Microcontrollers and Machine Learning

with MicroPython and emlearn

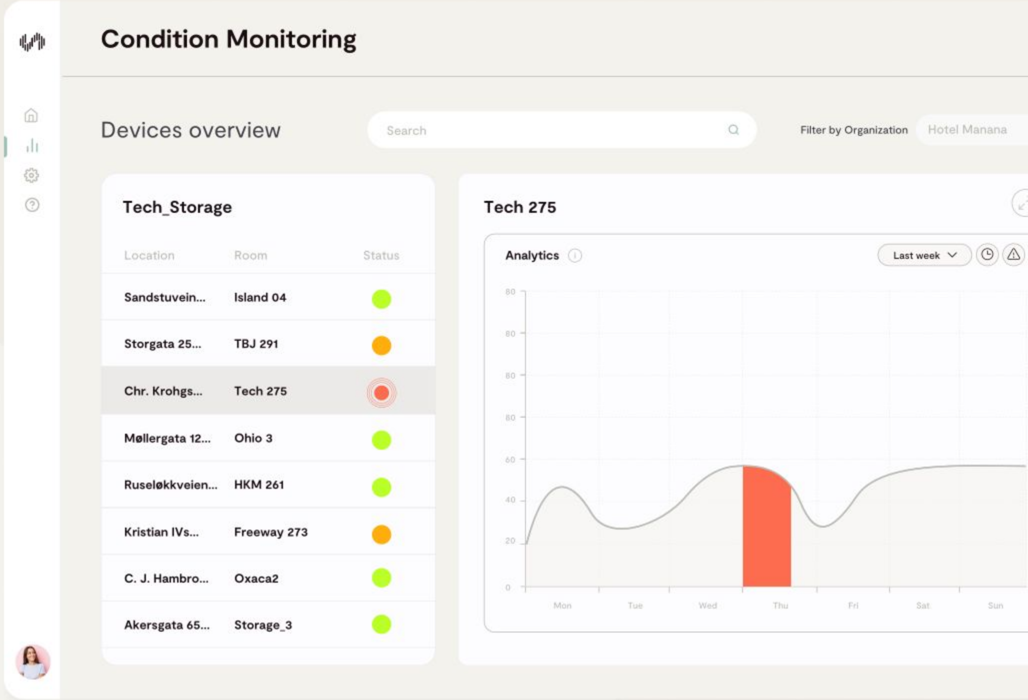
https://github.com/emlearn/emlearn-micropython

Jon Nordby jononor@gmail.com

foss-north 2025

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

Purpose of this presentation

You, as a **Python developer**,
can **build an IoT sensor**
on a **microcontroller**
using **MicroPython**

+ Including
on-edge Machine Learning
with **emlearn-micropython**

Python on Microcontroller

Jumping right into it

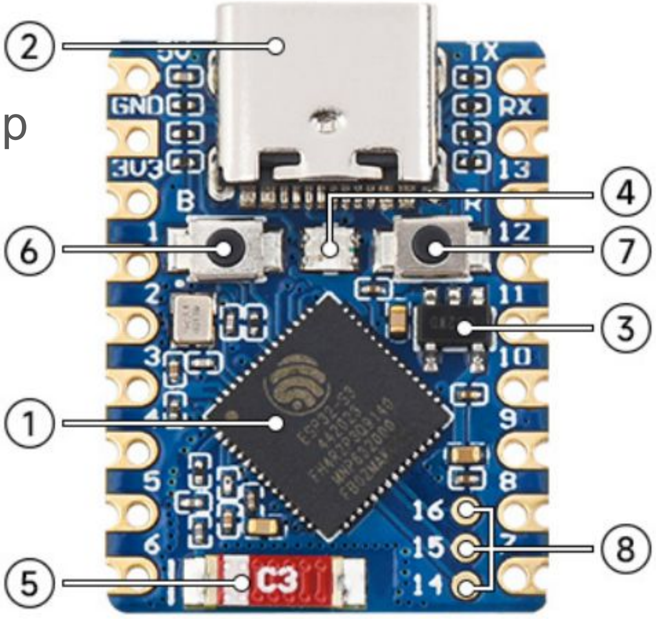
What is a microcontroller?

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

Hardware tinkering optional - complete devices available

Complete ESP32 based device with sensors etc.: 20 - 50 USD

Installing MicroPython

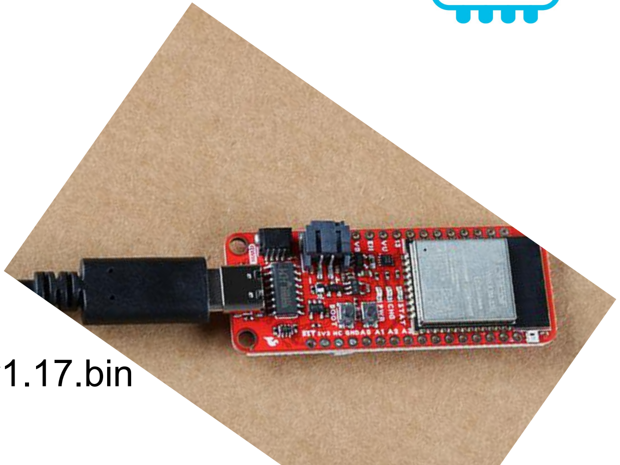
Download prebuilt firmware

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

Flash firmware to device

pip install esptool

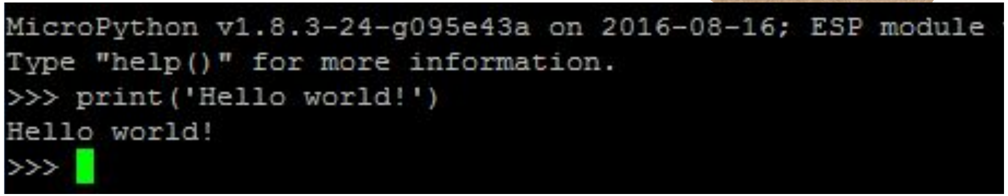
esptool.py --chip esp32 --port … erase_flash
esptool.py --chip esp32 --port … write_flash -z 0 micropython-v1.17.bin



Connect to device

pip install mpremote

mpremote repl



IDE (optional): Viper IDE, Thonny, et.c.

