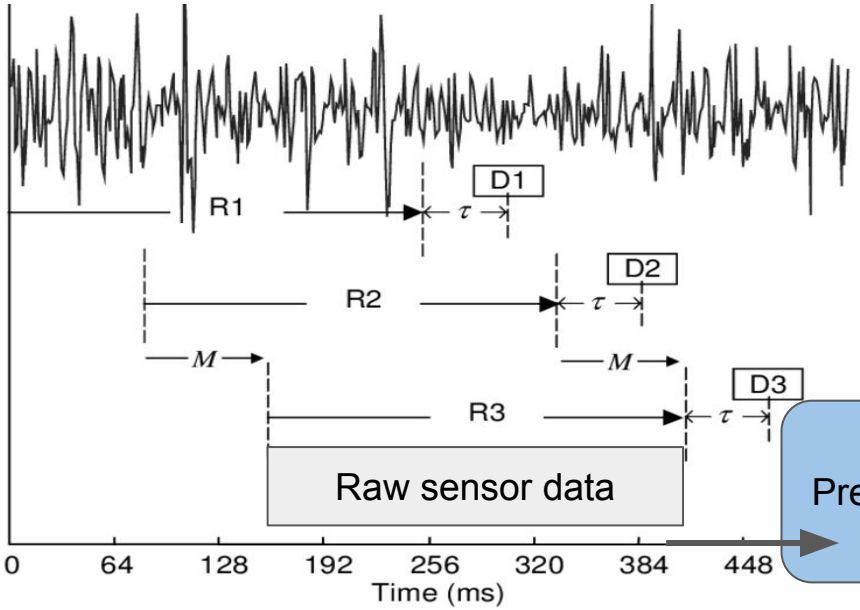
Health monitoring for cattle

Power Efficient Wireless Sensor Node through Edge Intelligence
Abhishek Damle (2022)

VIRGINIA TECH

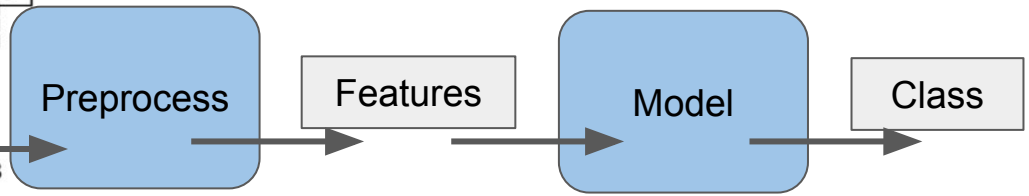
Reading sensors

ML on streams: Continuous classification



Example config

- 4 second window, every 1 second
- 100 Hz samplerate / 10 ms sample interval
- Window processing budget: 60 ms



[x1 y1 z1
x2 y2 z2
….]

[42, 4002, … , 329]

“Jumping Jacks”

Polling via sensor fetch&get

hread 1: Sensor **loop** | 100 Hz / 10 ms

```
ensor_sample_fetch(self->dev);

 Get data from each channel
or (size_t i = 0; i < self->n_channels; i++) {
  sensor_channel_get(self->dev, self->channels[i], &value);
  values[i] = sensor_value_to_double(&value);


…
 (self->read_samples_index == self->window_length) {
  self->read_samples_index = 0;
  // copy to dedicated output buffer for application
  memcpy(self->output_buffer, self->samples, out_length);

  // Send as message
  … msg = { 1, self->output_buffer, out_length };
  k_msgq_put(self->queue, &msg, K_NO_WAIT);

_usleep(1000000/self->samplerate);
```

Thread 2: Application **main** | 2 Hz / 500 ms

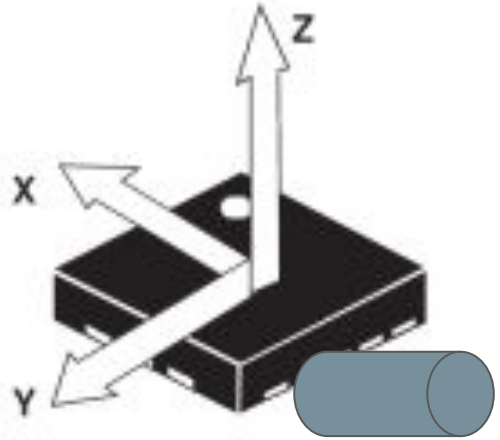
```
// Setup sensor
configure_sensor(lsm6dsl_dev);

// Start high-priority thread for collecting data
sensor_chunk_reader_start(&reader);

while (1) {
    const int get_error = k_msgq_get(reader.queue, &msg, K_NO_WAIT);
    if (get_error == 0) {
        // pre-process
        accelgyro_preprocessor_run(&preprocessor, msg.buffer, msg.length);
        // run machine learning model
        activity_class = \
            activity_model_predict(preprocessor.features, features_length);
        // TODO: do something with result
    }
    k_msleep(100);
}
```

ractical example
ttps://github.com/emlearn/emlearn/issues/108

Pros: Universal, simple

Cons: Limits CPU sleep, sample jitter on schedule variation

Using FIFO in IMU




```
// Setup sensor, including FIFO enable
configure_sensor(dev);

while (1) {
    const int fifo_count = imu_get_fifo_length(dev);
    if (fifo_count >= window_length) {
        // read in data from IMU
        imu_read_into(dev, buffer, window_length);
        // pre-process
        accelgyro_preprocessor_run(&preprocessor, buffer, length);
        // run machine learning model
        activity_class = \
            activity_model_predict(preprocessor.features, features_length);
        // TODO: do something with result
    }
    k_msleep(100);
}
```

Built-in FIFO for 32 - 128 samples
At 100 hz, gives **320 - 1280 ms to service**
Can (often) use application thread directly

Pros:
Perfectly sampled, CPU can sleep, fewer threads

Cons: No DMA support

Implementing an IMU/accelerometer/gyro driver? Use the FIFO!
https://github.com/orgs/micropython/discussions/15512

Batching via RTIO and sensor read&decode

Async interface

+ **Can utilize FIFO**
+ **Can utilize DMA (I2C/SPI)**

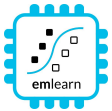
Introduced in Zephyr 4.x

As of Zephyr 4.2

- Only 18 drivers support read&decode

```
drivers/sensor/adi/adxl345/adxl345.c
drivers/sensor/adi/adxl362/adxl362.c
drivers/sensor/adi/adxl367/adxl367.c
drivers/sensor/adi/adxl372/adxl372.c
drivers/sensor/asahi_kasei/akm09918c/akm09918c.c
drivers/sensor/bosch/bma4xx/bma4xx.c
drivers/sensor/bosch/bme280/bme280.c
drivers/sensor/broadcom/afbr_s50/afbr_s50.c
drivers/sensor/maxim/ds3231/ds3231.c
drivers/sensor/melexis/mlx90394/mlx90394.c
drivers/sensor/memsic/mmc56x3/mmc56x3.c
drivers/sensor/pixart/paa3905/paa3905.c
drivers/sensor/pixart/pat9136/pat9136.c
drivers/sensor/pni/rm3100/rm3100.c
drivers/sensor/st/lis2dux12/lis2dux12.c
drivers/sensor/st/lsm6dsv16x/lsm6dsv16x.c
drivers/sensor/tdk/icm42688/icm42688.c
drivers/sensor/tdk/icm45686/icm45686.c
```

Sample: Batch processing with RTIO and read&decode
https://github.com/zephyrproject-rtos/zephyr/blob/main/samples/subsys/rtio/sensor_batch_processing/src/main.c



Creating dataset

Recording a dataset with “pre-labeling”

