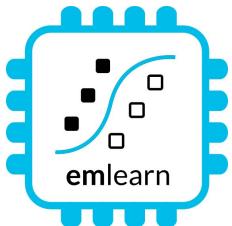


MicroPython - Python for microcontrollers and Embedded Linux

with focus on sensor-oriented applications

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



FOSDEM 2025, Brussels

Embedded devroom

Jon Nordby jononor@gmail.com



FOSDEM



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

 **sound sensing**

Condition Monitoring

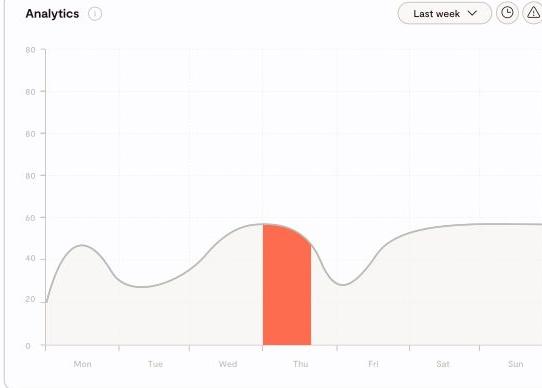
Devices overview Search Filter by Organization Hotel Manana

Tech_Storage

Location	Room	Status
Sandstuvein...	Island 04	Green
Storgata 25...	TBJ 291	Yellow
Chr. Krohgs...	Tech 275	Red
Møllergata 12...	Ohio 3	Green
Ruseløkkveien...	HKM 261	Green
Kristian IVs...	Freeway 273	Yellow
C. J. Hambro...	Oxaca2	Green
Akersgata 65...	Storage_3	Green

