TinyML :
IA embarquée sur microcontrôleur

email : kadionik@enseirb-matmeca.fr
web : kadionik.enseirb-matmeca.fr

Patrice KADIONIK
ENSEIRB-MATMECA

HISTORIQUE

- V 1.0 02/25 : Création du document.

CHAPITRE 0 :
INTRODUCTION

CHAPITRE 1 :
QUELQUES REFLEXIONS SUR l'IA

UN PEU D'HISTOIRE…

- Alan Turing (1912 - 1954), père de l'ordinateur, est le père de l'IA moderne (bien après les philosophes grecs).
- En 1950, dans son article « *Computing machinery and intelligence* », il pose la question : « est-ce que les machines peuvent penser ? »
- D'après lui, les IA ne peuvent pas avoir de conscience (donc la conscience de sa fin). La seule façon de savoir si un être a une conscience, c'est de pouvoir « rentrer » dans son corps.
- De l'extérieur, on a juste une impression de conscience.

IA=*Intelligence Artificielle*

UN PEU D'HISTOIRE…

- On demande à une « IA » de mimer et non d'être humain. On se contente d'un simulacre d'intelligence sachant que c'est finalement le résultat qui compte !
- Alan Turing préfère parler de machines apprenantes…
- Alan Turing produit alors le test de Turing qu'il appelle plutôt le jeu de l'imitation.

UN PEU D'HISTOIRE…

- Jeu de l'imitation (test de Turing) :

Le test de Turing est un jeu d'imitation à l'issue duquel une machine pourra être déclarée intelligente ou pas.

Dans un premier temps, un interrogateur doit dialoguer avec une femme et un homme ayant pour consigne de se faire passer pour une femme, puis déterminer lequel des deux est réellement une femme. Pour faire abstraction des apparences, de la voix et du visage, les échanges sont dactylographiés.

Dans un second temps, l'homme est remplacé par une machine à l'insu de l'interrogateur. Désormais, la machine imite l'homme qui imite la femme. Si l'interrogateur est incapable d'identifier la femme, qu'il ne peut dissocier l'homme et la machine, alors on peut considérer que la machine est intelligente.

UN PEU D'HISTOIRE…

- 1956 : conférence internationale de Dartmouth. Le mathématicien Johan McCarthy introduit le terme « intelligence artificielle ».
- 1957 : conception du perceptron. Ambigüité avec le terme réseau de « neurones ».
- 1959 : Arthur Samuel introduit le terme « apprentissage automatique ».

UN PEU D'HISTOIRE…

- 1960 : le mathématicien James Lighthill exprime auprès du gouvernement anglais ses inquiétudes au sujet de la recherche sur la robotique et le traitement de la parole.

- 1960 : le gouvernement anglais cesse alors le financement de l'IA.

- 1960 : premier hiver de l'IA.

UN PEU D'HISTOIRE…

- 1980 : création des systèmes experts : raisonnement à partir de faits et de règles pour en déduire la meilleure décision à prendre avec un moteur d'inférence. On utilise ici un langage de programmation impérative comme Prolog. La programmation est donc déterministe…

- 2000 : engouement pour l'apprentissage automatique via l'apprentissage supervisé.

- 2010 : boom de l'IA grand public avec l'usage des cartes graphiques équipées de GPU.

GPU=*Graphics Processing Unit*

QUELQUES REFLEXIONS…

- L'IA n'est pas un système déterministe mais un système probabiliste.
- L'IA est basée sur des statistiques.
- Cela veut dire que l'IA se trompe (on n'oublie pas la précision de son modèle, au mieux 95-98 %...).
- L'IA manque de transparence. C'est un boite noire un peu « magique ».

QUELQUES REFLEXIONS…

- L'IA a besoin de beaucoup de données pour être entraînée. 3 images de chats suffisent à un enfant pour pouvoir en reconnaître un par la suite. Il en faudra des dizaines de milliers pour qu'une IA fasse la même chose !

- L'IA (grand public) a besoin d'une grande capacité de calcul :
  - Coût d'entraînement de ChatGPT-4 : 100 millions USD.
  - Coût de fonctionnement : 700.000 USD par jour.
  - Coût d'entraînement de DeepSeek V3 : 5,5 millions USD.
  - Pourquoi cette différence de coût ?
    - Modèles d'IA *open source*.
    - Quantification des données sur 8 bits.

Article du Monde « La start-up chinoise DeepSeek bouleverse le secteur de l'intelligence artificielle ». 27 janvier 2025 : https://www.lemonde.fr/economie/article/2025/01/27/la-start-up-chinoise-deepseek-cree-une-onde-de-choc-sur-le-secteur-de-l-ia_6518928_3234.html

QUELQUES REFLEXIONS…

- Face à un humain, l'IA a ses limites. L'IA est limitée à un domaine, l'homme sait faire plus d'une chose et expliquer son raisonnement (et ne coûte pas 700.000 USD par jour ;-)).

- Il y a des dizaines de millions des personnes de l'ombre qui annotent les données (pour l'IA supervisée).

- Quoi faire quand il n'y aura plus de données réelles ?

QUELQUES REFLEXIONS…

- Enfin, l'IA grand public a un impact environnemental croissant de moins en moins négligeable :
  - Consommation d'électricité pour le refroidissement des *data centers*.
  - Consommation de l'eau pour le refroidissement des *data centers*.
  - Augmentation du CO2 (fabrication, fonctionnement, recyclage).
  - Augmentation des déchets électroniques.
- La sobriété doit être une obsession.
- Une solution moins pire : l'IA embarquée ?

CHAPITRE 2 :
LES SYSTEMES EMBARQUES : QUELLE APPROCHE POUR LEUR CONCEPTION ?

QUELLE APPROCHE ?

- Pour concevoir son système embarqué, on peut utiliser la :
  - Logique câblée : conception purement matérielle d'une fonctionnalité spécifique.
  - Logique programmée : conception purement logicielle d'une fonctionnalité (qui s'appuie sur une base matérielle standard).
- Le *codesign* est une combinaison maîtrisée d'un mélange savant de logique câblée et de logique programmée.
- Mais il y a une troisième voie…

