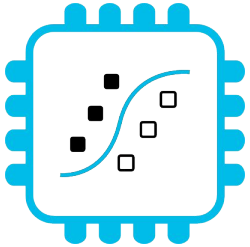
Sensor data processing on microcontrollers with MicroPython

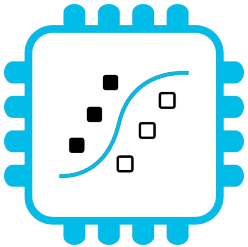


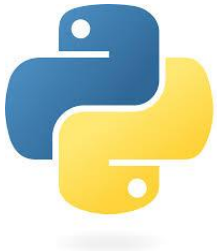
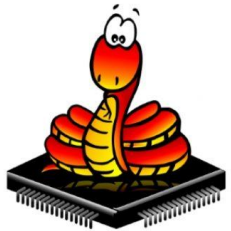
https://github.com/emlearn/emlearn-micropython

Jon Nordby <jononor@gmail.com>
EuroSciPy, Krakow, August 2025



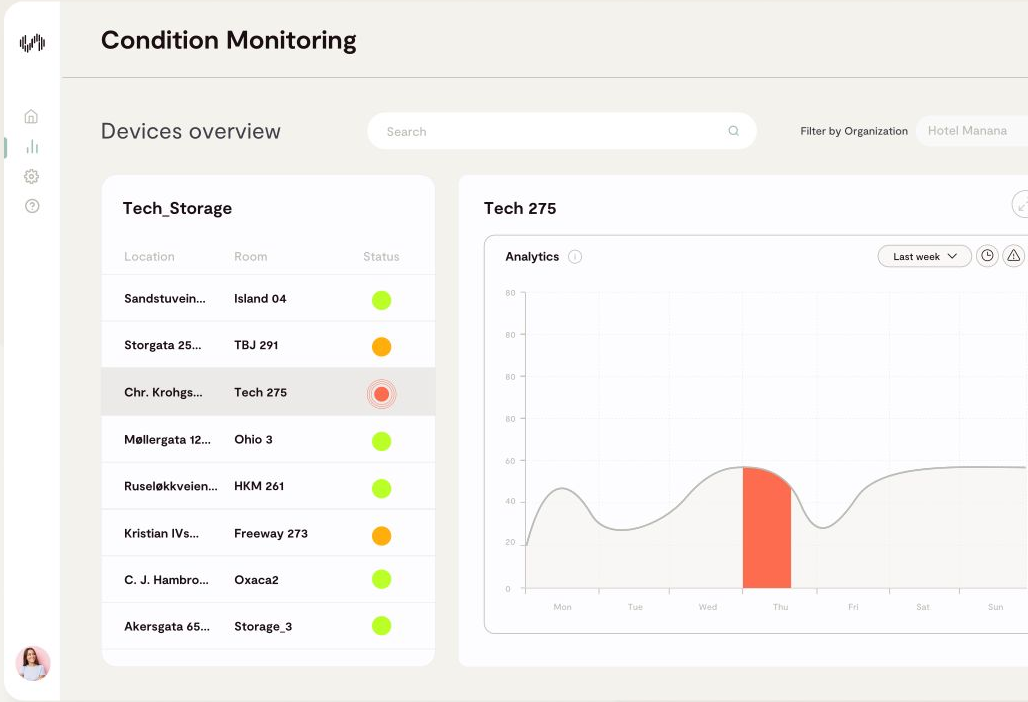
Sprint
MicroPython / emlearn
On Friday!



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

Goal of this talk

You as a **scientist, or engineer** supporting scientists,

become ***familiar*** with how to
create **custom sensor systems**

using **microcontrollers** and **(Micro)Python**

including on-sensor data processing

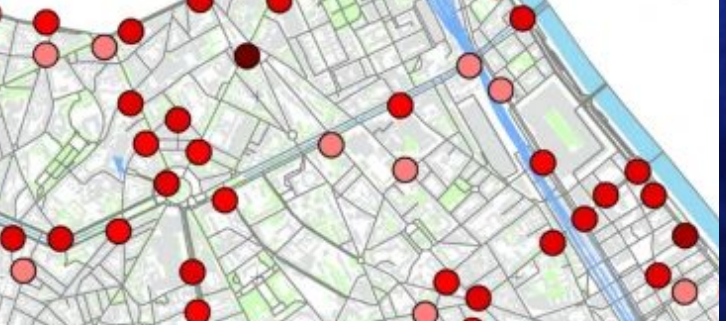
like **digital signal processing** and **machine learning**

Outline

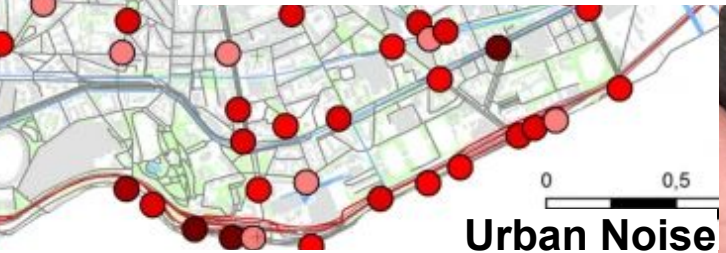
1. Wireless Sensor Networks
2. MicroPython
3. emlearn

Wireless Sensor Networks

for scientific applications

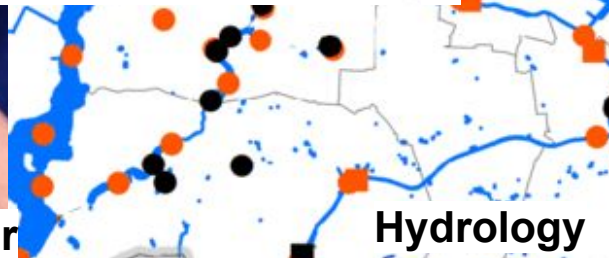


To understand the world, we must measure it



Urban Noise

Animal behavior



Hydrology

Want: Measurements over time
Many measurement locations

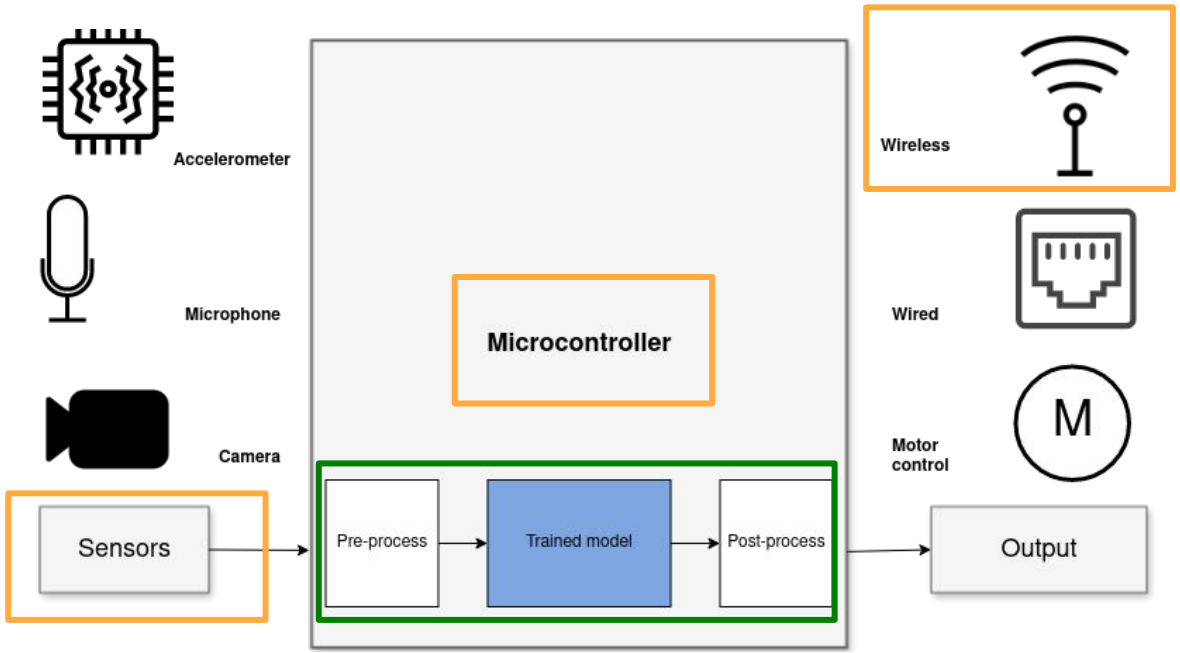
Need: Low-cost, unattended, data transfer