Temperature sensor - hardware

Example
MPU6886 accelerometer
built-in temperature sensor

Temperature Sensor
DS1820
Water-proof

Room Monitoring
+ CO2
+ humidity
+ temperature

Temperature sensor - code

Using https://viper-ide.org/

Zero-install. Connect to device via USB

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

The same approach can be used for sensing other slow changing phenomena

Using peterhinch/micropython-mqtt
and jonnor/micropython-mpu6886

test.py

```python
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}', qos = 0)
        await asyncio.sleep(30)

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

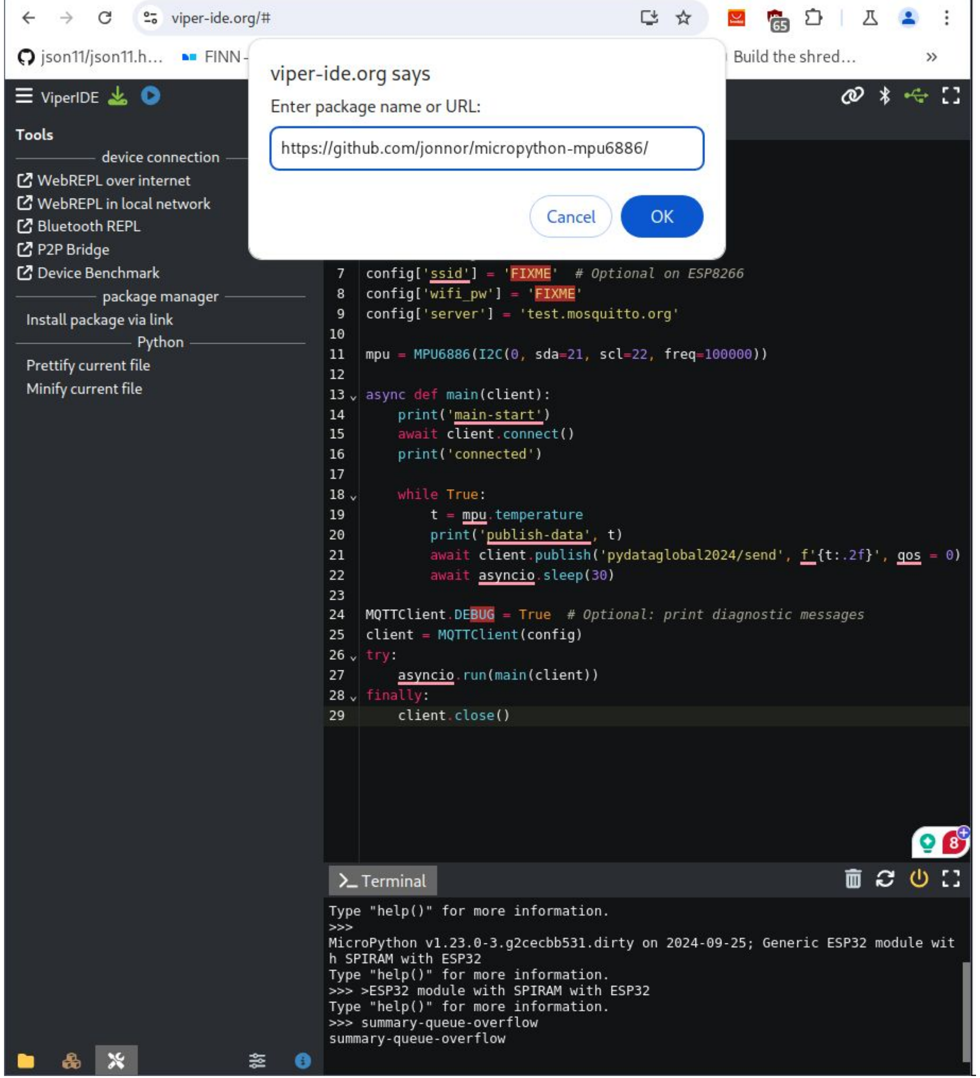
Installing packages

MicroPython has a package manager "**mip**"

Directly on device!

ViperIDE supports install via UI

Custom packages via URL



Sensor nodes with MicroPython

What can be done on an ESP32 microcontroller with MicroPython and how to make it work

Temperature logger

Temperature
0.1 Hz

Activity tracker

Accelerometer
100 Hz

Random Forest classifier
emlearn_trees

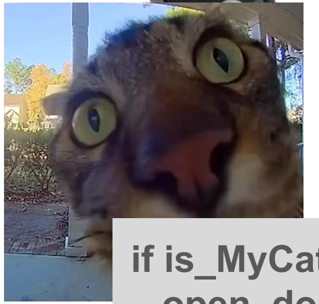
Noise monitor

Microphone
16000 Hz

Infinite Impulse Response filters
emlearn_iir

Image Classifier

Camera
27000 bytes/s



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

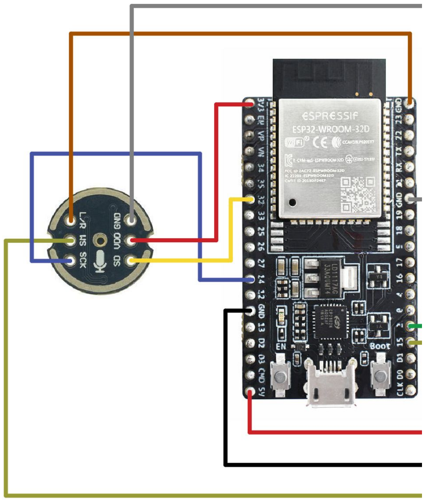
Convolutional Neural Network
emlearn_cnn

Feasibility — **Pure Python** — **Python with C modules**

Sound sensor

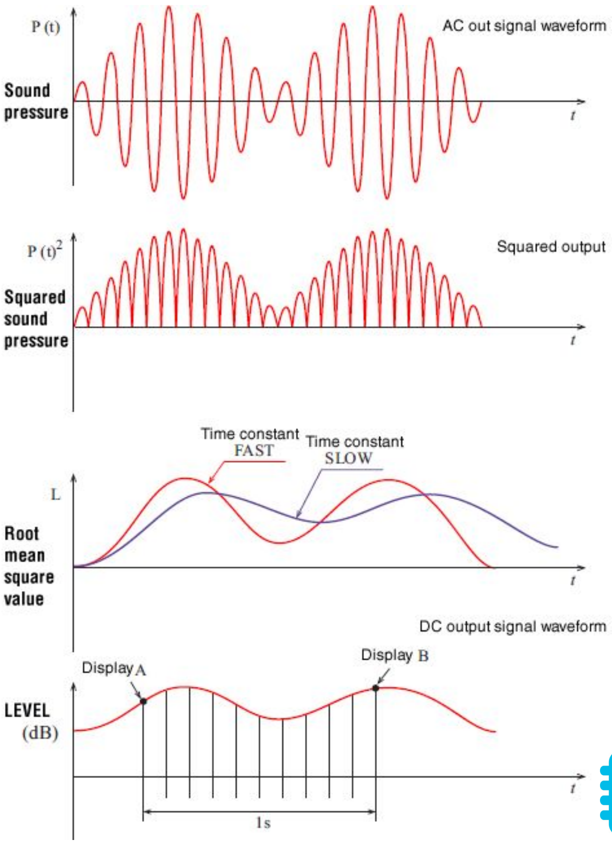
Sound sensor

I2S digital microphone
(Example wiring)



Complete hardware unit
LilyGo TTGO
T-Camera S3

Soundlevel calculation
Processing steps



Sound sensor - Install it

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

Running on device (ViperIDE)

The fastest and easiest to to install on your device is to use Viper IDE. This will install the library and the example code automatically.

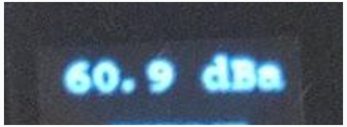
Run in ViperIDE

In Viper IDE, you can select which example file to run (described below), and hit the Play button to run it.

