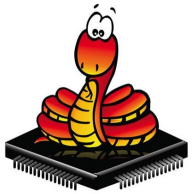
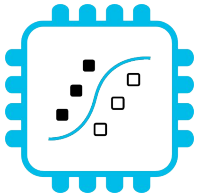
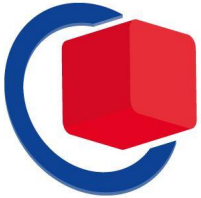
Bridging the TinyML language gap



with MicroPython and emlearn



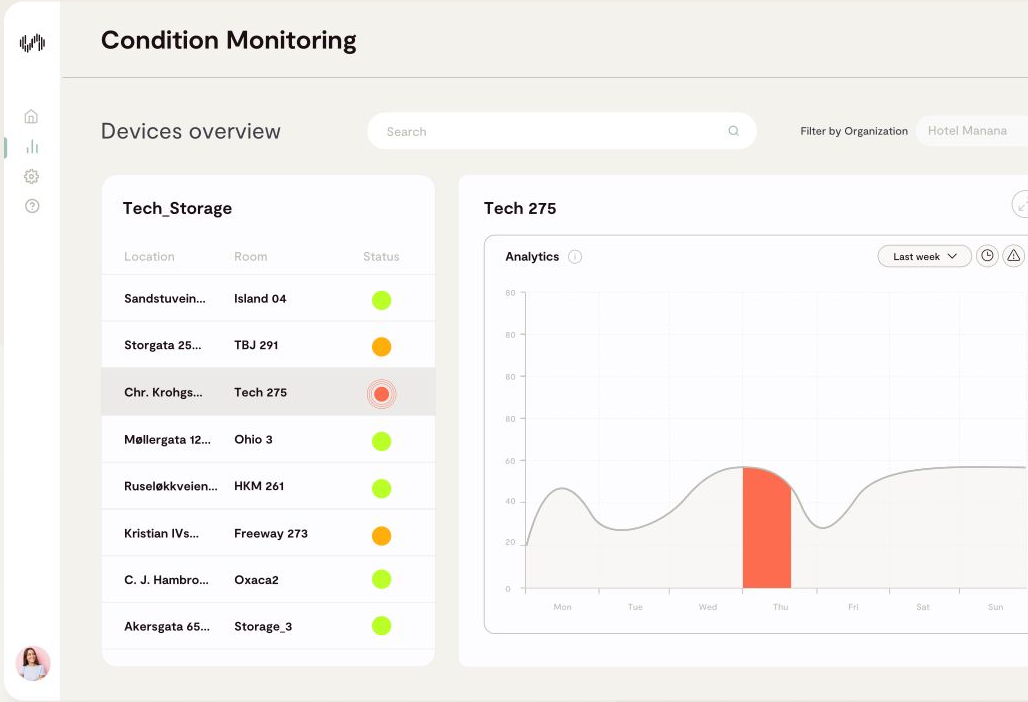
https://github.com/emlearn/emlearn



embeddedworld
Exhibition&Conference

Jon Nordby
jononor@gmail.com
March 2025, Nuremberg

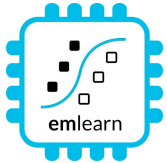
soundsensing



Trusted by Nordic market leaders

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

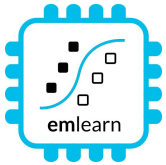
Outline / agenda



1. Brief introduction to **TinyML**
2. TinyML with emlearn
    - **emlearn** - overview, C library
    - emlearn + **Zephyr RTOS**

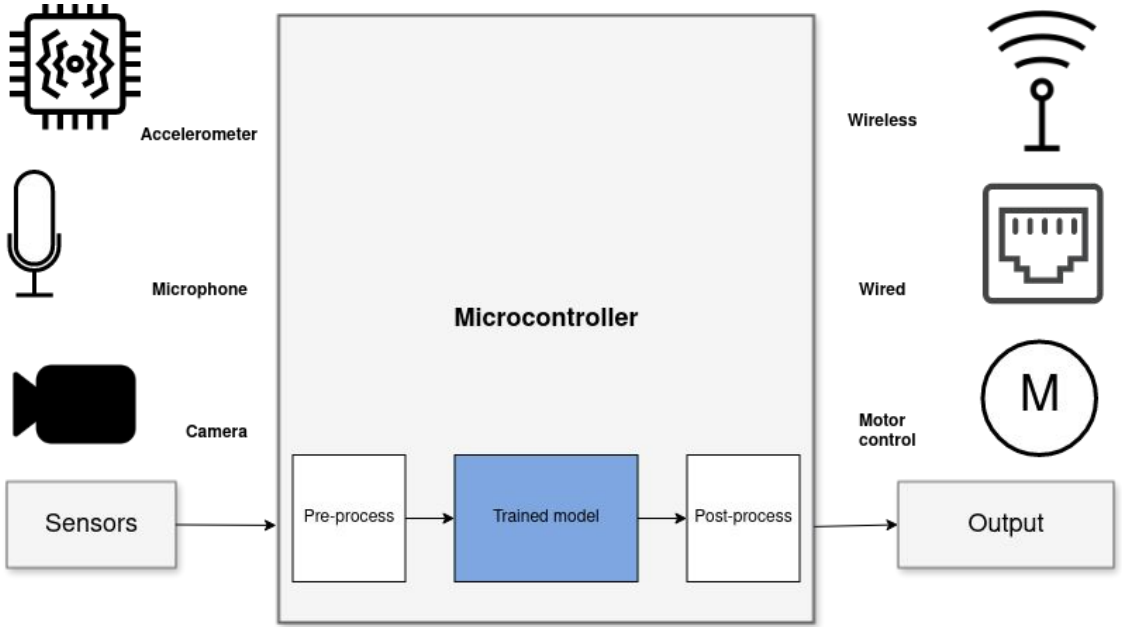
2. Bridging the TinyML language gap

    - **MicroPython**
    - emlearn-micropython



TinyML overview

Sensor data analysis with Machine Learning

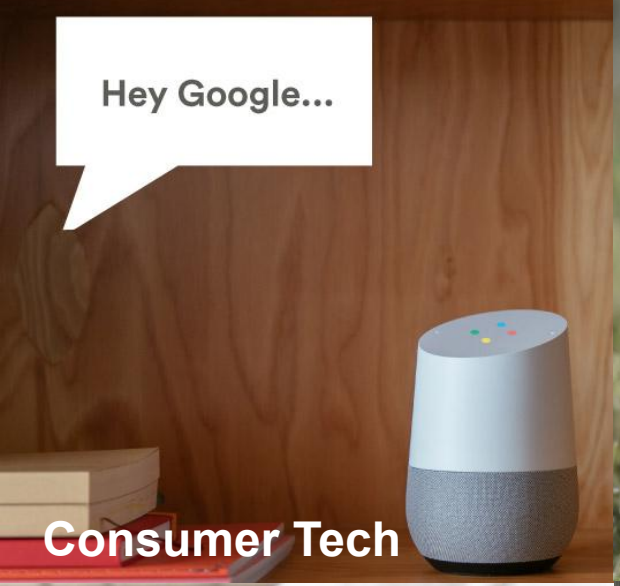


Benefits vs sending data to cloud

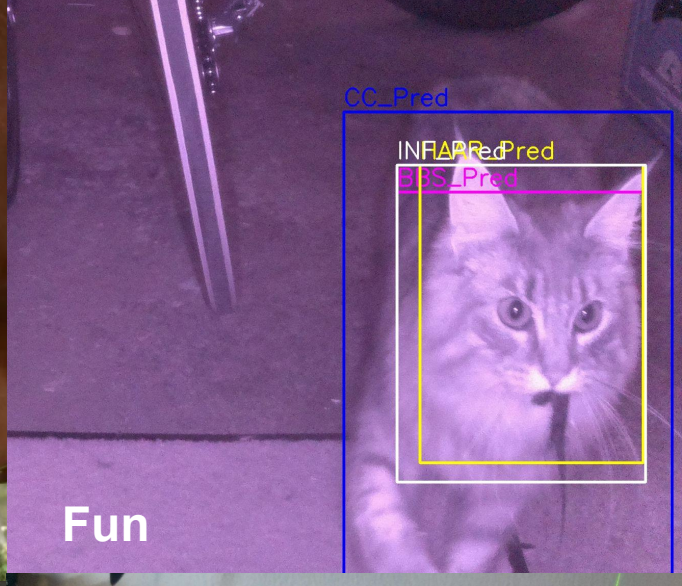
- Stand-alone
- Low latency
- Power efficiency
- Privacy
- Low cost