Tech 275

Analytics Last week More Less



Mon Tue Wed Thu Fri Sat Sun

Trusted by Nordic market leaders

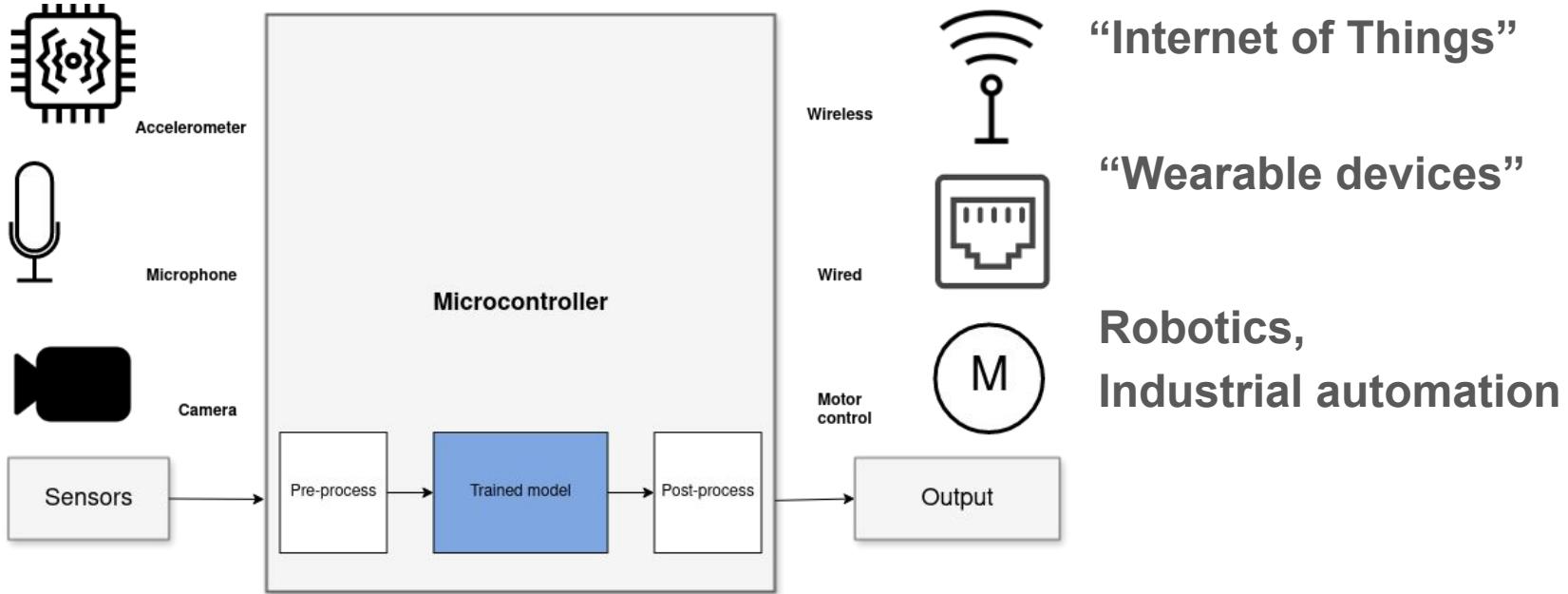
Goal

Purpose of this presentation

You as an
embedded / software developer
(professional or hobbyist)

will **learn enough about MicroPython**
to ***consider it for a future project***

Focus: Sensor node systems



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

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

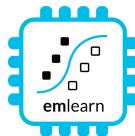
Environment logger

Temperature



Activity tracker

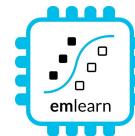
Accelerometer



Random Forest classifier
emlearn_trees

Noise monitor

Microphone



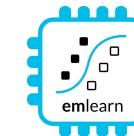
Infinite Impulse Response filters
emlearn_iir

Image Classifier

Camera



if is_MyCat(img):
open_door()



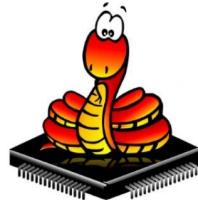
Convolutional Neural Network
emlearn_cnn

Outline / agenda

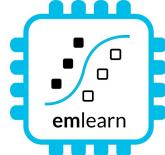
1. MicroPython project overview
2. Tour of MicroPython features
 - ...
 - Sensor communication
 - Connectivity