```python
# 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()
```

Microphone input

machine.I2S



Rendering text/widgets to screen
https://github.com/peterhinch/micropython-nano-gui

Audio filters using emlearn_iir

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

Implemented using **Infinite Impulse Response (IIR) filters**.

Must be computed within 125 ms

Python implementation
1100 ms   900% CPU

emlearn_iir C module
30 ms   20% CPU

```
import emlearn_iir

# setup
coefficients = array.array('f', [
    1.0383002230320646, 0.0, 0.0, 1.0, -0.016647242439959593, 6.92
    1.0, -2.0, 1.0, 1.0, -1.7070508390293027, 0.7174637059318595,
    1.0, -2.0, 1.0, 1.0, -1.9838868447331497, 0.9839517531763131
])
iir = emlearn_iir.new(coefficients)

# process
iir.run(samples)
```

```
# Use C module for data conversion
from emlearn_arrayutils import linear_map

def int16_to_float(inp, out):
    return linear_map(inp, out, -2**15, 2**15, -1.0, 1.0)

def float_to_int16(inp, out):
    return linear_map(inp, out, -1.0, 1.0, -2**15, 2**15)
```

C modules

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

Can be implemented by users, libraries or be part of MicroPython core.

Can be portable or specific to one hardware/platform

```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
}
```

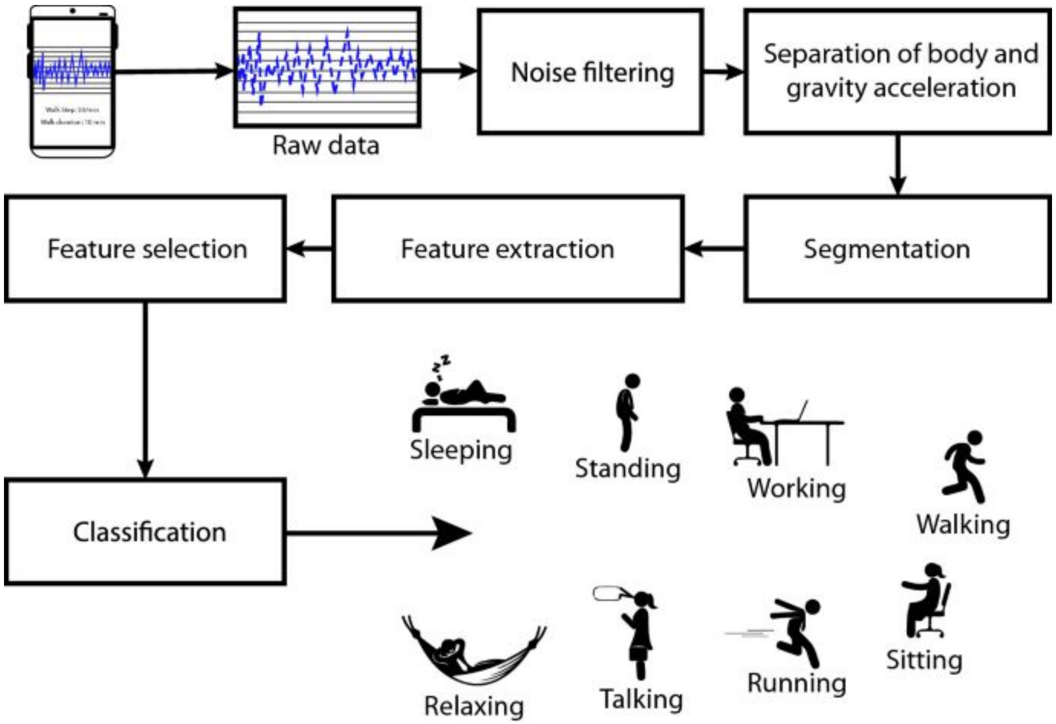
