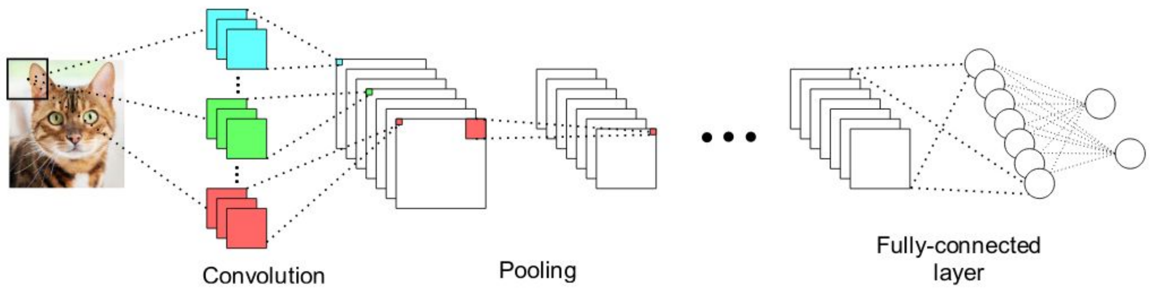
Machine Learning sur Microcontrôleurs

alexandre.abadie@inria.fr - Café LoOPS - mars 2021

TinyML en bref

➢ Techniques de machine learning sur microcontrôleurs



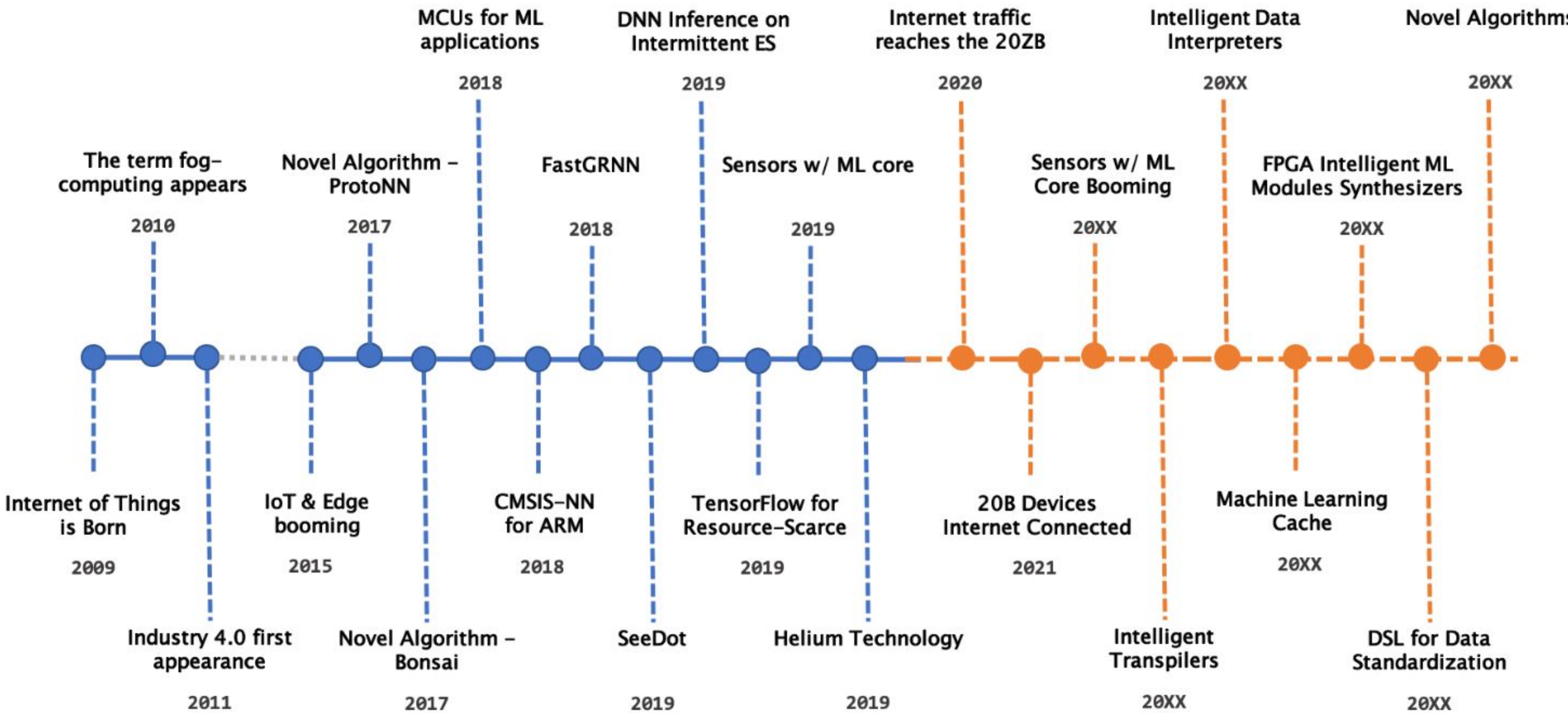
➢ Thème qui a émergé dans le courant 2019 (existe depuis 2018)

➢ Étend les possibilités des applications IoT

➢ Permet de limiter les coûts de natures diverses:
  - ➔ Réduire la consommation
  - ➔ Réduire l'utilisation des canaux de communication
  - ➔ Réduire la latence

➢ Autres bénéfices:
  - ➔ Améliorer la sécurité et le respect de la vie privée
  - ➔ Améliorer le passage à l'échelle

Timeline

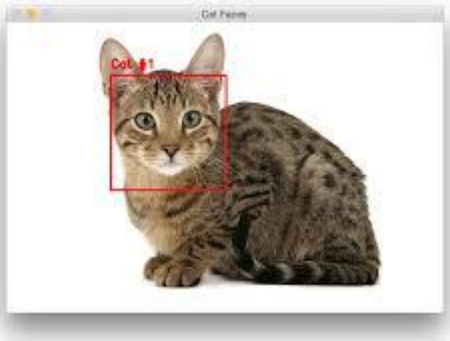


Sérgio Branco et al., Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey, Electronics 2019, 8, 1289

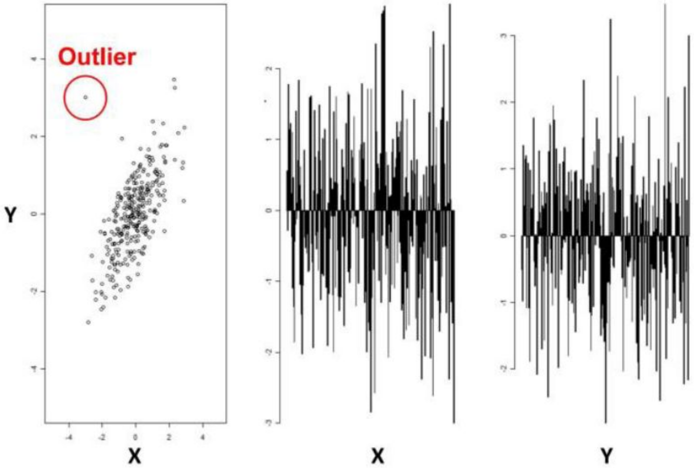
Domaines d'application

- Natural Language Processing
  - ➔ Reconnaissance vocale
  - ➔ Keyword spotting

- Vision et image
  - ➔ Reconnaissance d'image
  - ➔ Détection d'objets dans des vidéos



- Reconnaissance de gestes
- Santé
- Industrie 4.0
- Securité
- Protocoles réseaux