 - **Native C modules** for efficient data processing
3. Sensors using Digital Signal
4. MicroPython on (Embedded) Linux

MicroPython introduction

Jumping right into it



MicroPython



MicroPython introduction

Started in 2014

For devices with 64 kB+ RAM (256 kB+ recommended)

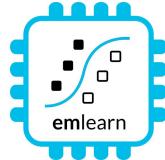
Supports 8+ microcontroller families

Tries to be as compatible with CPython as possible, within constraints.
Python 3.6 mostly implemented, partial after that.

Package manager “**mip install**”

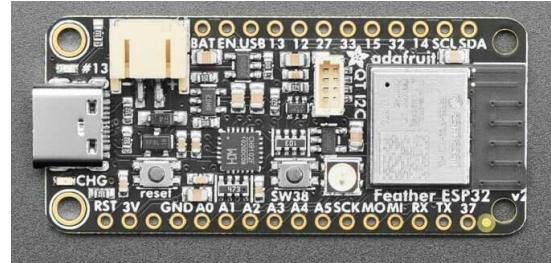
Has support for loading C modules at runtime!

More info: <https://micropython.org>

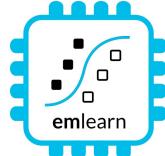


Hardware recommendation

- start with a **ESP32** device



Complete device with sensors etc.:	20 - 50 USD
Development boards:	5 - 20 USD
Chips / modules	1 - 5 USD



Installing MicroPython

Download prebuilt firmware

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

Flash firmware to device

pip install esptool

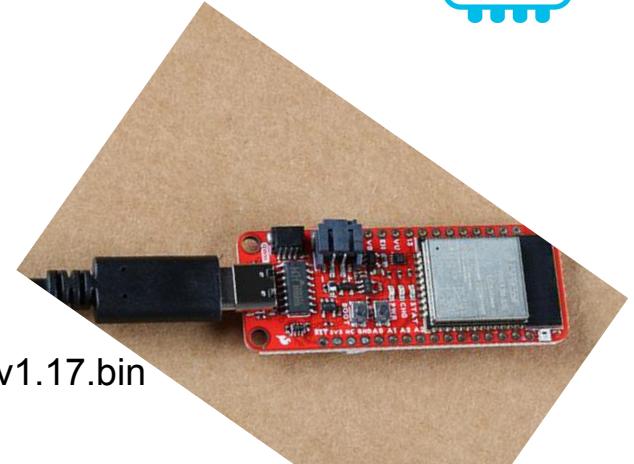
esptool.py --chip esp32 --port ... erase_flash