Controlled data collection

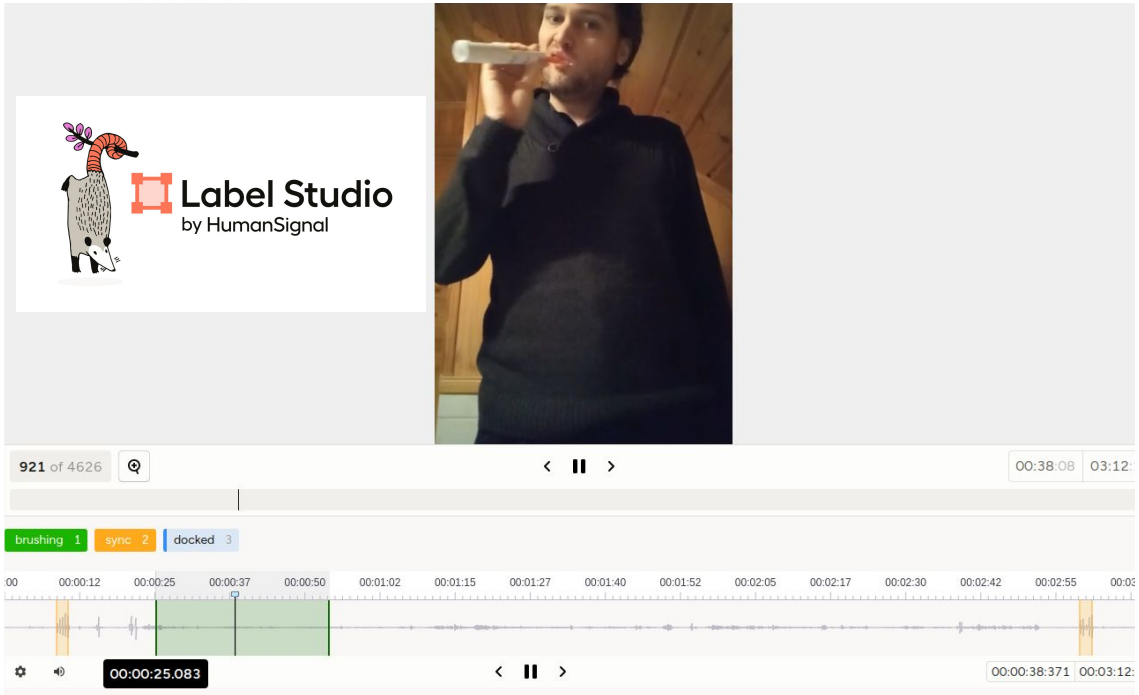
Activity to perform indicated up-front

Save files to internal file system (.CSV/.NPY)

Transfer to PC over USB

File system
https://docs.zephyrproject.org/latest/samples/subsys/fs/fs_sample/README.html#fs
USB Mass Storage
https://docs.zephyrproject.org/latest/samples/subsys/usb/mass/README.html#usb-mass
Alt: SMP server + smpclient (serial/BLE)
https://docs.zephyrproject.org/latest/samples/subsys/mgmt/mcumgr/smp_svr/README.html#smp-svr-sample-build

Better labeling



Video recorded on phone
used as reference for labeling

Start/end marked by tapping device - used to **sync video/labels** from phone **with sensor data**

Use **Label Studio for labeling**

https://labelstud.io/templates/timeseries_audio_video

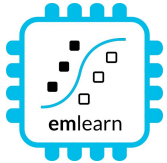
Supports time-series + video

Data collection tools:
https://github.com/emlearn/emlearn-micropython/tree/master/examples/har_trees

emlearn

Training model

Preprocessing for Activity Recognition



Gravity separation (simple)

Low-pass to separate gravity vector

Motion = Acceleration - Gravity

Feature extraction

Basic statistical summarizations are very useful.
Can be sufficient for Human Activity Recognition

```
l = sorted(list(v))
l2 = [x*x for x in l]
sm = sum(l)
sqs = sum(l2)
avg = sum(l) / len(l)

median = l[MEDIAN]
q25 = l[Q1]
q75 = l[Q3]
iqr = (l[Q3] - l[Q1])

energy = ((sqs / len(l2)) ** 0.5) # rms
std = ((sqs - avg * avg) ** 0.5)
```

Practical example
https://github.com/emlearn/emlearn/issues/108

Time-based features extraction
Are Microcontrollers Ready for Deep Learning-Based Human Activity Recognition?
Atis Elsts, and Ryan McConville
https://www.mdpi.com/2079-9292/10/21/2640

Training model on dataset

Using a scikit-learn based pipeline.

```
# Setup subject-based cross validation
splitter = GroupShuffleSplit(n_splits=n_splits, test_size=0.25,
    random_state=random_state)

# Random Forest classifier
clf = RandomForestClassifier(random_state = random_state,
    n_jobs=1, class_weight = "balanced")

# Hyper-parameter search
search = GridSearchCV(clf, param_grid=hyperparameters,
   scoring=metric, refit=metric, cv=splitter)
search.fit(X, Y, groups=groups)
```

```
(venv) [jon@jon-thinkpad har_trees]$ MIN_SAMPLES_LEAF=150,200,400 python har_train.py
-dataset har_exercise_1 --window-length 400 --window-hop 10
2024-12-04 12:54:52 [info     ] data-loaded                    dataset=har_exercise_1
uration=0.016095876693725586 samples=32000
2024-12-04 12:54:56 [info     ] feature-extraction-done        dataset=har_exercise_1
uration=4.534412145614624 labeled_instances=1952 total_instances=1952
Class distribution
 activity
jumpingjack    549
lunge          488
other          488
squat          427
Name: count, dtype: int64
```

```
Model written to ./har_exercise_1_trees.csv
Testdata written to ./har_exercise_1.testdata.npz
Results
    n_estimators  min_samples_leaf  mean_train_f1_micro  mean_test_f1_micro
0            10               150             0.996311            0.962705
1            10               200             0.995628            0.956557
2            10               400             0.986202            0.920902
```

har_train.py
https://github.com/emlearn/emlearn-micropython/blob/master/examples/har_trees/har_train.py

emlearn

Deploying model

Export model with emlearn

emlearn

Install Python package

$ pip install emlearn

```
import emlearn

cmodel = emlearn.convert(model, method='inline')

cmodel.save(file=mynet_model.h', name='mynet')
```

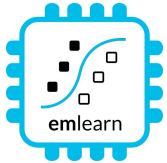
Use **emlearn.convert()** and **.save()**

mynet_model.h

```
#include <eml_net.h>

static const float mynet_layer_0_biases[8] = { -0.015587f, -0.005395f, -0.010957f, 0.015883f ….
static const float mynet_layer_0_weights[24] = { -0.256981f, 0.041887f, 0.063659f, 0.011013f, …
static const float mynet_layer_1_biases[4] = { 0.001242f, 0.010440f, -0.005309f, -0.006540f };
static const float mynet_layer_1_weights[32] = { -0.577215f, -0.674633f, -0.376140f, 0.646900f, …
static float mynet_buf1[8];
static float mynet_buf2[8];
static const EmlNetLayer mynet_layers[2] = {
{ 8, 3, mynet_layer_0_weights, mynet_layer_0_biases, EmlNetActivationRelu },
{ 4, 8, mynet_layer_1_weights, mynet_layer_1_biases, EmlNetActivationSoftmax }
};
static EmlNet mynet = { 2, mynet_layers, mynet_buf1, mynet_buf2, 8 };

    int32_t
    mynet_predict(const float *features, int32_t n_features)
    {
        return eml_net_predict(&mynet, features, n_features);
    }
.....
```

Example of generated code

emlearn module for Zephyr



1. Add project to west.yml

```
projects:
  #... exiting projects
  - name: emlearn
    url: https://github.com/emlearn/emlearn
    revision: master
```

2. Enable in your project

```
CONFIG_EMLEARN=y
```

3. Use in your code

```
#include <eml_net.h>
#include “my_model.h”
```

Module exists since February 2024

emlearn documentation

https://emlearn.readthedocs.io/en/latest/getting_started_zephyr.html

Zephyr documentation

https://docs.zephyrproject.org/latest/develop/manifest/external/emlearn.html

Using the C code

emlearn

3. **#include** and call **predict()**

```c
// Include the generated model code
#include "mynet_model.h"

// index for the class we are detecting
#define MYNET_VOICE 1

// Buffers for input data
#define N_FEATURES 6
float features[N_FEATURES];

#define DATA_LENGTH 128
int16_t sensor_data[DATA_LENGTH];
```