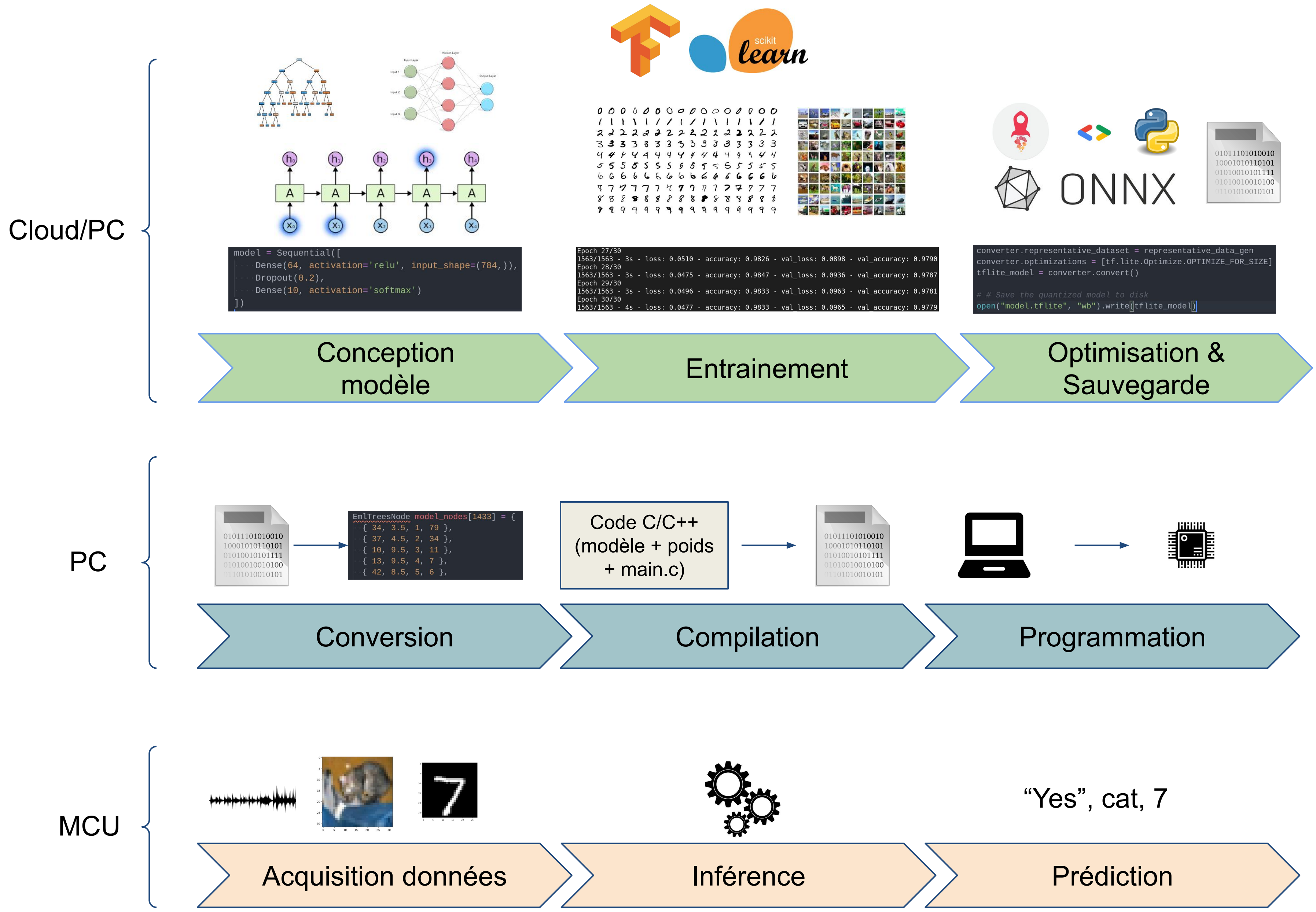


Domaines d'application

Input Type	Use Cases	Model Types	Datasets
Audio	Audio Wake Words Context Recognition Control Words Keyword Detection	DNN CNN RNN LSTM	Speech Commands (Warden, 2018a) Audioset (Gemmeke et al., 2017) ExtraSensory (Vaizman et al., 2017)
Image	Visual Wake Words Object Detection Image Classification Gesture Recognition Object Counting Text Recognition	DNN CNN SVM Decision Trees KNN Linear	Visual Wake Words (Chowdhery et al., 2019) CIFAR10 (Krizhevsky et al., 2009b) MNIST (LeCun & Cortes, 2010) ImageNet (Deng et al., 2009) DVS128 Gesture (Amir et al., 2017)
Physiological / Behavioral Metrics	Segmentation Forecasting Activity Detection	DNN Decision Tree SVM Linear	Physionet (Goldberger et al., 2000) HAR (Cramariuc, 2019) DSA (Altun et al., 2010) Opportunity (Roggen et al., 2010) UCI EMG (Lobov et al., 2018)
Industry Telemetry	Sensing (light, temp, etc) Anomaly Detection Motor Control Predictive Maintenance	DNN Decision Tree SVM Linear Naive Bayes	UCI Air Quality (De Vito et al., 2008) UCI Gas (Vergara et al., 2012) NASA's PCoE (Saxena & Goebel, 2008)

Colby R. Banbury et al., Benchmarking TinyML Systems

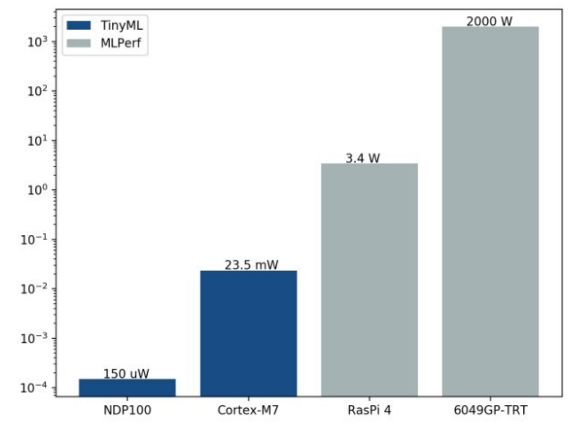
Workflow TinyML



Les microcontrôleurs

Les avantages

➔ Taille
➔ Consommation d'énergie
➔ Coût



Les contraintes

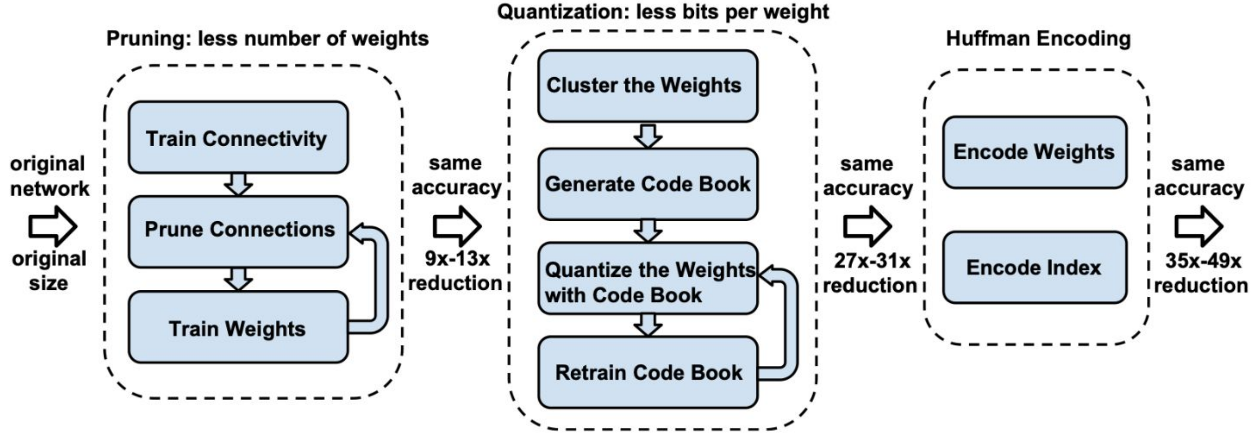
➔ Espace mémoire réduit
➔ Puissance de calcul limitée