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

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


Hey Google...
Consumer Tech


Industrial

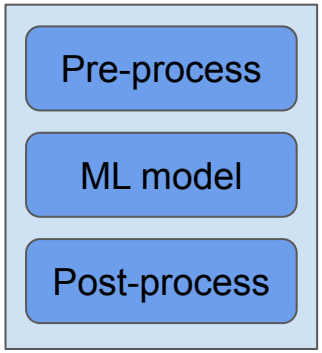

CC_Pred
INFRARed_Pred
BBS_Pred
Fun


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

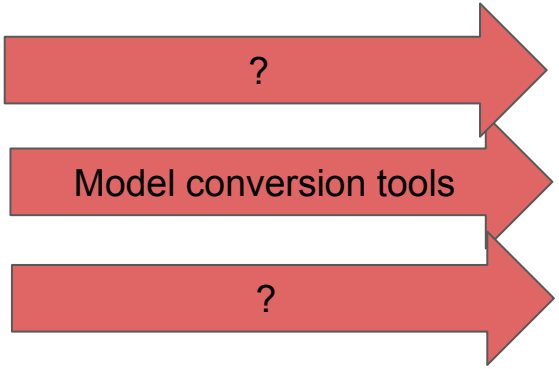
TinyML challenge: **PC/host** vs **embedded/device** pipeline

1) Train model
on PC

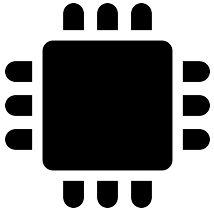
RAM (min)
10 GB+

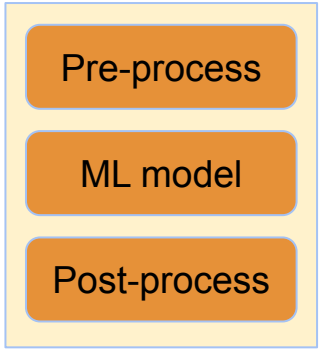


Deep-learning pipeline



2) Run
on Device





RAM (max):
1 MB

Embedded optimized pipeline

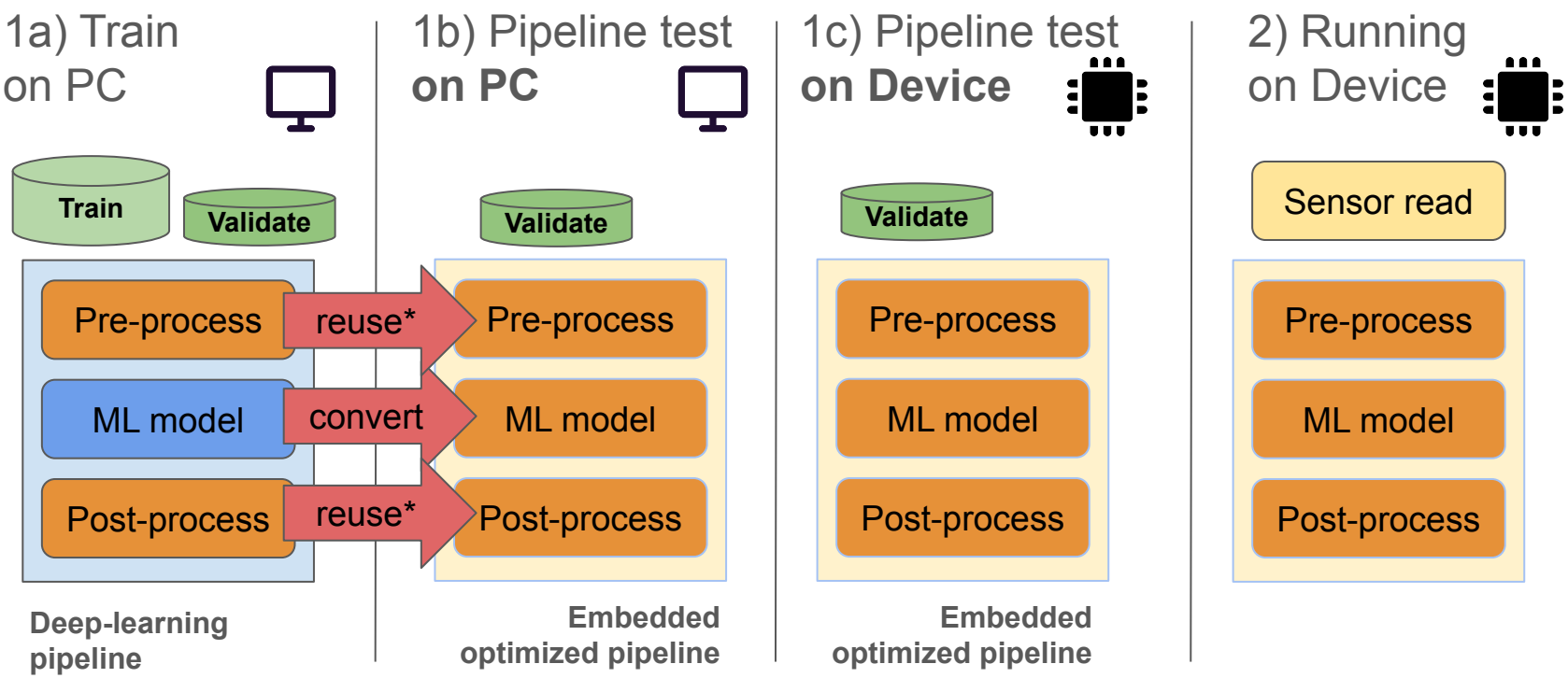
Challenges

- 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

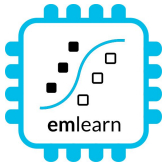
"Data Scientist land"

"Firmware Engineer land"

Strategy: Step-wise validation, code-sharing



* Requirements: Pre- and -post processing must be portable between device and PC
Can be done in embedded C/C++ etc, or (MicroPython)



About emlearn

Started in 2018

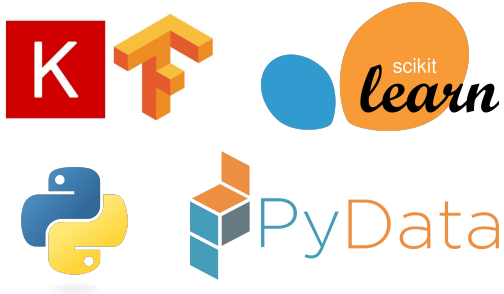
Open-source (MIT)

1. Train on PC

$ pip install emlearn

Convenient training

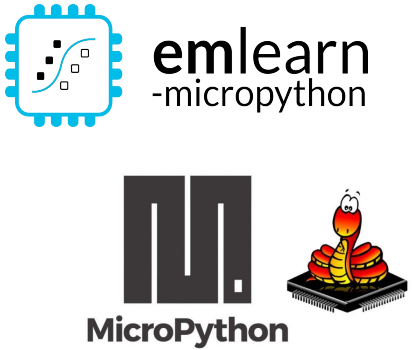
- Model creation **in Python**
- **Use standard libraries**
  a. scikit-learn
  b. Keras
- One-line to **export to device**
- Verification tools included