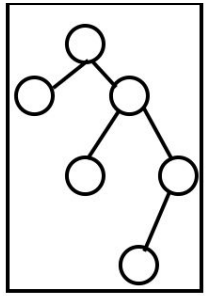
setup

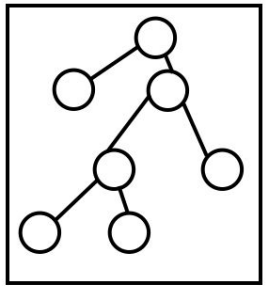
```c
// Get data and pre-process it
read_microphone(sensor_data, DATA_LENGTH);
preprocess_data(sensor_data, features);

// Run the model
out = mynet_predict(features, N_FEATURES);

// Do something with results
if (out == MYNET_VOICE) {
    set_display("voice detected");
} else {
    set_display("");
}
```

loop

Supported models

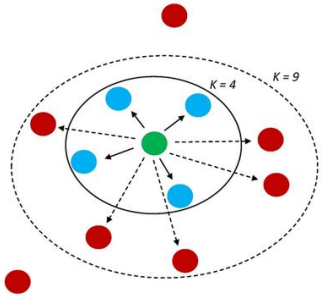
Selection of simple & effective embedded-friendly models

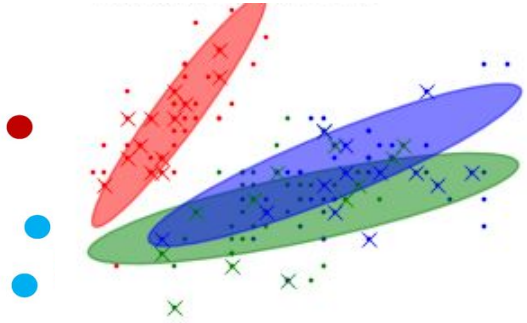


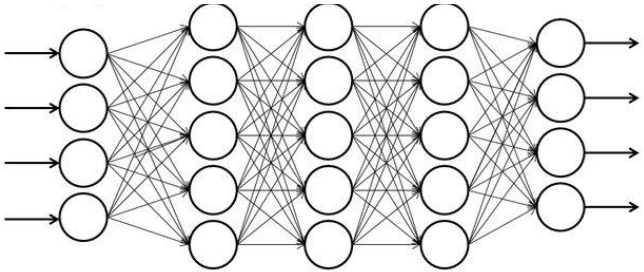


emlearn

- Decision Trees (DT)
- Random Forest (RF)
- K Nearest Neighbors (KNN)
- Gaussian Mixture Models (GMM)
- Multi-Layer-Perceptron (MLP)
- Convolutional Neural Network (CNN)







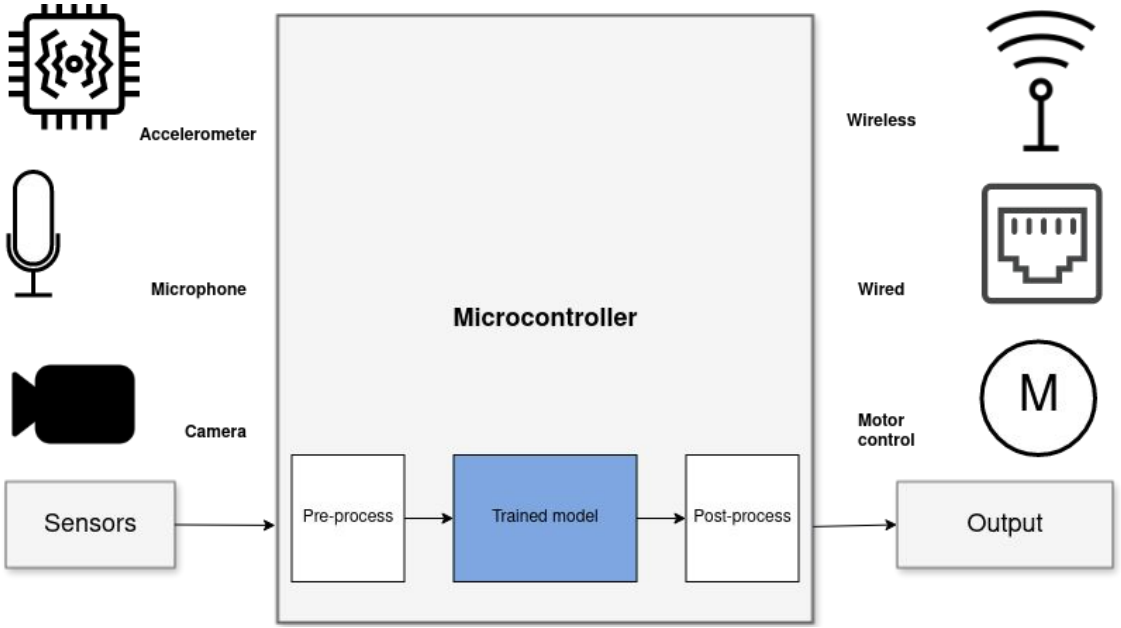
scikit learn

K

emlearn

Validating model

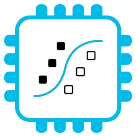
Sensor data analysis with Machine Learning



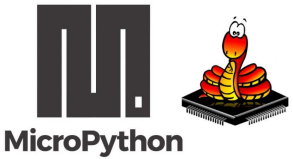
1) Read sensors –> 2) Process data * –> 3) Transmit/act

Bridging the TinyML language gap

with MicroPython



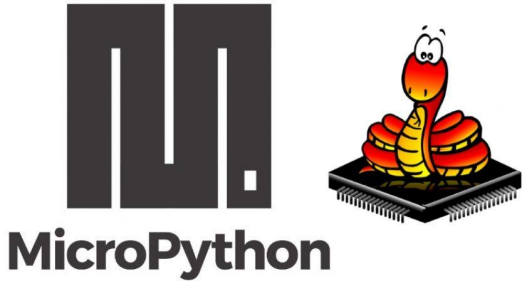
emlearn
-micropython



Python is the lingua franca for Machine Learning

How can we enable people to build TinyML solution also using Python?

MicroPython - Introduction



Implements a subset of Python 3.x
For devices with 64 kB+ RAM

Supports using Zephyr as HAL
Has matured a lot in MicroPython 1.26 (August 2025)

More testing & contributors wanted!

More info: https://micropython.org

C modules *

Defines a Python module with API. functions/classes et.c.

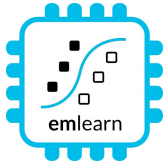
Implemented
by users,
libraries, or
part of MicroPython core.

Can be **portable** or
hardware/platform specific

* Or other language which compiles to C, or exposes a C API. C++, Rust, et.c.

```c
// Include the header file to get access to the MicroPython API
#include "py/dynruntime.h"

// Helper C function to compute factorial
static mp_int_t factorial_helper(mp_int_t x) {
    if (x == 0) {
        return 1;
    }
    return x * factorial_helper(x - 1);
}

// DEFINE FUNCTION. Callable from Python
static mp_obj_t factorial(mp_obj_t x_obj) {
    mp_int_t x = mp_obj_get_int(x_obj);
    mp_int_t result = factorial_helper(x);
    return mp_obj_new_int(result);
}
static MP_DEFINE_CONST_FUN_OBJ_1(factorial_obj, factorial);


// MODULE ENTRY
mp_obj_t mpy_init(mp_obj_fun_bc_t *self, size_t n_args, size_t n_kw, mp_obj_t *args) {
    // Must be first, it sets up the globals dict and other things
    MP_DYNRUNTIME_INIT_ENTRY

    // Register function in the module's namespace
    mp_store_global(MP_QSTR_factorial, MP_OBJ_FROM_PTR(&factorial_obj));

    // This must be last, it restores the globals dict
    MP_DYNRUNTIME_INIT_EXIT
}
```

MicroPython Data Science ecosystem



CPython

- Tensorflow
- scikit-learn
- OpenCV
- scipy
- numpy

MicroPython

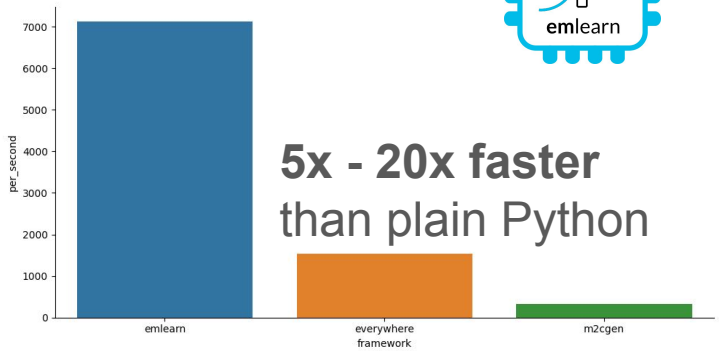
- emlearn-micropython — github.com/emlearn/emlearn-micropython
- OpenMV — https://openmv.io/
- ulab — github.com/v923z/micropython-ulab

