Milliwatt sized machine learning

on microcontrollers using emlearn

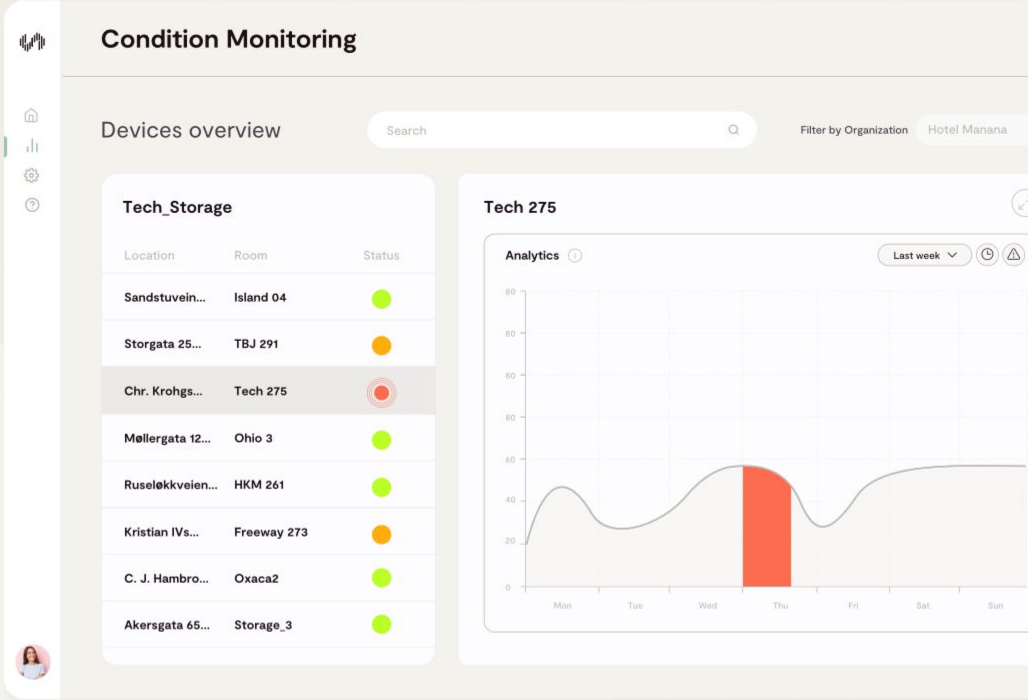
https://github.com/emlearn/emlearn

FOSDEM 2025, Brussels
Low level AI hacking devroom

Jon Nordby jononor@gmail.com

We utilise sound and vibration analysis to detect and warn you of upcoming errors in your technical infrastructure before they happen.

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

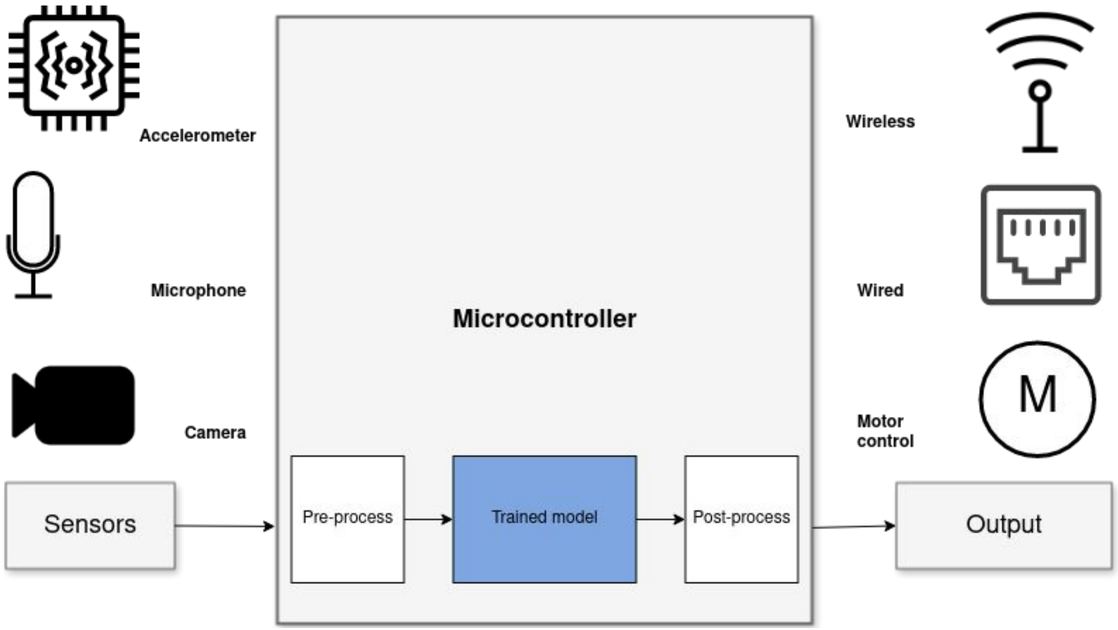
Outline / agenda

1. Why ML on microcontrollers (“**TinyML**”)
2. About the **emlearn project**
3. **Projects** made with emlearn
4. Quick how-to for **emlearn**

Introduction

How is machine learning relevant for constrained embedded devices and microcontrollers?