Sometimes: Small / portable

How:

- Wireless communication
- Battery powered
- Microcontroller
- Commodity sensors
- Smart on-sensor processing

On-sensor data analysis with Machine Learning



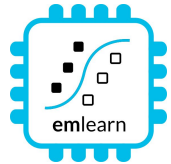
System diagram for “TinyML” solution

Focus of this talk

Benefits vs sending data to cloud

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

What is a microcontroller?

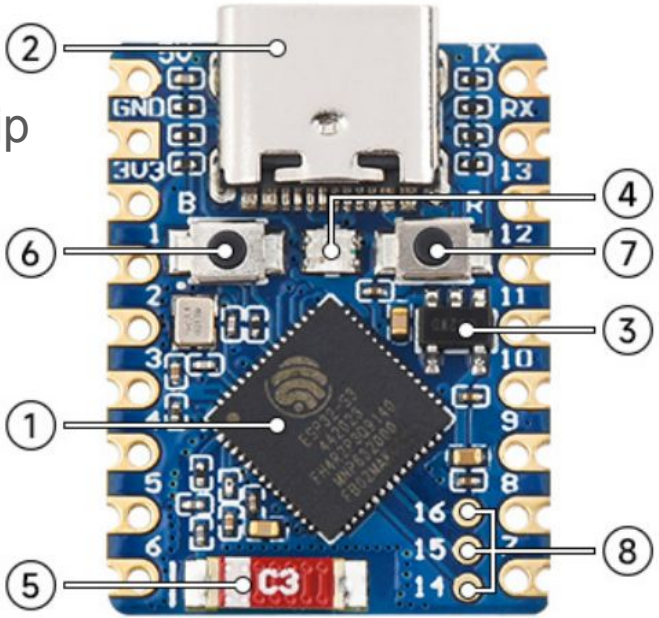


Modern microcontroller:
A complete programmable System-on-Chip

Example: ESP32-S3FH4R2

32 bit CPU, 240 Mhz
Floating Point Unit
2 MB RAM
4 MB FLASH

WiFi
Bluetooth Low Energy
USB-C



Espressif ESP32-S3FH4R2 chip:	2.5	USD
Waveshare ESP32-S3-Tiny board:	6	USD

Slow-changing phenomena

For slow-changing phenomena
(relative to measurement interval)