Libraries existing that implement a lot of embedded-relevant functionality from PyData stack.

emlearn-micropython: MicroPython bindings for emlearn

https://github.com/emlearn/emlearn-micropython/

Modules installable at runtime (.mpy)
1 - 10 kB in size



Module	Description	Corresponds to
emlearn_trees	Decision tree ensembles	sklearn RandomForestClassifier
emlearn_neighbors	Nearest Neighbors	sklearn KNeighborsClassifier
emlearn_cnn	Convolutional Neural Network	keras Model+Conv2D
emlearn_linreg	Linear Regression	sklearn ElasticNet
emlearn_fft	Fast Fourier Transform	scipy.fft.fft
emlearn_iir	Infinite Impulse Response filters	scipy.signal.sosfilt
emlearn_arrayutils	Fast utilities for array.array	N/A

Install & Export model

emlearn
-micropython

Export model on PC

```
import emlearn

estimator = train_model()
converted = emlearn.convert(estimator)
converted.save(name='gesture', format='csv', file='gesture_model.csv')
```

Copy model to device

mpremote fs cp **gesture_model.csv** :

Prebuilt binary
No need to setup
C toolchain and SDK

Copy library to device

*mpremote mip install https://emlearn.github.io/...../**xtensawin_6.2**/emltrees.mpy*

Load model & run

emlearn
-micropython

```
import emltrees
# Instantiate model. Capacity for trees, decision nodes, classes
model = emltrees.new(20, 200, 10)

# Load model "weights"
with open('gesture_model.csv') as f:
    emltrees.load_model(model, f)
```

Load model on device

```
# Read sensor data
sensor.fifo_read(samples)

# Preprocess features
preprocessor.run(samples, features)

# Run model
out = model.predict(features)
```

Run inference

Activity tracker

Accelerometer

Random Forest classifier
emlearn_trees

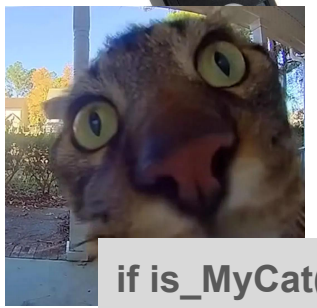
Noise monitor

Microphone

Infinite Impulse Response filters
emlearn_iir

Image Classifier

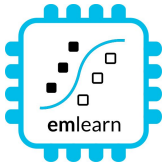
Camera



```
if is_MyCat(img):
    open_door()
```

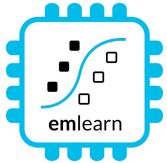
Examples at
https://github.com/emlearn/emlearn-micropython#examples

Convolutional Neural Network
emlearn_cnn



Outro

Summary



Machine learning inference on microcontroller useful in many applications

Zephyr is a good base

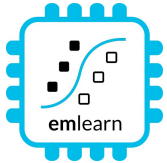
- Sensor API, filesystem, connectivity

emlearn makes deploying models simple

- Converting scikit-learn or Keras to efficient C code
- Easy to use via Zephyr module

Bonus: **MicroPython** can also be used **with Zephyr and emlearn-micropython**

More resources



Official documentation

https://emlearn-micropython.readthedocs.io
https://emlearn.readthedocs.io

TinyML EMEA 2024: **emlearn - scikit-learn for microcontrollers and embedded systems**
https://www.youtube.com/watch?v=LyO5k1VMdOQ

FOSDEM 2025: **Milliwatt sized Machine Learning on microcontrollers with emlearn**
https://fosdem.org/2025/schedule/event/fosdem-2025-4524-milliwatt-sized-machine-learning-on-microcontrollers-with-emlearn/

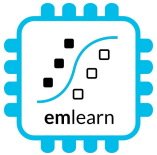
FOSDEM 2025: **MicroPython - Python for microcontrollers and Embedded Linux**
https://fosdem.org/2025/schedule/event/fosdem-2025-4525-micropython-python-for-microcontrollers-and-embedded-linux/

Machine Learning for Microcontrollers

with Zephyr and emlearn

https://github.com/emlearn/emlearn

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT EUROPE



Jon Nordby
jononor@gmail.com
August 2025, Amsterdam

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT EUROPE

Bonus

Potential Improvements

- Easier install (no zipties)
- Reduce size
- Reduce costs

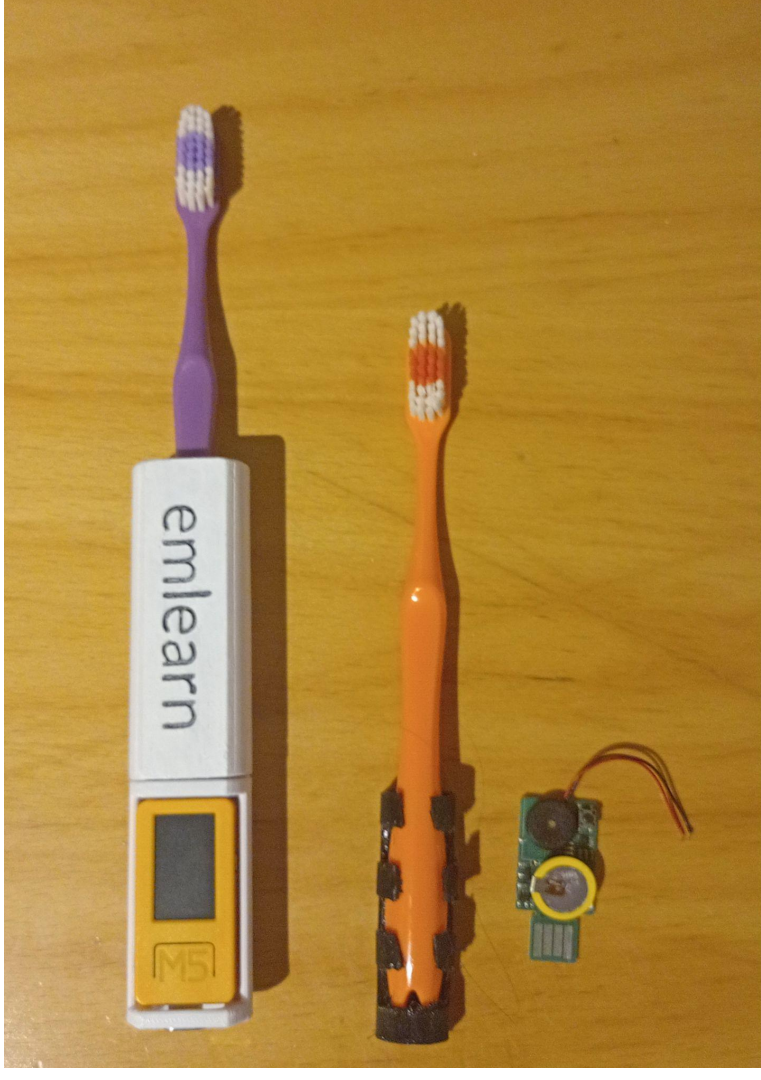
Flexible TPU casing

Custom PCB
Puya PY32 - 8 kB RAM / 64 kB FLASH

Using **emlearn C library**

https://hackaday.io/project/194511-1-dollar-tinyml

PCBWay



1 dollar TinyML

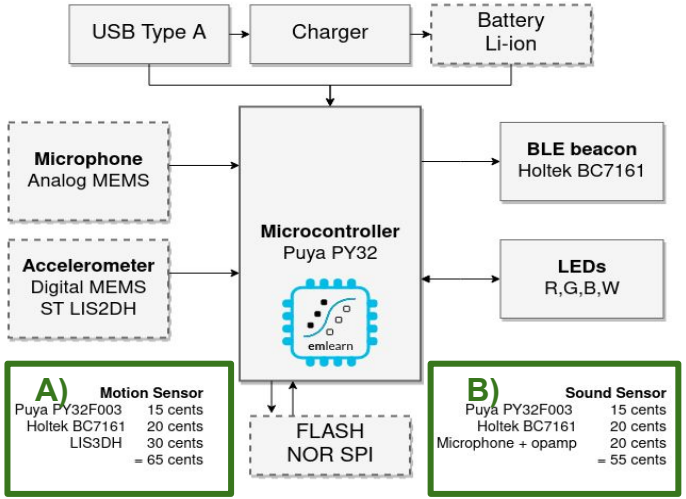
https://hackaday.io/project/194511-1-dollar-tinyml