Sensor data analysis with Machine Learning



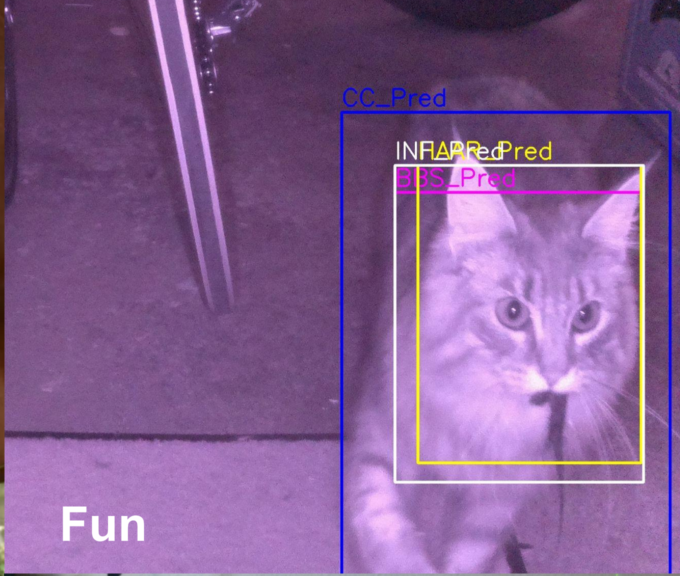
System diagram for “TinyML” solution

Benefits vs sending data to cloud

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


Hey Google...
Consumer Tech


Industrial

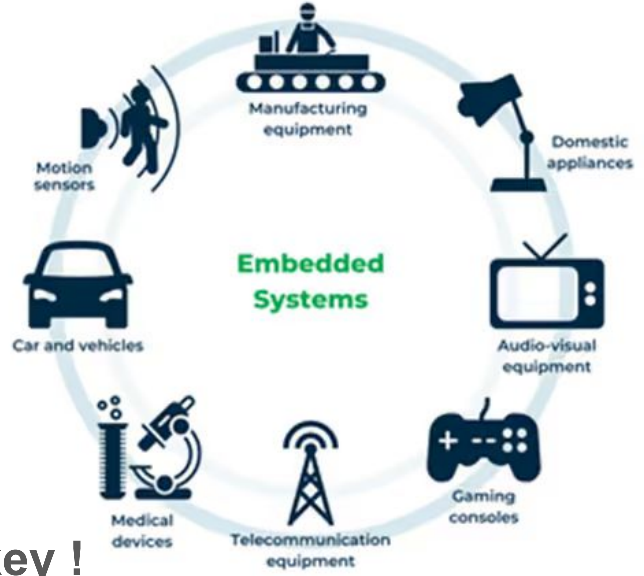

CC_Pred
BBS_Pred
Fun


Heart Rate
15:34

Microcontroller - tiny programmable chip

Compute power: 1 / 1000x of a smartphone

- **RAM:** **0.10 - 1 000 kB**
- **Program space:** **1.0 - 10 000 kB**
- **Compute** **10 - 1000 DMIPS**
- **Price:** **0.10 - 10 USD**



Over 20 *billions* shipped *per year*

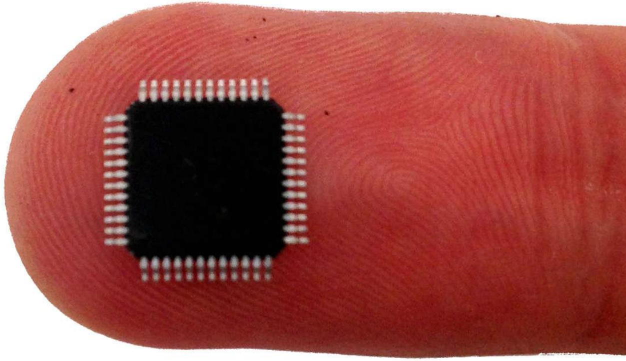
Efficiency is key !
Memory, compute, power