2. Deploy on device

Embedded Friendly

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

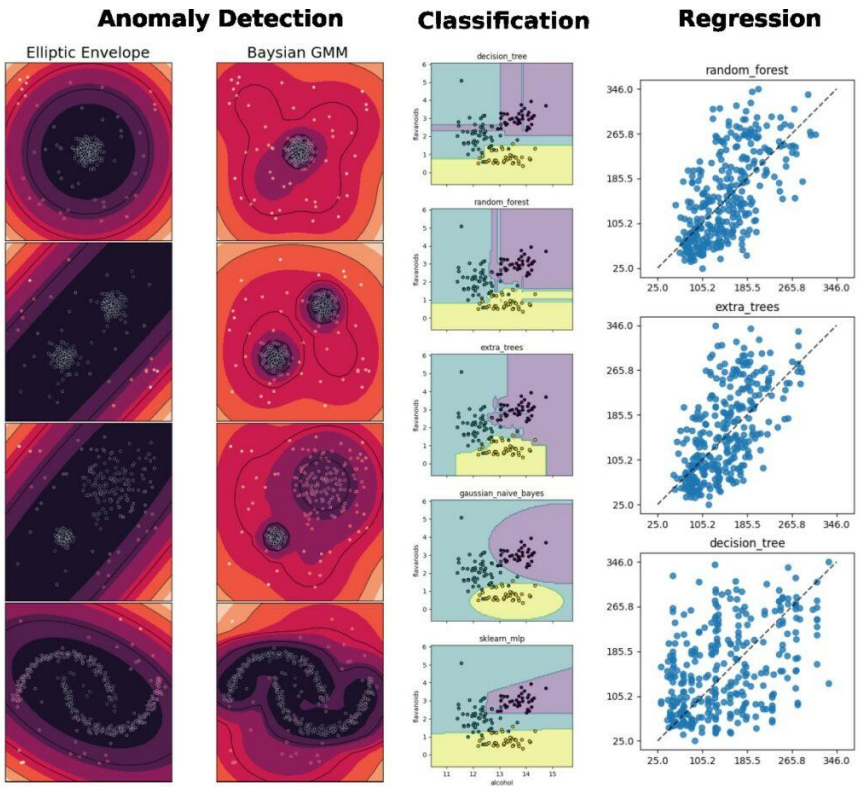


Convenient & Efficient

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

All code runs both on host/PC and device

Supported tasks

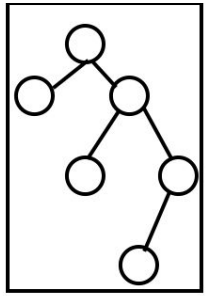


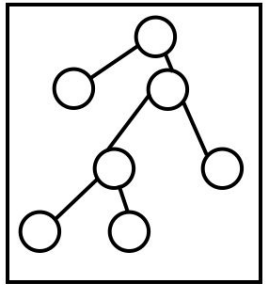
Supports the most common tasks for embedded
& sensor data use cases.

- Classification
- Regression
- Anomaly Detection

Supported models

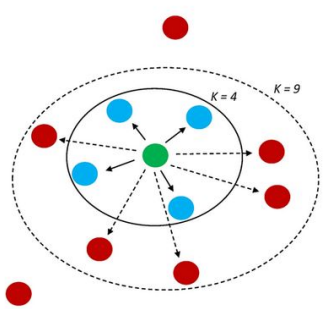
Selection of simple & effective embedded-friendly models

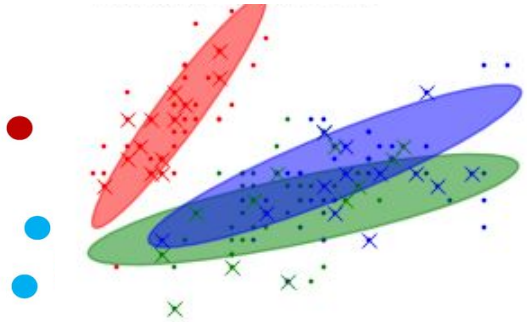


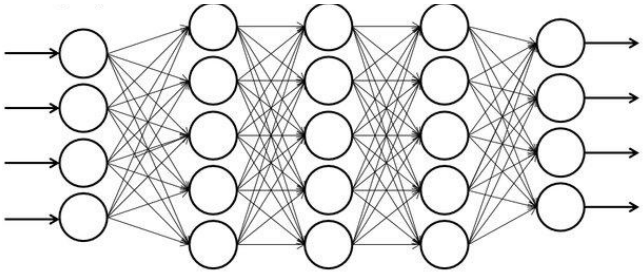


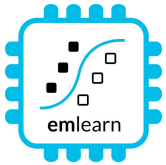
emlearn

- 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

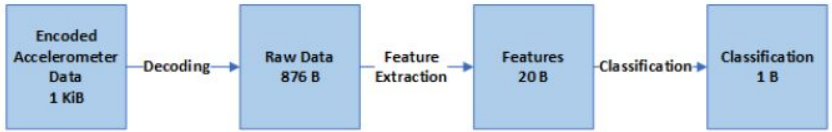
Accelerometer data
used to **detect abnormal behavior**.
Can indicate **health issues** or other problems

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

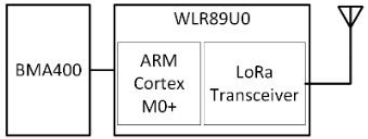
Activity is transmitted using **LoRaWAN**