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

Connect to device

pip install mpremote

mpremote repl



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

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

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

Using [peterhinch/micropython-mqtt](https://github.com/peterhinch/micropython-mqtt) and [jonnor/micropython-mpu6886](https://github.com/jonnor/micropython-mpu6886)

```
1 from mqtt_as import MQTTClient, config
2 import asyncio
3 from mpu6886 import MPU6886
4 from machine import I2C
5
6 # Local configuration
7 config['ssid'] = 'FIXME' # Optional on ESP8266
8 config['wifi_pw'] = 'FIXME'
9 config['server'] = 'test.mosquitto.org'
10
11 mpu = MPU6886(I2C(0, sda=21, scl=22, freq=100000))
12
13 v async def main(client):
14     print('main-start')
15     await client.connect()
16     print('connected')
17
18 v     while True:
19         t = mpu.temperature
20         print('publish-data', t)
21         await client.publish('pydataglobal2024/send', f'{t:.2f}')
22         await asyncio.sleep(30)
23
24 MQTTClient.DEBUG = True # Optional: print diagnostic messages
25 client = MQTTClient(config)
26 v try:
27     asyncio.run(main(client))
28 v finally:
29     client.close()
```

Try it now - running in the browser!

WOKWI SAVE SHARE esp32-micropython-neopixels by urish

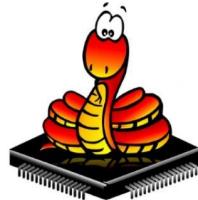
main.py • diagram.json • Simulation

```
1 # NeoPixels Rainbow on MicroPython
2 # Wokwi Example https://wokwi.com/arduino/projects/30556906
3
4 from machine import Pin
5 from neopixel import NeoPixel
6 from time import sleep
7
8 rainbow = [
9     (126, 1, 0), (114, 13, 0), (102, 25, 0), (90, 37, 0),
10    (30, 97, 0), (18, 109, 0), (6, 121, 0), (0, 122, 5),
11    (0, 62, 65), (0, 50, 77), (0, 38, 89), (0, 26, 101),
12    (33, 0, 94), (45, 0, 82), (57, 0, 70), (69, 0, 58),
13
14 pixels = NeoPixel(Pin(15), 16)
15 while True:
16     rainbow = rainbow[-1:] + rainbow[:-1]
17     for i in range(16):
18         pixels[i] = rainbow[i]
19     pixels.write()
20     sleep(0.05)
21
```

Diagram showing the connection between an ESP32 module and a NeoPixel ring. The ESP32's D15 pin is connected to the NeoPixel's GND pin. The ESP32's 3V3 pin is connected to the NeoPixel's VIN pin. The NeoPixel ring has 16 segments, each consisting of three pixels (red, green, blue) arranged in a triangle.

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x000,q_drv:0x000,d_drv:0x000,cs0_drv:0x000,hd_drv:0x000,wp_drv:0x000
mode:DIO, clock div:2
load:0x7fffef0030 len:4540
```

MicroPython tour



MicroPython

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

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

File system

Enabled on most ports/hardware
(with sufficient resources)

Internal FLASH and/or SDCard

LittleFS or FAT32

Save/load from standard files

[https://docs.micropython.org
/en/latest/reference/filesystem.html](https://docs.micropython.org/en/latest/reference/filesystem.html)

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

mpremote

Tool for PC <-> microcontroller communication

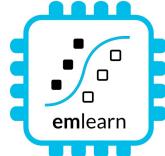
[https://docs.micropython.org
/en/latest/reference/mpremote.html](https://docs.micropython.org/en/latest/reference/mpremote.html)

Copy from device

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

Copy to device

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



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

Connectivity

BLE - Bluetooth Low Energy

[aioble](#) - high-level application API, asyncio

[bluetooth](#) - low-level hardware-layer

Ports: ESP32, RP2, Unix

WiFi

[network.WLAN](#)

Ports: ESP32, RP2, STM32, etc

Ethernet

[network.LAN](#)

Ports: ESP32, RP2, STM32, etc

Hardware: Wiznet, +++

C modules *

Defines a Python module with API.
functions/classes etc.

**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](https://github.com/vshymanskyy/wasm2mpy)

C++, Rust, Zig, TinyGo, TypeScript