Sample, Send, Sleep

1. Read the sensor
2. Send data
3. Wait until next measurement

Processing
Sensor reading
Send

Continuous measurement

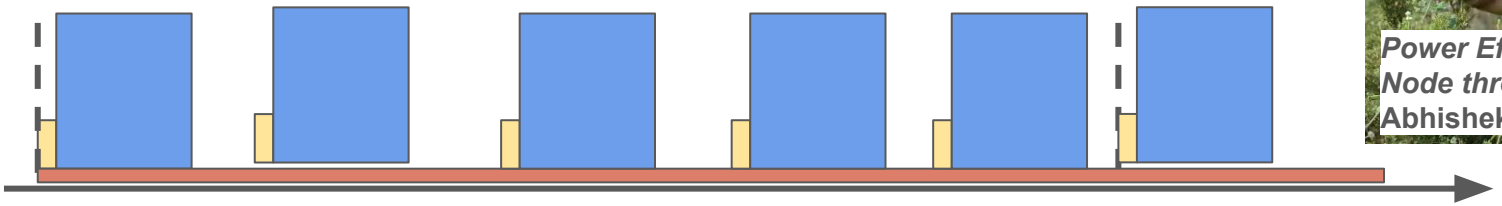
Processing

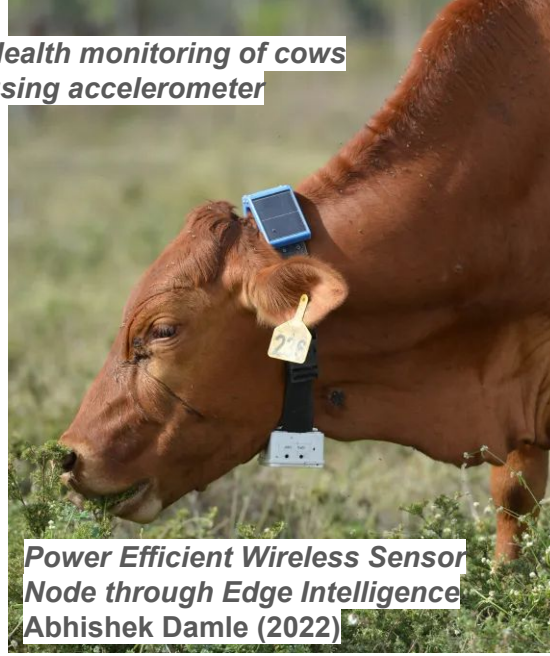
Sensor reading

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

Act locally

Task:
Give vaccinated bait *only* to Tasmanian devils

Trigger: PIR sensor

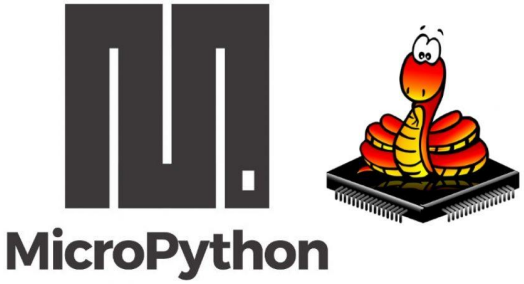
```
Wake up
Classify image using CNN
if is_Taz:
    dispense_bait()

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

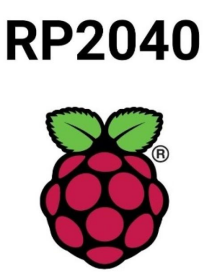
MicroPython tour



MicroPython - Introduction

Implements a subset of Python 3.x

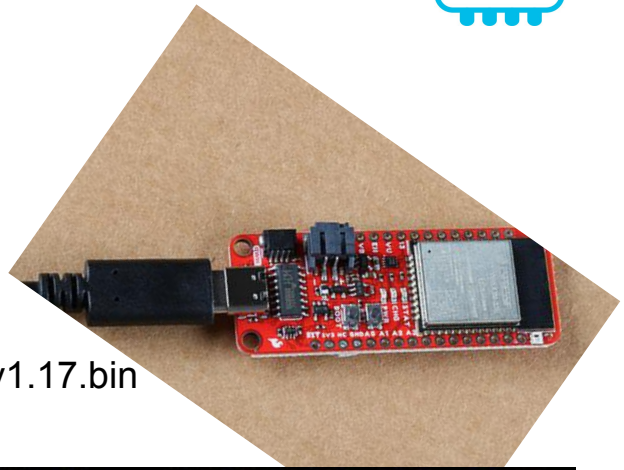
For devices with 128 kB+ RAM
Supports 8+ microcontroller families directly,
many more via Zephyr RTOS



Official webpage: https://micropython.org

MicroPython

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

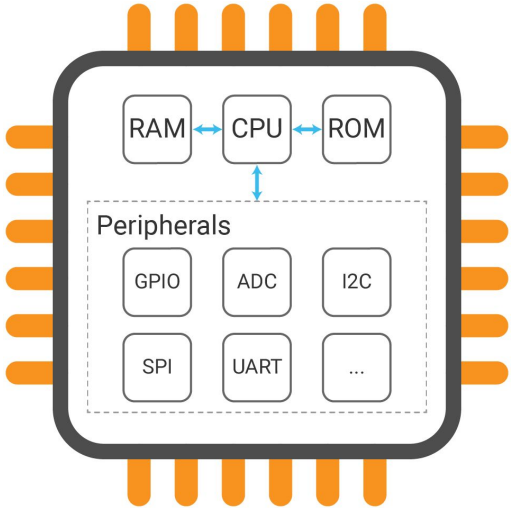
Hardware access - the **machine** module

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

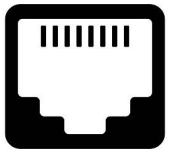


MicroPython

Connectivity

WiFi
network.WLAN

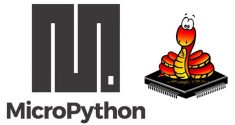
Ethernet
network.LAN



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

C modules *

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

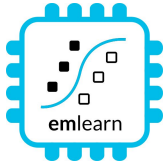
Implemented
by users,
libraries, or
part of MicroPython core.

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



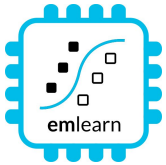
CPython

- 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 for MicroPython

Python is the lingua franca for Machine Learning

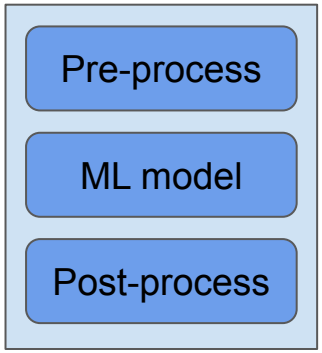
How can we enable people to build microcontroller-grade machine learning using Python?

For easier learning and prototyping

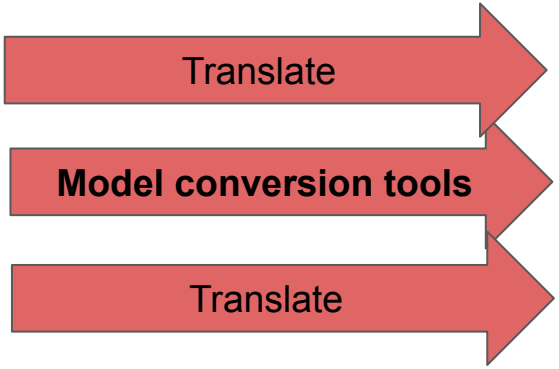
TinyML challenge: **PC/host** vs **embedded/device** pipeline

1) Train model
on PC

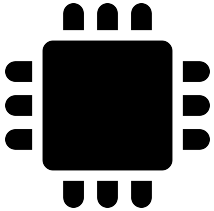
RAM (min)
10 GB+

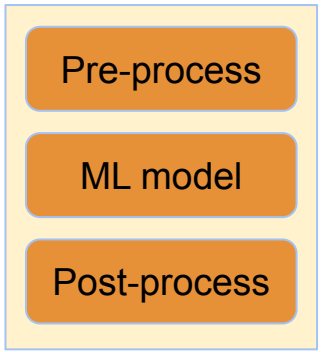


Deep-learning pipeline



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"

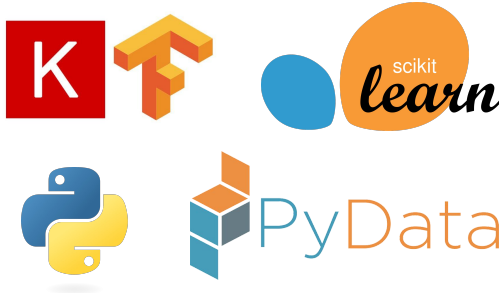