Increasingly accessible for hobbyists

2010: Arduino Uno
2016: ESP32
2021: RP2040
2024: RP2350

Arduino IDE C/C++

MicroPython



About emlearn

Started in 2018

Open-source (MIT)

1. Train on PC

pip install emlearn

Convenient training

- Model creation **in Python**
- **Use standard libraries**
  a. scikit-learn
  b. Keras
- One-line to **export to device**
- Verification tools included

2. Deploy on device

emlearn

Embedded Friendly

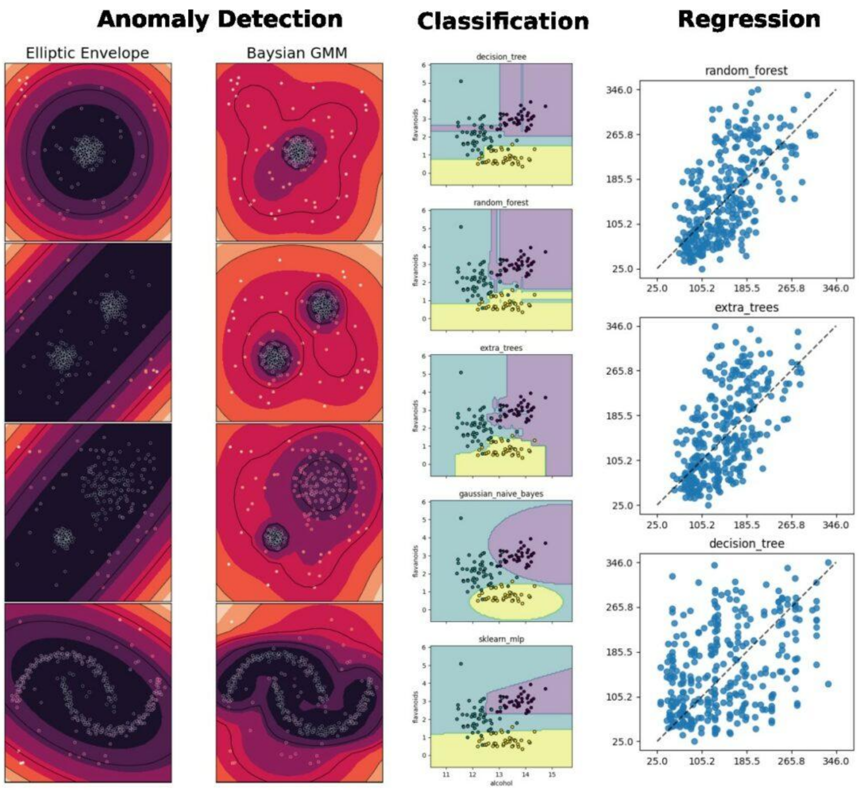
- **Portable C99** code
- No dynamic allocations
- **Header-only**
- High **test coverage**
- Integer/**fixed-point** math *
- **Small**. 2 kB+ FLASH

emlearn
-micropython

Convenient & Efficient

- **Portable MicroPython**
- **Single .mpy file**
- No dynamic allocations
- High **test coverage**
- Integer/**fixed-point** math *
- **Small**. 2 kB+ FLASH

Supported tasks



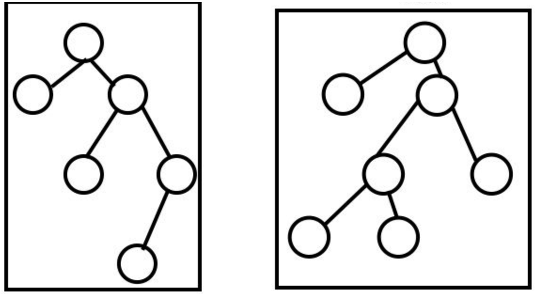
Supports the most common tasks for embedded
& sensor data use cases.

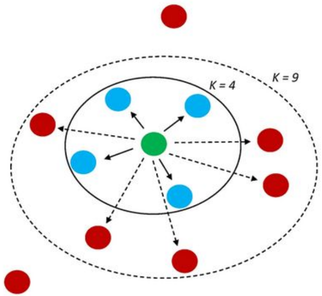
- Classification
- Regression
- Anomaly Detection