```
// 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
}
```

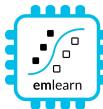


Native module (.mpy) VS External C module

	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



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()
```

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](https://github.com/cnadler86/micropython-camera-API)

OpenMV

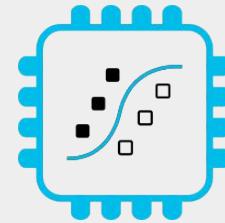
<https://openmv.io/>

Custom MicroPython distribution
Focused on Computer Vision / Machine Vision



Sensor nodes

with MicroPython
and emlearn



emlearn
-micropython

[https://github.com/
emlearn/emlearn-micropython](https://github.com/emlearn/emlearn-micropython)

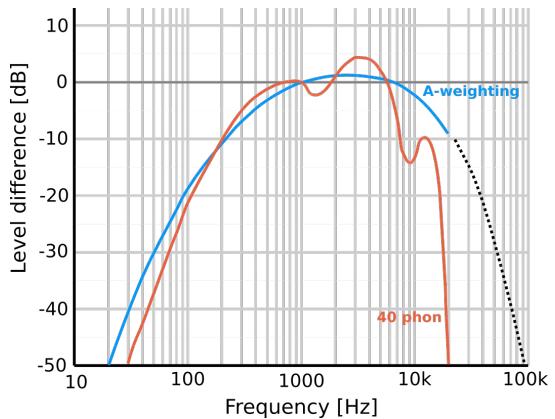
Noise sensor using emlearn_iir

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



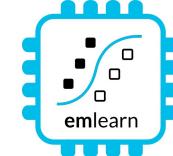
```
# 6th order filter. 3x Second Order Sections "biquad"
a_filter_16k = [
    1.0383002230320646, 0.0, 0.0, 1.0, -0.016647242439959593, 6.928267
    1.0, -2.0, 1.0, 1.0, -1.7070508390293027, 0.7174637059318595,
    1.0, -2.0, 1.0, 1.0, -1.9838868447331497, 0.9839517531763131
]
self.frequency_filter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

Image classification with emlearn_cnn

Convolutional Neural Network (CNN)

Running on ESP32-S3 (example)

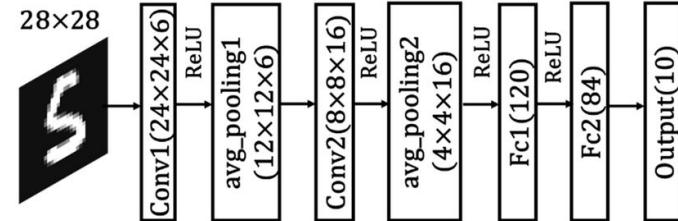
Input dimensions: 32x32 px - 96x96 px

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)

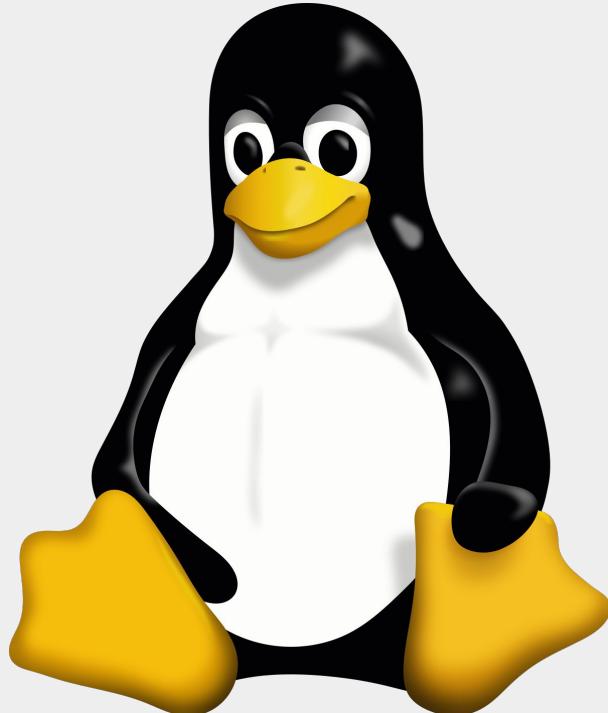
while True:

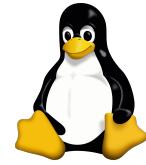
    raw = read_camera()
    img = preprocess(raw)
    classification = model.predict(img)

    if classification == MY_CAT:
        open_door()

    machine.lightsleep(500)
```