"Firmware Engineer land"

1. Train on PC

pip install emlearn

Convenient training

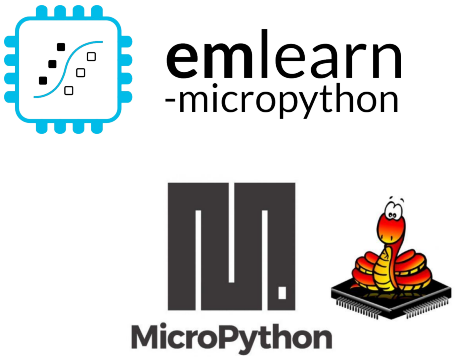
- Model creation **in Python**
- **Use standard libraries**
  a. scikit-learn
  b. Keras
- One-line to **export to device**
- Verification tools included



2. Deploy on device

Embedded Friendly

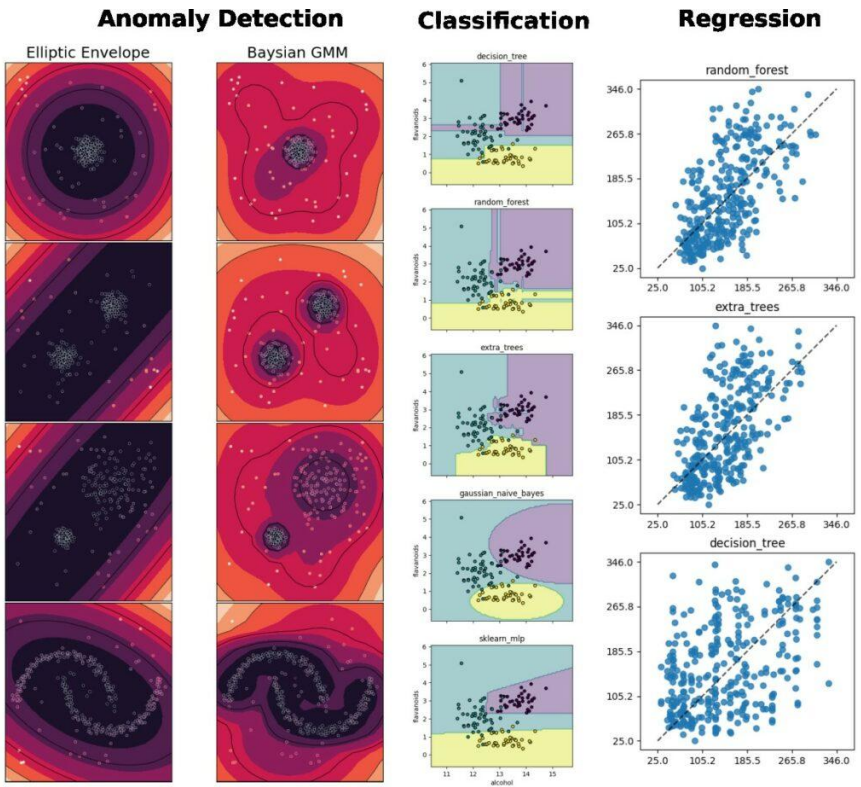
- **Portable C99** code
- No dynamic allocations
- **Header-only**
- High **test coverage**
- Integer/**fixed-point** math *
- **Small**. 2 kB+ FLASH



Convenient & Efficient

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

Supported tasks



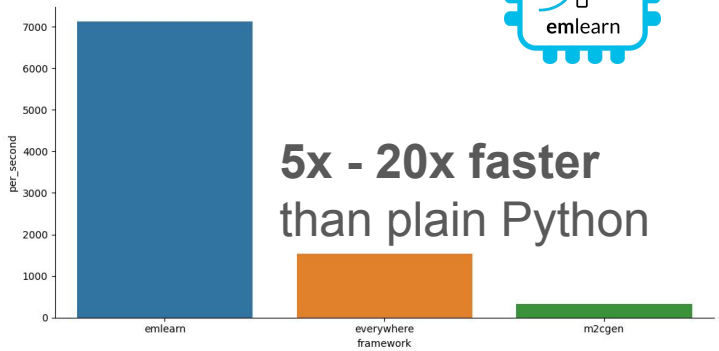
Supports the most common tasks for embedded
& sensor data use cases.

- Classification
- Regression
- Anomaly Detection

emlearn-micropython: MicroPython bindings for emlearn

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

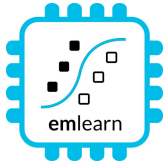
Modules installable at runtime (.mpy)
1 - 10 kB in size



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

PyData friendly

emlearn-micropython: Install & Export model



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

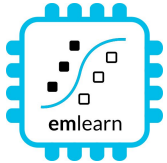
mpremote fs cp **gesture_model.csv** :

Install MicroPython library on device

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

Prebuilt binary
No need to setup
C toolchain and SDK

emlearn-micropython: 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

Activity tracker

Accelerometer

Random Forest classifier
emlearn_trees

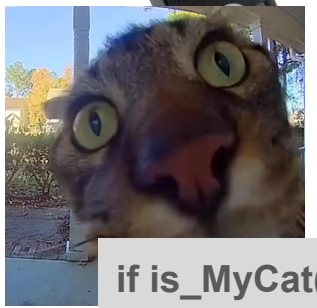
Noise monitor

Microphone

Infinite Impulse Response filters
emlearn_iir

Image Classifier

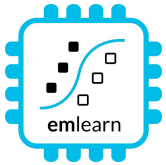
Camera



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

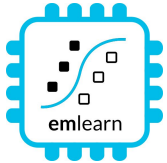
Examples at
https://github.com/emlearn/emlearn-micropython#examples

Convolutional Neural Network
emlearn_cnn



Outro

Summary



Many scientific applications rely on collecting a lot of physical data

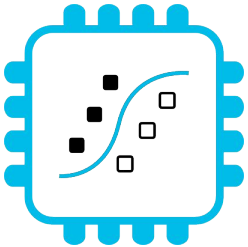
Large deployments possible with **low-cost Wireless Sensor Networks**

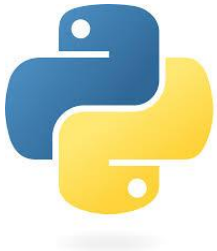
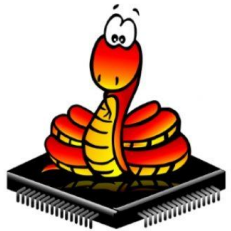