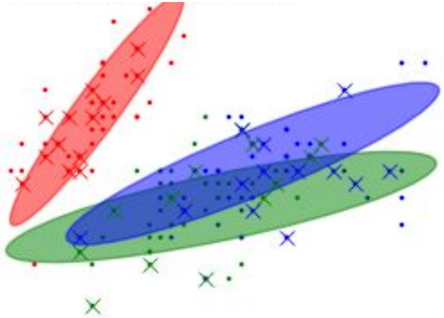
Supported models

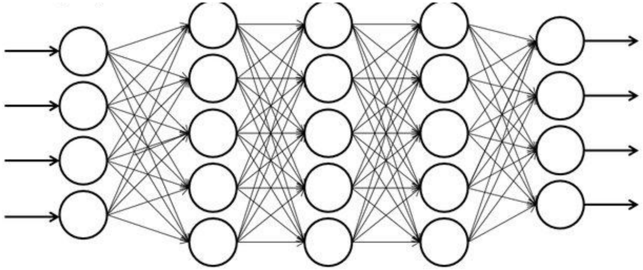
Selection of simple & effective
embedded-friendly models

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









Projects using emlearn

Many real-world uses across the globe

Referred in 40+ publications

Health monitoring for cattle

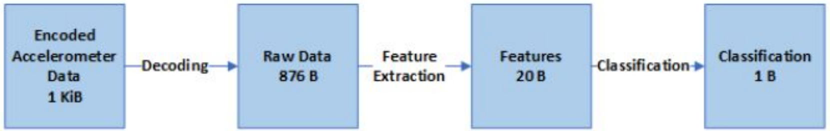
Accelerometer data
used to **detect abnormal behavior**.
Can indicate **health issues** or other problems

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

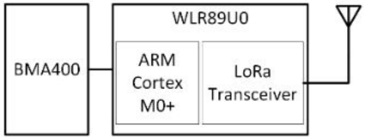
Activity is transmitted using **LoRaWAN**