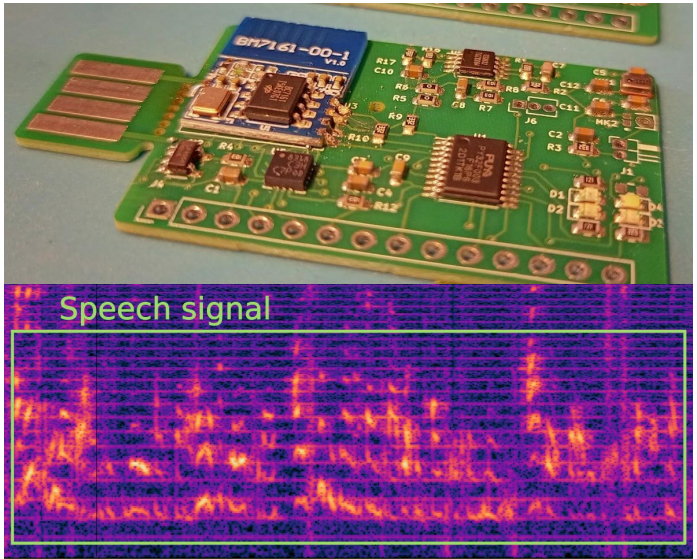
Research question:
Can we make a useful TinyML system
<1 USD in total component cost (BOM)?

Preliminary answer: YES!
Either motion (accelerometer)
or sound sensor (microphone)

Constraints: 8 kB RAM / 64 kB FLASH

Current status:
Audio streaming works
Revision 2 ordered

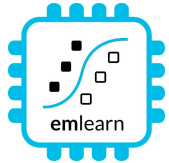




Task feasibility versus microcontroller type

	RAM (kB)	**FLASH** (kB)	**CPU** (CoreMark)		**IMU** 100 hz 5 s	**Sound** 16 kHz 1 s	**Image** 64x64 px 1 fps
Arduino Uno ATMega 328 @ 16 Mhz	2	32	4.0		✓	❌	❌
Arduino Nano BLE 33 NRF52840 ARM Cortex M4F @ 64 Mhz	256	1 000	215		✓	✓	❌
Arduino Nano ESP32 ESP32-S3 @ 240 Mhz	8 000	16 000	613 (per core)		✓	✓	✓
Teensy 4.0 ARM Cortex M7 @ 600 Mhz	1 000	2 000	2313 (per core)		✓	✓	✓

Human Activity Detection with **emlearn_trees**



Using **Random Forest** classifier trained with **scikit-learn**

```
import emltrees
model = emltrees.new(10, 1000, 10)
with open('eml_digits.csv', 'r') as f:
    emltrees.load_model(model, f)

features = array.array('h', ...)
out = model.predict(features)
```

Performance comparison

10 trees, max 100 leaf nodes, “digits” dataset

	Inference time	Program space
m2cgen	60.1 ms	179 kB
everywhereml	17.7 ms	154 kB
emlearn	**1.3 ms**	**15 kB**

https://github.com/emlearn/emlearn-micropython/tree/master/examples/har_trees

emlearn is **10x faster and 10x more space efficient** compared to generating Python code

Activity Tracker - Feature Extraction

Statistical summarizations are useful time-series features, sufficient for basic Human Activity Recognition.

! Preprocessing *must be compatible between training* on host PC (CPython) *and device* (MicroPython)

Solution: Write preprocessor for MicroPython, re-use in Python

subprocess('micropython preprocess.py data.npy features.npy')

Alternative: (when using common MicroPython/CPython subset)

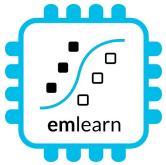
import mypreprocessor.py

Using micropython-npyfile to read/write Numpy .npy files
https://github.com/jonnor/micropython-npyfile/

```
l = sorted(list(v))
l2 = [x*x for x in l]
sm = sum(l)
sqs = sum(l2)
avg = sum(l) / len(l)

median = l[MEDIAN]
q25 = l[Q1]
q75 = l[Q3]
iqr = (l[Q3] - l[Q1])

energy = ((sqs / len(l2)) ** 0.5)
std = ((sqs - avg * avg) ** 0.5)
```

https://github.com/emlearn/emlearn-micropython/blob/master/examples/har_trees/timebased.py

Time-based features extraction
Are Microcontrollers Ready for Deep Learning-Based Human Activity Recognition?
Atis Elsts, and Ryan McConville
https://www.mdpi.com/2079-9292/10/21/2640

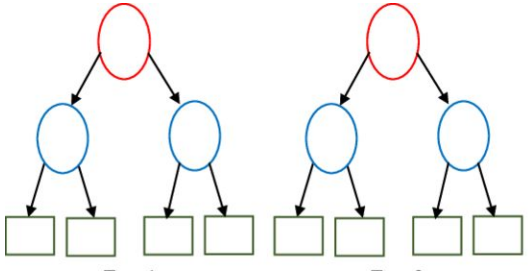


Tree-based ensembles

Random Forest / Decision Tree

The most popular model in emlearn

emlearn trees: Leaf compression

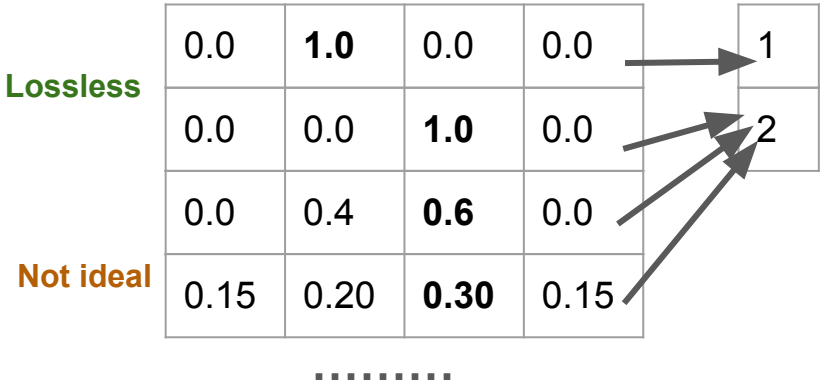


scikit-learn decision trees store class probabilities in leaf-nodes. Size: n_leaves * n_classes * 4 bytes
Consequence: Leaves take a large amount of space

emlearn (since 2019) does **hard majority voting** instead
And does **leaf-node de-duplication** *across* all trees

Size: n_classes * 1 * 1 byte
Saving 50-80% of space

Tradeoff: May read to reduced predictive performance with some leaf values



emlearn trees: Inference modes

Data structure (“loadable”)

+ Can be loaded at runtime
- Only supports float

```
static const EmlTreesNode xor_model_nodes[14] = {
  { 1, 0.197349f, 1, 2 },
  .........
  { 0, 0.421164f, -2, -1 }
};
EmlTrees xor_model = {  14,  xor_model_nodes,  ……  };

static int32_t
eml_trees_predict_tree(const EmlTrees *f, int32_t tree_root,
                       const float *features, int8_t features_length) {
    int32_t n = tree_root;
    while (n >= 0) {
        const float value = f[forest->nodes[node_idx].feature];
        const float point = forest->nodes[node_idx].value;
        const int16_t child = (value < point) ?
                    f->nodes[n].left : f->nodes[n].right;
        …..
    }
    return leaf;
}
```

Code generation (“inline”)