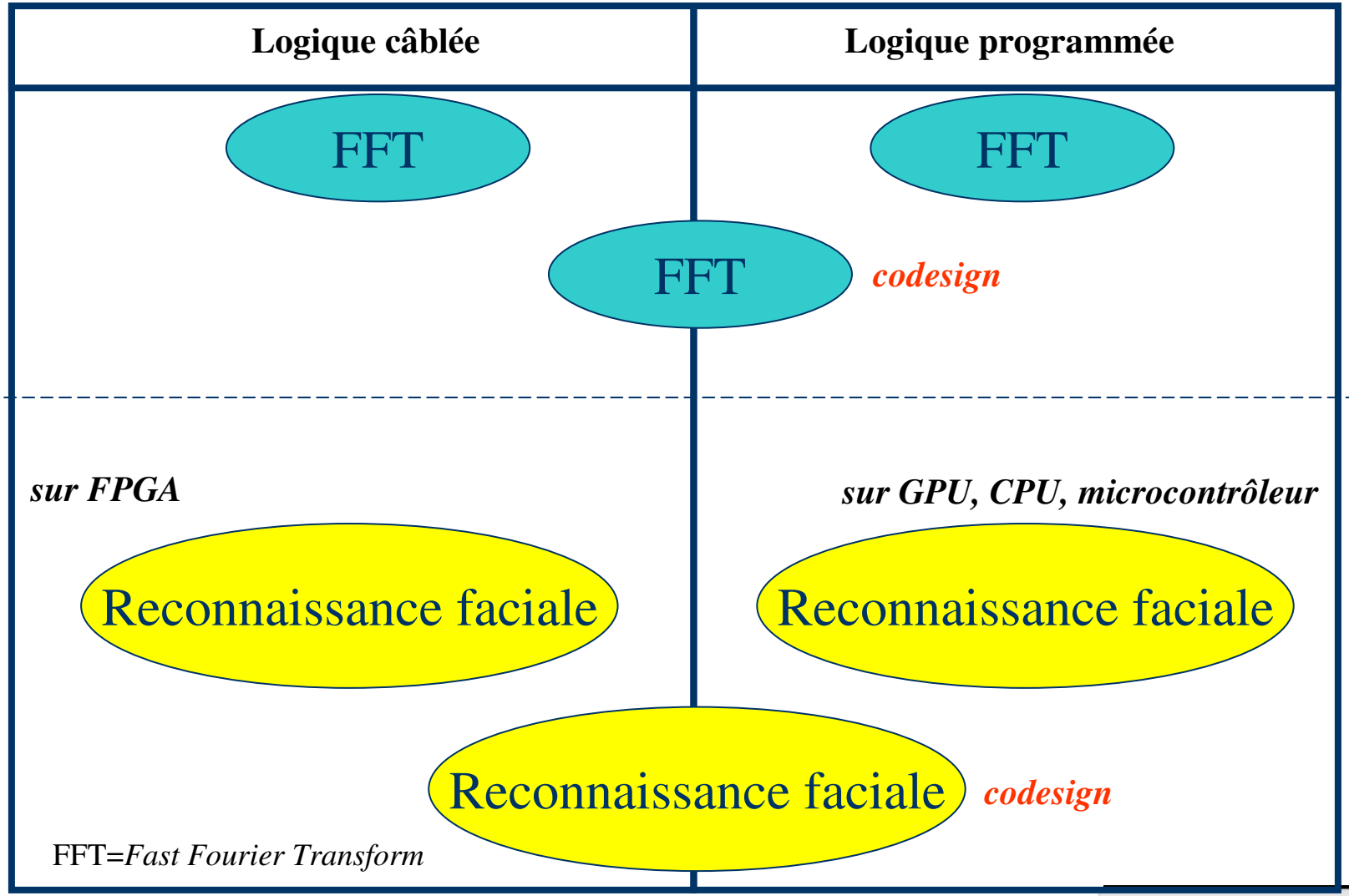
QUELLE APPROCHE ?

- La logique par apprentissage !
- La logique par apprentissage inclut le *Machine Learning* et le *Deep Learning*.
- La logique par apprentissage peut reposer sur une logique programmée et/ou sur une logique câblée.

QUELLE APPROCHE ?

- Le choix de la « bonne » approche parmi les 3 possibles dépendra des contraintes que doit respecter la fonctionnalité du système embarqué comme :
  - Performances.
  - Consommation.
  - Temps Réel.
  - Coûts.
  - Système déterministe ou système probabiliste.
  - …

QUELLE APPROCHE ?



QUELLE APPROCHE ?

- L'IA peut implémenter finalement n'importe quel algorithme.
- On pourrait même remplacer tout le traitement du signal et de l'image par de l'IA !

QUELLE APPROCHE ?

- Mais il y a des limites avec l'IA :
  - On a une approche statistique et non déterministe.
  - On atteint au mieux une précision de 98-99 %. On fait quoi du 1 % restant quand l'IA se plante dans sa prédiction ?
  - Que se passe-t-il en cas de besoin de respecter des contraintes Temps Réel dures ?
  - Qu'en est-il de l'usage de l'IA quand la sécurité des personnes est critique ?
  - Qu'en est-il de l'empreinte mémoire et de la puissance de calcul nécessaires ?
  - Qu'en est-il de la consommation électrique ?

QUELLE APPROCHE ?

- Toutes ces questions sont primordiales quand on fait de l'embarqué.
- Il faudra donc voir comment on peut utiliser de l'IA dans l'embarqué.
- Cela n'a rien à voir avec l'IA « classique »…

CHAPITRE 3 :
LE PROJET TINYML OU
L'IA EMBARQUEE SUR MICROCONTROLEUR

TINYML : UNE DEFINITION

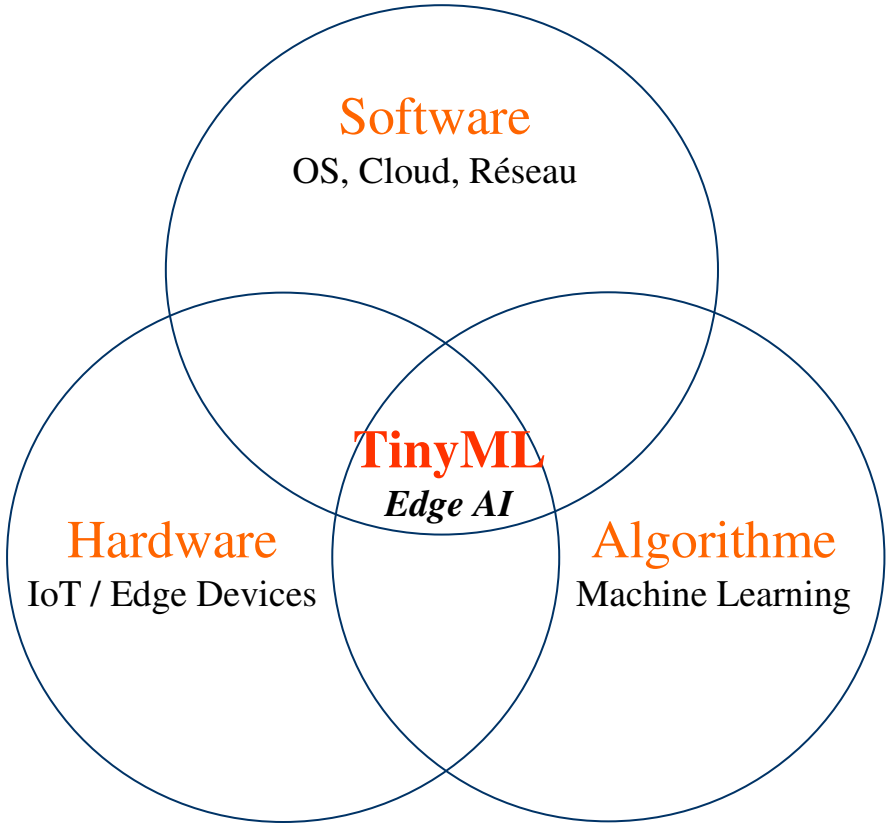
- TinyML est un domaine de l'apprentissage automatique ou *Machine Learning* (ML) axé sur le développement de modèles pouvant être exécutés dans des systèmes embarqués à faible consommation.

- TinyML est donc un projet d'IA embarquée.

- D'après P.Warden [1], ingénieur chez Google et développeur de *Tensorflow Lite*, TinyML a été conçu pour des des équipements qui exécuteront une IA consommant une puissance électrique autour du mW !!!