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!!!
```

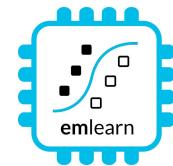
emlearn_trees with MicroPython:

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





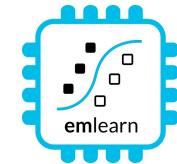
Unix MicroPython port: Limited hardware access

Not implemented:

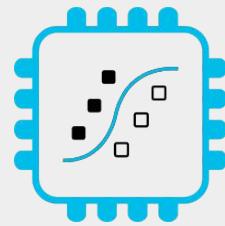
- GPIO. machine.Pin / machine.PWM / machine.ADC
- Digital busses. machine.I2C / SPI / USART / USB
- Watchdog Timer. machine.WDT
- Power management. machine.lightsleep() /deepsleep()
- Audio input/microphone
- Camera access

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

Already very useful for unit testing



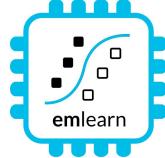
Summary



emlearn
-micropython

[https://github.com/
emlearn/emlearn-micropython](https://github.com/emlearn/emlearn-micropython)

Take aways



1. MicroPython productive environment (for sensor devices)

Python familiarity and ease-of-use

Good connectivity

mip package manager

mpremote tool for device communication

2. Can implement advanced processing of sensor data

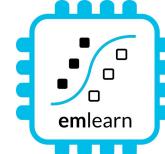
Accelerometer, audio, image, radar,

C modules a killer feature

emlearn-micropython: modules for DSP and Machine Learning

More!

FOSDEM 2025 - Low-level AI Engineering and Hacking - Sunday 16:40 (Lameere)
[Milliwatt sized machine learning on microcontrollers with emlearn](#)



FOSDEM 2025 - MicroPython & Espruino stand - AW building, level 1
See you there after this talk / later today?!

Official documentation

<https://micropython.org/>

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

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

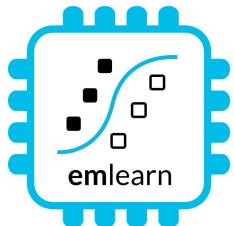
TinyML EMEA 2024: *emlearn - scikit-learn for microcontrollers and embedded systems*
<https://www.youtube.com/watch?v=LyO5k1VMdOQ>

PyData ZA 2024: *Sensor data processing on microcontrollers with MicroPython (video soon)*
<https://za.pycon.org/talks/31-sensor-data-processing-on-microcontrollers-with-micropython/>

MicroPython - Python for microcontrollers and Embedded Linux

with focus on sensor-oriented applications

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



FOSDEM 2025, Brussels

Embedded devroom

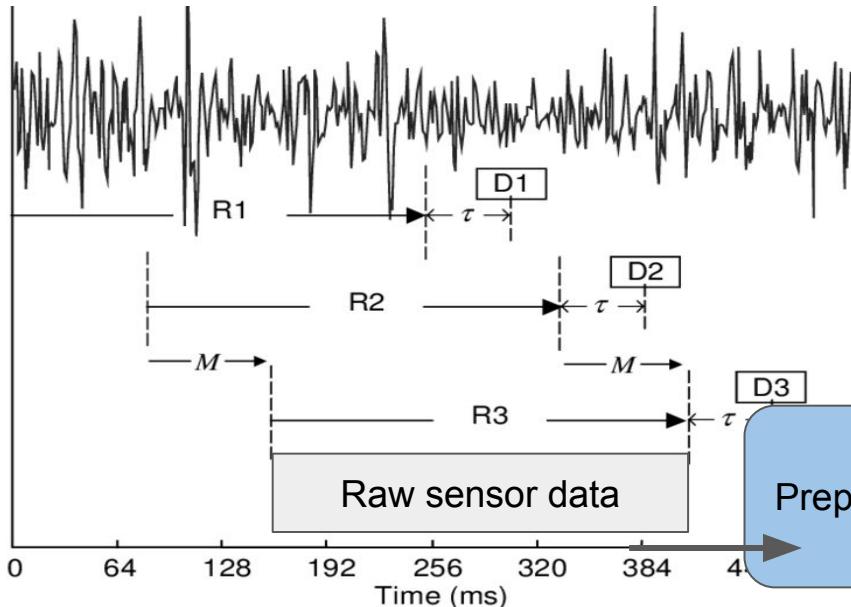
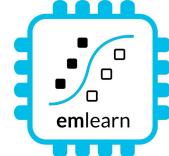
Jon Nordby jononor@gmail.com



FOSDEM

Bonus

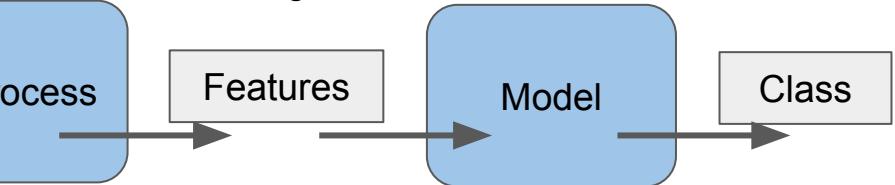
ML on streams: Continuous classification



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



[42, 4002, ..., 329]

“Jumping Jacks”

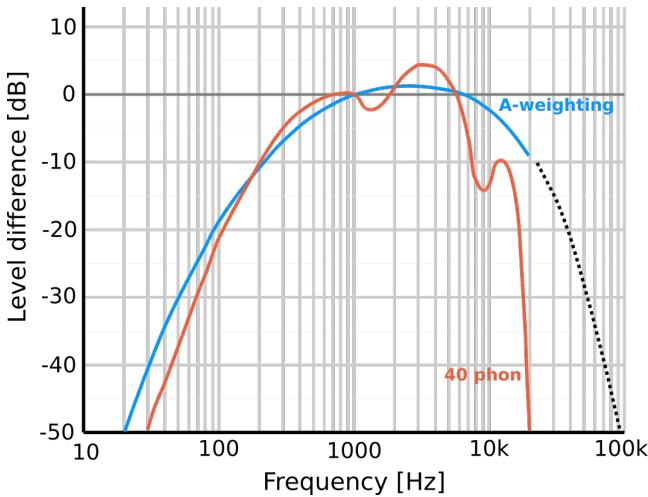
Implementing an IMU/accelerometer/gyro driver? Use the FIFO!

<https://github.com/orgs/micropython/discussions/15512>

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, 6.9282670,
    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

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