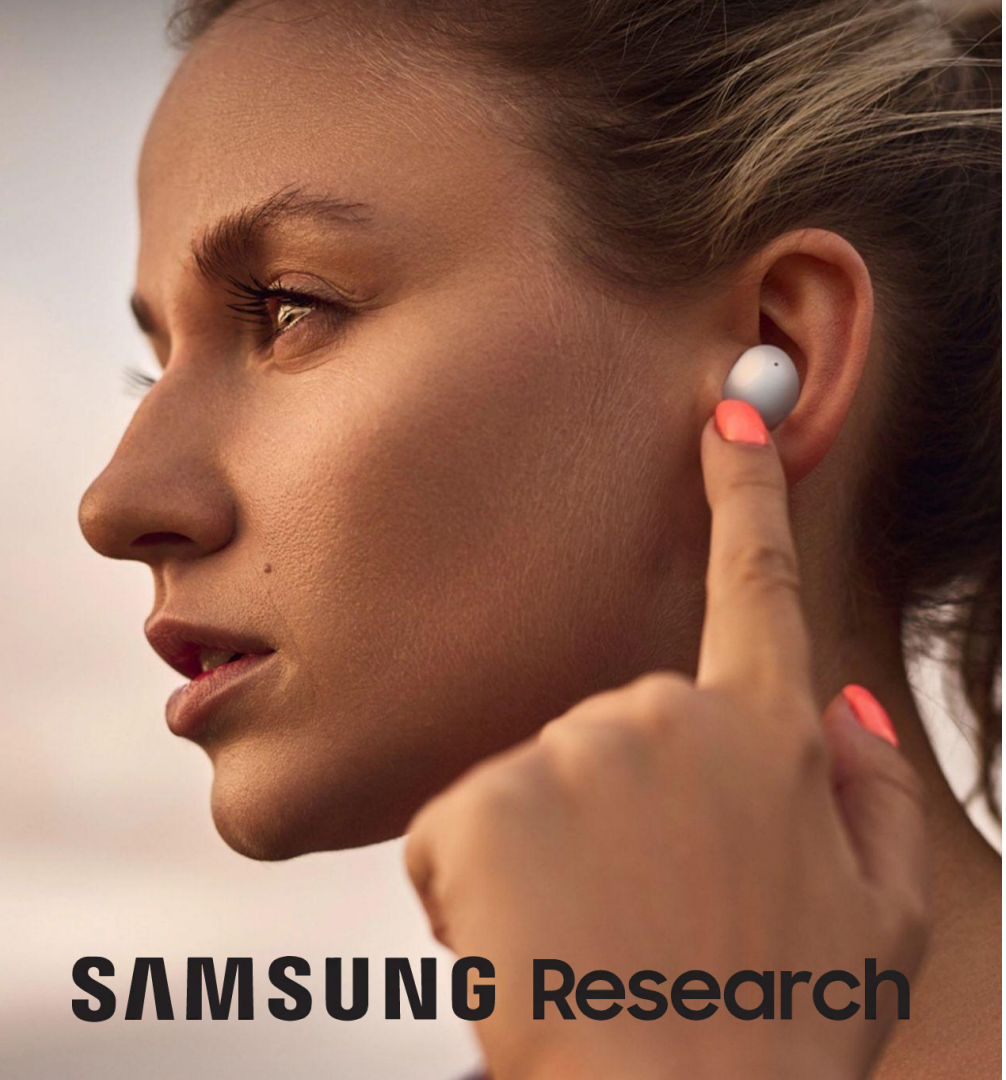
Running ML on-sensor: **under 1 mW**

50 times lower power than sending raw data



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



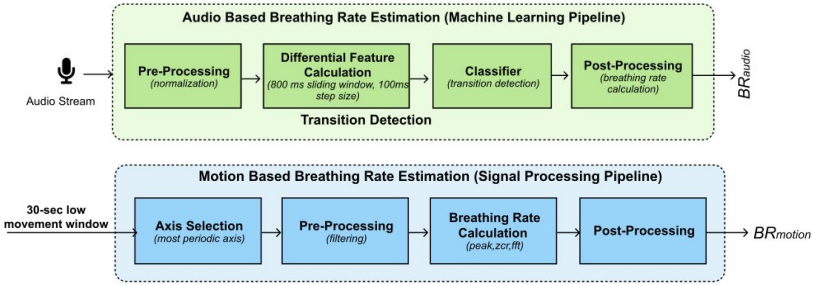


Health monitoring earable

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.

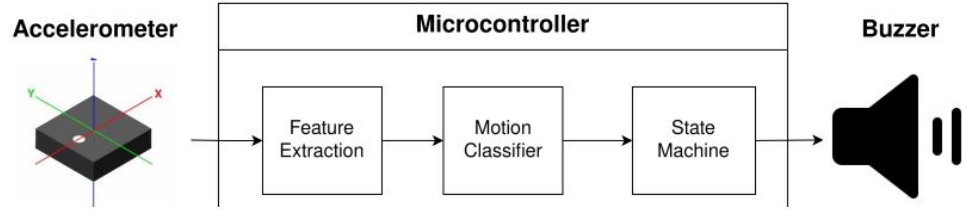
Random Forest classifier for audio.

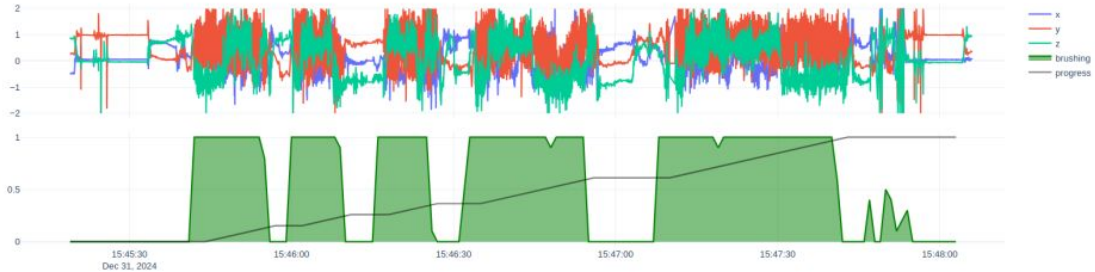


Remote Breathing Rate Tracking in Stationary Position Using the Motion and Acoustic Sensors of Earables
Tousif Ahmed et.al (2023)

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

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')
```

Example of generated code

```
#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);
    }
.....
```

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("");
}
```

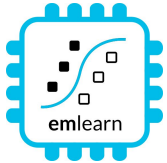
loop

emlearn

emlearn on Zephyr

Zephyr®

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
```

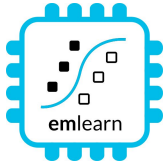
3. Use in your code

```
#include <eml_net.h>
```

First class support
since February 2024

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

Zephyr sensor data APIs

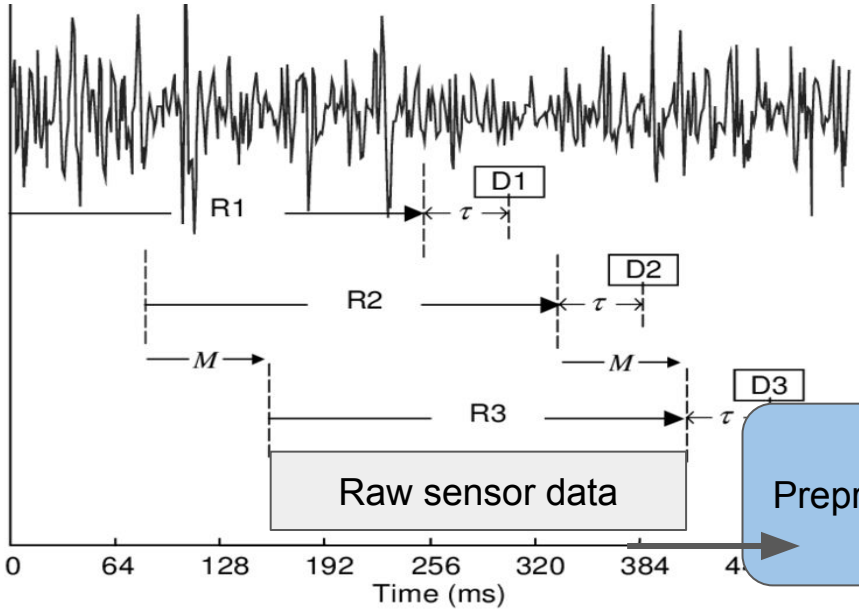


★ Unified hardware API across all platforms
★ Large amount of drivers included out-of-the-box

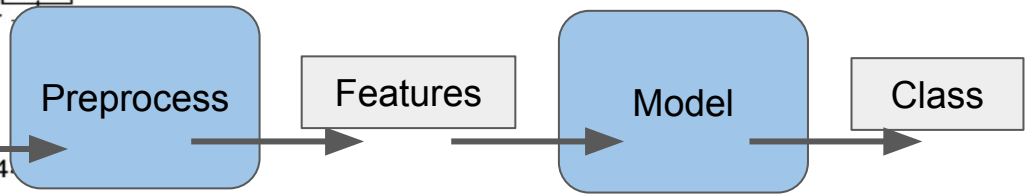
Examples

IMU (accelerometer/gyro)	**Sensors API**	https://docs.zephyrproject.org/latest/hardware/peripherals/sensor/index.html
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

ML on streams: Continuous classification



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

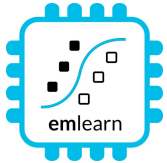


[x1 y1 z1
x2 y2 z2
….]

[42, 4002, … , 329]

“Jumping Jacks”

Chunked sensor reader for Zephyr




```
struct sensor_chunk_msg chunk;

if (!device_is_ready(sensor) {
    printk("sensor: device %s not ready.\n", sensors[i]->name);
    return 0;
}

// Setup sensor
set_sampling_freq(sensor, SAMPLINGRATE);