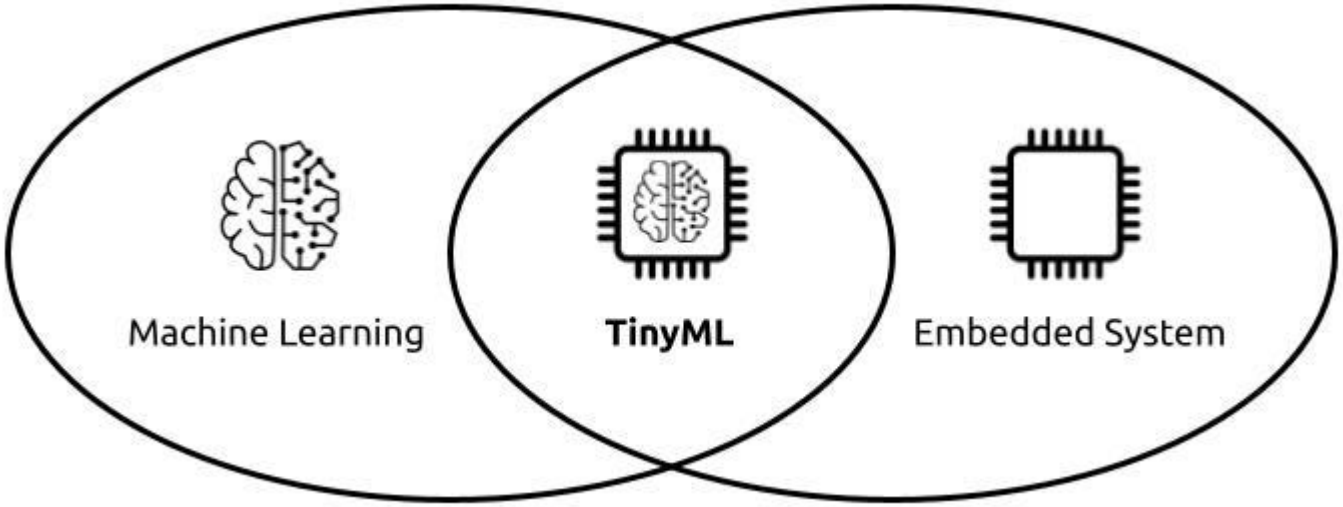
ML=*Machine Learning*

TINYML : UNE DEFINITION



AI=*Artificial Intelligence*

TINYML : UNE DEFINITION



Edge Computing

Source [10]

TINYML : UNE DEFINITION

- Consommation électrique de quelques équipements électroniques :

Equipement	Puissance consommée	Facteur
Carte GPU de PC	500 W	x 500000
Processeur de PC	100 W	x 100000
Carte GPU Jetson	10 W	x 10000
Carte Raspberry PI	qq centaines de mW	x 100
Carte Arduino	qq mW	x 1

TINYML : UNE DEFINITION

- On va donc s'orienter vers un microcontrôleur :
  - Popularité et disponibilité, usage massif dans l'IoT.
    - En 2018 (d'après ICD), 28,1 milliards de microcontrôleurs ont été vendus pour 67,2 millions de PC.
  - Facilité de programmation.
  - Facilité d'exécution d'un algorithme de ML.

IoT=*Internet of Things*

TINYML : UNE DEFINITION

- Dans les technologies de l'embarqué, le microcontrôleur a des capacités réduites :
  - Fréquence de fonctionnement peu élevée : 1 MHz à 400 MHz.
  - Puissance de calcul limitée.
  - Mémoire vive limitée : 2 Ko à 512 Ko.
  - Mémoire de stockage limitée : 32 Ko à 2 Mo.

TINYML : UNE DEFINITION

- Le microcontrôleur présente aussi des avantages :
  - Faible consommation électrique. On peut donc utiliser une batterie de type pile bouton.
  - Prix très faible. Au plus quelques USD.
  - Accès direct aux capteurs et actionneurs.

- Cela veut dire que le ML doit s'adapter au microcontrôleur d'où le projet TinyML.

TINYML : UNE DEFINITION

- TinyML permet ainsi le déploiement de petits modèles de ML sur des dispositifs embarqués possédant des limitations en puissance de calcul, en mémoire et en consommation électrique.

- L'analyse des données et leur interprétation se font en local par l'IA locale sans transfert des données vers le *cloud* pour un traitement par un IA déportée puis récupération du résultat depuis le *cloud*.

- Il n'y a donc pas d'*offloading*.

TINYML : UNE DEFINITION

- Les dispositifs embarqués évaluent les données issues des capteurs en Temps Réel et in situ sans utiliser de ressources externes.

- D'après [3], avec une carte Arduino BLE Sense, dans une application IoT :
  - 65 % de la consommation électrique est pour la transmission sans fil BLE.
  - 21 % de la consommation électrique est pour la lecture des capteurs.
  - 14 % de la consommation électrique est pour le calcul.

- Il vaut donc mieux **moins** transmettre mais calculer un peu **plus**.

TINYML : UNE DEFINITION

- Avec cette approche à rebours de l'*offloading*, on réduit :
  - Les coûts.
  - La consommation d'énergie.

- Avec cette approche à rebours de l'*offloading*, on augmente :
  - La protection des données. Elles ne sont pas transmises sur Internet.

CARACTERISTIQUES

- Faible latence :
  - L'inférence de ML s'exécute directement sur le dispositif embarqué.
  - Pas d'envoi de données sur le *cloud*.
  - Traitement local.

- Faible consommation électrique :
  - Dispositif embarqué à faible consommation électrique donc usage de batterie compacte ou de techniques de récupération d'énergie (*energy harvesting*).
  - Optimisation des ressources *hardware* et *software*.

CARACTERISTIQUES

- Faible bande passante :
  - Les données collectées n'ont pas besoin d'être envoyées à un service externe.
  - La communication vers l'extérieur généralement par une connexion sans fil est moins utilisée.

- Facteur d'échelle :
  - On peut facilement déployer en quantité des dispositifs TinyML notamment grâce au faible prix de revient et à l'autonomie des dispositifs embarqués utilisés.

CARACTERISTIQUES

- Confidentialité :
  - Les données utilisées dans les modèles TinyML sont collectées et analysées in situ en temps réel.
  - Les données ne sont pas envoyées ou partagées vers des services externes.
  - Sécurité des données accrue et vie privée mieux préservée.
  - Réduction du risque d'interception lors d'une cyber attaque.

APPLICATIONS

- *Smart environment* :
  - Surveillance de la qualité l'air.
  - Surveillance de la pollution.
  - Surveillance des départs de feux de forêt.
  - Monitoring de la neige.
  - Prévision des catastrophes naturelles (inondations…).
  - Surveillance des mers : suivi des baleines.

APPLICATIONS

- *Smart agriculture* :
  - Aide à la réduction de la quantité de fertilisants et de produits chimiques.
  - Détection des invasions d'insectes ravageurs dans les cultures par écoute des bruits.
  - Détection de maladies par analyse d'images.

APPLICATIONS

- *Smart health* :
  - Prédiction avant accident d'une crises cardiaque, fibrillation ou crise d'épilepsie.
  - Diagnostic médical : bracelet et montre pour la mesure des constantes (rythme cardiaque, tension artérielle).
  - Détection de chute.

APPLICATIONS

- *Smart appliances* :
  - Contrôle de la TV par gestes et voix.
  - Détection d'odeurs dans le réfrigérateur (produits pourris).

- *Smart homes* :
  - Thermostats, mesure de consommation…
  - Contrôle de l'air conditionné dans les pièces où il y a des personnes.
  - Assistant vocal commandé par la parole.

APPLICATIONS

- *Smart cities* :
  - Reconnaissance faciale (vie privée ?).
  - Surveillance de la structure des bâtiments.
  - Gestion du trafic auto.

- *Smart industry* :
  - Maintenance prédictive pour l'anticipation des pannes afin d'éviter un arrêt de la production.

- *Défense* :
  - Détection d'intrusion, détection d'objets, drones.

CHALLENGE AVEC TINYML

- La mise en œuvre de TinyML apporte son lot de difficultés dans sa mise en œuvre sur les 3 principaux aspects suivants :
  - Les ressources sont limitées : petits systèmes embarqués avec mémoire limitée, puissance de calcul limitée et stockage limité.
  - Taille du modèle : le modèle d'IA doit tenir dans le système embarqué. Il faudra mettre en œuvre des techniques d'optimisation : quantification (réduction de la précision), *pruning* (suppression des poids du modèle non nécessaires).
  - Efficacité énergétique : les équipements sont alimentés par batterie et la consommation électrique doit être optimisée : minimisation du *processing* et maximisation des performances du modèle d'IA.

QUELS LOGICIELS POUR TINYML ?