Sensor nodes can be **developed in Python**,
using **MicroPython on microcontroller**

MicroPython has **good tools for sensor data processing**,
including connectivity, storage, performance

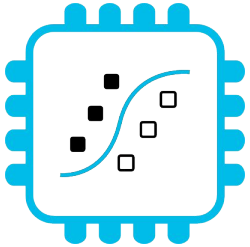
Good **ecosystem for data processing**,
including **machine learning with emlearn-micropython**

Sprint
MicroPython / emlearn
On Friday!



Sensor data processing on microcontrollers with MicroPython

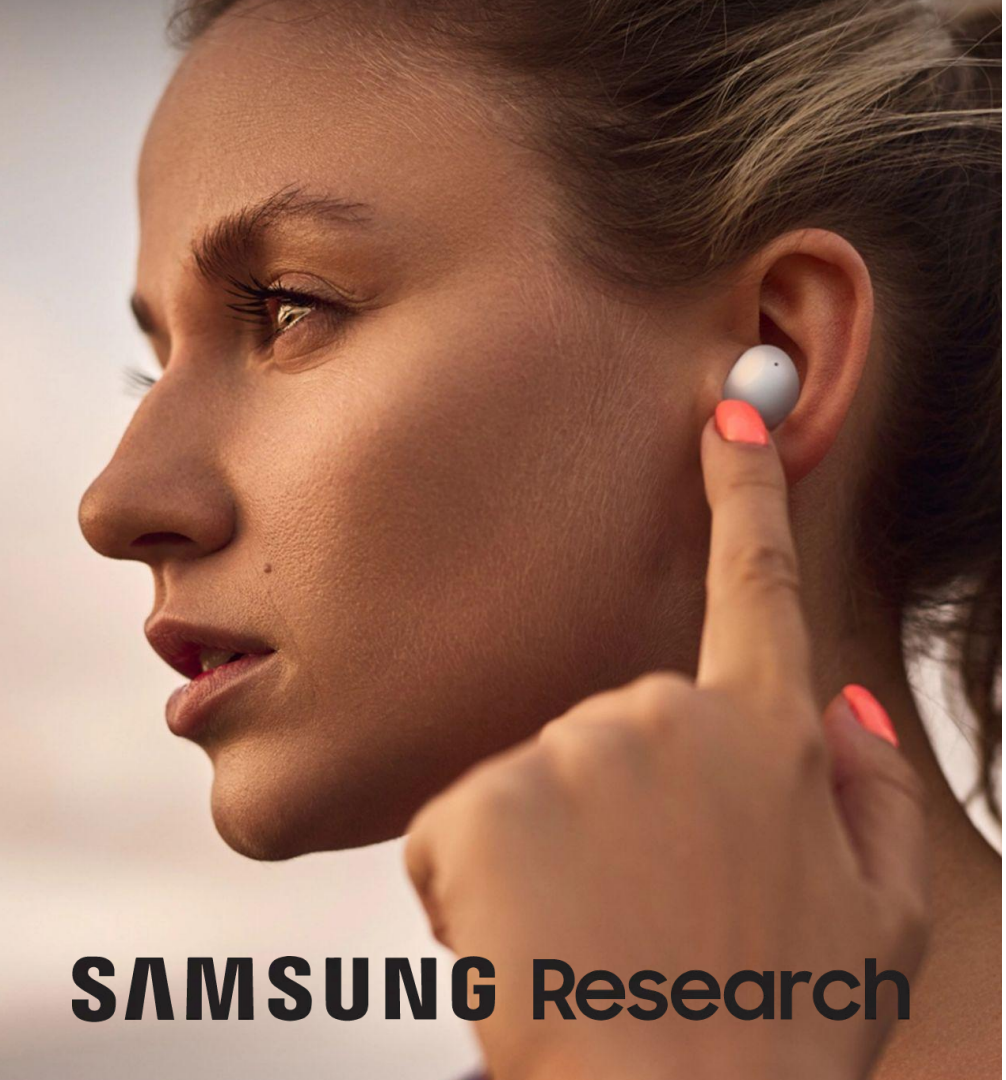


https://github.com/emlearn/emlearn-micropython

Jon Nordby <jononor@gmail.com>
EuroSciPy, Krakow, August 2025



Bonus

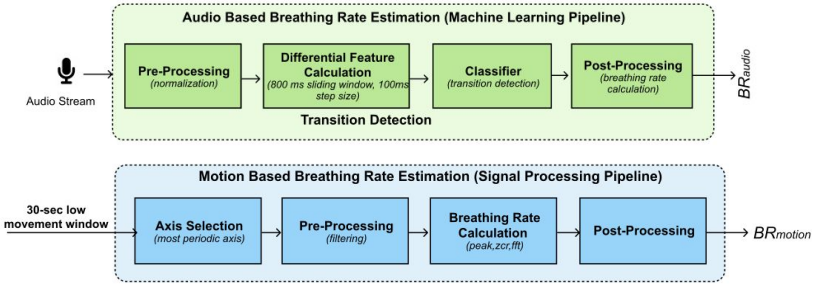


Health monitoring earable

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.

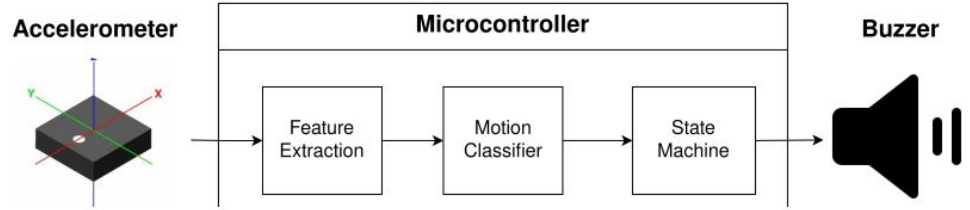
Random Forest classifier for audio.

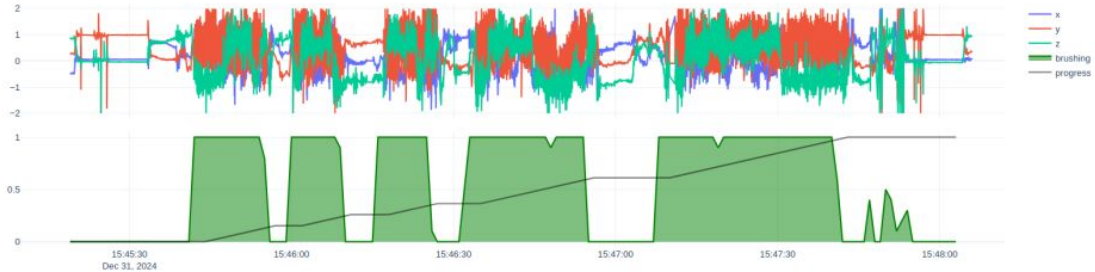


Remote Breathing Rate Tracking in Stationary Position Using the Motion and Acoustic Sensors of Earables
Tousif Ahmed et.al (2023)

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

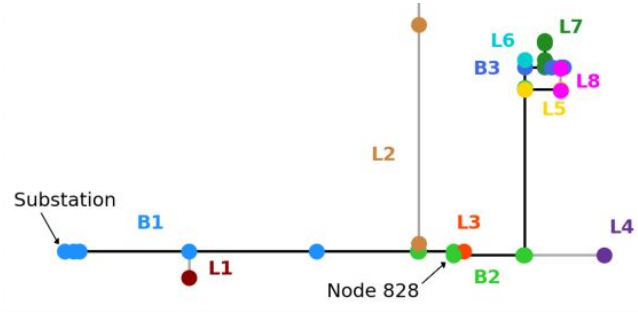
Data collection and training tools:
https://github.com/emlearn/emlearn-micropython/tree/master/examples/har_trees

Electric grid monitoring

Detect and localize faults in the electric grid.

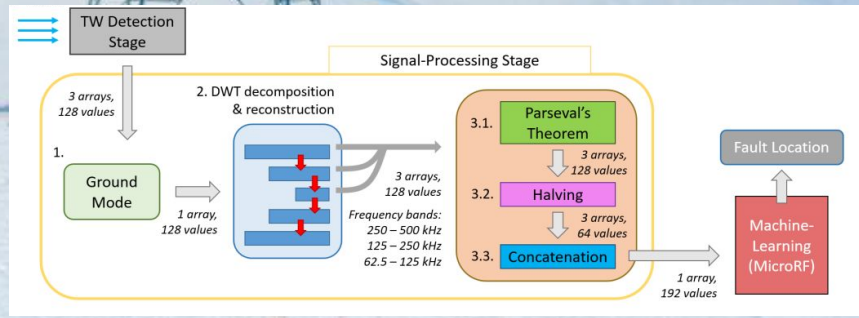
Analyzes the voltage signals at a single location. Able to detect where the fault originated.

Runs on a DSP board from Texas Instruments.

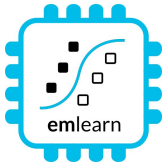


Micro Random Forest: A Local, High-Speed Implementation of a Machine-Learning Fault Location Method for Distribution Power Systems
Tousif Ahmed et.al (2023)

Sandia National Laboratories



Misc



Model optimization

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

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

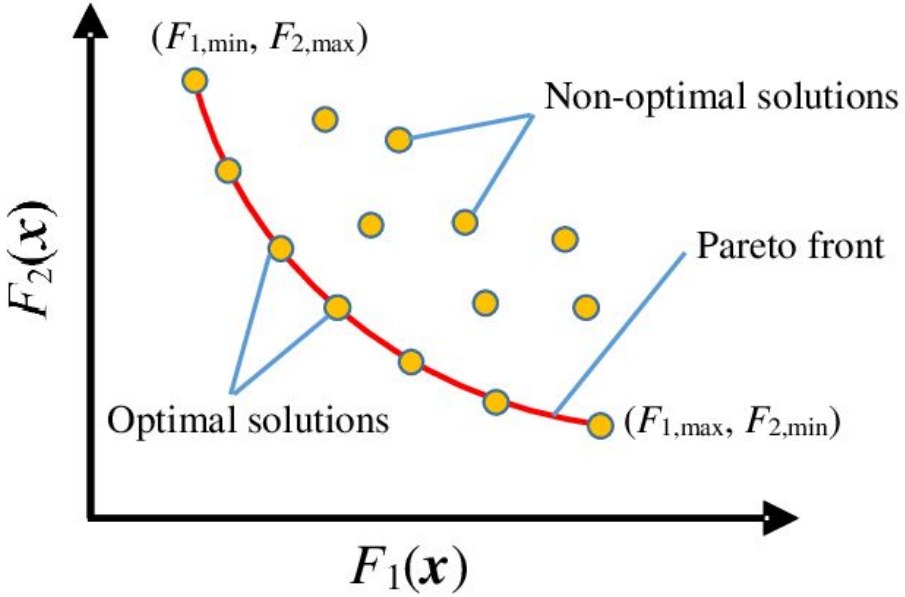
Pareto front - performance/compute tradeoffs

Many possible combinations of **predictive performance** vs **computational cost**

No point considering the non-optimal solutions!

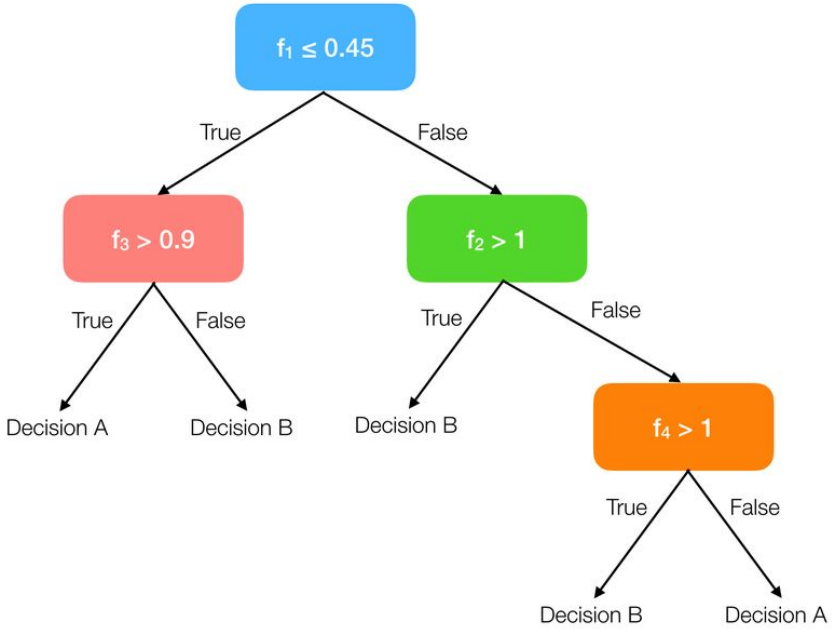
Pareto front is formed by the set of optimal solutions - that dominate the non-optimal ones

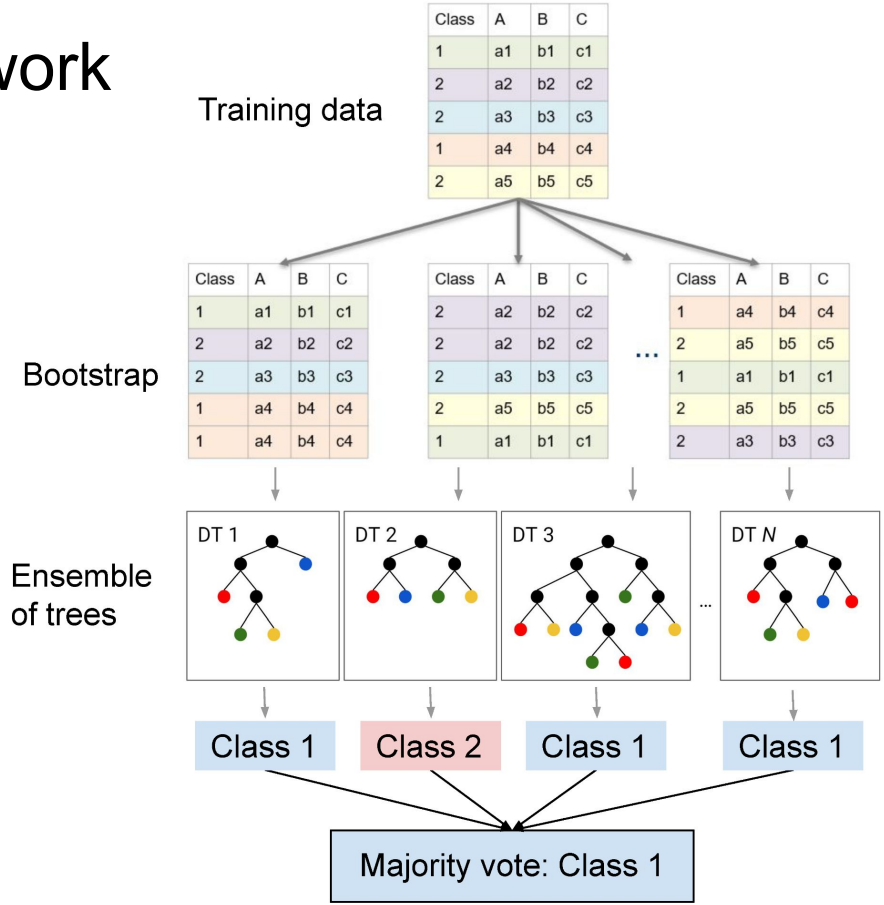