Native module (.mpy) VS External C module

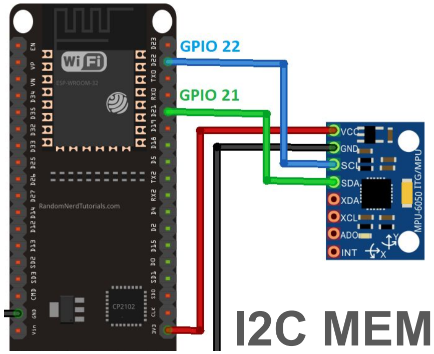
	Native module	External C module
Installable at runtime	Yes, as .mpy file	No. Must be included in firmware image
Requires SDK/toolchain	No (only to build)	Yes
Code executes from	RAM	FLASH
Limitations	No libc / libm linked * No static BSS *	None
Maturity	Low *	Excellent
Documentation	https://docs.micropython.org/en/latest/develop/natmod.html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

Activity tracker

Activity tracker - concept



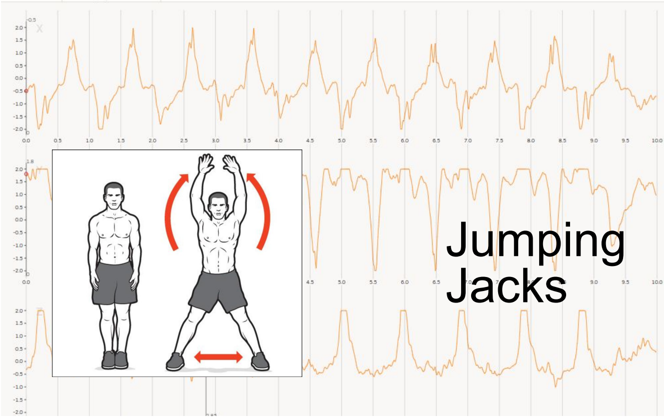


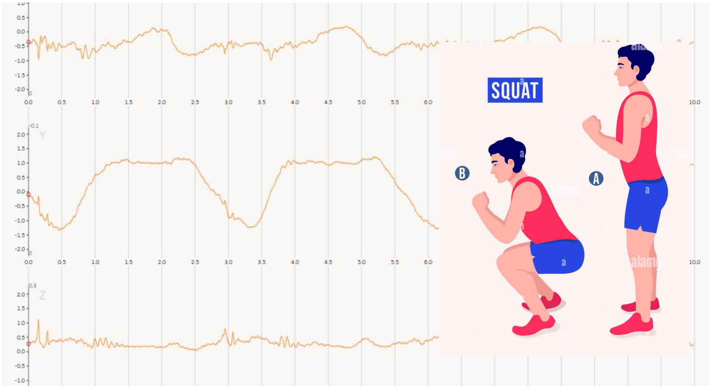
I2C MEMS IMU accelerometer/gyro
(Example wiring)

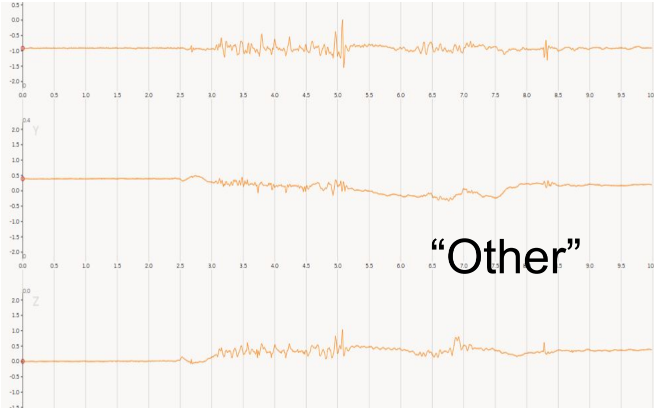
Complete hardware unit
LilyGo
T-Watch S3

Recording a dataset

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







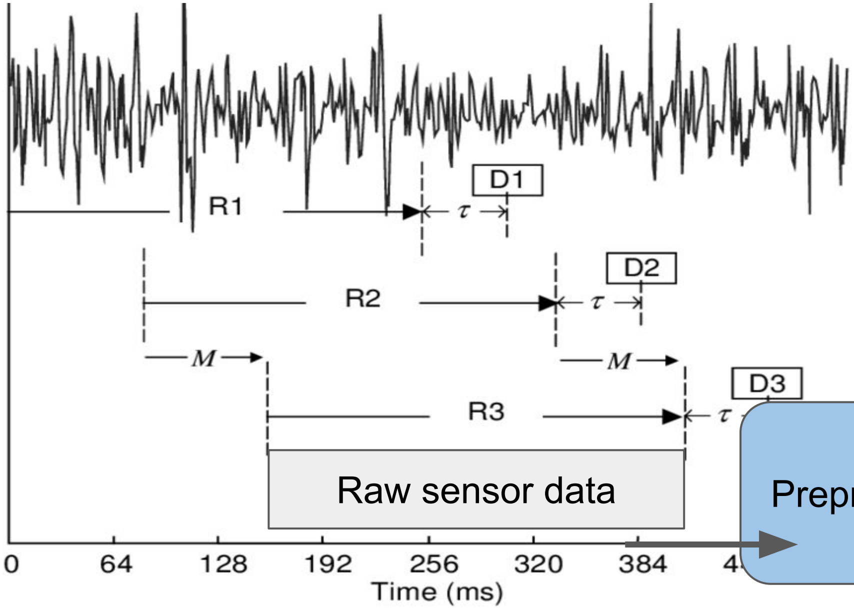
3 separate series,
1 minute per activity

Saves 10 sec .npy files

Transfer over USB

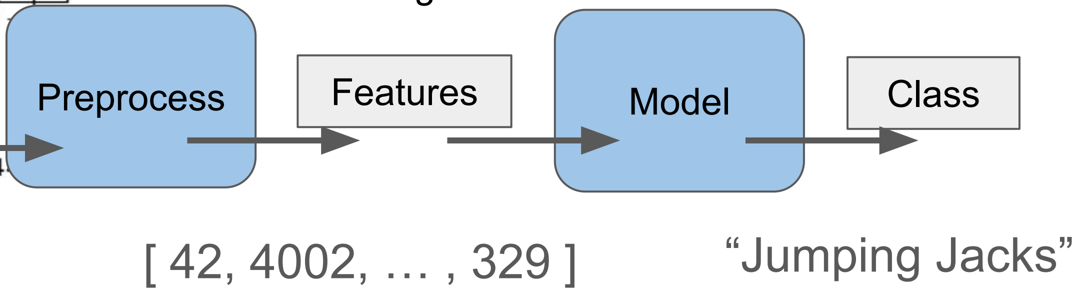
Cleaned labels in
Label Studio

ML on streams: Continuous classification



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

Exercise activity detection:
- 4 second window, every 1 second
- 100 Hz samplerate
- Processing time 200 ms



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

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')
```