- TinyML vise les petits systèmes embarqués.
- On peut aussi faire la distinction entre ceux qui peuvent mettre en œuvre un système d'exploitation (comme Linux) et ceux qui ne le peuvent pas.
- On fait donc une distinction entre microcontrôleur et autres processeurs :
  - Microcontrôleur : pas de MMU, qq centaines de Ko de RAM, 1 Go de mémoire Flash embarquée.
  - Processeurs : MMU, qq Go de RAM, mémoire Flash externe sur carte SD.

MMU=*Memory Management Unit*

QUELS LOGICIELS POUR TINYML ?

- Différents logiciels sont disponibles pour TinyML.

- On peut les classer comme suit :
  - Langage de programmation : C/C++, Python (1991), R (1993), Java (1995).
  - Editeurs de texte, IDE.
  - Compilateurs croisés.
  - Canevas de programmation ou *frameworks*.

IDE=*Integrated Design Entry*

QUELS LOGICIELS POUR TINYML ?

- Langage de programmation :
  - Le langage Python est utilisé à 80 % pour le *Machine Learning*.
  - Le langage R l'est aussi…

- Editeurs de texte, IDE :
  - *Spyder*.
  - *Jupyter Notebook* avec sauvegarde dans le source des résultats d'exécution (cahier de TP).
  - *Google Colaboratory* avec sauvegarde dans le source des résultats d'exécution (cahier de TP).

QUELS LOGICIELS POUR TINYML ?

- Compilateurs croisés :
  - Ils sont plutôt utilisés indirectement à travers l'IDE.
- Canevas de programmation ou *frameworks* :
  - Cela correspond à une bibliothèque de programmation pour pouvoir faciliter le développement d'applications TinyML.
  - Pour information, un *framework* peut aussi proposer un langage de programmation dédié ou DSL pour faciliter le développement logiciel.

DSL=*Domain Specific Language*

QUELS LOGICIELS POUR TINYML ?

- *Frameworks* :
  - La base : scikit-learn, TensorFlow, Keras, Pytorch.
  - **TensorFlow Lite** de Google (maintenant LiteRT) pour systèmes embarqués ayant un système d'exploitation comme IOS, Android ou Linux.
  - **TensorFlow Lite for Micro** de Google (maintenant LiteRT for Micro) pour systèmes embarqués à base de microcontrôleur sans système d'exploitation.
  - Edge Impulse : site Internet pour construire facilement des applications TinyML.
  - Pytorch Mobile de Meta (Facebook).

QUELS LOGICIELS POUR TINYML ?

- *Frameworks* :
  - AIMET de Qualcomm (*AI Model Efficiency Toolkit*).
  - Caffe (*Convolutional Architecture for Fast Feature Embedding*).
  - CoreML d'Apple.
  - ONNX (*Open Neural Network Exchange*).
  - OpenVINO (*OPEN Visual Inference and Neural Network Optimization*).
  - CMIX-NN : *framework* optimisé pour microcontrôleur ARM Cortex-M.
  - uTensor pour microcontrôleur ARM.
  - STM32Cube.AI et NanoEdge AI Studio de STMicroelectronics.

QUELS MATERIELS POUR TINYML ?

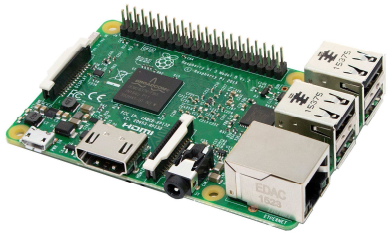
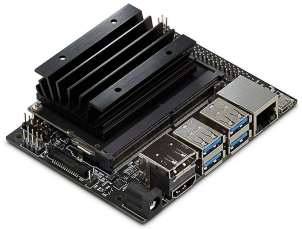
- On peut avoir des architectures matérielles embarquées optimisées :
  - FPGA.
  - ASIC.
  - GPU.
  - TPU : Google Coral Edge TPU.
  - CPU optimisé : Qualcomm Snapdragon, STM32, NXP i.MX RT.

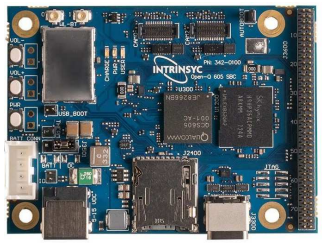
TPU=*Tensor Processing Unit*

QUELS MATERIELS POUR TINYML ?

- Nous allons focaliser sur les matériels utilisables avec TensorFlow Lite et TensorFlow Lite for Micro.

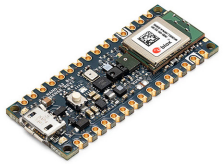
- Matériels performants pour TensorFlow Lite :
  - Carte Raspberry Pi.
  - Carte NVIDIA Jetson Nano.
  - Carte Google Coral Edge TPU.
  - Carte Qualcomm QCS605.



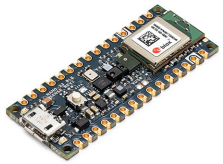
QUELS MATERIELS POUR TINYML ?

- Matériels à base de microcontrôleur plutôt adaptés à TensorFlow Lite for Micro :
  - Carte Arduino BLE Sense.
  - Carte Raspberry Pi Pico.
  - Carte Arduino Nicla Sense ME.
  - Carte Arduino Nicla Vision.
  - Carte Arduino Portenta H7 + VisionShield.
  - Carte Adafruit Feather.
  - Carte SparkFun Edge.
  - Carte STMicroelectronics STM32L4.



QUELS MATERIELS POUR TINYML ?

- Matériels à base de microcontrôleur plutôt adaptés à TensorFlow Lite for Micro :
  - Carte Intel Curie.
  - Carte Syntiant TinyML.
  - Carte Espressif ESP32.
  - Carte Espressif ESP-EYE.
  - Carte Himax WE-I Plus.
  - Carte Open MV Cam H7 Plus.
  - Carte SiLabs xG24 Dev Kit.
  - Carte Sony's Spresense.



QUELS MATERIELS POUR TINYML ?

- Pour les TP de TinyML, la carte Arduino Nano 33 BLE Sense sera mise en œuvre.

- La carte Arduino Nano 33 BLE Sense possède ainsi les éléments suivants :
  - Un SoC (*System on Chip*) Nordic nRF52840 avec un processeur ARM Cortex-M4 à 64 MHz.
  - 256 ko de RAM et 1 Mo de mémoire Flash.
  - 1 port USB.
  - 8 entrées analogiques.
  - 14 E/S GPIO.
  - Bluetooth BLE.

QUELS MATERIELS POUR TINYML ?

- La carte Arduino Nano 33 BLE Sense possède ainsi les éléments suivants :
  - Accéléromètre et gyroscope 3 axes BMI270.
  - Magnétomètre 3 axes BMM150.
  - Microphone MP34DT06JTR.
  - Capteur de geste, de lumière ambiante et de proximité APDS9960.
  - Capteur de pression et baromètre LPS22HB.
  - Capteur de température et d'humidité HS3003.
  - 1 UART, 1 bus I2C et 1 bus SPI.

CHAPITRE 4 :
CANEVAS DE PROGRAMMATION TENSORFLOW LITE

INTRODUCTION

- Quelques dates importantes pour s'y retrouver :
  - 2015 : création de TensorFlow par Google.
  - 2017 : TensorFlow Lite.
  - 2018 : TensorFlow Lite for Micro.
  - 2024 : TensorFlow Lite devient LiteRT.
  - 2024 : TensorFlow Lite for Micro devient LiteRT for Micro.

INTRODUCTION

- TensorFlow Lite est un *framework open source* pour le *Machine Learning* développé par Google.

- Il supporte diverses plateformes embarquées :
  - Android. Avec OS.
  - iOS. Avec OS.
  - Linux embarqué. Avec OS.
  - Microcontrôleurs. Sans OS avec la version TensorFlow Lite for Micro.