Describes the **possible tradeoffs between predictive performance and compute**



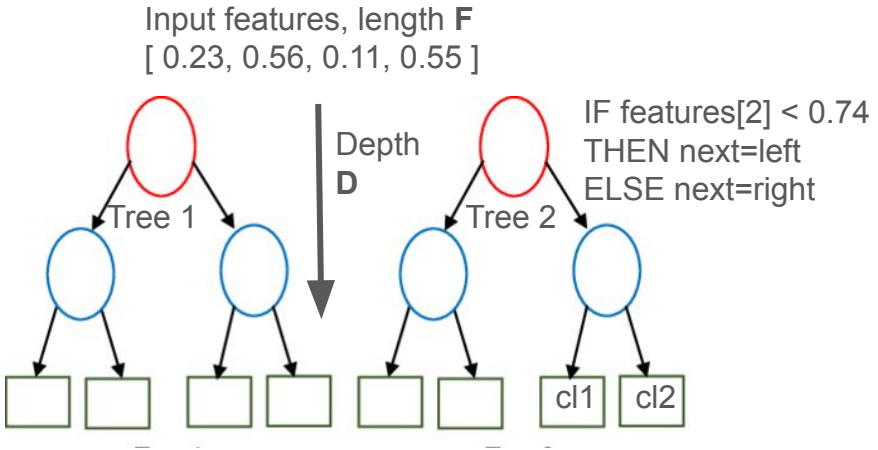
Utilities for finding Pareto front in **emlearn.evaluate.pareto**
https://emlearn.readthedocs.io/en/latest/evaluate.html
Example: https://emlearn.readthedocs.io/en/latest/auto_examples/trees_hyperparameters.html

How tree-ensemble models work





Tree-based ensembles - costs



No. features	**F** [0]
No. trees	**T**
Tree Depth	**D_max** [1] / **D_eff** [2]
No. nodes	**N ≈ T** * (2 ** **D**)

Computational costs

Memory (RAM)	O(**F**)
Program (FLASH)	O(**N**) [3]
Exc. time (CPU)	sum(**D_eff**(t), t -> **T**)

Utilities for estimating costs

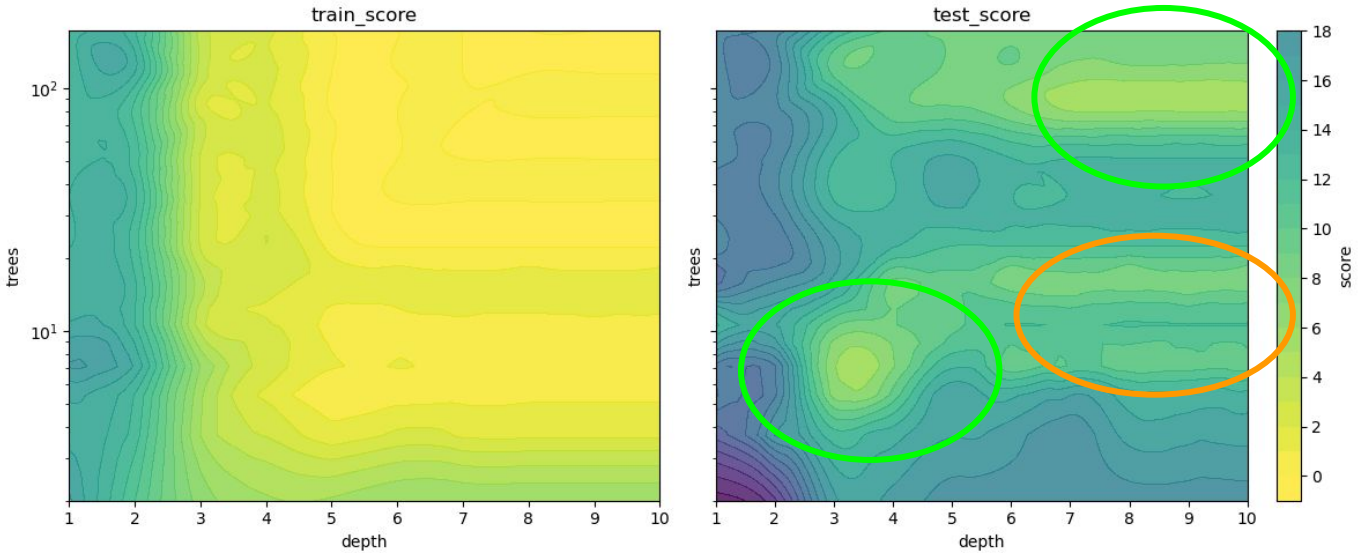
from **emlearn.evaluate.trees** import
model_size_bytes,
compute_cost_estimate

[0] May also enable deeper trees
[1] Depth might differ across trees
[2] Execution path is data dependent! Can estimate average using a dataset, or worst-case from model
[3] Assuming leaf and decision nodes same size

Tree-based ensemble - predictive performance

No. features	F	increases capacity
No. trees	T	increases capacity, **decreases overfitting**
Tree Depth	D_max / D_eff	increases capacity, **increases overfitting**
No. nodes	N ≈ T * D	

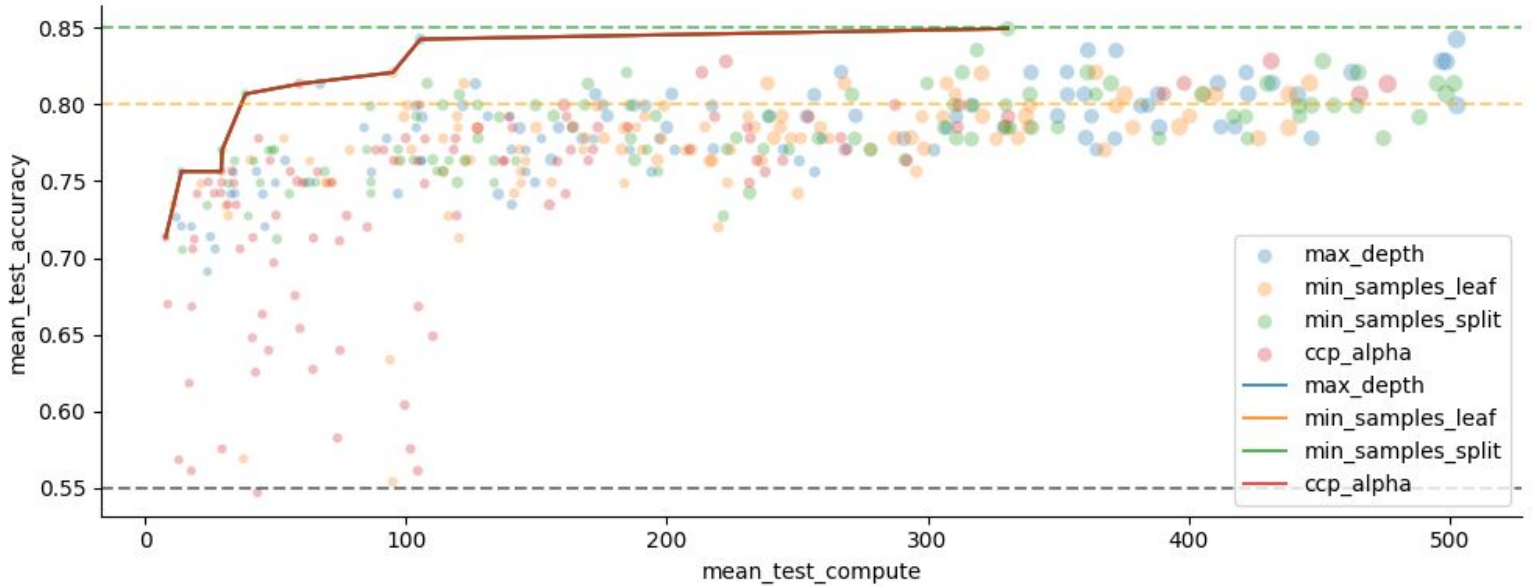
For **generalized performance** (on unseen data), **need to balance overfitting.**



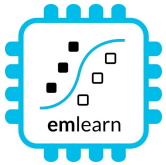
Hyperparameter tuning with scikit-learn

Vary **n_estimators** (trees)
and *one* of the **depth limiting** hyperparameters

from sklearn.model_selection import **GridSearchCV**, **RandomizedSearchCV**



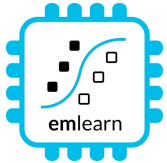
https://emlearn.readthedocs.io/en/latest/auto_examples/trees_hyperparameters.html



Activity Recognition

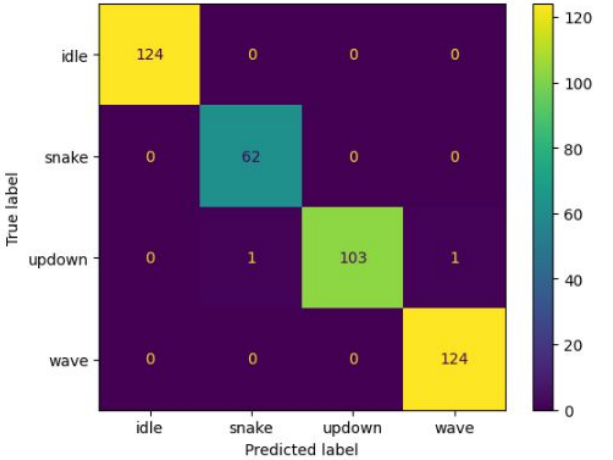
Using accelerometer / IMU

Training a RandomForest model

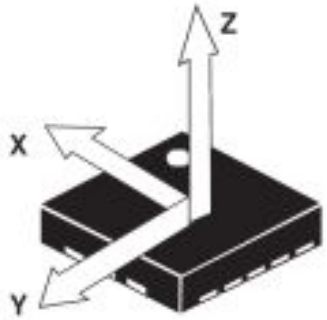


Standard process - familiar to anyone who has used scikit-learn.
But - Make sure model size is not huge.

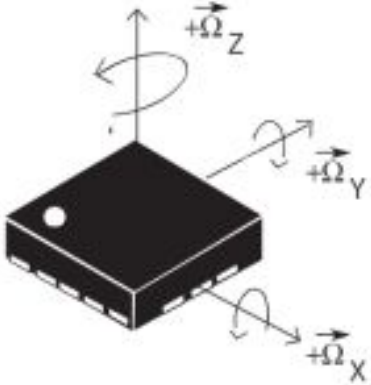
```
# preprocess data
windows = train.groupby(....).apply(create_windows,
        length=int(2000/15), hop=int(260/16))
features = windows.groupby(...).apply(spectral_features)