Deploying trained model

Copy model to device

mpremote fs cp **gesture_model.csv** :

Load and run on device

```
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**

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

emlearn for C

Embedded Friendly

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

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

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

Summary

Conclusions

1. **Modern microcontrollers are very accessible**

Runs (Micro)Python!

ESP32 recommended

ViperIDE easy start

2. **MicroPython productive environment for sensor devices**

Python familiarity and ease-of-use

mip package manager

Good connectivity

3. **Can implement advanced processing of sensor data**

Accelerometer, audio, image, radar, ….

C modules a killer feature

emlearn-micropython: modules for DSP and Machine Learning

More resources

emlearn-micropython

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

Ulab: Numpy implementation for MicroPython https://github.com/v923z/micropython-ulab

OpenMV: Computer/Machine Vision for MicroPython https://openmv.io/

PyCon Berlin 2024: **Machine Learning on microcontrollers using MicroPython and emlearn**
https://www.youtube.com/watch?v=S3GjLr0ZIE0
PyData ZA 2024: **Sensor data processing on microcontrollers with MicroPython**
https://za.pycon.org/talks/31-sensor-data-processing-on-microcontrollers-with-micropython/

Microcontrollers and Machine Learning

with MicroPython and emlearn

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

Jon Nordby jononor@gmail.com
foss-north 2025

Bonus

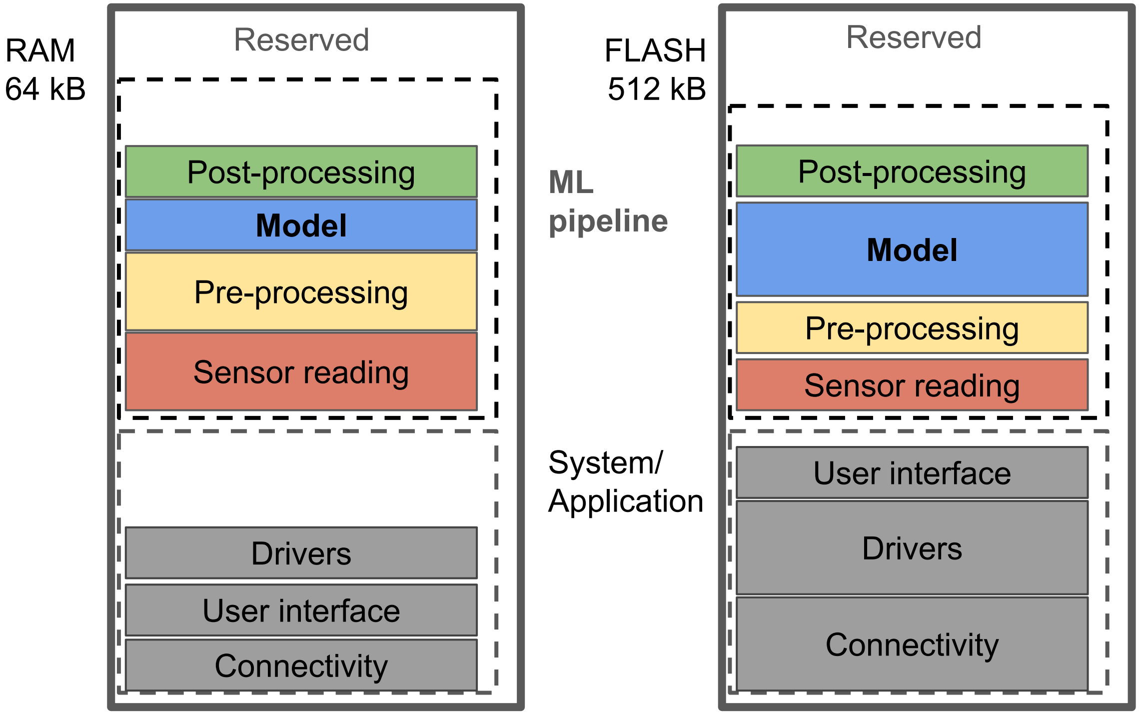
Model optimization

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

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

Compute constraints

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



CPU
Cortex M4F

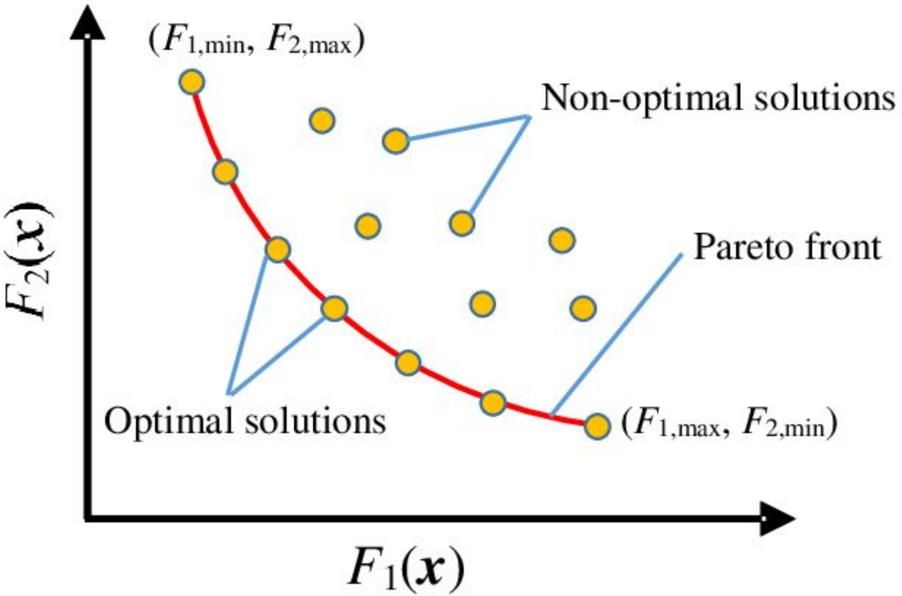
Pareto front - performance/compute tradeoffs

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

No point considering the non-optimal solutions!

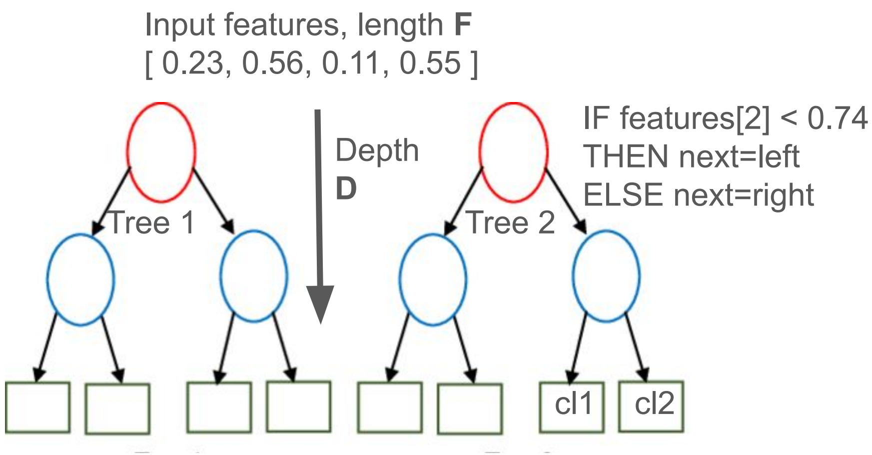
Pareto front is formed by the set of optimal solutions - that dominate the non-optimal ones

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

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

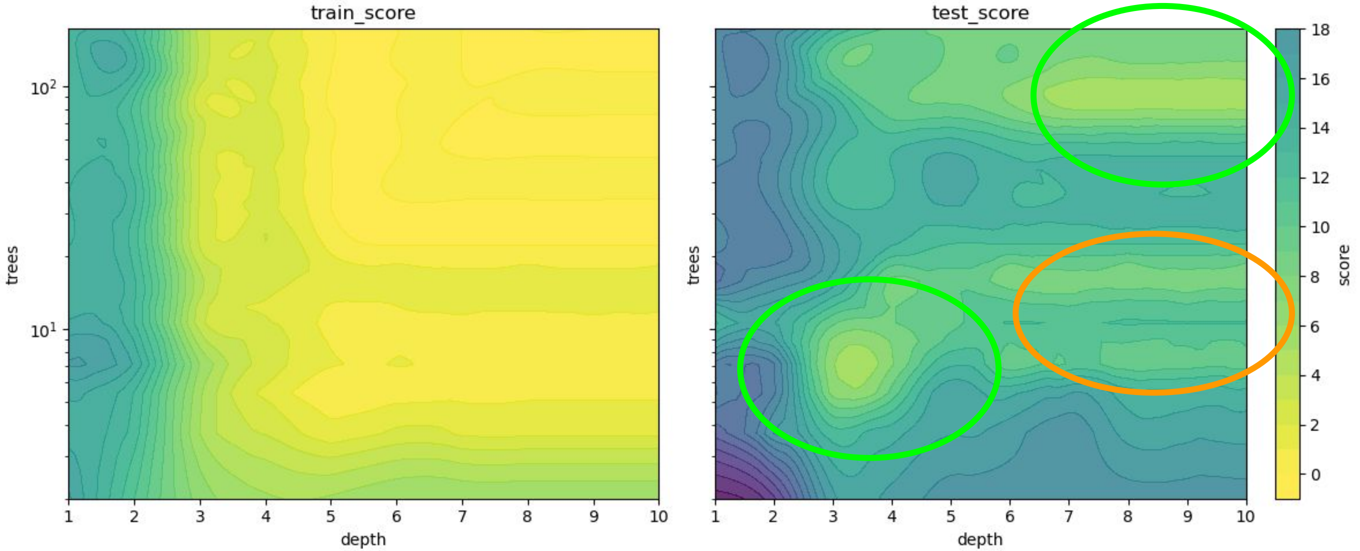
from **emlearn.evaluate.trees** import
model_size_bytes,
compute_cost_estimate

[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	

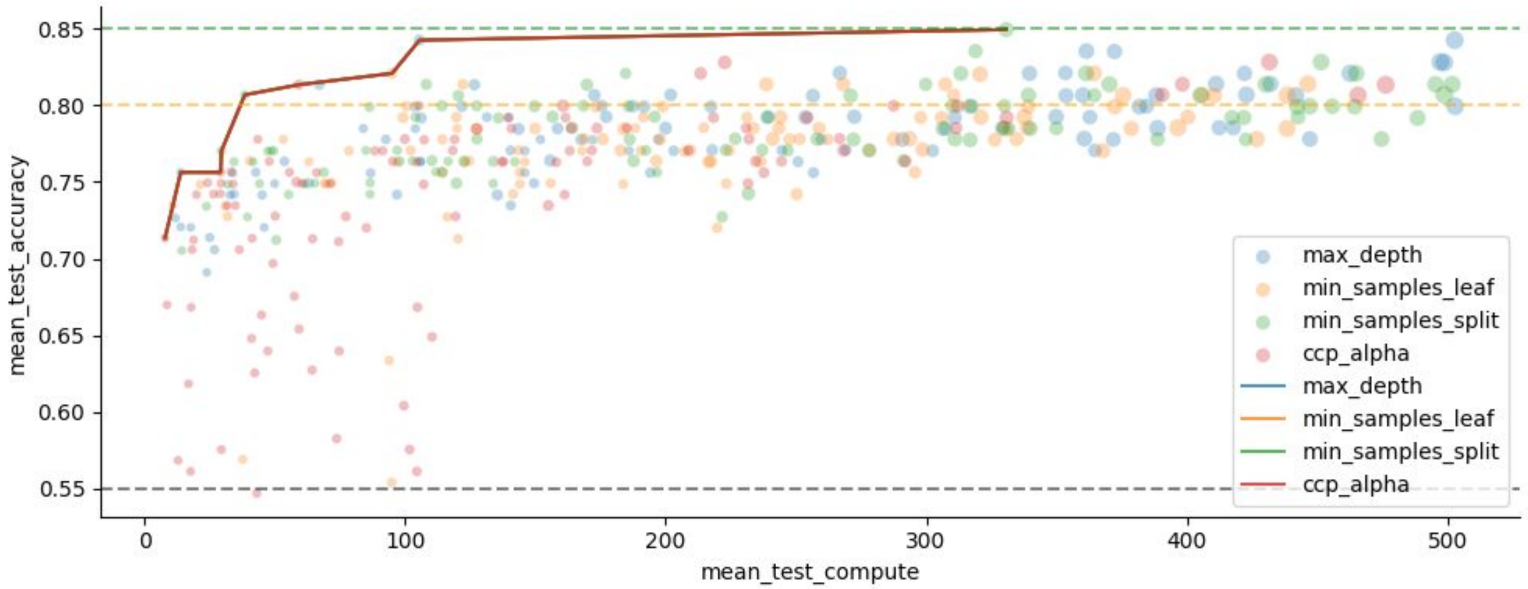