OS=*Operating System*

INTRODUCTION

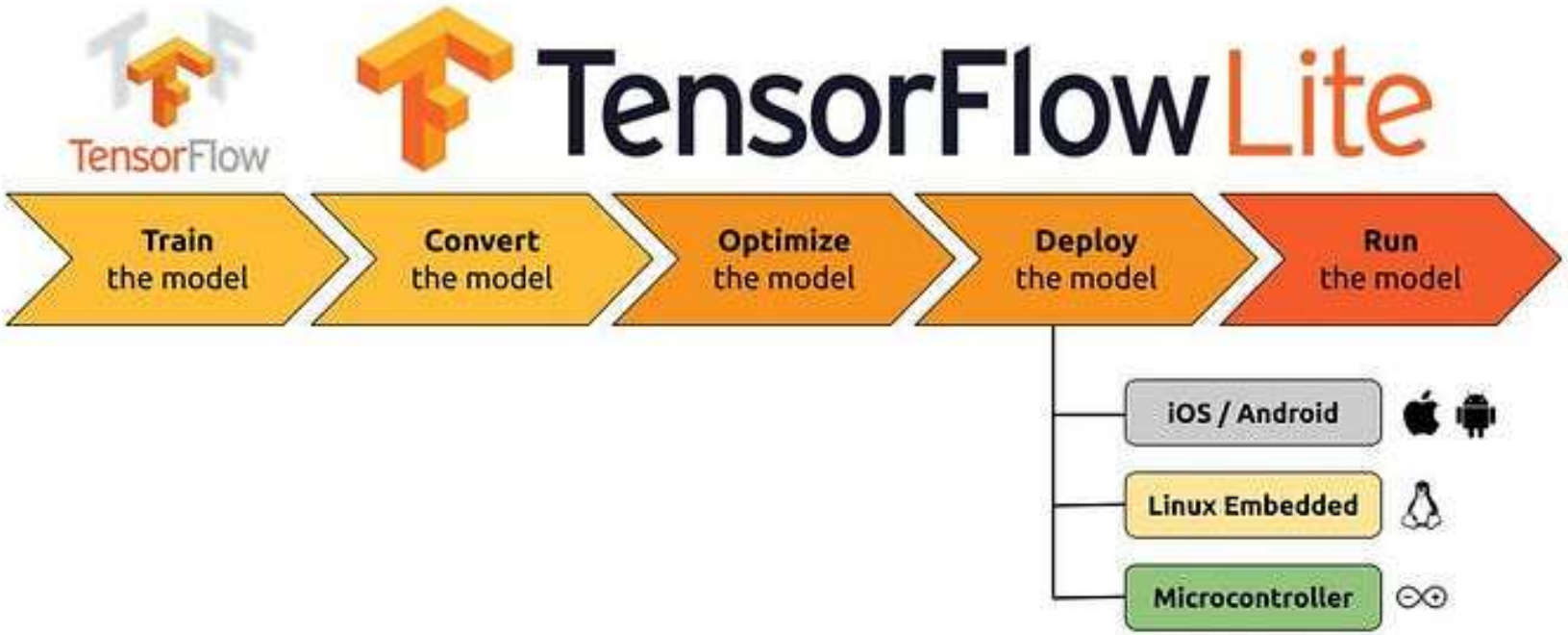
- TensorFlow Lite est une une version optimisée de TensorFlow.
- TensorFlow Lite permet l'exécution de modèles de *Machine Learning* (ML) sur des équipements à ressources limitées.
- Il est conçu pour :
  - Augmenter la rapidité d'exécution du modèle de ML.
  - Réduire l'empreinte mémoire.
  - Réduire la consommation énergétique.

FONCTIONNALITES

- TensorFlow Lite est :
  - Léger : initialisation et démarrage rapide. Faible empreinte mémoire.
  - Multi plateforme : Android, iOS, Linux embarqué, microcontrôleurs.
  - Faible latence : pas besoin d'accès d'envoyer les données vers l'extérieur (*Edge Computing*).

FONCTIONNALITES

- TensorFlow Lite est :
  - Sécurisé : les données personnelles ne sortent pas de l'équipement.
  - Langages de programmation : Python, Java, Swift, Objective-C, C++.
  - Support et aides importants : beaucoup d'exemples sont disponibles.


TensorFlow
TensorFlow Lite
Train the model
Convert the model
Optimize the model
Deploy the model
Run the model
iOS / Android
Linux Embedded
Microcontroller

Source [8]

MISE EN OEUVRE

- On distingue différentes étapes dans la mise en œuvre de TensorFlow Lite :
  - 1. Entraînement d'un modèle TensorFlow TF: création et entraînement d'un modèle classique de ML avec TensorFlow.
  - 2. Conversion et optimisation du modèle TF: le modèle TensorFlow est converti en modèle TensorFlow Lite (TFL). La taille du modèle TFL est réduite et des optimisations (quantification, *pruning*…) sont introduites et qui n'affectent pas ou peu (compromis) la précision du modèle.
  - 3. Déploiement et exécution du modèle TFL: le modèle TFL est déployé sur la cible et exécuté par un interpréteur TFL.

TF=*TensorFLow* TFL=*TensorFLow Lite*

MISE EN OEUVRE

- L'étape 1 correspond à l'entraînement du modèle avec TensorFlow/Keras (vu durant le semestre S8 et approfondi dans le module IA embarquée de l'option SE) .

- L'étape 2 correspond à la conversion et l'optimisation du modèle TF en modèle FTL.

- L'étape 3 correspond au déploiement et l'exécution du modèle TFL (inférence).