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

while (1) {

    // check for new data
    const int get_status = k_msgq_get(sensor_reader.queue, &chunk, K_NO_WAIT);
    if (get_status != 0) {

        process(chunk.buffer, chunk.length);
    }

        k_msleep(100);
}
```

Work-in-progress

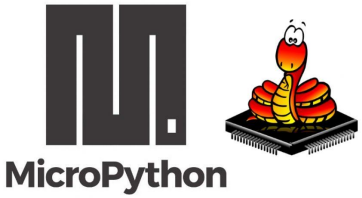
Practical examples using sensor data for Zephyr

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

Dedicated thread handling time-sensitive fetching

Queue with complete chunks of data

Allows application to process at convenient time

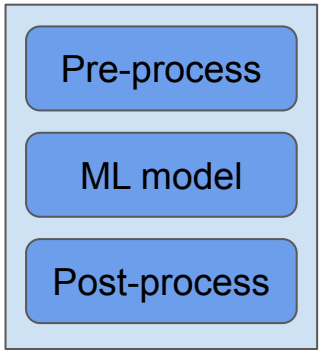


Bridging the TinyML language gap

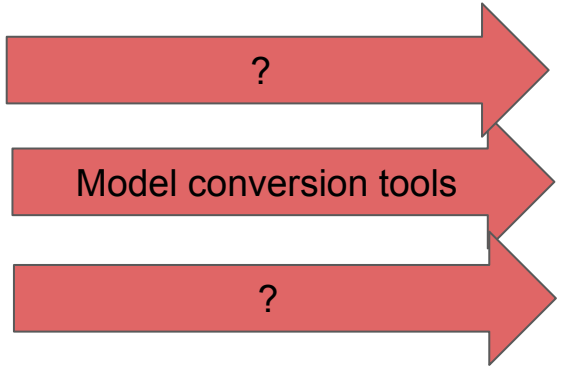
TinyML challenge: **PC/host** vs **embedded/device** pipeline

1) Train model
on PC

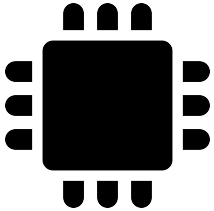
RAM (min)
10 GB+

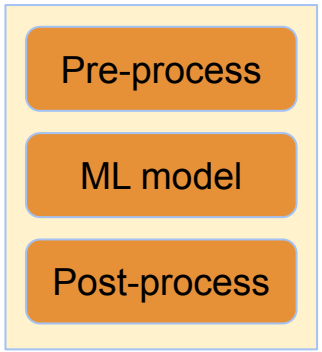


Deep-learning pipeline



2) Run
on Device





RAM (max):
1 MB

Embedded optimized pipeline

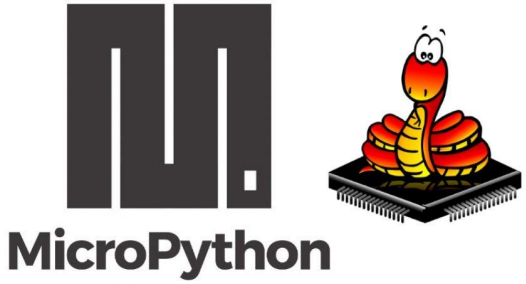
Challenges

- 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"

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 standard PyData stack

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

More info: https://micropython.org

C modules *

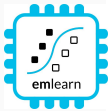
Defines a Python module with API. functions/classes et.c.

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

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

* Or **other language** which **compiles to C**, or **exposes C API**
https://github.com/vshymanskyy/wasm2mpy
C++, Rust, Zig, TinyGo, TypeScript

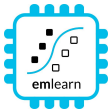
```c
// Include the header file to get access to the MicroPython API
#include "py/dynruntime.h"

// 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
}
```



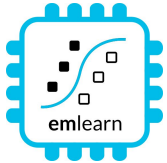
Native module (.mpy) VS External C module

	Native module	**External C module**
Installable at runtime	Yes, as .mpy file	No. Must be included in firmware image
Requires SDK/toolchain	No (only to build)	Yes
Code executes from	RAM	FLASH
Limitations	No libc / libm linked * No static BSS *	None
Maturity	Low *	Excellent
Documentation	https://docs.micropython.org/en/latest/develop/natmod.html	https://docs.micropython.org/en/latest/develop/cmodules.html

* Improved greatly in upcoming MicroPython (1.25+).
Contributions by Volodymyr Shymanskyy, Alessandro Gatti, Damien George, and others



MicroPython Data Science ecosystem



CPython

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

MicroPython

- emlearn-micropython
- OpenMV

https://openmv.io/

- ulab

https://github.com/v923z/micropython-ulab

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

emlearn for MicroPython

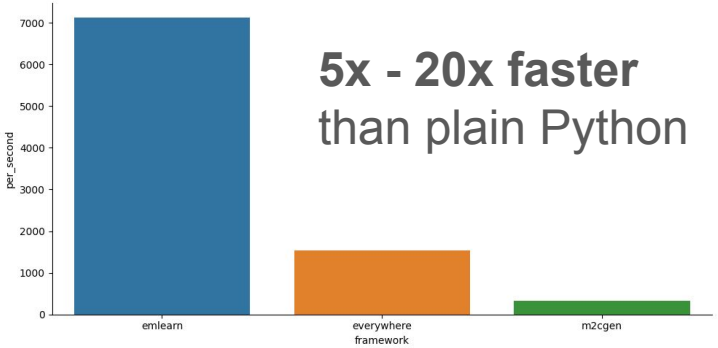
Python is the lingua franca
for Machine Learning

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

MicroPython library for emlearn

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
emlearn_neighbors	K-nearest Neighbors

Image classification+
On-device learning

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

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

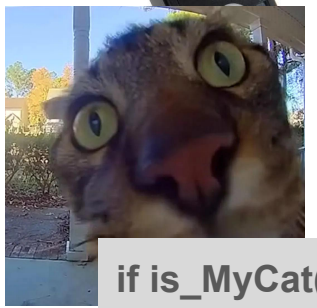
Noise monitor

Microphone

Infinite Impulse Response filters
emlearn_iir

Image Classifier

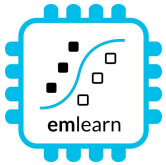
Camera



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

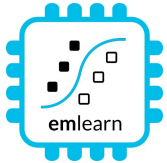
Examples at
https://github.com/emlearn/emlearn-micropython#examples

Convolutional Neural Network
emlearn_cnn



Outro

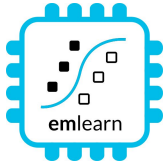
Summary 1/2



TinyML pipelines must be
compatible between train/PC and **deploy/device**

Recommend using **portable code** with
automated validation tests on both device and PC

Summary 2/2

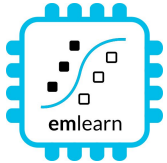


emlearn is an portable open-source project
to help **deploy ML pipelines to microcontrollers**

emlearn supports both **C** and **MicroPython**

MicroPython with C modules enables writing
TinyML **applications primarily in Python**

More resources



Official documentation

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

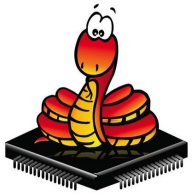
PyCon Berlin 2024: **Machine Learning on microcontrollers using MicroPython and emlearn**
https://www.youtube.com/watch?v=S3GjLr0ZIE0

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

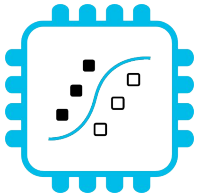
FOSDEM 2025: **MicroPython - Python for microcontrollers and Embedded Linux**
https://fosdem.org/2025/schedule/event/fosdem-2025-4525-micropython-python-for-microcontrollers-and-embedded-linux/

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/

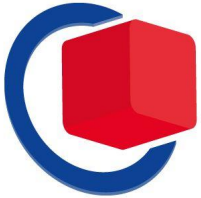
Bridging the TinyML language gap



with MicroPython and emlearn



https://github.com/emlearn/emlearn



embeddedworld
Exhibition&Conference

Jon Nordby
jononor@gmail.com
March 2025, Nuremberg

Bonus

1 dollar TinyML

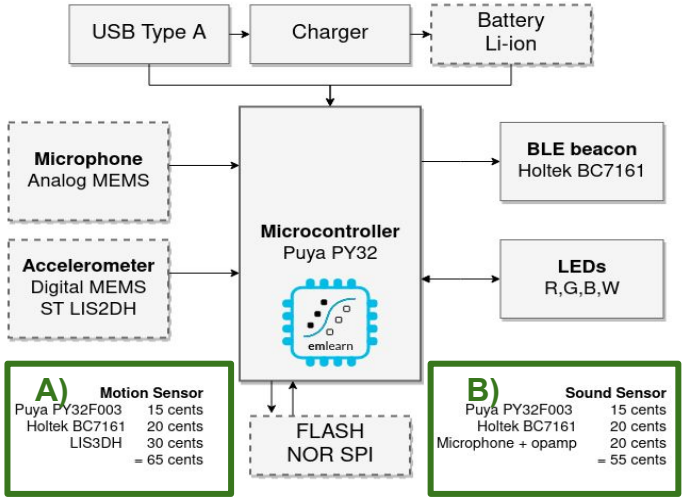
https://hackaday.io/project/194511-1-dollar-tinyml

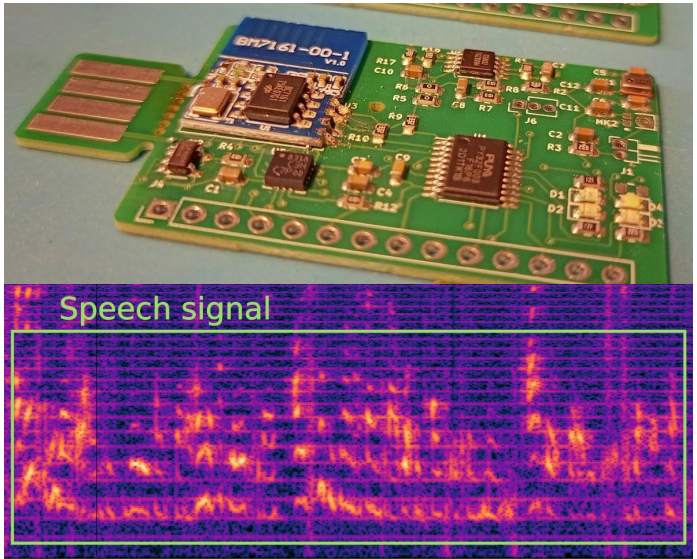
Research question:
Can we make a useful TinyML system
<1 USD in total component cost (BOM)?

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

Constraints: 8 kB RAM / 64 kB FLASH

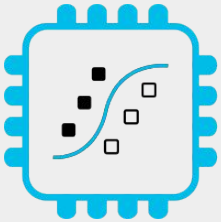
Current status:
Audio streaming works
Revision 2 ordered





Sensor nodes

with MicroPython
and emlearn



emlearn
-micropython

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

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)		✓	✓	✓

Audio input - **machine.I2S**

Digital microphone or **external audio ADC**

Can be done using I2S protocol

On ports **ESP32, STM32, RP2, NRF52**

PDM protocol not supported :(

Example code
https://github.com/emlearn/emlearn-micropython/tree/master/examples/soundlevel_iir
https://github.com/miketeachman/micropython-i2s-examples