For **generalized performance** (on unseen data), **need to balance overfitting.**



Hyperparameter tuning with scikit-learn

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

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



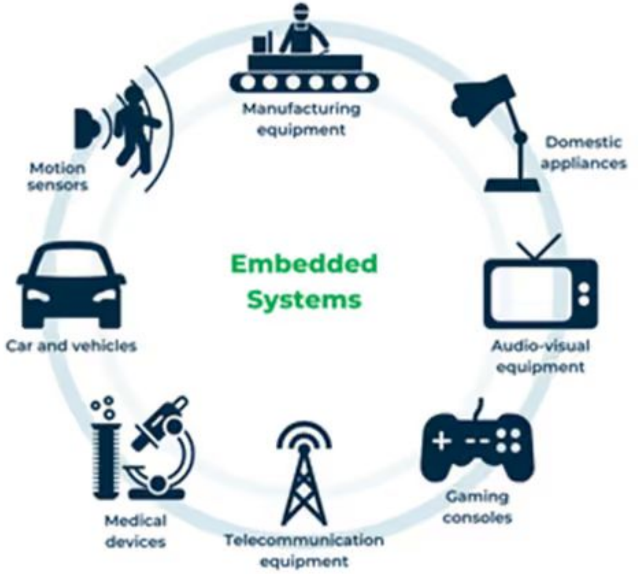
https://emlearn.readthedocs.io/en/latest/auto_examples/trees_hyperparameters.html

Bonus

TinyML for MicroPython - comparisons

Project	Deployment	Models	Size	Compute time
emlearn -micropython	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)

Microcontroller - tiny programmable chip



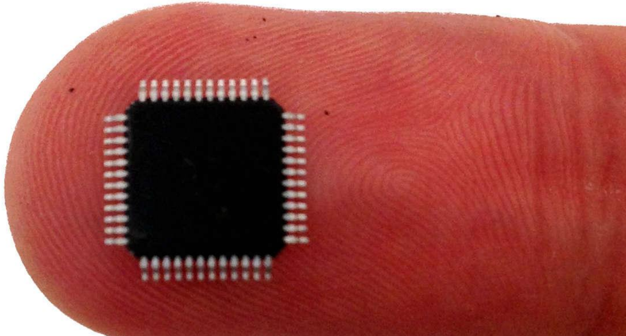
Compute power: 1 / 1000x of a smartphone

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

Efficiency is key !
Memory, compute, power

Over 20 *billions* shipped *per year*!

Increasingly accessible for hobbyists



2010: Arduino Uno
2014: MicroPython
2019: MicroPython 1.10 - ESP32 PSRAM

Inline Assembly

MicroPython can expose Assembler opcodes as Python statements.

Allows to write a function in Assembler
inline in the Python program
Can compile and execute on device

Supported on ARM Cortex M chips
Not supported (yet) on ESP32

For the most hardcore hackers!

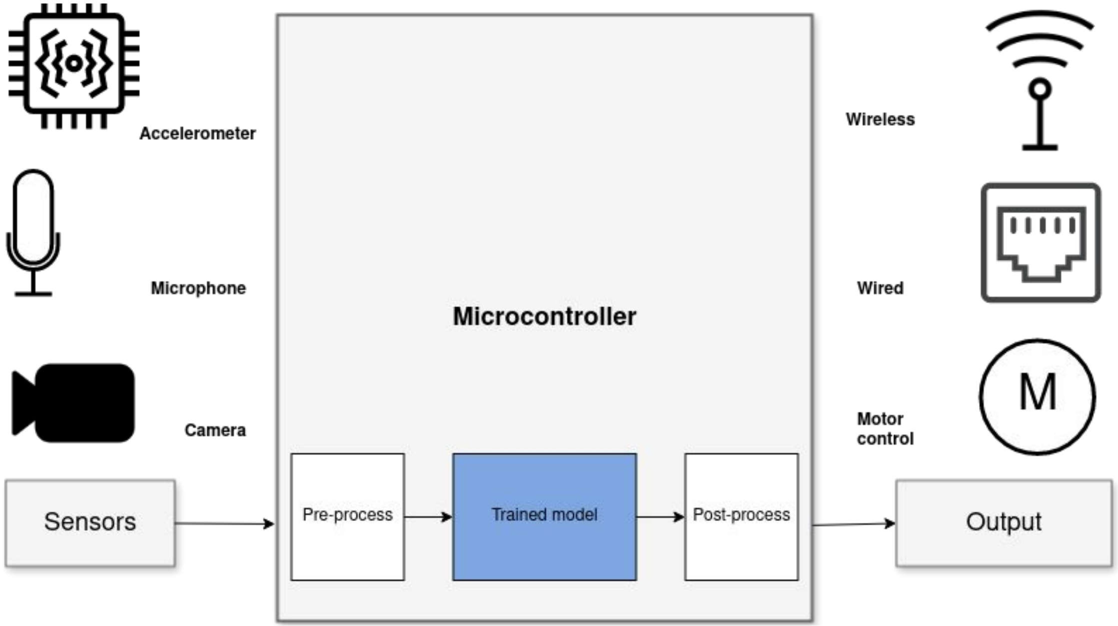
Official Documentation:
https://docs.micropython.org/en/latest/reference/asm_thumb2_index.html