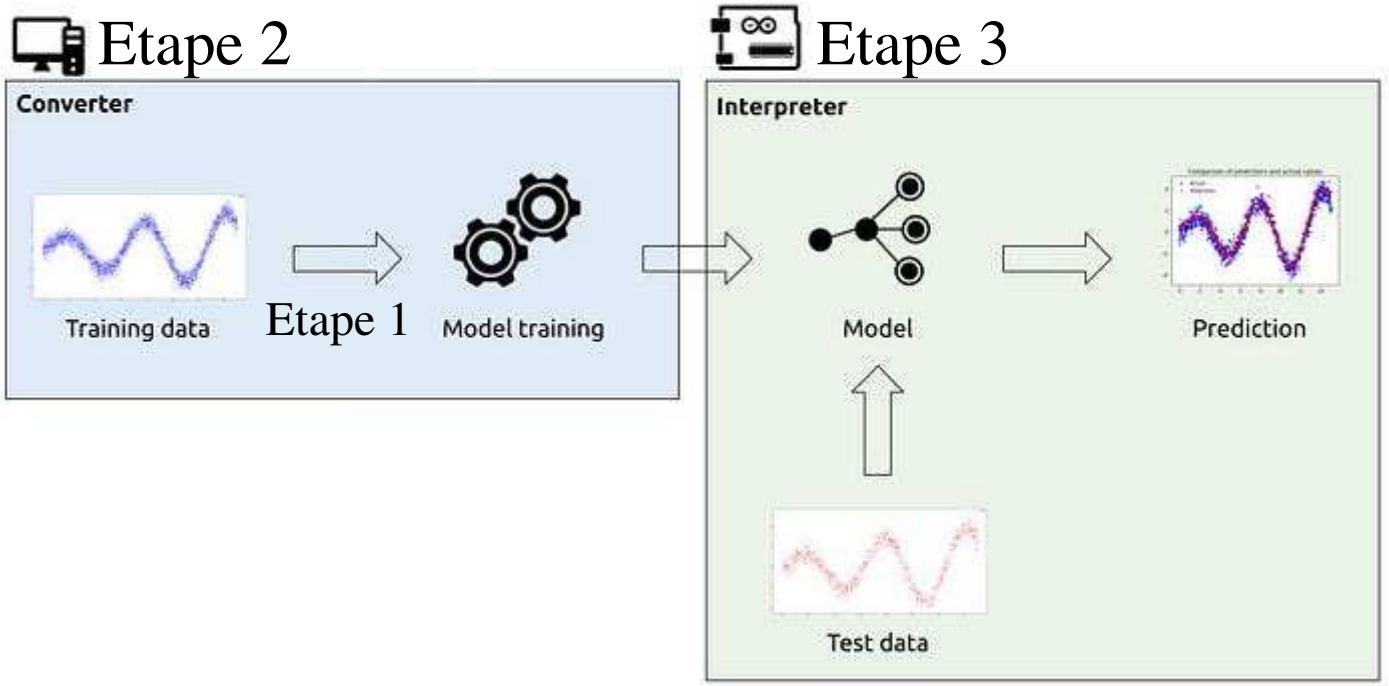
MISE EN OEUVRE

- On va maintenant différencier TensorFlow Lite (TFL) et TensorFlow Lite for Micro (TFLM), chacun étant destiné à des types d'équipements différents :
  - TFL est conçu pour des systèmes embarqués assez puissants et dotés d'un système d'exploitation (Raspberry Pi, *smartphones*, microprocesseurs…).
  - TFLM est adapté aux microcontrôleurs peu puissants sans MMU avec une capacité mémoire vive et non volatile limitée gérée de façon statique ne nécessitant ni OS ni allocation dynamique de mémoire (pas de `malloc()`).

- Seule, l'étape 3 diffère entre TFL et TFLM.

TFLM=*TensorFLow Lite for Micro*

MISE EN OEUVRE



Source [10]

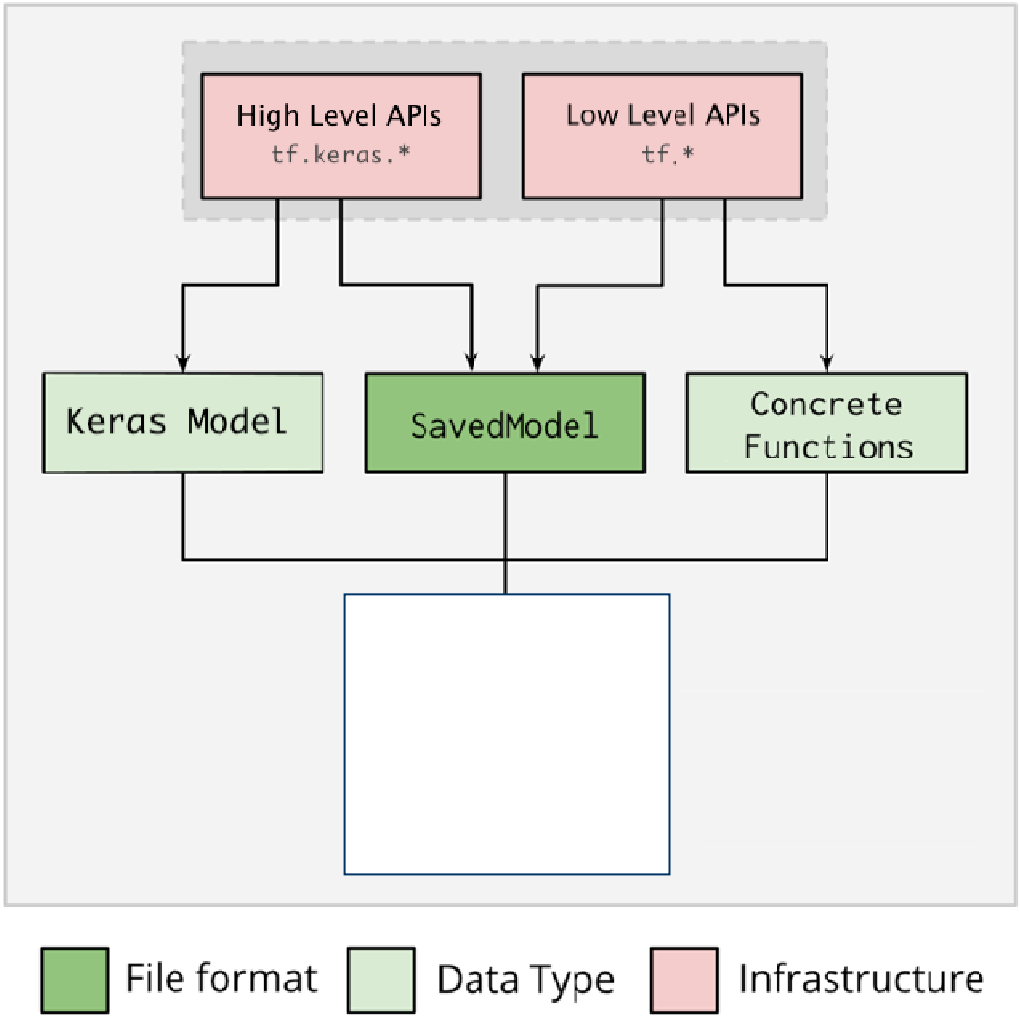
MISE EN ŒUVRE : ETAPE 1 PREPARATION

- Préparation des données :
  - Collecte des données : formation de sa base de données.
  - Nettoyage et normalisation : Les données brutes ont été nettoyées et mises à l'échelle pour améliorer l'apprentissage (mise dans la plage [0, 1] par exemple, distribution gaussienne, normalisation Z-Score).
  - Labellisation : les données sont labellisées si l'on fait un apprentissage supervisé.

MISE EN ŒUVRE : ETAPE 1 ENTRAINEMENT

- Entraînement du modèle TF :
  - Définition du modèle : *Machine Learning*, *Deep Learning* par réseaux de neurones convolutifs.
  - Entraînement avec le *dataset* d'entraînement (80 % du *dataset* initial) : choix d'une fonction de coût et d'un optimiseur.
  - Tests avec le *dataset* de test (10 % du *dataset* initial) .
  - Validation avec le *dataset* de validation (10 % du *dataset* initial) : mesure de la précision et détection du sur apprentissage (*overfitting*) du sous apprentissage (*underfitting*) .

MISE EN ŒUVRE : ETAPE 1



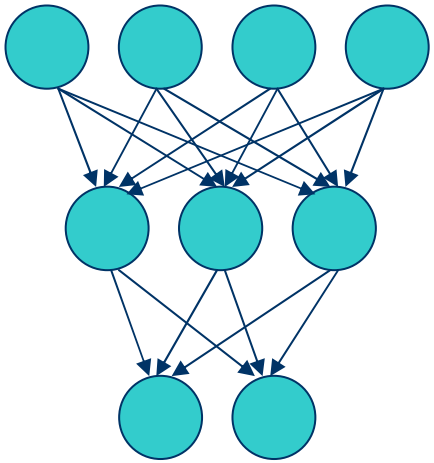
Source [8]

MISE EN ŒUVRE : ETAPE 2

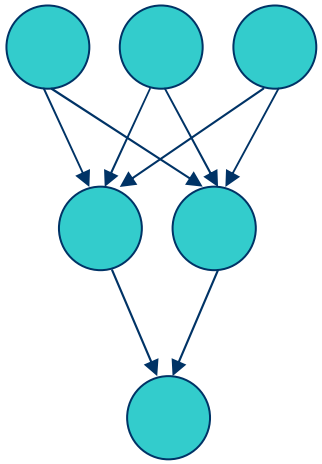
- Conversion et optimisation du modèle TFL :
  - Conversion du modèle TF en modèle TFL au format *FlatBuffer* : le modèle est compressé et enregistré dans un fichier *.tflite*.
  - Lors de la conversion du modèle TF, des techniques d'optimisation peuvent être appliquées :
    - Quantification : réduction de la précision des nombres.
    - *Pruning* : suppression des poids du modèle non nécessaires en DL.

MISE EN ŒUVRE : ETAPE 2 PRUNING

- Des coefficients du modèle TF sont mis à zéro : neutralisation.