```
# I2S audio input
from machine import I2S
audio_in = I2S(0, sck=Pin(26), ws=Pin(32), sd=Pin(33),
    mode=I2S.RX, bits=16,format=I2S.MONO, rate=16000,
)

# allocate sample arrays
chunk_samples = int(AUDIO_SAMPLERATE * 0.125)
mic_samples = array.array('h', (0 for _ in range(chunk_samples))) # int16
# memoryview used to reduce heap allocation in while loop
mic_samples_mv = memoryview(mic_samples)
# global to share state between callback and main
soundlevel_db = 0.0

meter = SoundlevelMeter(buffer_size=chunk_samples, samplerate=16000)

def audio_ready_callback(arg):
    # compute soundlevel
    global soundlevel_db
    soundlevel_db = meter.process(mic_samples)
    # re-trigger audio callback
    _ = audio_in.readinto(mic_samples_mv)

def main():
    # Use Non-Blocking I/O with callback
    audio_in.irq(audio_ready_callback)
    # Trigger first audio readout
    audio_in.readinto(mic_samples_mv)

    while True:
        render_display(db=soundlevel_db)
        time.sleep_ms(200)

if __name__ == '__main__':
    main()
```

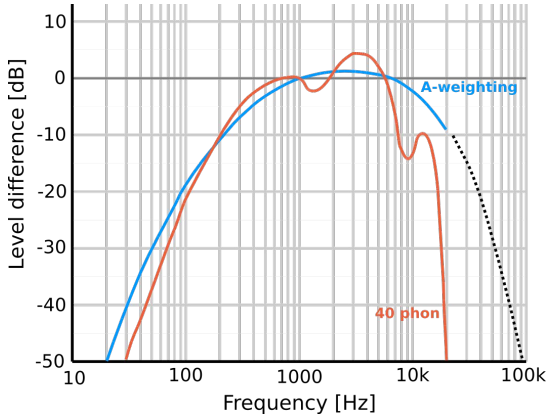
Noise sensor using **emlearn_iir**

Using **machine.I2S**
16 kHz samplerate

A weighting implemented with
Infinite Impulse Response (IIR) filter

emlearn_iir
25% CPU usage total

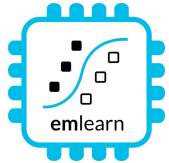
pure MicroPython
900% CPU - not feasible




```
# 6th order filter. 3x Second Order Sections "biquad"
a_filter_16k = [
    1.0383002230320646, 0.0, 0.0, 1.0, -0.016647242439959593, 6.928267
    1.0, -2.0, 1.0, 1.0, -1.7070508390293027, 0.7174637059318595,
    1.0, -2.0, 1.0, 1.0, -1.9838868447331497, 0.9839517531763131
]
self.frequency_filter = IIRFilter(a_filter_16k)
...
self.frequency_filter.process(samples)
```

Complete example code
https://github.com/emlearn/emlearn-micropython/tree/master/examples/soundlevel_iir

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

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
```

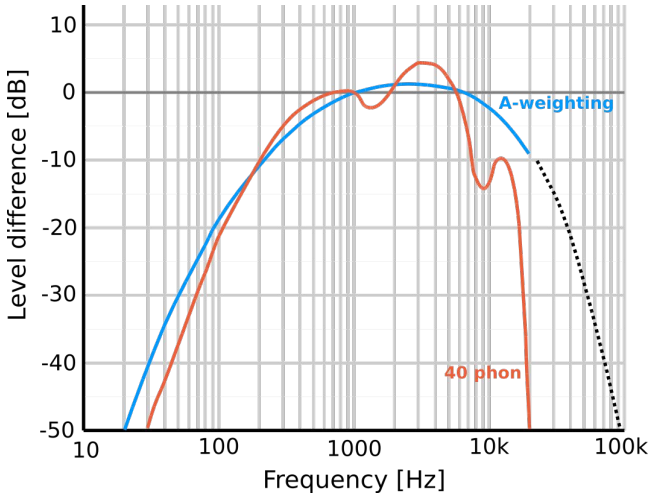
har_train.py

```
import emlearn
converted = emlearn.convert(clf)
converted.save(name='gesture', format='csv', file='model.csv')
```

Sound sensor - IIR filter

Standard sound level measurements are **A-weighted**. To approximate human hearing.