```
@micropython.asm_thumb
def fir(r0, r1, r2):
    mov(r3, r8)          # For Pico: can't push({r8}). r0-r7 only.
    push({r3})
    ldr(r7, [r0, 0])     # Array length
    mov(r6, r7)          # Copy for filter
    mov(r3, r0)
    add(r3, 12)          # r3 points to ring buffer start
    sub(r7, 1)
    add(r7, r7, r7)
    add(r7, r7, r7)      # convert to bytes
    add(r5, r7, r3)      # r5 points to ring buffer end (last valid address)
    ldr(r4, [r0, 8])     # Current insertion point address
    cmp(r4, 0)           # If it's zero we need to initialise
    bne(INITIALISED)
    mov(r4, r3)          # Initialise: point to buffer start
    label(INITIALISED)
    str(r2, [r4, 0])     # put new data in buffer and post increment
    add(r4, 4)
    cmp(r4, r5)          # Check for buffer end
    ble(BUFOK)
    mov(r4, r3)          # Incremented past end: point to start
    label(BUFOK)
    str(r4, [r0, 8])     # Save the insertion point for next call
                         # *** Filter ***
    ldr(r0, [r0, 4])     # Bits to shift
```

Example: FIR filter implementation (cut out)
https://github.com/peterhinch/micropython-filters/blob/master/fir.py

Sensor node systems

Including on-sensor data processing
with Digital Signal Processing (DSP) and Machine Learning (ML)



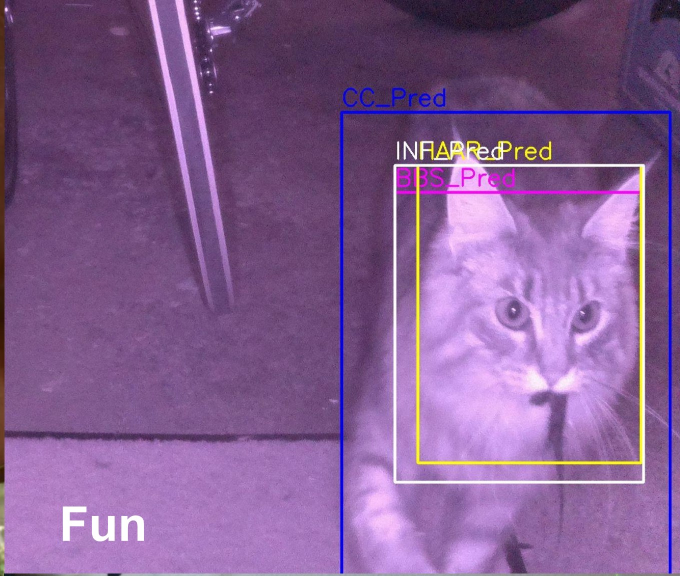
“Internet of Things”

“Wearable devices”

Robotics,
Industrial automation


Hey Google...
Consumer Tech


Industrial


CC_Pred
BBS_Pred
Fun


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

Cat Detector - Data Acquisition

Classify *low-res* images, at *1 FPS*+

Using mp_camera by cnadler86
https://github.com/cnadler86/micropython-camera-API

96x96 px grayscale.
Takes ~100 ms - OK

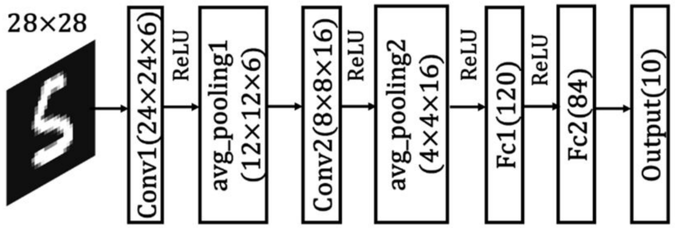
Cat Detector - Classification with CNN

emlearn_tinymaix_cnn

Go-to solution for image classification:
Convolutional Neural Network (CNN)

32x32 px input. 3 layers. Under 100 ms - OK

Preprocessing (untested)
- Downscaling 96 px -> 32 px
- Image brightness normalization



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)
```

See also OpenMV - MicroPython for Computer Vision- https://docs.openmv.io/index.html