Avant *pruning*



Après *pruning*

- La mise en œuvre du *pruning* ne sera pas abordée. Voir [4]

MISE EN ŒUVRE : ETAPE 2 QUANTIFICATION

- La quantification est une stratégie d'optimisation pour réduire la précision du modèle donc sa taille.

- TFL permet la :
  - Quantification post entraînement : *Post-Training Quantization* (PTQ).
  - Quantification avec nouvel entraînement : *Quantization-Aware Training* (QAT).

PTQ=*Post-Training Quantization* QAT=*Quantization-Aware Training*

MISE EN ŒUVRE : ETAPE 2 QUANTIFICATION

- La quantification PTQ est la plus simple.
- Le modèle de base TF utilisant des nombres réels simple précision de 32 bits (*float*) est quantifié pour utiliser généralement des nombres entiers de 8 bits.
- Cela introduit des erreurs (acceptables ?) qui ne sont pas par la suite compensées. Ce sont les erreurs de quantification.
- Il est alors important de contrôler les performances de son modèle TFL après quantification PTQ.

MISE EN ŒUVRE : ETAPE 2 QUANTIFICATION

- La quantification QAT est plus compliquée à mettre en œuvre.

- Après la quantification, le modèle TFL est ré-entraîné sur quelques époques (*epoch*) pour réajuster des coefficients du modèle TFL et compenser l'erreur de quantification.

MISE EN ŒUVRE : ETAPE 2 QUANTIFICATION

- Les techniques de quantification et de *pruning* permettent de réduire la taille du modèle d'un facteur 4 entre le modèle TF et le modèle TFL.

Technique	Data requirements	Size reduction	Accuracy
Post-training float16 quantization	No data	Up to 50%	Insignificant accuracy loss
Post-training dynamic range quantization	No data	Up to 75%	Smallest accuracy loss
Post-training integer quantization	Unlabelled representative sample	Up to 75%	Small accuracy loss
Quantization-aware training	Labelled training data	Up to 75%	Smallest accuracy loss

Source [8]

MISE EN ŒUVRE : ETAPE 2 QUANTIFICATION

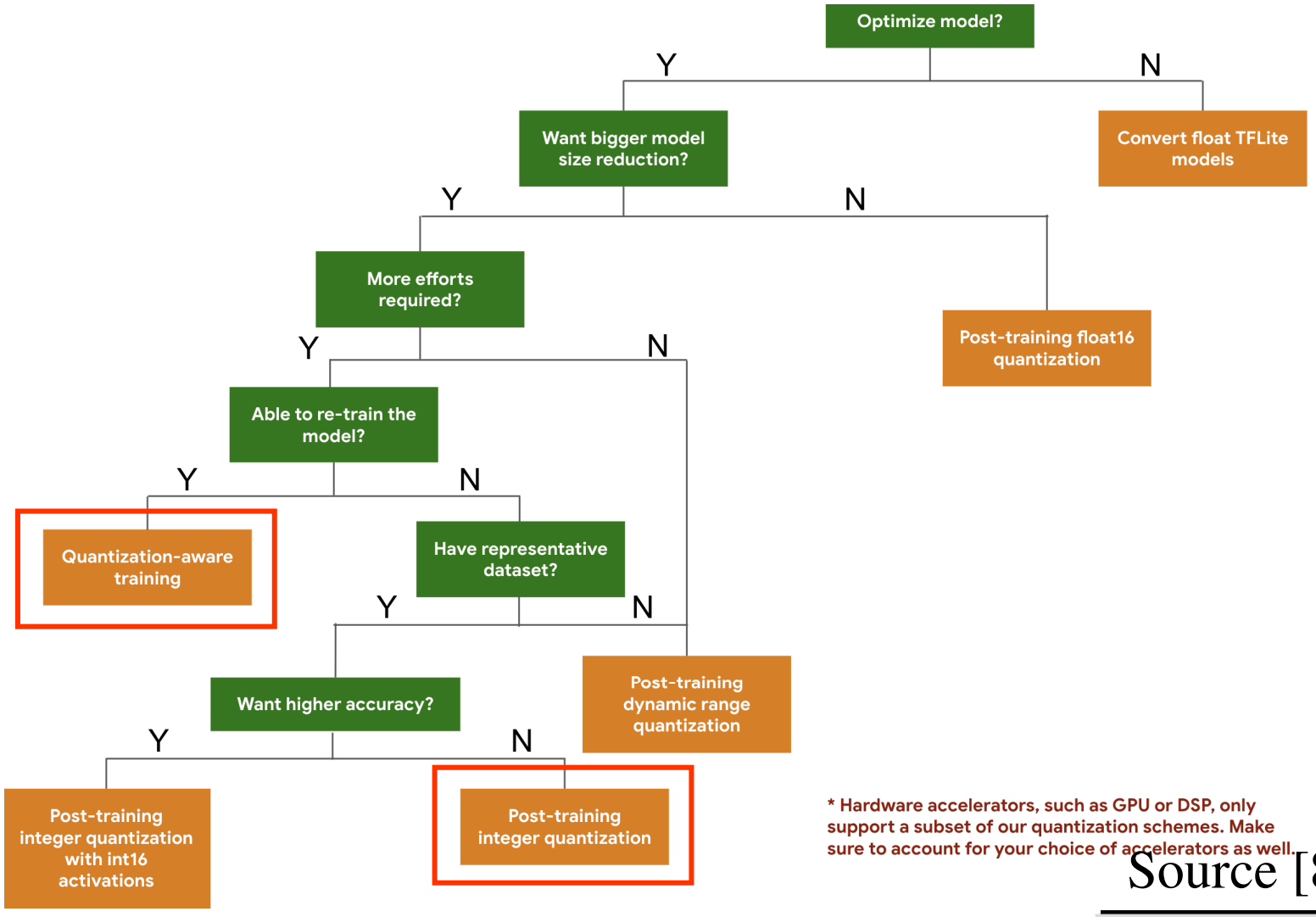
Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)	Size (Original) (MB)	Size (Optimized) (MB)
Mobilenet-v1-1-224	0.709	0.657	0.70	124	112	64	16.9	4.3
Mobilenet-v2-1-224	0.719	0.637	0.709	89	98	54	14	3.6
Inception_v3	0.78	0.772	0.775	1130	845	543	95.7	23.9
Resnet_v2_101	0.770	0.768	N/A	3973	2868	N/A	178.3	44.9

Source [8]

Résultats sur quelques modèles de réseaux de neurones

Diminution de la taille et du temps de latence

MISE EN ŒUVRE : ETAPE 2 QUANTIFICATION



Source [8]

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- L'exécution d'un modèle TFL (fichier *.tflite*) se fait à l'aide d'un interpréteur.