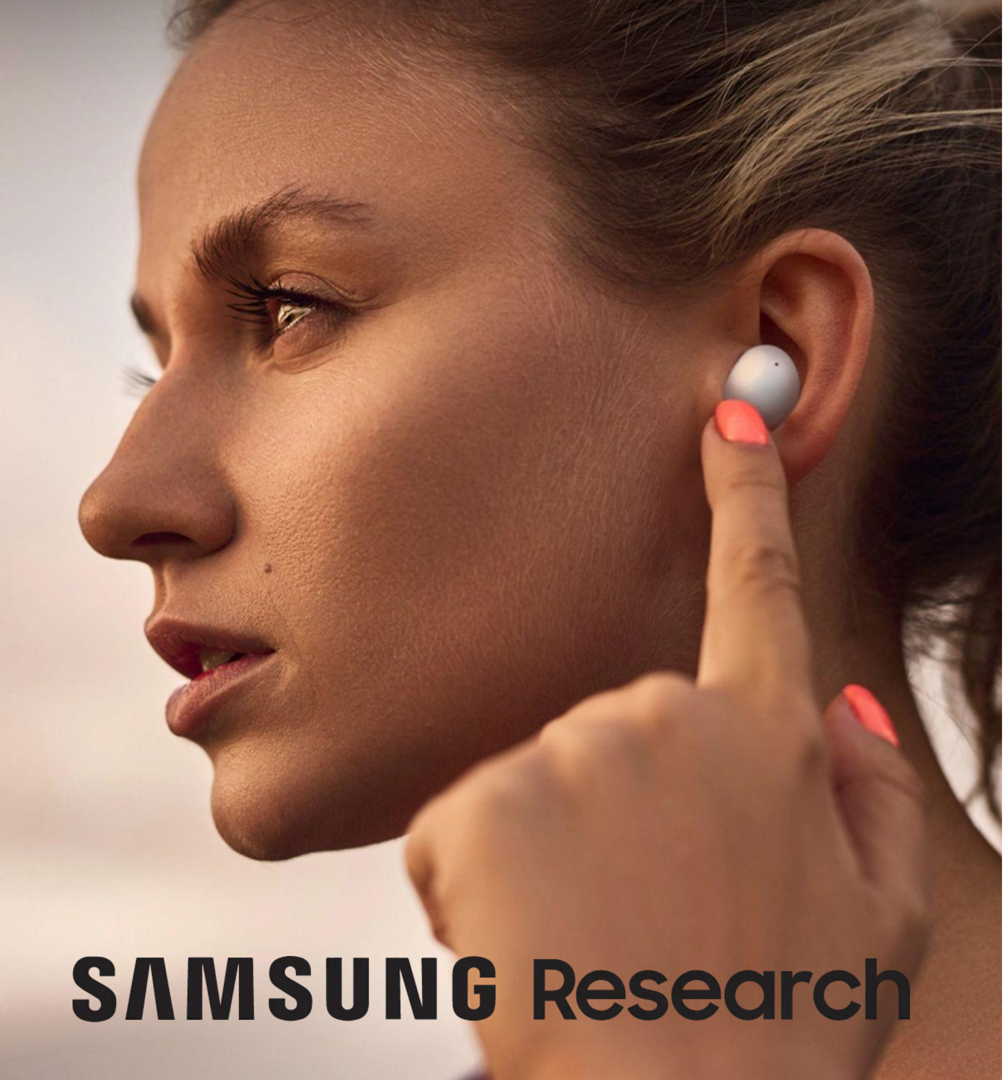
Running ML on-sensor: **under 1 mW**

50 times lower power than sending raw data



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



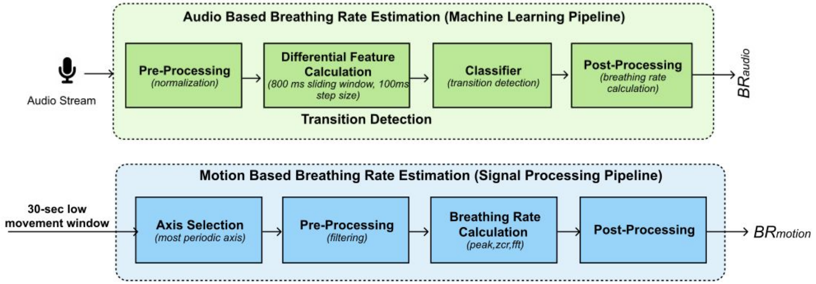


Health monitoring earable

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.

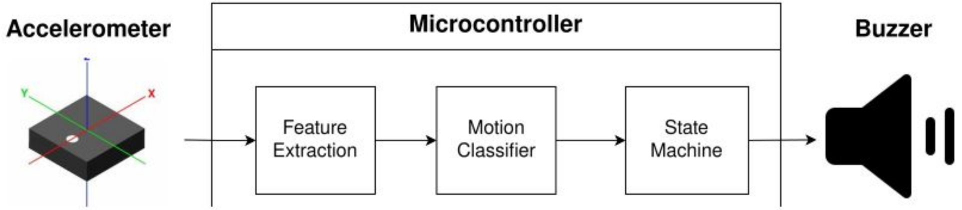
Random Forest classifier for audio.

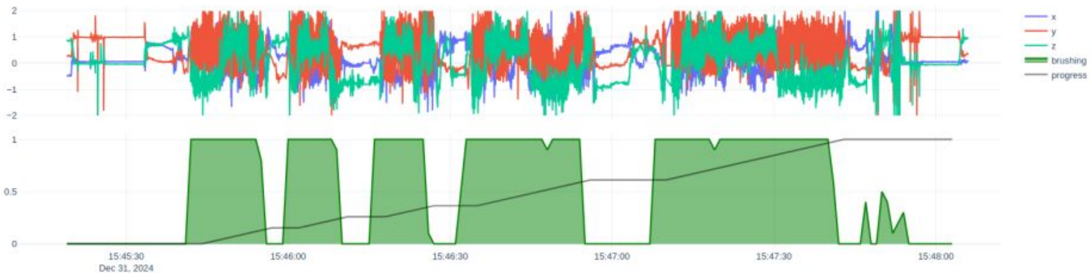


Remote Breathing Rate Tracking in Stationary Position Using the Motion and Acoustic Sensors of Earables
Tousif Ahmed et.al (2023)

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

1 dollar TinyML

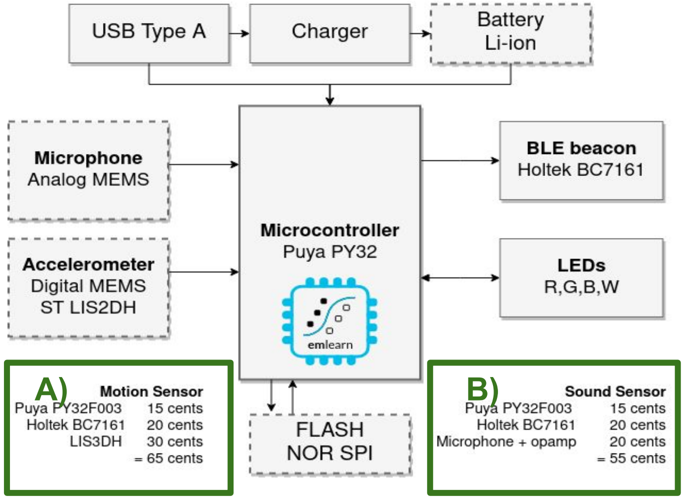
https://hackaday.io/project/194511-1-dollar-tinyml

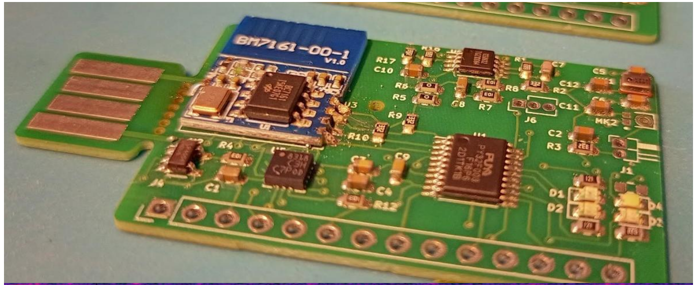
Research question:
Can we make a useful TinyML system
<1 USD in total component cost (BOM)?