MCU Platform	Processor	Frequency	SRAM	Flash
Arduino Nano 33 BLE Sense [6]	ARM Cortex M4	64 MHz	256 KB	1 MB
ESP32 [7]	Tensilica Xtensa LX6	160 MHz	512 KB	2 MB
Sparkfun Edge Appolo3 Blue [8]	ARM Cortex M4F	48 MHz	384 KB	1 MB
ST Nucleo Boards [9]	ARM Cortex M7	216 MHZ	320 KB	1 MB
Adafruit EdgeBadge [10]	ATSAMD51	120 MHz	192 KB	512 KB

Stanislava Soro, TinyML for Ubiquitous Edge AI, MITRE Technical report

Il faut adapter les modèles

- ➔ Machine learning classique: millions de paramètres, architectures complexes
- ➔ Relation en taille/complexité du modèle et précision
- ➔ Utilisation de techniques de compression de modèles:
  - ◆ Suppression des connexions non pertinentes (“pruning”)
  - ◆ “Quantization” des paramètres: représentation sur 8/16bit
  - ◆ Compression lossless des poids



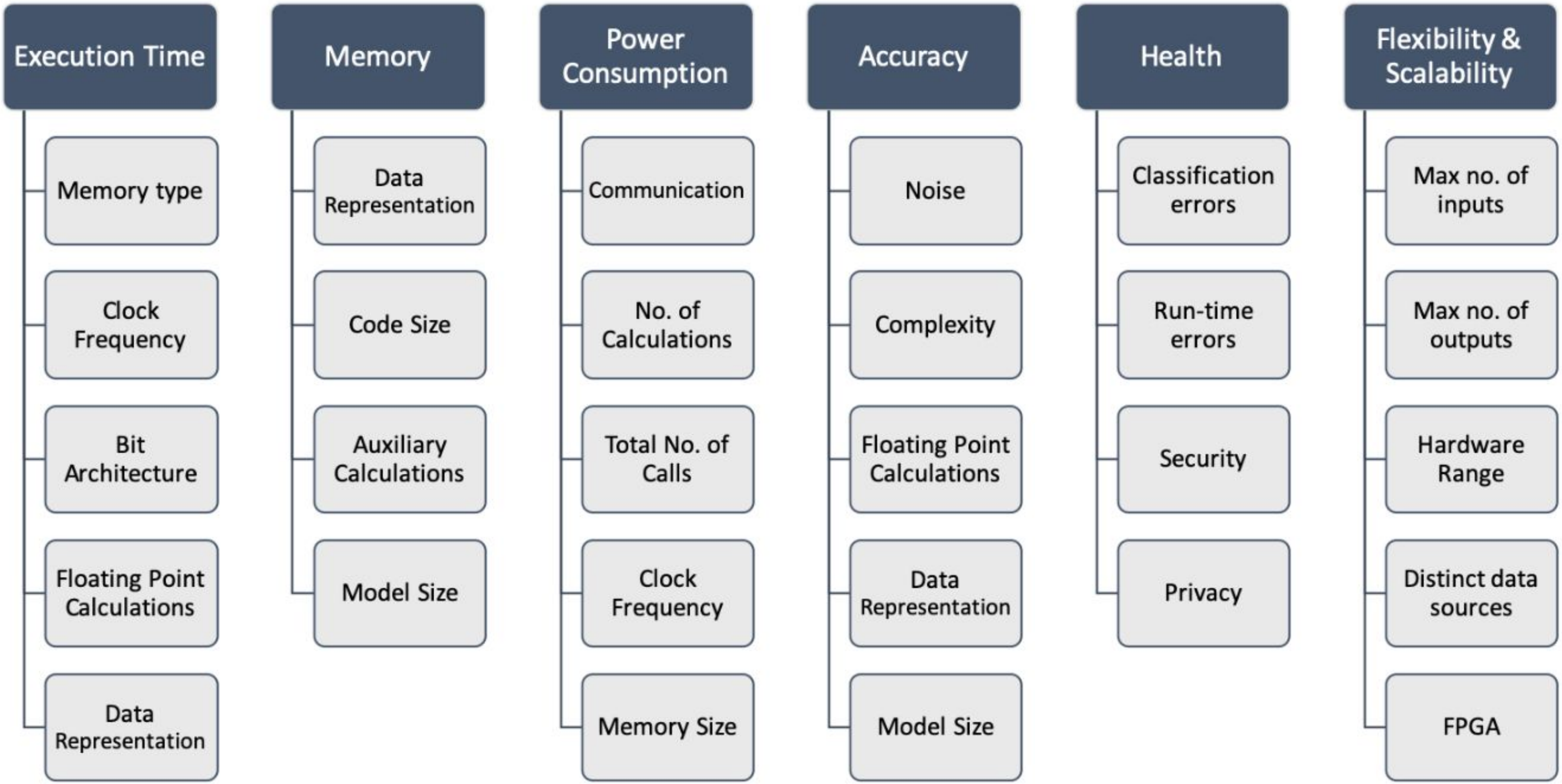
Task	Network Type	Network Architecture	Number of Parameters
Voice activity detection [23]	MLP[2]	60-24-11-2-FC[3]	5 K
Keyword spotting [26]	CNN[4]	1CL[5]-FCL-3-FCNL[6]	54 K
Speaker recognition [27]	CNN	1CL-3-FCNL	234 K
Speaker verification [24]	RNN[7]	2x220 GRU[8]	900 K
Speech enhancement [24]	RNN	500-1024-1024 FC	500 K
Speech recognition [25]	RNN	5x465 GRU	10 M

Stanislava Soro, TinyML for Ubiquitous Edge AI, MITRE Technical report

NN model	S(80KB, 6MOps)			M(200KB, 20MOps)			L(500KB, 80MOps)		
	Acc.	Mem.	Ops	Acc.	Mem.	Ops	Acc.	Mem.	Ops
DNN	84.6%	80.0KB	158.8K	86.4%	199.4KB	397.0K	86.7%	496.6KB	990.2K
CNN	91.6%	79.0KB	5.0M	92.2%	199.4KB	17.3M	92.7%	497.8KB	25.3M
Basic LSTM	92.0%	63.3KB	5.9M	93.0%	196.5KB	18.9M	93.4%	494.5KB	47.9M
LSTM	92.9%	79.5KB	3.9M	93.9%	198.6KB	19.2M	94.8%	498.8KB	48.4M
GRU	93.5%	78.8KB	3.8M	94.2%	200.0KB	19.2M	94.7%	499.7KB	48.4M
CRNN	94.0%	79.7KB	3.0M	94.4%	199.8KB	7.6M	95.0%	499.5KB	19.3M
DS-CNN	94.4%	38.6KB	5.4M	94.9%	189.2KB	19.8M	95.4%	497.6KB	56.9M

Yundong Zhang et al. Hello Edge: Keyword Spotting on Microcontrollers

Les challenges



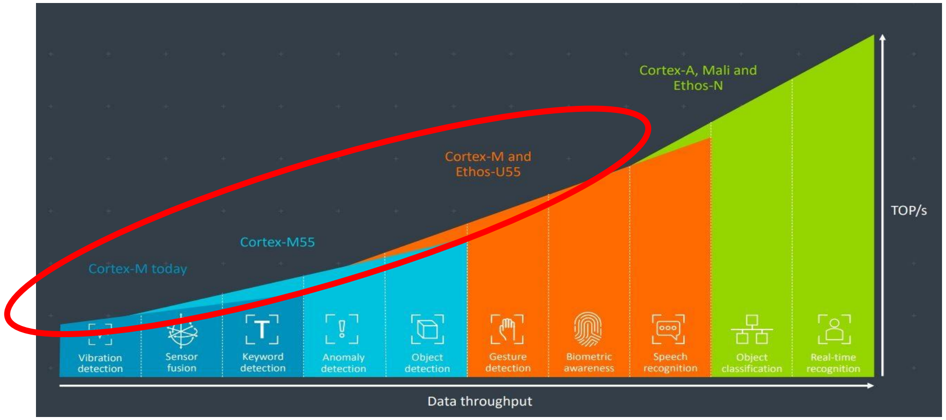
Sérgio Branco et al., Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey, Electronics 2019, 8, 1289