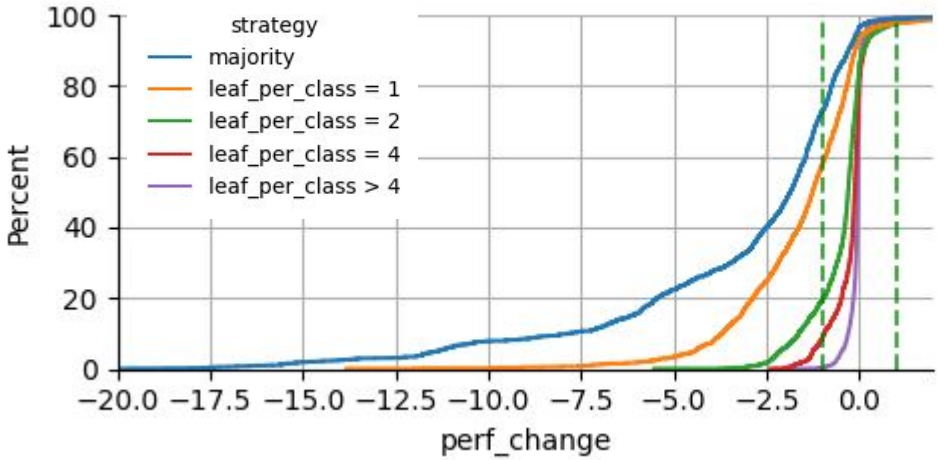
+ Supports int8/int16/int32 and float

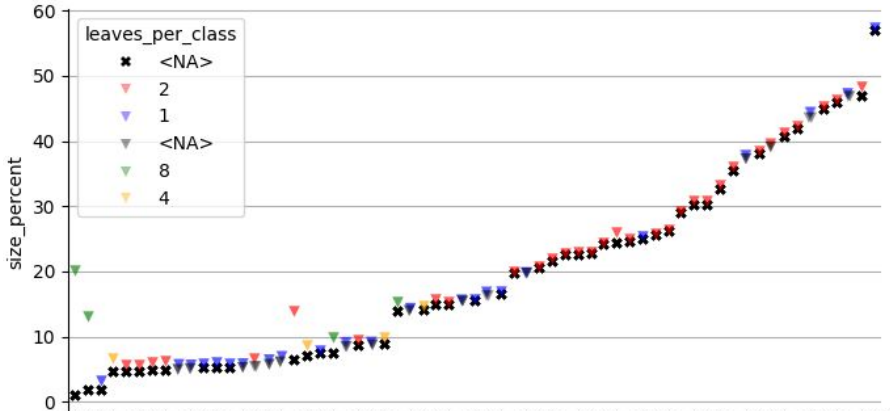
```
static inline int32_t
xor_model_tree_0(const float *features, int32_t
features_length)
{
    if (features[1] < 0.197349f) {
        if (features[0] < 0.466316f) {
            return 0;
        } else {
            return 1;
        }
    } else {
        if (features[1] < 0.256702f) {
            if (features[0] < 0.464752f) {
                return 0;
            } else {
                return 1;
            }
        } else {
....
}
```

emlearn trees: K-means clustering for leaf de-duplication

Optimizing for: Small model size
Constraint: Under 1 percentage point drop in AUC ROC
Evaluating on 48 datasets from OpenML CC18 suite

Preliminary results
- paper to follow





Hard majority
sometimes bad:
Failed perf constraint on 75% of datasets

Clustering with
N leaves per class
Allows adaptable size/perf tradeoff

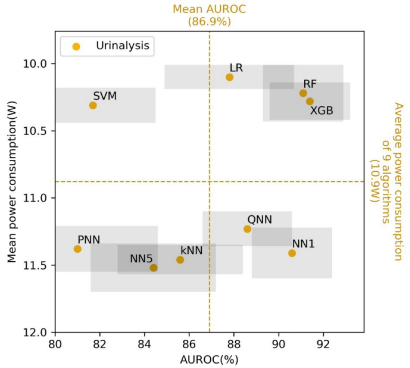
2-8 leaves per class sufficient
0% of datasets failed performance constraint
High model size compression 50-90%

Trees - relevant across the task complexity range

Low

Go-to for tabular data

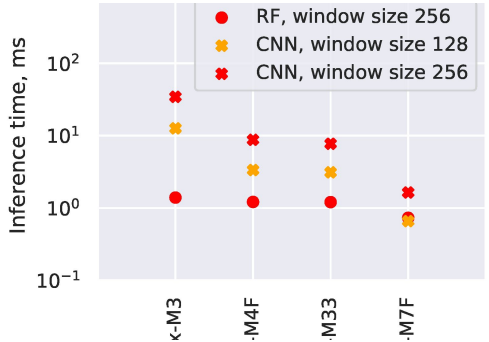
Very easy to get good performance, with minimal tweaking.



Medium

Alternative to deep learning

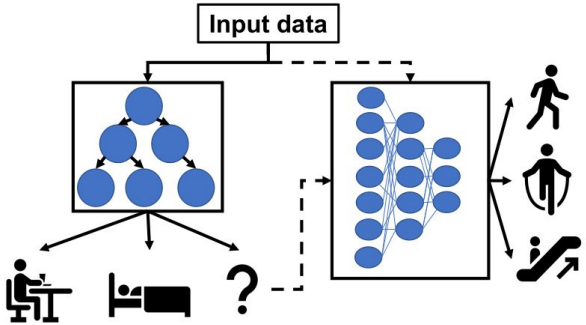
Elsts et.al [1] showed performance matching a deep-learning approach, at 10x to 100x better resource usage



High

Combine with neural network

Daghero et.al [2] showed up to 67.7% energy saving using two-stage approach

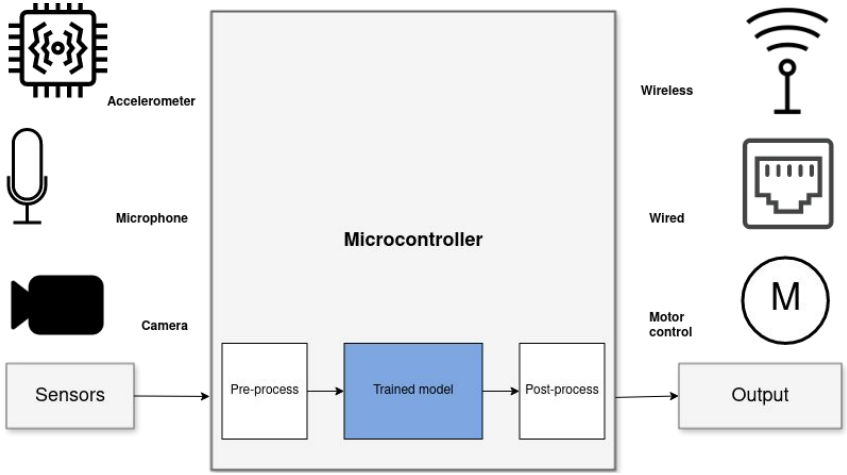


1. *"Are Microcontrollers Ready for Deep Learning-Based Human Activity Recognition?"*
   Atis Elsts, Ryan McConville (2021) https://www.mdpi.com/2079-9292/10/21/2640
2. *"Two-stage Human Activity Recognition on Microcontrollers with Decision Trees and CNNs".*
   Francesco Daghero, Daniele Jahier Pagliari, Massimo Poncino (2022) https://arxiv.org/abs/2206.07652
3. *"Energy Efficiency of Inference Algorithms for Clinical Laboratory Data Sets: Green Artificial Intelligence Study"*
   Jia-Ruei Yu et. al (2022) https://www.jmir.org/2022/1/e28036/

Focus of this presentation

TinyML development phases

- Task definition
- Hardware bringup
- **Data collection**
- Exploratory Data Analysis
- Data annotation/curation
- Model pipeline setup
- **Iterating. Model and dataset**
- **Lab/field testing**
- Monitoring in production



ML pipeline

- Data reading/loading
- Pre-processing
- **Machine Learning model (inference/training)**
- Post-processing
- **Acting on outputs**

Act locally

Task:
Give vaccinated bait *only* to Tasmanian devils

Trigger: PIR sensor

Wake up

Classify image using CNN
if is_Taz:
    dispense_bait()

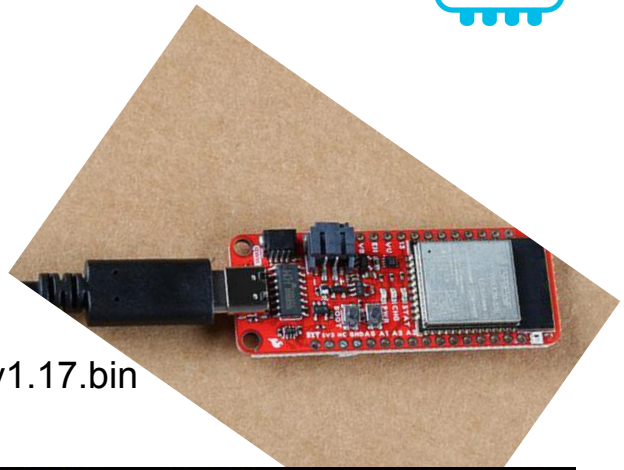
Send event over LoRaWAN

EdgeAI bait dispenser to vaccinate the Tasmanian devil (Sarcophilus harrisii) against the devil facial tumour disease
Prithul Chaturvedi[1], Andrew S. Flies, et al.