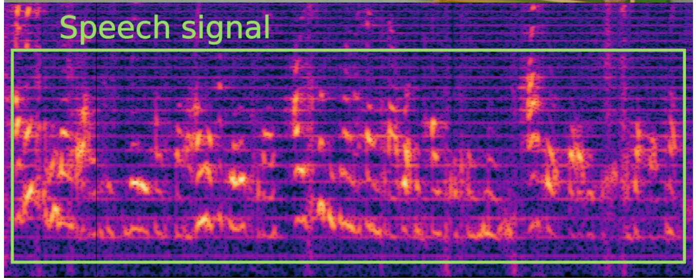
Preliminary answer: YES!
Either motion (accelerometer)
or sound sensor (microphone)

Constraints: 8 kB RAM / 64 kB FLASH

Current status:
Audio streaming works
Revision 2 ordered







emlearn for C

Quick walkthrough

Train and export a model

emlearn

1. Train using standard Python ML libraries.

A) keras neural network

```
from keras import ...

model = Sequential([
   Dense(16, input_dim=n_features, activation='relu'),
   Dense(8, activation='relu'),
   Dense(1, activation='sigmoid'),
])
model.compile(....)
model.fit(X_train, Y_train, epochs=1, batch_size=10)
```

B) scikit-learn neural network

```
from sklearn.neural_network import MLPClassifier
model = MLPClassifier(hidden_layer_sizes=(100,50,25))

model.fit(X_train, Y_train)
```

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

```
import emlearn

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

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

```
#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

Using the C code

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

emlearn

```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

emlearn
-micropython

emlearn for 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 16 kB+ RAM
Supports 8+ microcontroller families

As compatible with CPython as possible, within constraints.
No CFFI or C module compatibility!
TLDR: Small .py scripts will mostly work with minor mods
But not PyData stack

“mip” package manager
Can load C modules at runtime!

More info: https://micropython.org

MicroPython

RP2040

STM32

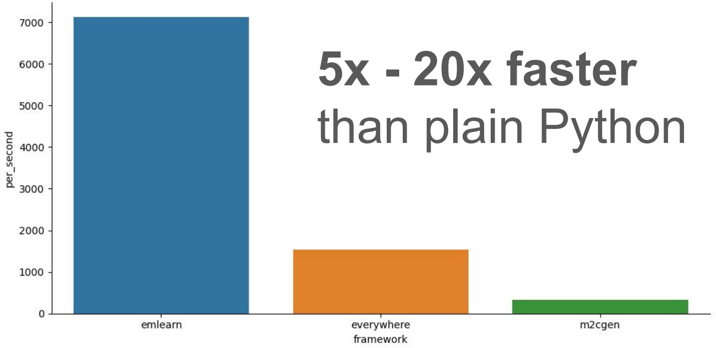
ESP32

MicroPython library for emlearn

emlearn -micropython

Combine the **convenience, familiarity and productivity of Python** - with the **speed of C**

Modules installable at runtime (.mpy)



5x - 20x faster
than plain Python

emlearn_iir	Infinite Impulse Response filters	
emlearn_trees	Random Forest	
emlearn_fft	Fast Fourier Transform	
emlearn_cnn	Convolutional Neural Networks	**Image classification+**
emlearn_neighbors	K-nearest Neighbors	**On-device learning**

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

Install & Export model

emlearn
-micropython