Solutions → hardware et software

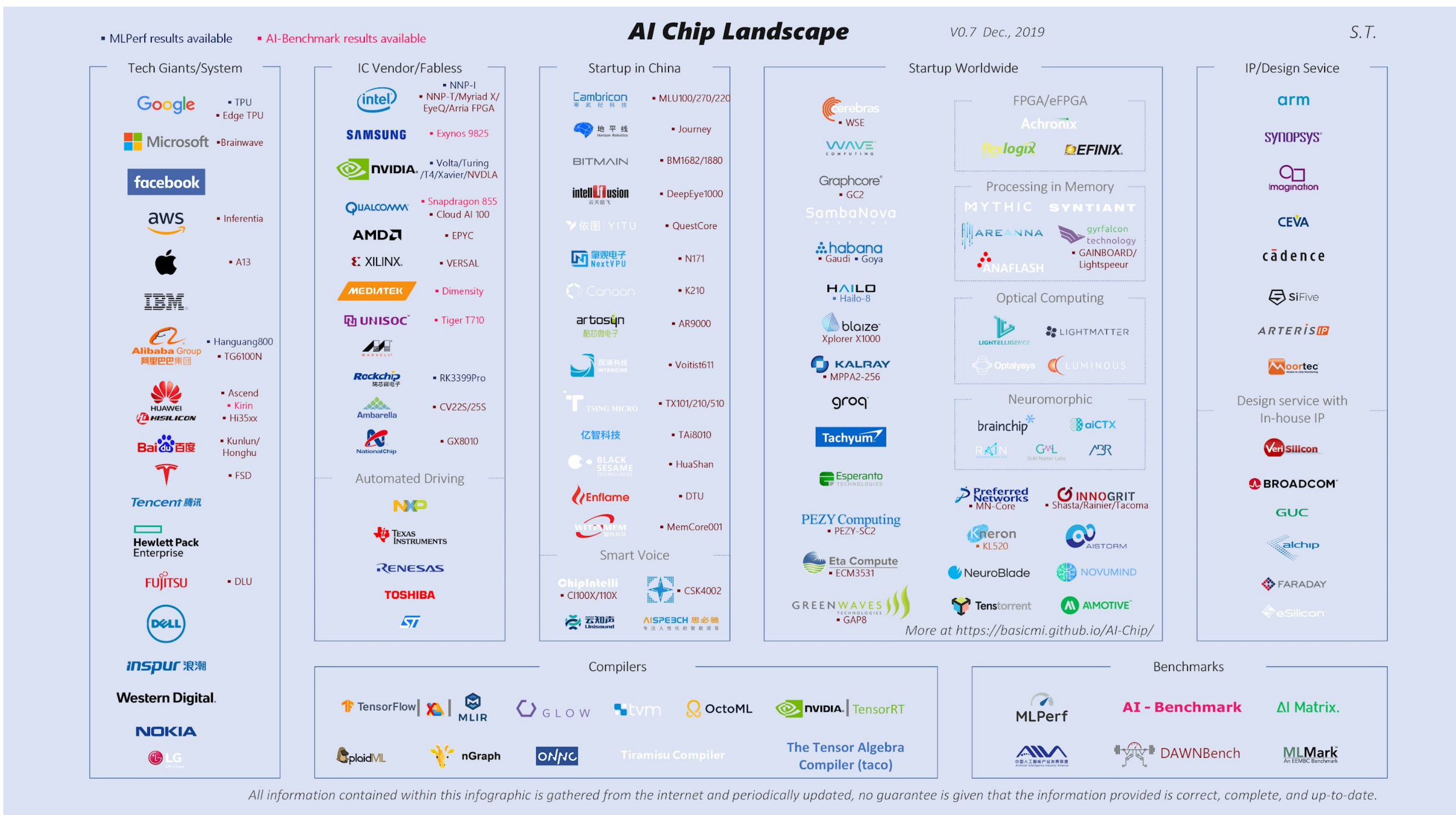
Plateformes matérielles

Ecosystème

- ➔ Guidé par les fabricants
- ➔ Besoins et contraintes de l'application
- ➔ Acteurs de plus en plus nombreux



https://www.eetasia.com/arm-tackles-tinyml-with-new-cores



https://github.com/basicmi/AI-Chip

ARM Cortex-M4/M7

Arduino Nano 33 BLE Sense
➔ Nordic nRF52840 (ARM Cortex-M4)
➔ 256KB SRAM - 1MB Flash
➔ 64MHz

Adafruit EdgeBadge
➔ Microchip SAMD51 (ARM Cortex-M4)
➔ 192KB of SRAM / 512KB of FLASH
➔ 120MHz

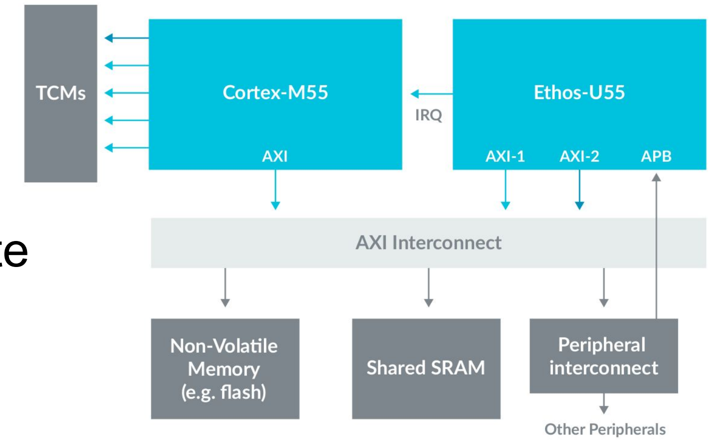
Applications: détection de mots clé, reconnaissance de geste

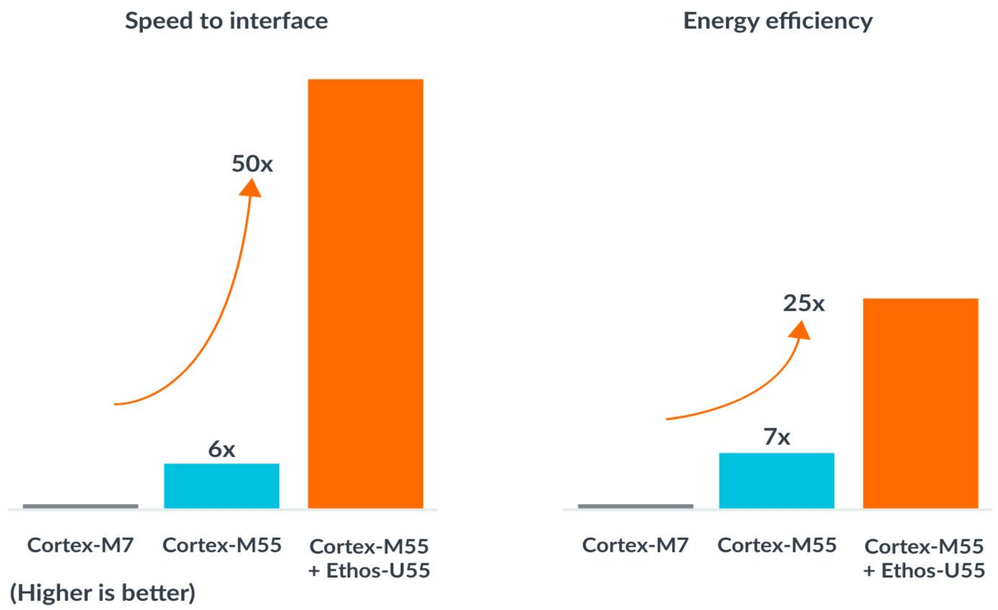
Arduino Portenta:
➔ Dual core STM32 H7: M4 (200MHz) + M7 (480MHz)
➔ 8MB SRAM / 128MB Flash
➔ Principe: microcontrôleur + co-processeur spécialisé