```
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)
```

```
@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
```

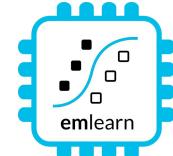
```
int16_to_float(samples, self.float_array)
self.frequency_filter.process(self.float_array)
float_to_int16(self.float_array, samples)
```

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



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	<u>(build-your-own)</u> <u>Using ndarray</u> <u>primitives</u>	High initial size	Unknown (assume good)



**Make it Work,
Make it Right,
Make it Fast**

- Ken Beck

Write simple automated tests,
Code in straightforward Python,
Measure performance with benchmarks

Optimize *if needed*
Start with simple techniques
Go more advanced *if needed*

time.time and **assert**
for benchmark and tests

```
import time

repeats = 100
expect = 18965.39
input = .....

start = time.ticks_us()
for r in range(repeats):
    out = rms_python(input)
    assert p == expect, (out, expect)
t = time.ticks_diff(time.ticks_us(), start) / repeats
print('python', t)

start = time.ticks_us()
for r in range(repeats):
    out = rms_micropython_native(input)
    assert out == expect, (out, expect)
t = time.ticks_diff(time.ticks_us(), start) / repeats
print('native', t)
```

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

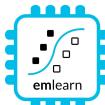
For the most hardcore hackers!

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)
    mov(r3, r0)
    add(r3, 12)                     # r3 points to ring buffer start
    sub(r7, 1)
    add(r7, r7, r7)
    add(r7, r7, r7)
    add(r5, r7, r3)
    ldr(r4, [r0, 8])
    cmp(r4, 0)
    bne(INITIALISED)
    mov(r4, r3)
    label(INITIALISED)
    str(r2, [r4, 0])
    add(r4, 4)
    cmp(r4, r5)
    ble(BUFOK)
    mov(r4, r3)
    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>



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
duration=0.016095876693725586 samples=32000
2024-12-04 12:54:56 [info] feature-extraction-done dataset=har_exercise_1
duration=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')
```

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/>

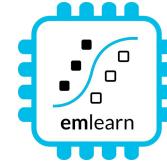
```
l = sorted(list(v))
l2 = [x*x for x in l]
sm = sum(l)
sqsum = sum(l2)
avg = sum(l) / len(l)

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

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

[https://github.com/emlearn/emlearn-micropython
/blob/master/examples/har_trees/timebased.py](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>



What is a microcontroller?

Modern microcontroller:
A complete programmable System-on-Chip

Example: ESP32-S3FH4R2

32 bit CPU, 240 Mhz

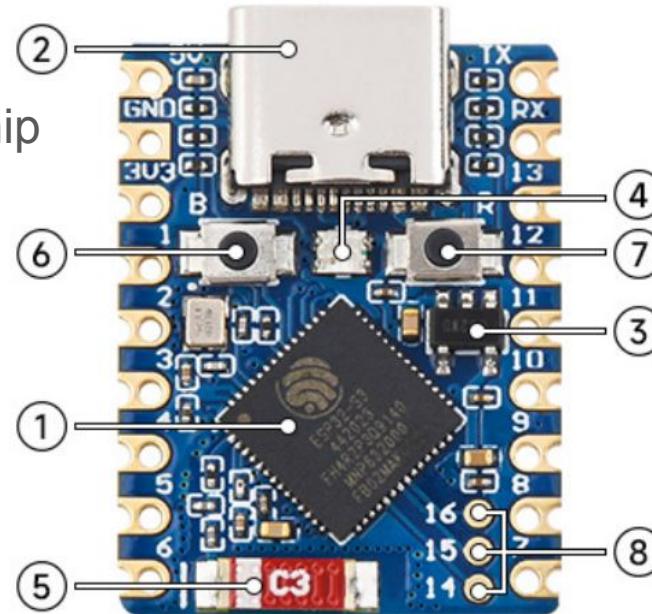
Floating Point Unit

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

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

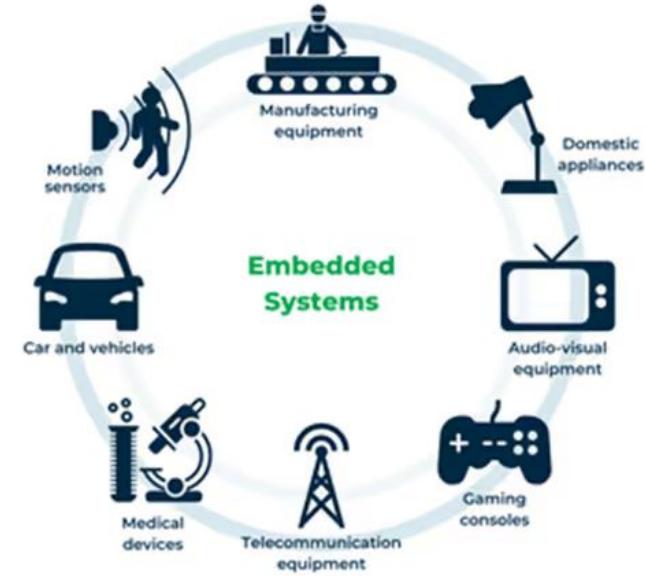
Over 20 *billions* shipped per year!

Increasingly accessible for hobbyists

2010: Arduino Uno

2014: MicroPython

2019: MicroPython 1.10 - ESP32 PSRAM



Efficiency is key !
Memory, compute, power