```
import emlearn

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

Export model on PC

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

```
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

Outro

Summary

1. Machine Learning used in embedded systems
   to **automatically analyze sensor data**

2. Practical applications possible with just a
   kilo-bytes RAM, **milli-watts of power**, **couple dollars**

3. **emlearn** is an open-source project
   to help **deploy ML models to microcontrollers**
   - running **C** or **MicroPython**

More resources

Official documentation

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

PyCon Berlin 2024: **Machine Learning on microcontrollers using MicroPython and emlearn**
https://www.youtube.com/watch?v=S3GjLr0ZIE0

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

PyData ZA 2024: **Sensor data processing on microcontrollers with MicroPython**
https://za.pycon.org/talks/31-sensor-data-processing-on-microcontrollers-with-micropython/

Milliwatt sized machine learning

on microcontrollers using emlearn

https://github.com/emlearn/emlearn

FOSDEM 2025, Brussels
Low level AI hacking devroom
Jon Nordby jononor@gmail.com

Bonus

MicroPython - Installing

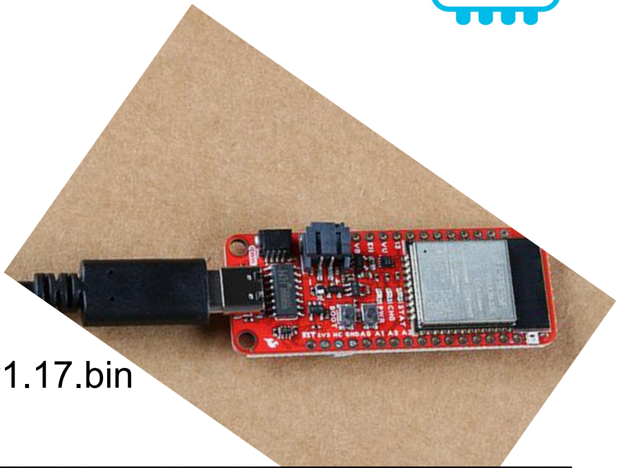
Download prebuilt firmware

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

Flash firmware to device

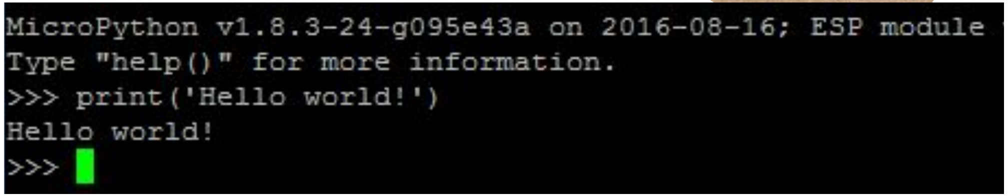
```
pip install esptool

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
```



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

TinyML for MicroPython - comparisons

Project	Deployment	Models	Program size	Compute time
emlearn	Easy. Native mod .mpy	DT, RF, KNN, CNN	Good	Good
everywhereml	Easy. Pure Python .py	DT, RF, SVM, KNN,	High with large models	Poor
m2cgen	Easy. Pure Python .py	DT, RF, SVM, KNN, MLP	High with large models	Poor
OpenMV.tf	Hard. Custom Fork	CNN	High initial size	Good
ulab	Hard. User C module	(build-your-own) Using ndarray primitives	High initial size	Unknown (assume good)

Task feasibility versus microcontroller type

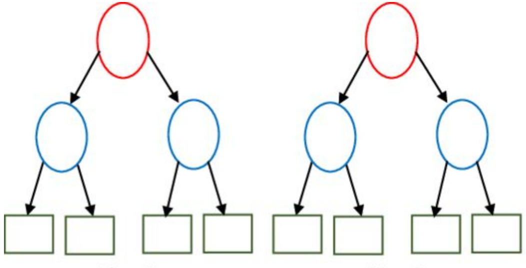
	RAM (kB)	**FLASH** (kB)	**CPU** (CoreMark)		**IMU** 100 hz 5 s	**Sound** 16 kHz 1 s	**Image** 64x64 px 1 fps
Arduino Uno ATMega 328 @ 16 Mhz	2	32	4.0		✓	❌	❌
Arduino Nano BLE 33 NRF52840 ARM Cortex M4F @ 64 Mhz	256	1 000	215		✓	✓	❌
Arduino Nano ESP32 ESP32-S3 @ 240 Mhz	8 000	16 000	613 (per core)		✓	✓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

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

Lossless	0.0	**1.0**	0.0	0.0
	0.0	0.0	**1.0**	0.0
	0.0	0.4	**0.6**	0.0
Not ideal	0.15	0.20	**0.30**	0.15

....

1
2

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")

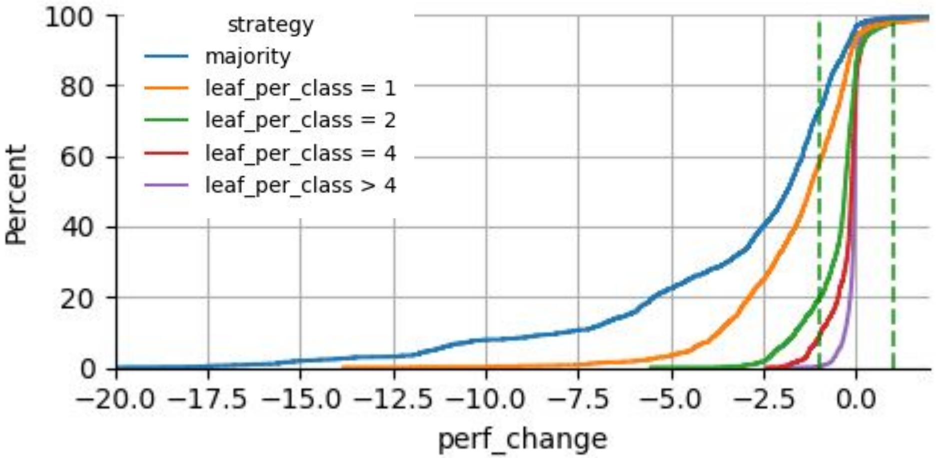
- \+ Supports int8/int16/int32 and float

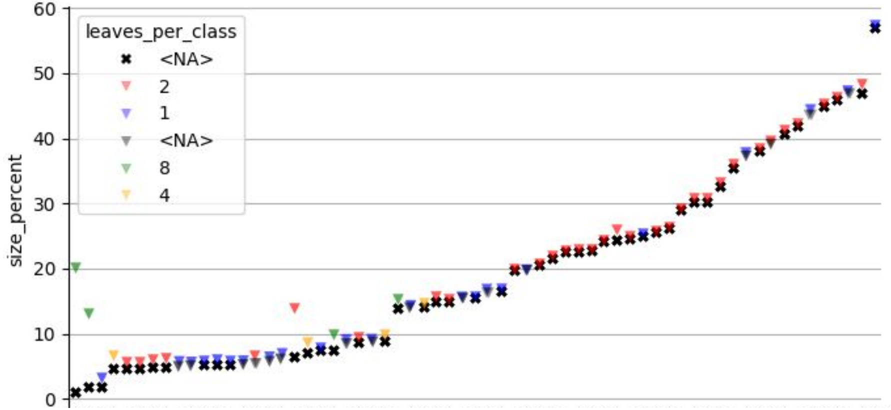
```
static inline int32_t
xor_model_tree_0(const float *features, int32_t
features_length)
{
      if (features[1] < 0.197349f) {
          if (features[0] < 0.466316f) {
              return 0;
          } else {
              return 1;
          }
      } else {
          if (features[1] < 0.256702f)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