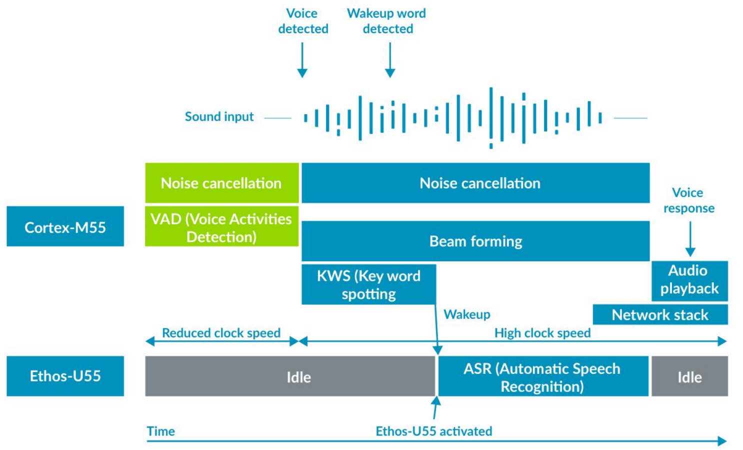
Applications: Détection d’objets, vision par ordinateur, etc

ARM Cortex-M55 + Ethos U55

➔ Microcontrôleur + co-processeur microNPU
➔ Ethos U55: amélioration des performances
➔ Temps de calcul plus faible → consommation réduite



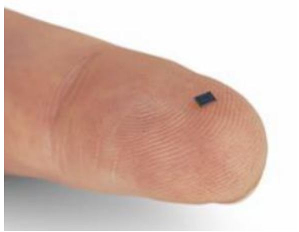


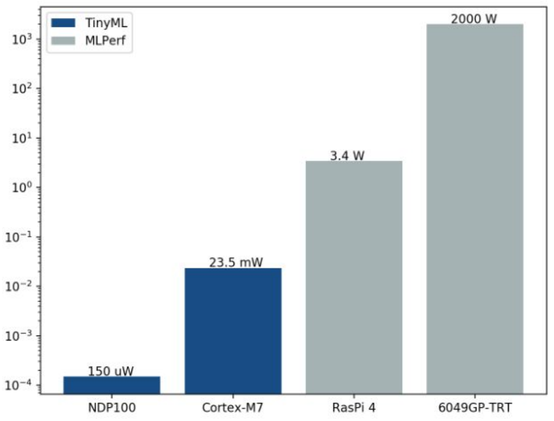


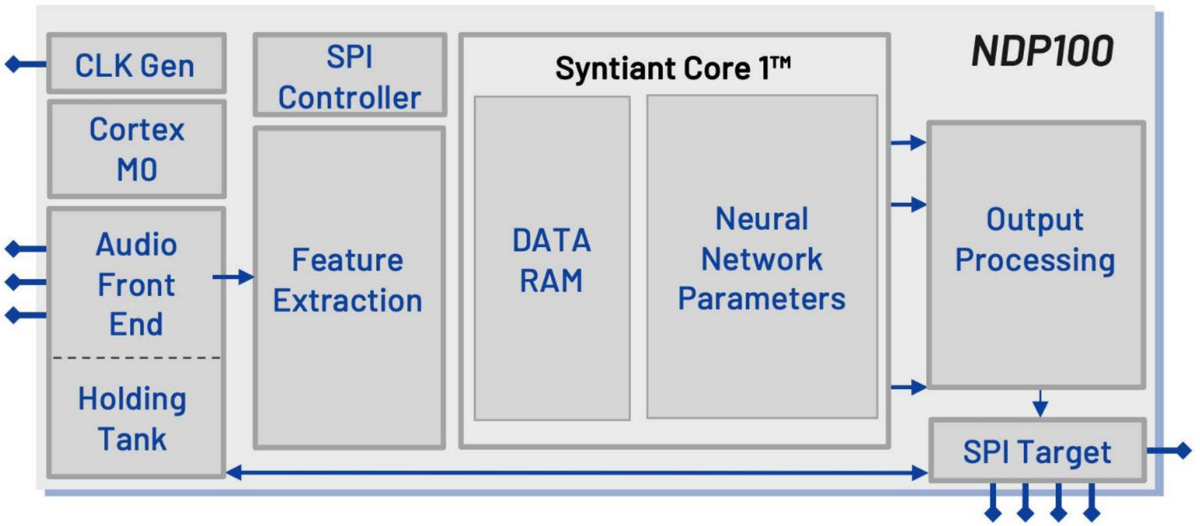
https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/introduction-to-arm-cortex-m55-processor.pdf

Syntiant NDP100

- ➔ ARM Cortex-M0 (112kB RAM) + Syntiant NPU
- ➔ Performance ML améliorée d'un facteur 100x
- ➔ Ultra low-power
- ➔ Applications:
  - ➢ Détection de mots clés
  - ➢ Identification de voix







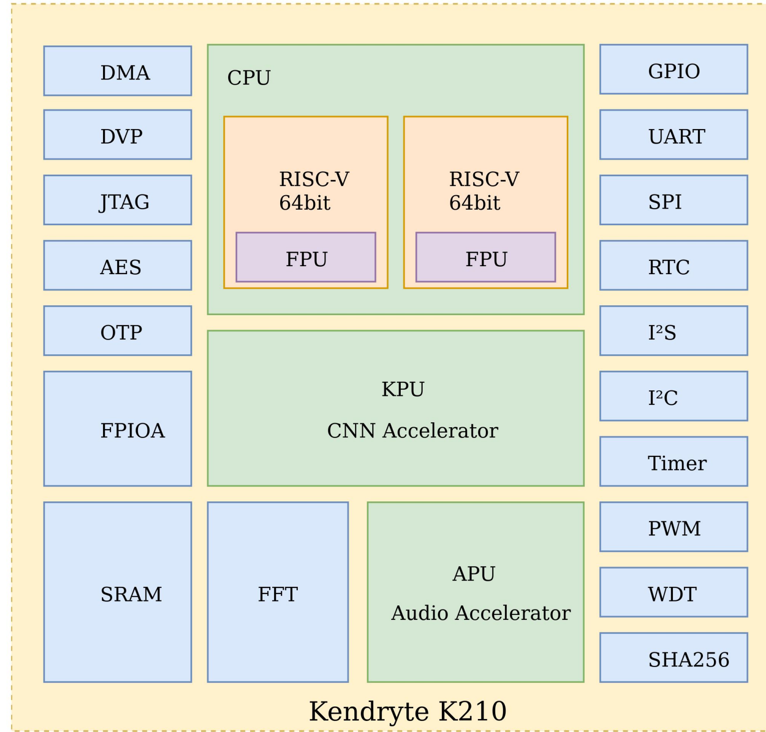
https://aac7932a-14cd-4db8-a195-7d329044d81a.filesusr.com/ugd/799fc8_1c152b8ec97e4abfb038f4bbe41c60ab.pdf

Kendrytes K210

Dual core 64bit RISC-V
- + KPU: Knowledege Processing Unit
- + APU: Audio Processing Unit