UNIVERSITY of TASMANIA
MENZIES
Institute for Medical Research

SAVE THE TASMANIAN DEVIL PROGRAM

MicroPython - Installing



Download prebuilt firmware

https://micropython.org/download/?port=esp32

Flash firmware to device

pip install esptool

esptool.py --chip esp32 --port … erase_flash
esptool.py --chip esp32 --port … write_flash -z 0 micropython-v1.17.bin

Connect to device

pip install mpremote

mpremote repl

```
MicroPython v1.8.3-24-g095e43a on 2016-08-16; ESP module
Type "help()" for more information.
>>> print('Hello world!')
Hello world!
>>>
```

IDE (optional): Thonny, VS Code, et.c.

MicroPython *is* Python - but not CPython

High degree of compatibility - but never 100%
Continuous job to keep up with CPython
Some differences inherent - from < 1 MB RAM and FLASH

Included libraries are minimal
micropython-lib has more extensive/featured
https://github.com/micropython/micropython-lib

Known incompatibilities
https://docs.micropython.org/en/latest/genrst/index.html
#micropython-differences-from-cpython

Not implemented (by CPython major release)
https://github.com/micropython/micropython/issues/
7919#issuecomment-1025221807

No CFFI or C module compatibility!
But there is another C API

=> No standard PyData stack
But there are alternatives

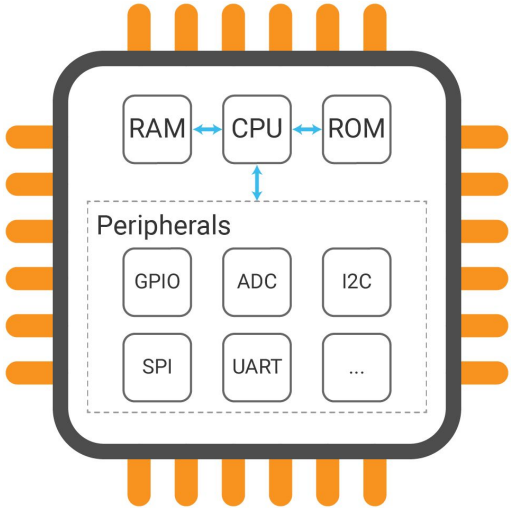
Hardware access - the **machine** module

https://docs.micropython.org/en/latest/library/machine.html

- class Pin – control I/O pins
- class Signal – control and sense external I/O devices
- class ADC – analog to digital conversion
- class ADCBlock – control ADC peripherals
- class PWM – pulse width modulation
- class UART – duplex serial communication bus
- class SPI – a Serial Peripheral Interface bus protocol (controller side)
- class I2C – a two-wire serial protocol
- class I2S – Inter-IC Sound bus protocol
- class RTC – real time clock
- class Timer – control hardware timers
- class WDT – watchdog timer
- class SD – secure digital memory card (cc3200 port only)
- class SDCard – secure digital memory card
- class USBDevice – USB Device driver

Hardware Abstraction Layer
for microcontroller peripherals

Same on all hardware/ports
* with exceptions

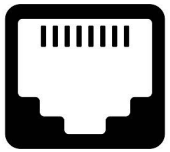


MicroPython

Connectivity

WiFi
network.WLAN

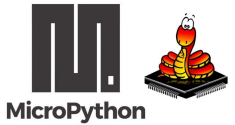
Ethernet
network.LAN



BLE - Bluetooth Low Energy
aioble - high-level application API, asyncio
bluetooth - low-level hardware-layer

Cellular (4G / LTE)
network.PPP

LoRa / LoRaWAN
micropython-lib: lora



File system

Enabled on most ports/hardware (with sufficient resources)

Internal FLASH and/or **SDCard**

LittleFS or **FAT32**

Save/load from standard files [1]

https://docs.micropython.org/en/latest/reference/filesystem.html

mpremote

Tool for **PC <-> microcontroller communication**

https://docs.micropython.org/en/latest/reference/mpremote.html

Copy from device

```
mpremote cp -r :images/ ./data/
```

Copy to device

```
mpremote cp ./model.trees.csv ./models/
```

1. Using micropython-npyfile to read/write Numpy .npy files https://github.com/jonnor/micropython-npyfile/

MicroPython

mip - package manager

Install from micropython-lib

mpremote install requests

Third party packages

mpremote install github:jonnor/micropython-zipfile

Can run directly on device [1]

```
import mip
mip.install('requests')
```

Install native C modules at runtime

mpremote mip install https://example.net/
xtensawin_6.2*/emlearn_trees.mpy*

1. Assuming device has Internet over WiFi/Ethernet

* Specify architecture + MicroPython ABI version

Efficient data processing in MicroPython

Specialized code emitters - numba.jit style

@micropython.native / @viper
https://docs.micropython.org/en/latest/reference/speed_python.html
#the-native-code-emitter

Inline assembler

https://docs.micropython.org/en/latest/pyboard/tutorial/assembler.html

C modules

https://docs.micropython.org/en/latest/develop/cmodules.html
https://docs.micropython.org/en/latest/develop/natmod.html

PyData ZA 2024

Sensor data processing on microcontrollers with MicroPython

https://www.youtube.com/watch?v=vBiji9IFJJs
https://za.pycon.org/talks/31-sensor-data-processing-on-microcontrollers-with-micropython/

MicroPython

Continuous measurement

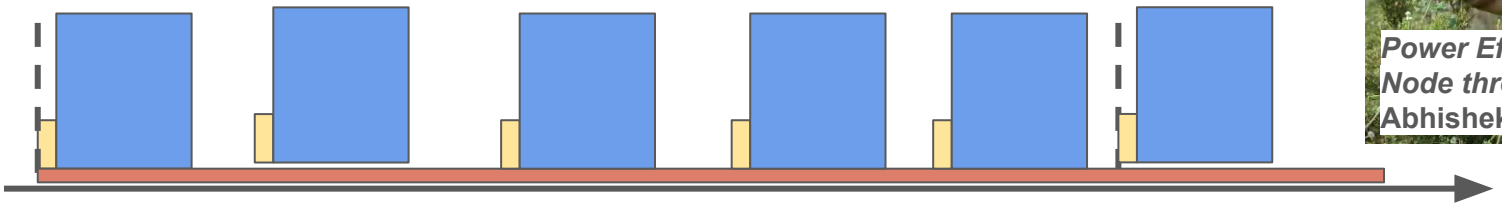
Processing

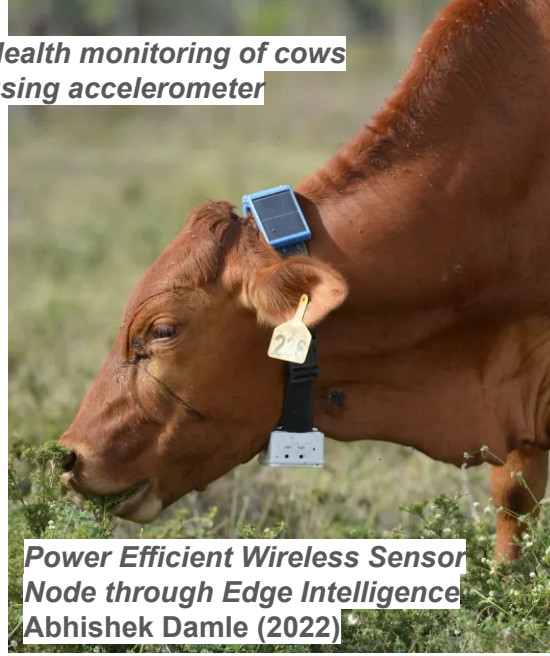
Sensor reading

Send

Send accelerometer data continuously

6 kB bytes/minute





Health monitoring of cows using accelerometer

Power Efficient Wireless Sensor Node through Edge Intelligence
Abhishek Damle (2022)