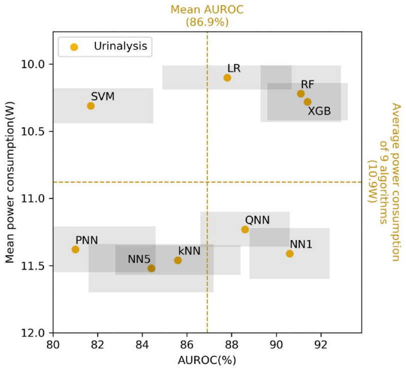
2-8 leaves per class sufficient
0% of datasets failed performance constraint
High model size compression 50-90%

Trees - relevant across the task complexity range

Low

Go-to for tabular data

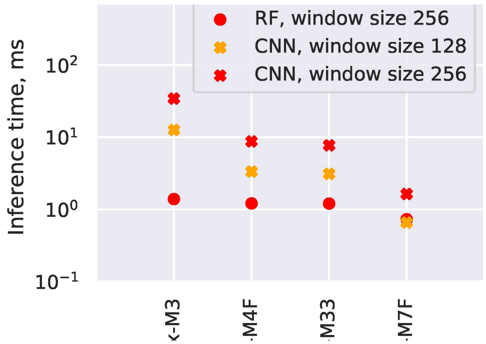
Very easy to get good performance, with minimal tweaking.



Medium

Alternative to deep learning

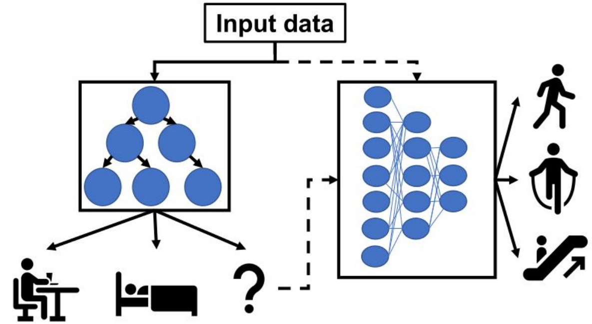
Elsts et.al [1] showed performance matching a deep-learning approach, at 10x to 100x better resource usage



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/