➔ 400MHz
➔ 8MB SRAM: peut faire fonctionner Linux
➔ Support MicroPython

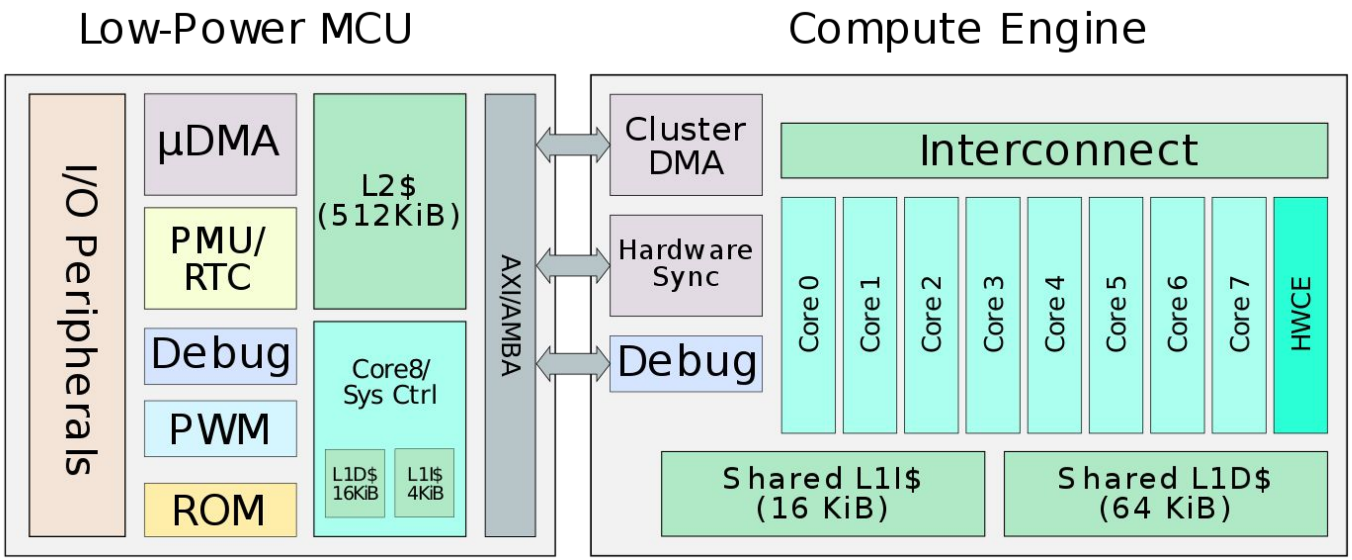
Applications:
➔ Détection d'objet (Yolov2)
➔ Classification d'image
➔ Reconnaissance de visages
➔ Reconnaissance vocales
➔ Détection de mots clés



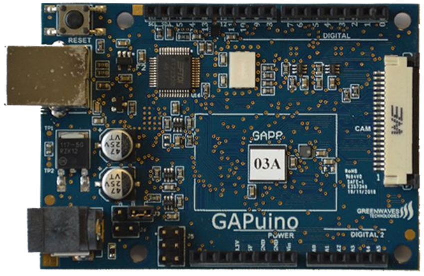

```python
import sensor
import image
import KPU as kpu
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.run(1)
task = kpu.load(0x300000)
anchor = (
    1.889, 2.5245, 2.9465,
    3.94056, 3.99987, 5.3658,
    5.155437, 6.92275, 6.718375, 9.01025
)
a = kpu.init_yolo2(task, 0.5, 0.3, 5, anchor)
while(True):
    img = sensor.snapshot()
    code = kpu.run_yolo2(task, img)
    if code:
        for i in code:
            print(i)
            a = img.draw_rectangle(i.rect())
a = kpu.deinit(task)
```

Greenwaves Gap8 (et maintenant Gap9)

➔ Développés par une société Grenobloise
➔ Multi-coeurs RISC-V
➔ 22.65 GOPS / 4.24 mW/GOP / 512KB RAM / 250MHz
➔ Support FreeRTOS



Applications:
➔ Vision par ordinateur
➔ Reconnaissance vocale
➔ Détection de geste

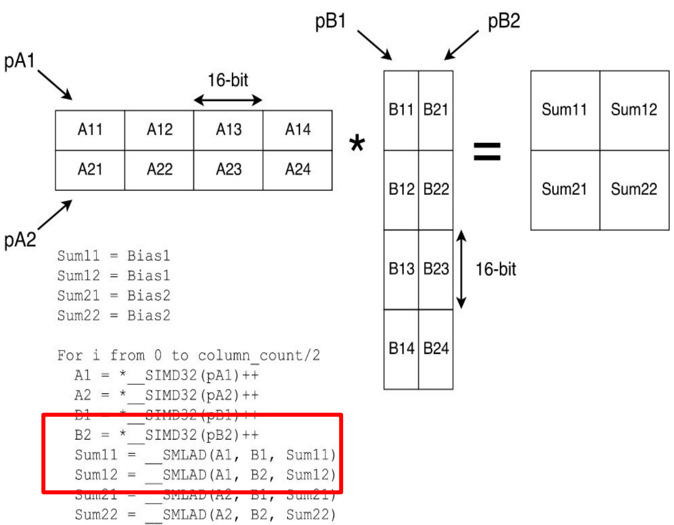


Plateformes logicielles

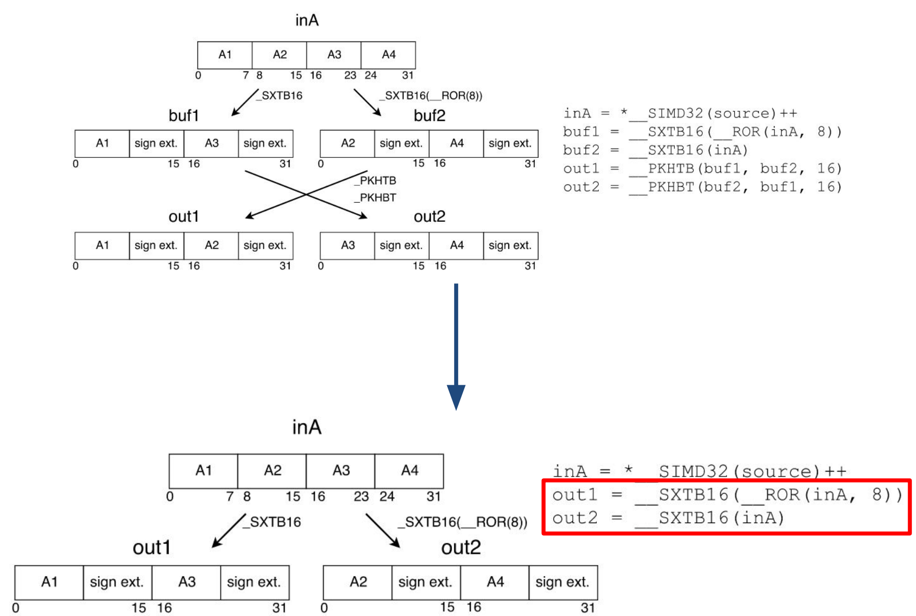
ARM CMSIS-NN

- ➔ Existe depuis 2018
- ➔ Code optimisé pour ARM Cortex-M4+
- ➔ API avec opérateurs pour NN
- ➔ Performances x4

Layer type	Baseline runtime	New kernel runtime	Improvement	
			Throughput	Energy Efficiency
Convolution	443.4 ms	96.4 ms	4.6X	4.9X
Pooling	11.83 ms	2.2 ms	5.4X	5.2X
ReLU	1.06 ms	0.4 ms	2.6X	2.6X
Total	456.4ms	99.1 ms	4.6X	4.9X



Nouvelle instruction "__SMLAD" pour les opérations MAC



Nouvelle instruction "__SXTB16" pour les conversions vers 16bit

- ➔ Implémentation "à la main" des paramètres (poids) et de l'architecture du modèle

L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for Arm Cortex-M CPUs,", arXiv, 2018 - Jan 2018

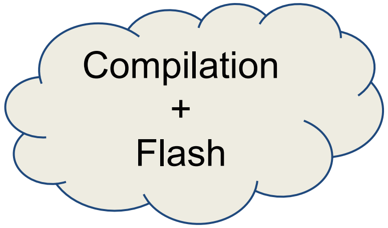
ARM CMSIS-NN: exemple de classification d'image avec CNN