# train model
from sklearn.ensemble import RandomForestClassifier
est = RandomForestClassifier(n_estimators=10, max_depth=5)
est.fit(features, features.reset_index()['class'])

# evalutate on test set
…
```



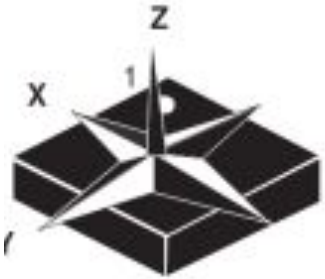
Sensors for motion



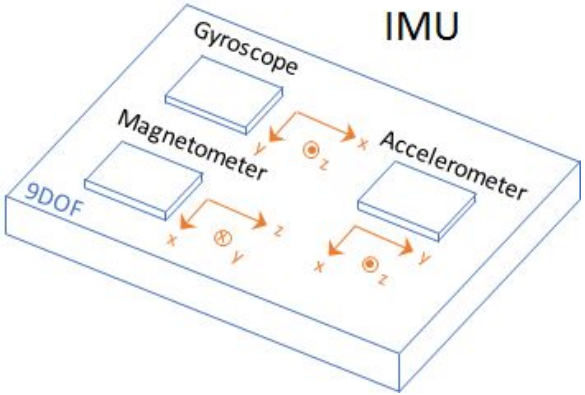
Acceleromter



Gyroscope



Magnetometer



Sensor Type	Components + Measures	What It Can Do	Limitations
3-Axis Accelerometer	X, Y, Z acceleration (g-force)	Detect tilt, orientation	Cannot measure rotation or heading
6-Axis Accel + Gyro	Acceleration + Angular velocity	Detect tilt, rotation	Drifts over time, no heading
9-Axis Accel + Gyro + Compass	Acceleration + Rotation + Magnetic heading	Full 3D orientation	Sensitive to magnetic interference

Activity Recognition usecases

Human Activity Recognition. Health, medical, sports
https://github.com/jonnor/embeddedml/blob/master/applications/human-activity-recognition.md

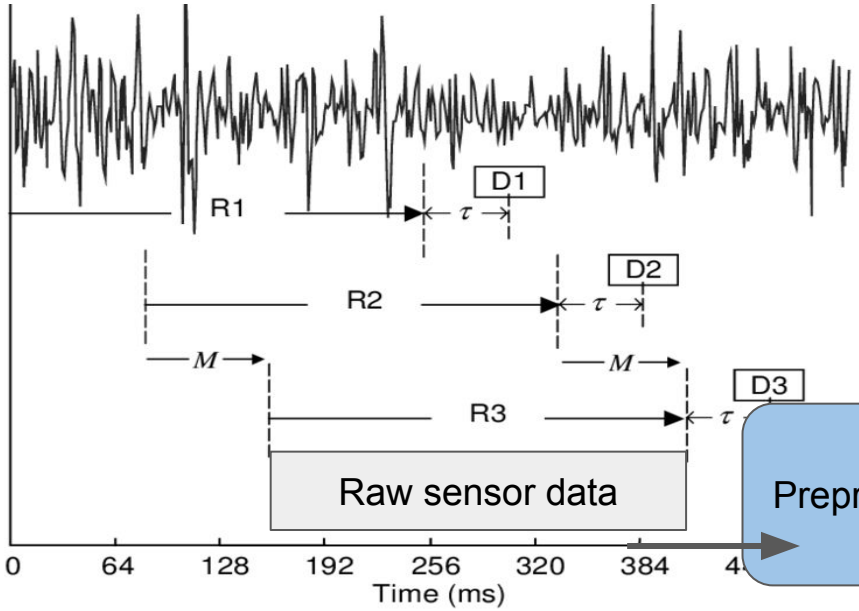
Animal Activity Recognition. Pets, livestock, wild animals
https://github.com/jonnor/embeddedml/blob/master/applications/animal-activity-recognition.md

Other TinyML tasks related to motion / IMUs

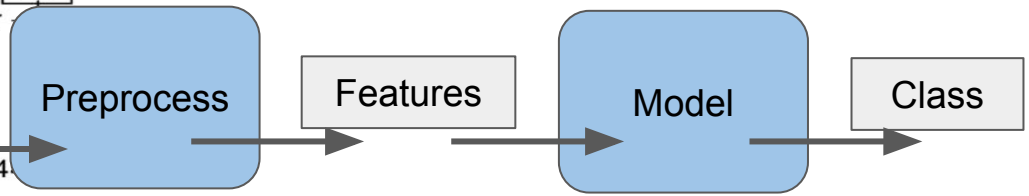
Gesture Recognition
https://github.com/jonnor/embeddedml/blob/master/applications/gesture-recognition.md

Condition Monitoring of Machinery
https://github.com/jonnor/condition-monitoring

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”

har_trees - Machine Learning pipeline

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)
```

```
toothbrush_jonnor': dict(
    groups=['session'],
    label_column = 'is_brushing',
    time_column = 'time',
    data_columns = ['x', 'y', 'z'],
    classes = [
        'True', 'False',
    ],
```

```
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)
```

```
(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
```

```
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
```