Typically, implemented using **Infinite Impulse Response (IIR) filters**.




```
class IIRFilter():
    def __init__(self, coefficients : array.array):
        stages = len(coefficients)//6
        self.sos = coefficients
        self.state = array.array('f', [0.0]*(2*stages))

    @micropython.native
    def process(self, samples : array.array):
        stages = len(self.sos)//6
        for i in range(len(samples)):
            x = samples[i]
            for s in range(stages):
                b0, b1, b2, a0, a1, a2 = self.sos[s*6:(s*6)+6]
                # compute difference equations of transposed direct form II
                y = b0*x + self.state[(s*2)+0]
                self.state[(s*2)+0] = b1*x - a1*y + self.state[(s*2)+1]
                self.state[(s*2)+1] = b2*x - a2*y
                x = y
            samples[i] = x
```

1100 ms 900% CPU
Native emitter
Float

Too slow by ~10x

```
# 6th order filter. 3x Second Order Sections "biquad"
a_filter_16k = [
    1.0383002230320646, 0.0, 0.0, 1.0, -0.016647242439959593, 6.928267
    1.0, -2.0, 1.0, 1.0, -1.7070508390293027, 0.7174637059318595,
    1.0, -2.0, 1.0, 1.0, -1.9838868447331497, 0.9839517531763131
]
self.frequency_filter = IIRFilter(a_filter_16k)
...
self.frequency_filter.process(samples)
```

Sound sensor - IIR filter

Using **emliir.mpy** native module helps a lot.

BUT - conversion from float/int16 too slow
Also needs a native module

```
import emliir

class IIRFilterEmlearn:

    def __init__(self, coefficients):
        c = array.array('f', coefficients)
        self.iir = emliir.new(c)
    def process(self, samples):
        self.iir.run(samples)
```

IIR filter only
Using emliir.mpy
native module
30 ms 20% CPU - OK

```
@micropython.native
def float_to_int16(inp, out):
    for i in range(len(inp)):
        out[i] = int(inp[i]*32768)

@micropython.native
def int16_to_float(inp, out):
    for i in range(len(inp)):
        out[i] = inp[i]/32768.0

int16_to_float(samples, self.float_array)
self.frequency_filter.process(self.float_array)
float_to_int16(self.float_array, samples)
```

But need to convert data types
Adds 70ms+ with micropython.native
Too slow! Total > 100 ms
Must create native module

emliir.mpy from emlearm-micropython
https://github.com/emlearn/emlearn-micropython/

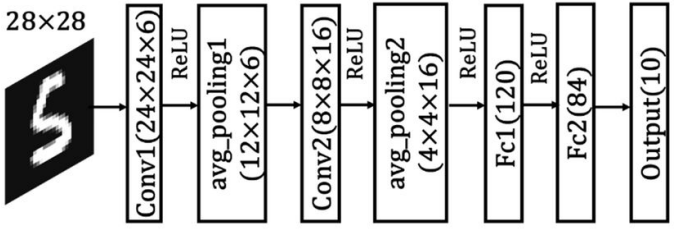
Camera input

Not part of standard APIs yet

micropython-camera-API
Proposed API and implementation
(ESP32 only, for now)
https://github.com/cnadler86/micropython-camera-API

OpenMV
https://openmv.io/
Custom MicroPython distribution
Focused on Computer Vision / Machine Vision

Image classification with **emlearn_cnn**



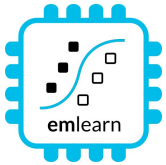
Convolutional Neural Network (CNN)

Running on ESP32-S3 (example)

Input dimensions:	32x32 px - 96x96 px
Layers:	3 - 4 layers.
Framerate:	1 - 10 FPS

Complete example code
https://github.com/emlearn/emlearn-micropython/tree/master/examples/mnist_cnn

```
import tinymaix_cnn # from emlearn-micropython

with open('cat_classifier.tmdl', 'rb') as f:
    model_data = array.array('B', f.read())
    model = tinymaix_cnn.new(model_data)

while True:

    raw = read_camera()
    img = preprocess(raw)
    classification = model.predict(img)

    if classification == MY_CAT:
       open_door()

    machine.lightsleep(500)
```

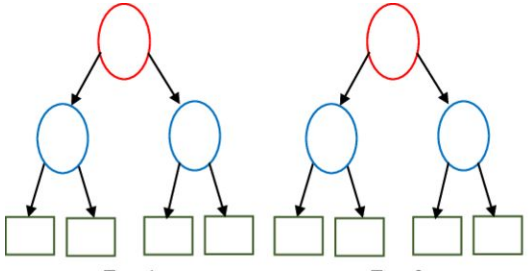


Tree-based ensembles

Random Forest / Decision Tree

The most popular model in emlearn

emlearn trees: Leaf compression

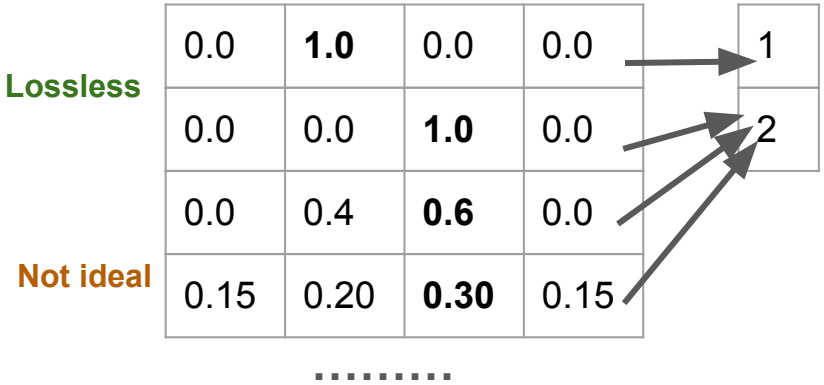


scikit-learn decision trees store class probabilities in leaf-nodes. Size: n_leaves * n_classes * 4 bytes
Consequence: Leaves take a large amount of space

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”)