- ➔ Dataset CIFAR10:
  - ◆ 60k images couleur, 32x32
  - ◆ 10 classes
- ➔ Structure du modèle:
  - ◆ 3 couches de convolution avec activation ReLU et max pooling
  - ◆ 1 couche fully-connected

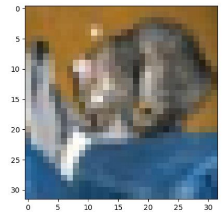
```c
/* conv1 img_buffer2 -> img_buffer1 */
arm_convolve_HWC_q7_RGB(img_buffer2, CONV1_IM_DIM, CONV1_IM_CH, co
                        CONV1_STRIDE, conv1_bias, CONV1_BIAS_LSHIF
                        (q15_t *)col_buffer, NULL);

arm_relu_q7(img_buffer1, CONV1_OUT_DIM * CONV1_OUT_DIM * CONV1_OUT

/* pool1 img_buffer1 -> img_buffer2 */
arm_maxpool_q7_HWC(img_buffer1, CONV1_OUT_DIM, CONV1_OUT_CH, POOL1
                   POOL1_PADDING, POOL1_STRIDE, POOL1_OUT_DIM, NUL

/* conv2 img_buffer2 -> img_buffer1 */
arm_convolve_HWC_q7_fast(img_buffer2, CONV2_IM_DIM, CONV2_IM_CH, c
                         CONV2_PADDING, CONV2_STRIDE, conv2_bias,
                         CONV2_OUT_DIM, (q15_t *)col_buffer, NULL)

arm_relu_q7(img_buffer1, CONV2_OUT_DIM * CONV2_OUT_DIM * CONV2_OUT

/* pool2 img_buffer1 -> img_buffer2 */
arm_maxpool_q7_HWC(img_buffer1, CONV2_OUT_DIM, CONV2_OUT_CH, POOL2
                   POOL2_PADDING, POOL2_STRIDE, POOL2_OUT_DIM, col

* conv3 img_buffer2 -> img_buffer1 */
arm_convolve_HWC_q7_fast(img_buffer2, CONV3_IM_DIM, CONV3_IM_CH, c
                         CONV3_PADDING, CONV3_STRIDE, conv3_bias,
                         CONV3_OUT_DIM, (q15_t *)col_buffer, NULL)

arm_relu_q7(img_buffer1, CONV3_OUT_DIM * CONV3_OUT_DIM * CONV3_OUT

/* pool3 img_buffer-> img_buffer2 */
arm_maxpool_q7_HWC(img_buffer1, CONV3_OUT_DIM, CONV3_OUT_CH, POOL3
                   POOL3_PADDING, POOL3_STRIDE, POOL3_OUT_DIM, col

arm_fully_connected_q7_opt(img_buffer2, ip1_wt, IP1_DIM, IP1_OUT,
                           output_data, (q15_t *)img_buffer1);

arm_softmax_q7(output_data, CLASSES_NUMOF, output_data);

int val = -1;
uint8_t class_idx = 0;
for (unsigned i = 0; i < CLASSES_NUMOF; i++) {
    if (output_data[i] > val) {
        val = output_data[i];
        class_idx = i;
    }
}

if (val > 0) {
    printf("Predicted class: %s\n", classes[class_idx]);
}
```

tableaux de paramètres

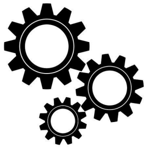
```c
#define CONV1_WT {
    -9, -1, 2, 6, -4, 6, 4, -11, 8, -9, -11, 10, -
    -18, #define CONV1_BIAS {
    34, -    -49, -18, -7, -20, -12, -15, 7, 2, -10, -84,
             -28, -28, -4, -3, -10, -52, -15, -5, -7, -31,
```



Input



Runtime

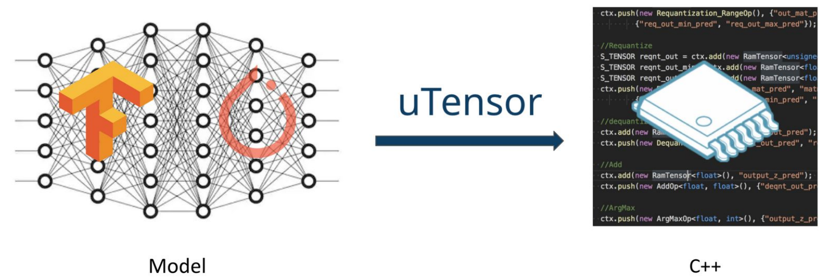


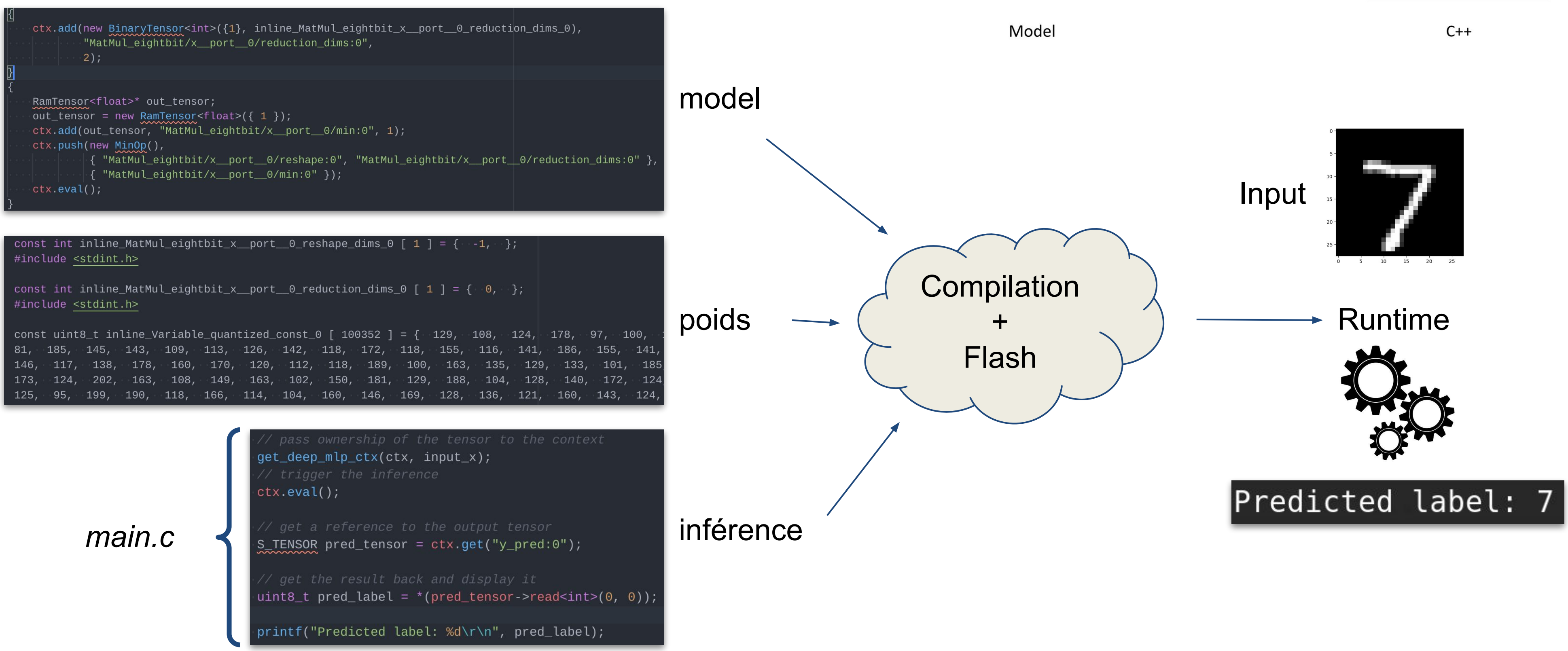
```
Predicted class: cat
```

https://github.com/ARM-software/ML-examples/tree/master/cmsisnn-cifar10

uTensor

- ➔ Moteur d’inférence pour models TensorFlow
- ➔ Générateur de classes C++ en ligne de commande (utensor_cli)
- ➔ Modèle compilé en dur dans le firmware
- ➔ https://utensor.ai
- ➔ Github: https://github.com/uTensor/uTensor