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

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

Native module (.mpy) VS External C module

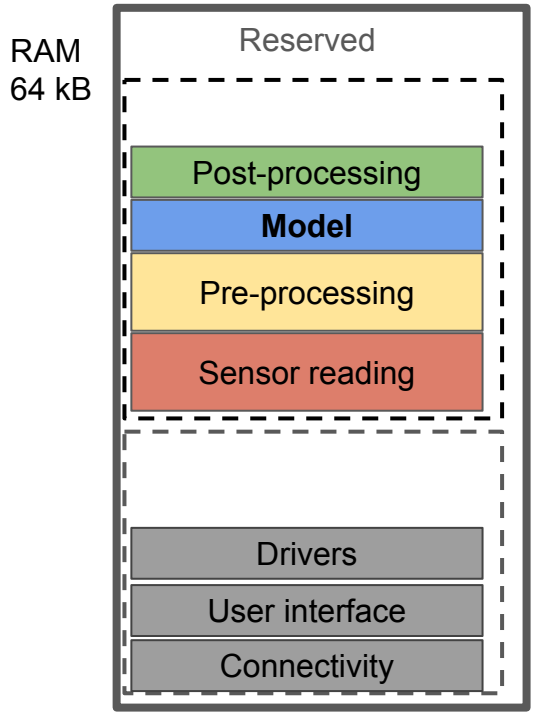
	Native module	**External C module**
Installable at runtime	Yes, as .mpy file	No. Must be included in firmware image
Requires SDK/toolchain	No (only to build)	Yes
Code executes from	RAM	FLASH
Limitations	Limited hardware access	None
Maturity	Medium *	Excellent
Documentation	https://docs.micropython.org/en/latest/develop/natmod.html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

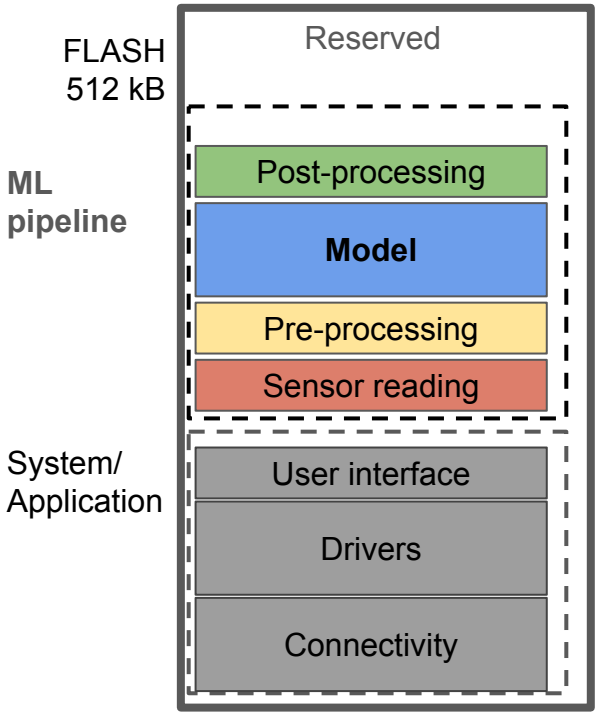
* Improved greatly in MicroPython (1.25+).
Thanks to Volodymyr Shymanskyy, Alessandro Gatti, Damien George, and others

MicroPython

Compute constraints

- Memory (RAM)
- Program space (FLASH)
- Inference time (CPU)
- Energy (battery)





CPU
Cortex M4F

Temperature sensor - code

1. Read the sensor in a loop
2. Send data using MQTT
3. Wait until next measurement

The same approach usable for other slow-changing phenomena

Using https://viper-ide.org/ with Chromium

Zero-install. Connect to device via USB

```
from mqtt_as import MQTTClient, config
import asyncio
from mpu6886 import MPU6886
from machine import I2C

# Local configuration
config['ssid'] = 'FIXME'  # Optional on ESP8266
config['wifi_pw'] = 'FIXME'
config['server'] = 'test.mosquitto.org'

mpu = MPU6886(I2C(0, sda=21, scl=22, freq=100000))

async def main(client):
    print('main-start')
    await client.connect()
    print('connected')

    while True:
        t = mpu.temperature
        print('publish-data', t)
        await client.publish('pydataglobal2024/send', f'{t:.2f}'
        await asyncio.sleep(30)

MQTTClient.DEBUG = True  # Optional: print diagnostic messages
client = MQTTClient(config)
try:
    asyncio.run(main(client))
finally:
    client.close()
```