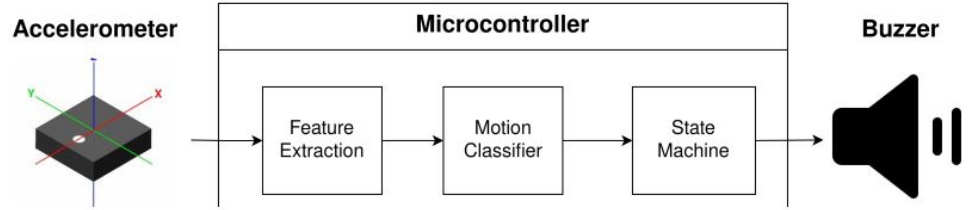
Classify data using a **Decision Tree**

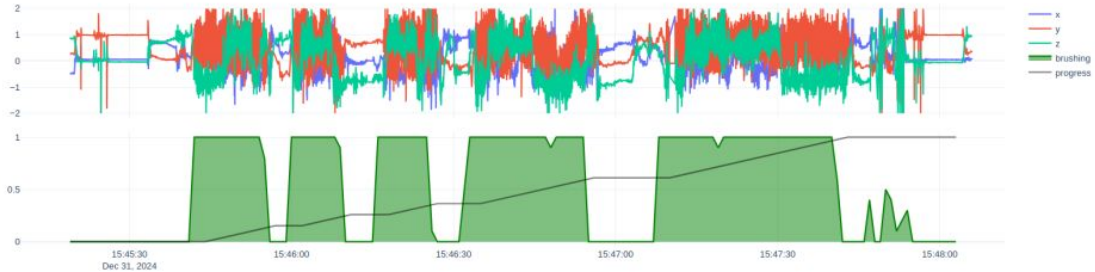
lying/walking/standing/grazing/ruminating

6 bytes / minute

Automatic toothbrush timer

https://github.com/jonnor/toothbrush/





- Stock hardware: M5StickC PLUS 2 by M5Stack
- ~500 lines of MicroPython code
- Collected and labeled data in ~1 hour

Data collection and training tools:
https://github.com/emlearn/emlearn-micropython/tree/master/examples/har_trees

Health monitoring for cattle

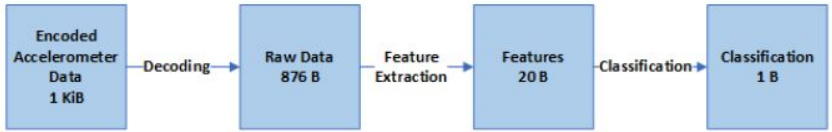
Accelerometer data
used to **detect abnormal behavior**.
Can indicate **health issues** or other problems

Classified using a **Decision Tree**
lying/walking/standing/grazing/ruminating

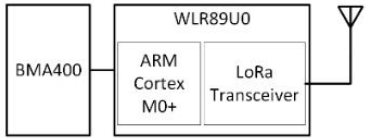
Activity is transmitted using **LoRaWAN**

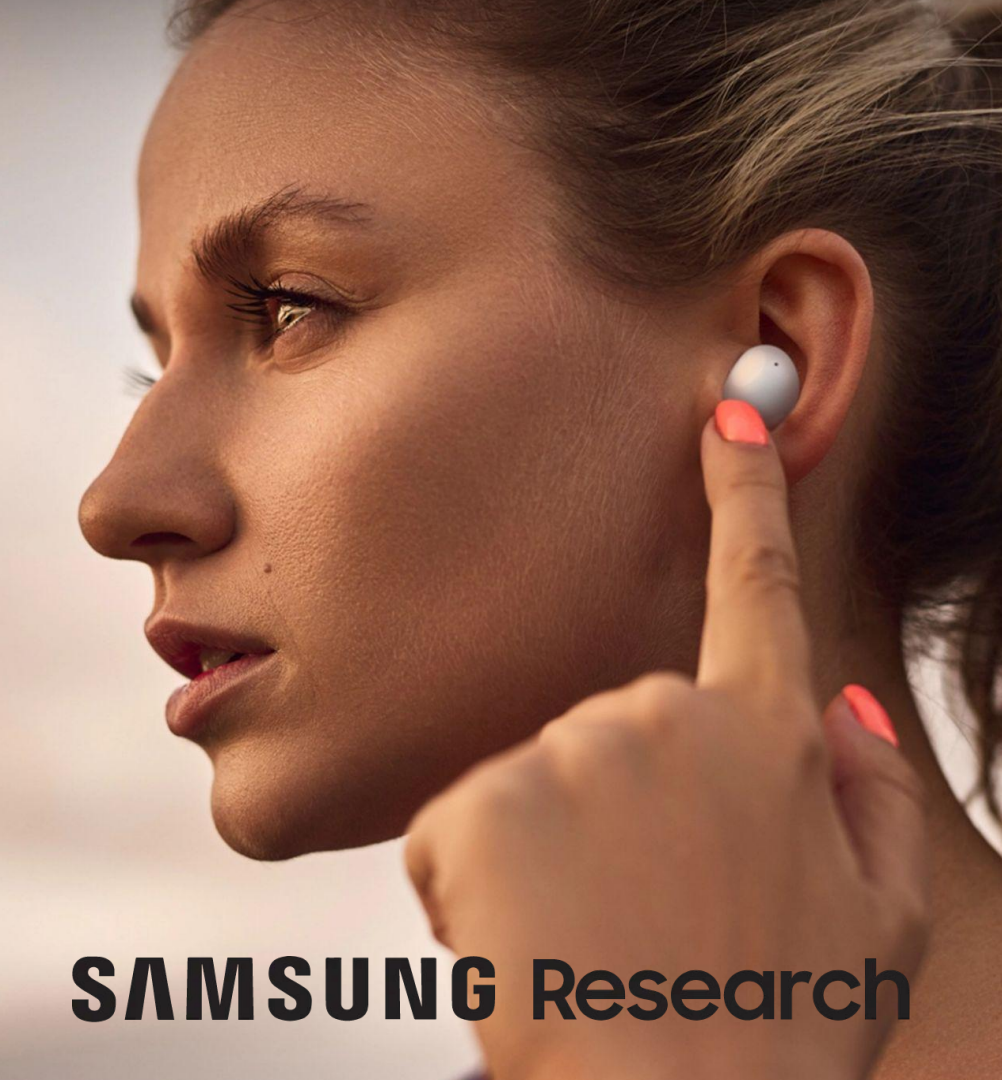
Running ML on-sensor: **under 1 mW**

50 times lower power than sending raw data



Power Efficient Wireless Sensor Node through Edge Intelligence
Abhishek Damle (2022)



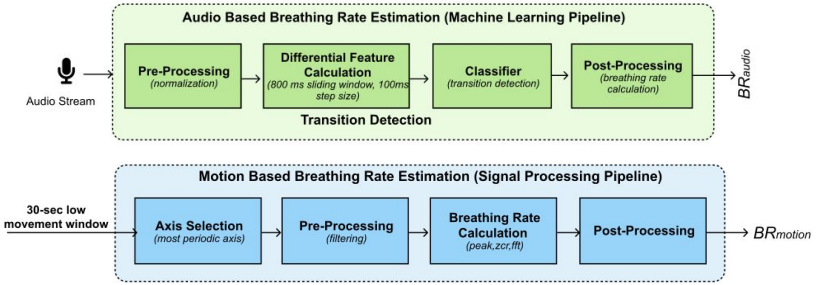


Health monitoring earable

Breathing rate is critical to monitor for persons with **respiratory health problems**.

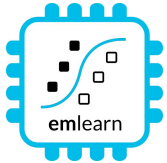
Monitoring **for every-day use**, not just in clinical context. Using earable to replace specialized devices.

Random Forest classifier for audio.



Remote Breathing Rate Tracking in Stationary Position Using the Motion and Acoustic Sensors of Earables
Tousif Ahmed et.al (2023)

Audio API



Digital mic (DMIC)
provides batching natively

Audio	**Digital mic (DMIC)** Digital Audio Interface (DAI)	https://docs.zephyrproject.org/latest/hardware/peripherals/audio/dmic.html
Images (camera)	**Video API**	https://docs.zephyrproject.org/latest/hardware/peripherals/video.html