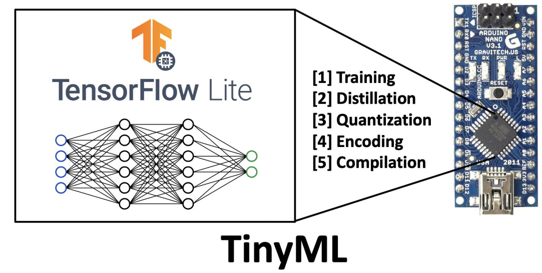


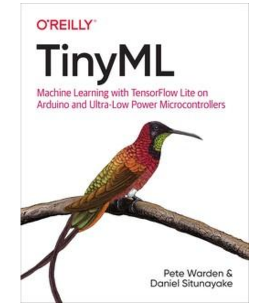


- ➔ Intégration dans RIOT: https://github.com/RIOT-OS/RIOT/tree/master/tests/pkg_utensor

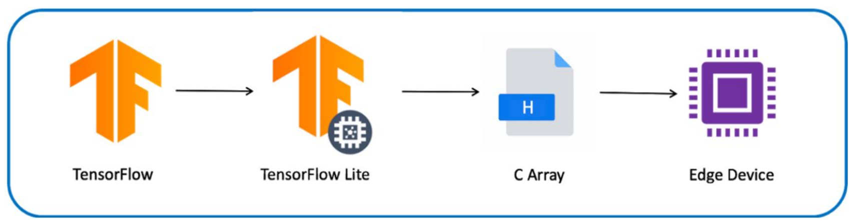
TensorFlow Lite

- ➔ TensorFlow adapté au microcontrôleurs
- ➔ Langage C++
- ➔ Optimisation CMSIS-NN pour MCUs ARM
- ➔ Intégration Arduino, RIOT, ARM mbed, etc





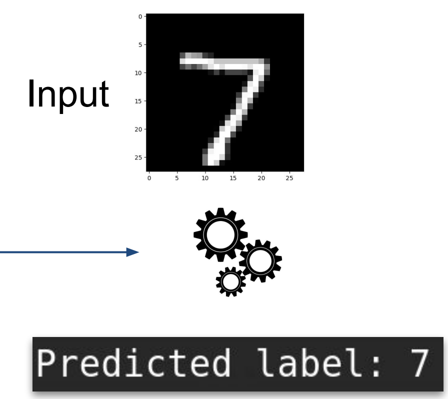
- ➔ Modèle sérialisé au format FlatBuffer (tableau d'octets)



- ➔ Utilisation d'un interpréteur au runtime:

```cpp
// Explicitly load required operators
static tflite::MicroMutableOpResolver micro_mutable_op_resolver;
micro_mutable_op_resolver.AddBuiltin(
    tflite::BuiltinOperator_FULLY_CONNECTED,
    tflite::ops::micro::Register_FULLY_CONNECTED(), 1, 4);
micro_mutable_op_resolver.AddBuiltin(
    tflite::BuiltinOperator_SOFTMAX,
    tflite::ops::micro::Register_SOFTMAX(), 1, 2);
micro_mutable_op_resolver.AddBuiltin(
    tflite::BuiltinOperator_QUANTIZE,
    tflite::ops::micro::Register_QUANTIZE());
micro_mutable_op_resolver.AddBuiltin(
    tflite::BuiltinOperator_DEQUANTIZE,
    tflite::ops::micro::Register_DEQUANTIZE(), 1, 2);

// Build an interpreter to run the model with.
static tflite::MicroInterpreter static_interpreter(
    model, micro_mutable_op_resolver, tensor_arena, kTensorArenaSize, error_reporter);
interpreter = &static_interpreter;

// Allocate memory from the tensor_arena for the model's tensors.
TfLiteStatus allocate_status = interpreter->AllocateTensors();
if (allocate_status != kTfLiteOk) {
    puts("AllocateTensors() failed");
    return;
}
```

```cpp
// Obtain pointers to the model's input and output tensors.
input = interpreter->input(0);
output = interpreter->output(0);

// Copy digit array in input tensor
for (unsigned i = 0; i < digit_len; ++i) {
    input->data.f[i] = static_cast<float>(digit[i]) / 255.0;
}

// Run inference, and report any error
TfLiteStatus invoke_status = interpreter->Invoke();
if (invoke_status != kTfLiteOk) {
    puts("Invoke failed");
    return;
}
```



https://www.tensorflow.org/lite/microcontrollers

Autres plateformes logicielles

➔ emlearn: https://github.com/emlearn/emlearn
  - Langage C
  - Support modèles Scikit-Learn (Decision Tree, MLP)
  - Intégration dans RIOT:
    https://github.com/RIOT-OS/RIOT/tree/master/tests/pkg_emlearn

➔ deepC: https://cainvas.ai-tech.systems
  - Langage C++
  - Intégration Arduino: https://www.arduino.cc/reference/en/libraries/deepc
  - Gallerie d'exemples: https://cainvas.ai-tech.systems/use-cases/tags/tinyml

➔ micromlgen: https://github.com/eloquentarduino/micromlgen
  - Langage C
  - Support modèles Scikit-Learn : DecisionTree, SVM, RandomForest, etc
  - Simple d'utilisation:

```
#include "model.h"

void classify() {
    Serial.print("Predicted class: ");
    Serial.println(classIdxToName(predict(features)));
}
```

Conclusion

- ➔ Ecosystèmes matériels et logiciels hétérogènes
- ➔ Les grands acteurs sont présents (Google, ARM, etc)
- ➔ Accès aux bases de données d’entraînement ?
- ➔ Comment généraliser un modèle avec des données différentes ?
- ➔ Nouveaux modèles ? Nouvelles architectures ?

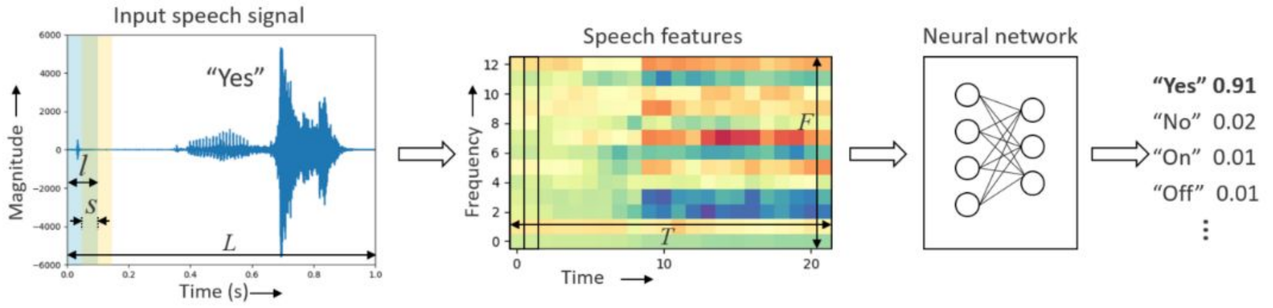
Merci !

Démo: Keyword Spotting and TensorflowLite

Objectif: détecter les mots “yes” et “no”

- “yes”: led verte allumée
- “no”: led rouge allumée
- mot inconnu: led bleue allumée

Arduino Nano BLE Sense
ARM Cortex-M4



https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/micro/examples/micro_speech

Questions ?