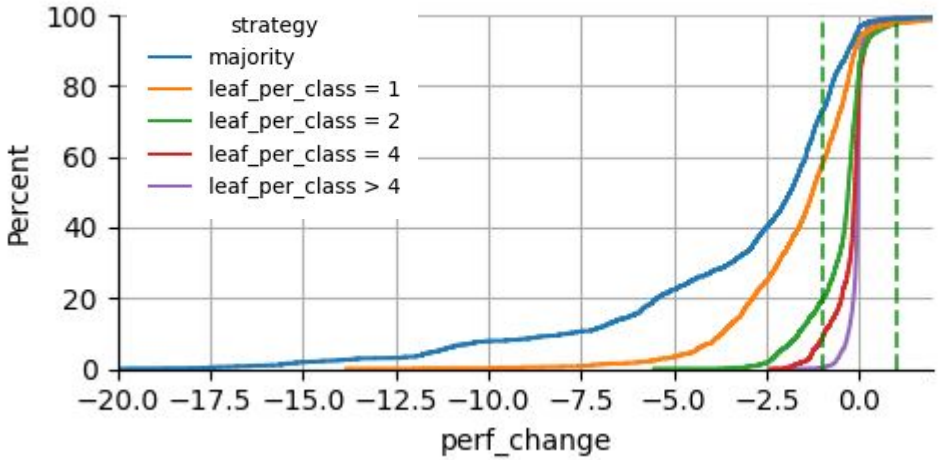
+ Supports int8/int16/int32 and float

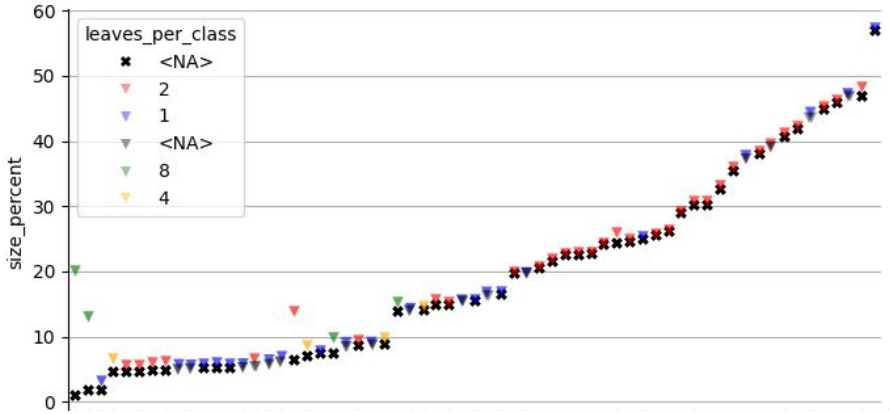
```
static inline int32_t
xor_model_tree_0(const float *features, int32_t
features_length)
{
    if (features[1] < 0.197349f) {
        if (features[0] < 0.466316f) {
            return 0;
        } else {
            return 1;
        }
    } else {
        if (features[1] < 0.256702f) {
          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

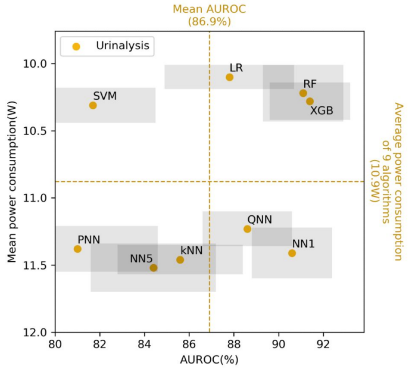
2-8 leaves per class sufficient
0% of datasets failed performance constraint
High model size compression 50-90%

Trees - relevant across the task complexity range

Low

Go-to for tabular data

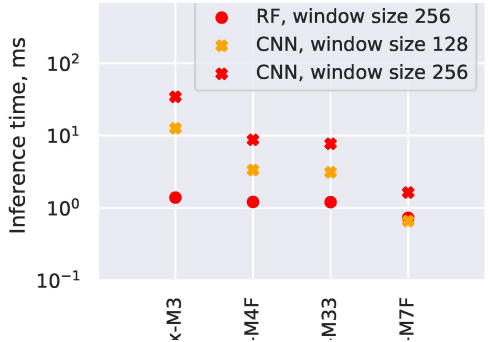
Very easy to get good performance, with minimal tweaking.



Medium

Alternative to deep learning

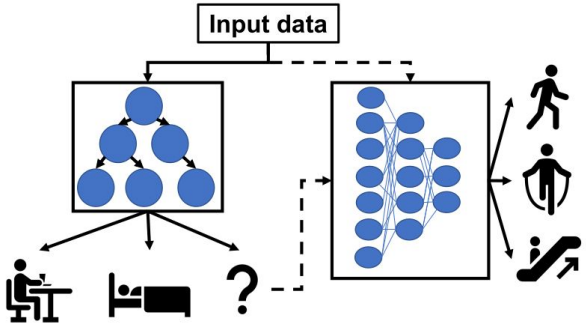
Elsts et.al [1] showed performance matching a deep-learning approach, at 10x to 100x better resource usage



High

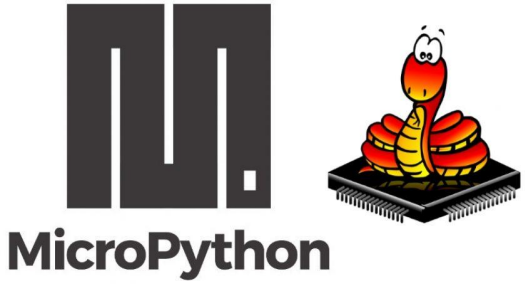
Combine with neural network

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/

More MicroPython



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)

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 extensisive/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

Garbage collected
One GC cycle *will take 1-10 ms* (typ)
Some control, limited (gc module)

TLDR:

- Will be very familiar to Python devs
- Small scripts will mostly work with minor mods.
- Larger programs/modules may need refactoring or rewrite to fit target

Hardware access - the **machine** module

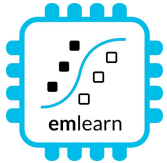
https://docs.micropython.org/en/latest/library/machine.html

- 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

mip - package manager



Install from micropython-lib

```
mpremote install requests
```

Third party packages

```
mpremote install github:jonnor/micropython-zipfile
```

Can run directly on device *

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

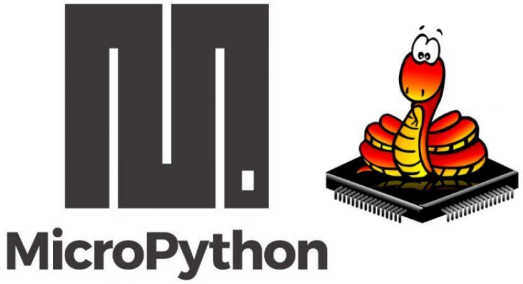
* Assuming device has Internet over WiFi/Ethernet

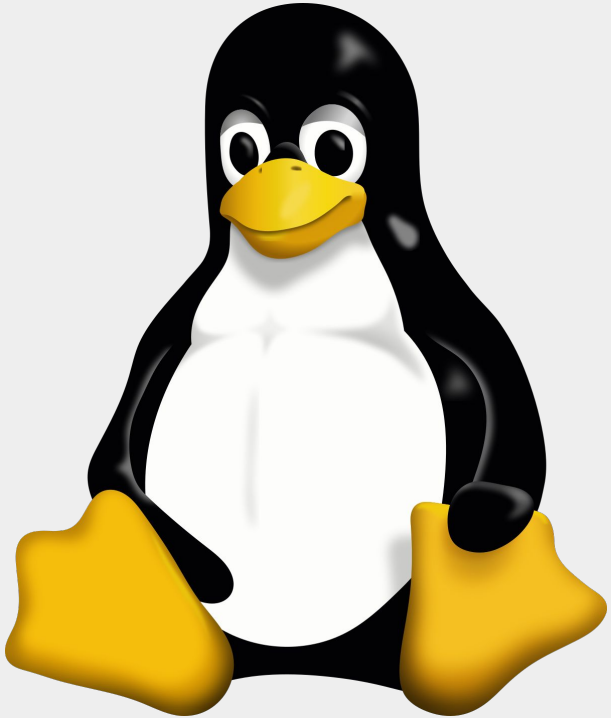
Install native C modules at runtime

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

* Specify architecture + MicroPython ABI version

MicroPython for Embedded Linux





Motivation: **Memory efficiency**

CPython is quite RAM hungry. Especially “standard” Python/PyData ecosystem

scikit-learn with **CPython:** **13 - 128 MB**

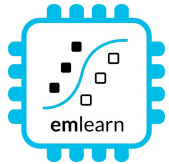
```
from sklearn.ensemble import RandomForestClassifier
estimator = RandomForestClassifier()
# model not even trained yet!!!
```

emlearn_trees with **MicroPython:** **0.1 - 6 MB**

100 trees, 4000 nodes per tree (max_depth=12)

```
import emlearn_trees
model = emlearn_trees.new(100, 4000, 100)
```

MicroPython attractive for Linux devices with < 512 MB RAM



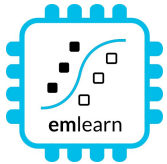
Unix MicroPyton port: Limited hardware access

Not implemented:

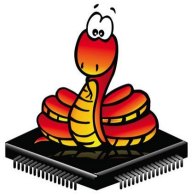
- GPIO. machine.Pin / machine.PWM / machine.ADC
- Digital busses. machine.I2C / SPI / USART / USB
- Watchdog Timer. machine.WDT
- Power management. machine.lightsleep() /deepsleep()
- Audio input/microphone
- Camera access

Need C or Python modules for this. Or call external programs.
Contributions welcome!

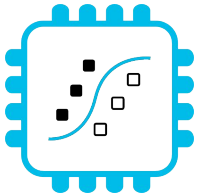
Already very useful for unit testing

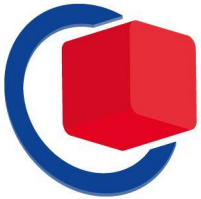


Bridging the TinyML language gap



with MicroPython and emlearn





embeddedworld
Exhibition&Conference

Jon Nordby
jononor@gmail.com
March 2025, Nuremberg

https://github.com/emlearn/emlearn