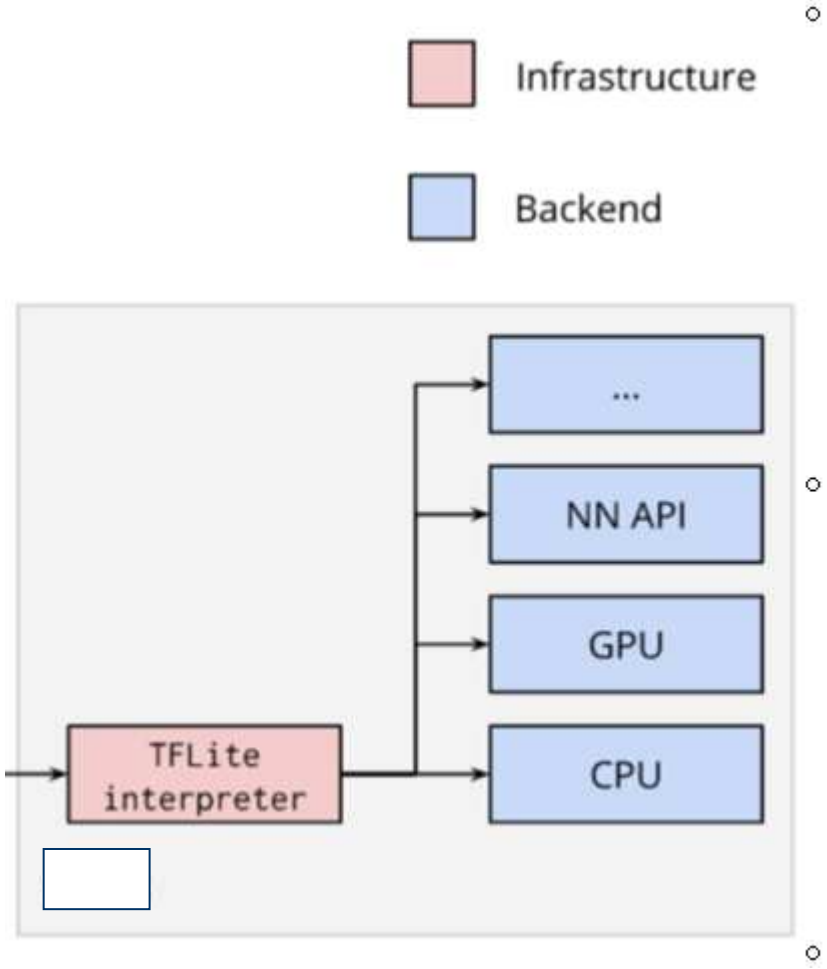
- Les différentes étapes sont :
  - 1. Création d'une instance de la classe `Interpreter`. Il prend comme paramètre d'entrée le fichier *.tflite*.
  - 2. Allocation mémoire pour l'interpréteur avec la méthode `allocate_tensors()`.
  - 3. Récupération de l'entrée et de la sortie de type tenseur avec les méthodes `get_input_details()` et `get_output_details()`.

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- Les différentes étapes suivies sont :
  - 4. Alimentation du tenseur d'entrée avec une donnée d'entrée avec la méthode `set_tensor()`.
  - 5. Invocation de l'interpréteur avec la méthode `invoke()`.
  - 6. Récupération de la valeur du tenseur de sortie.
  - 7. Conversion du tenseur de sortie en valeur prédite.

- La démarche décrite pour TFL sera la même avec TFLM.

MISE EN ŒUVRE : ETAPE 3 EXECUTION



Source [8]

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- Un exemple de code en langage Python est donné ci-après :
- Comme on utilise TFL, on a donc un système embarqué ayant un OS.
- Donc Python s'impose alors avec TFL…

MISE EN ŒUVRE : ETAPE 3 EXECUTION

```
# Etape 1
interpreter = tf.lite.Interpreter('model.tflite')

# Etape 2
interpreter.allocate_tensors()

# Etape 3
input_index = interpreter.get_input_details()[0]["index"]
output_index = interpreter.get_output_details()[0]["index"]

pred_list = []
for images in X_test:
  input_data = np.array(images, dtype=np.float32)

  input_data = input_data.reshape(1, input_data.shape[0], input_data.shape[1],
   1)
```

MISE EN ŒUVRE : ETAPE 3 EXECUTION

```
# Etape 4
  interpreter.set_tensor(input_index, input_data)

# Etape 5
  interpreter.invoke()

# Etape 6
  prediction = interpreter.get_tensor(output_index)

# Etape 7
  prediction = np.argmax(prediction)
  pred_list.append(prediction)
```

MISE EN ŒUVRE : ETAPE 3 EXECUTION

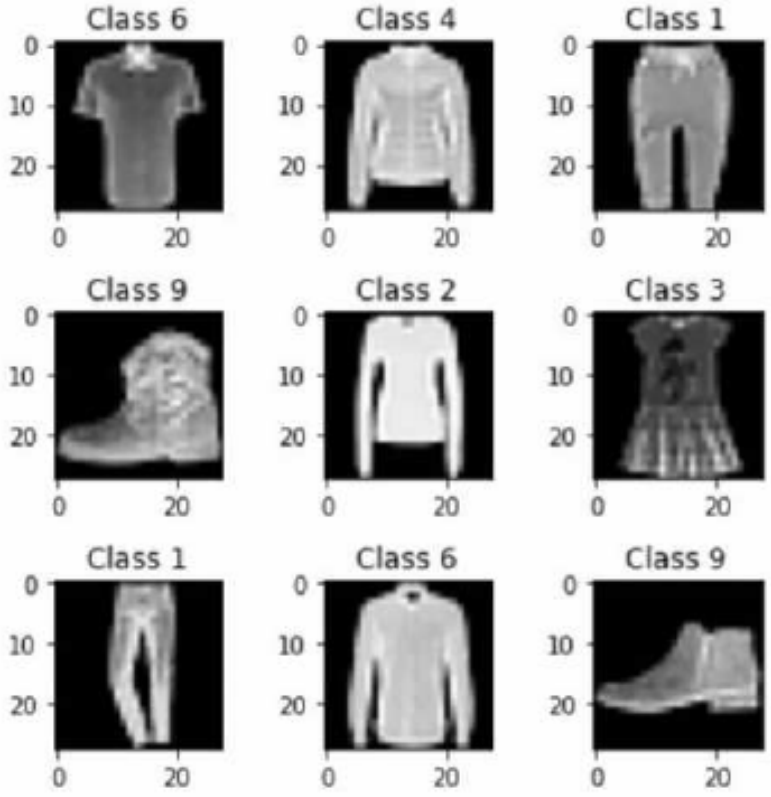
```
accurate_count = 0
for index in range(len(pred_list)):
  if pred_list[index] == np.argmax(y_test[index]):
      accurate_count += 1
accuracy = accurate_count * 1.0 / len(pred_list)

print('\n\n')
print('Accuracy tflite = ', accuracy)
print('\n\n')
```

EXEMPLE COMPLET TFL

- Nous allons décrire un exemple complet (tiré de [2]) qui utilise le *dataset* Fashion MNIST.
- Le *dataset* Fashion MNIST est en accès libre.
- C'est le pendant du *dataset* MNIST avec ses chiffres manuscrits de 0 à 9.
- Il s'agit là aussi d'images 28x28 pixels en niveau de gris :
  - 60000 images d'entraînement.
  - 10000 images de test.

EXEMPLE COMPLET TFL

- Il y a 10 vêtements identifiés par un label de 0 à 9 (apprentissage supervisé).



Source [2]

EXEMPLE COMPLET TFL

- La mise en œuvre est faite sur un PC sous Windows avec l'environnement suivant :
  - Python 3.12.
  - Bibliothèque, TensorFlow, numpy...
  - Spyder.

- On peut aussi utiliser une carte Raspberry équipée de l'OS Raspberry Pi OS avec l'environnement Python complet comme précédemment.

- Pour l'exécution du modèle, on peut aussi utiliser la bibliothèque *tftlite-runtime*.

EXEMPLE COMPLET TFL : TRAINING

```
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
import pathlib

import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from tensorflow_model_optimization.python.core.keras.compat import keras

# Load Fashion_MNIST database
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# Data pre-processing
num_class = 10
X_train = X_train.astype('float32')   # change integers to 32-bit floating point numbers
X_test = X_test.astype('float32')

X_train /= 255                        # normalize the input
X_test /= 255
```

EXEMPLE COMPLET TFL : TRAINING

```
print("shape of X_train ",X_train.shape)
print("shape of X_test ",X_test.shape)
X_train = X_train.reshape(-1, X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(-1, X_test.shape[1], X_test.shape[2], 1)
print("shape of X_train after ",X_train.shape)
print("shape of X_test after ",X_test.shape)

y_train = to_categorical(y_train, num_class)
y_test = to_categorical(y_test, num_class)

# Model definition the baseline model
model = keras.Sequential([
keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape = (28,28,1)),
keras.layers.MaxPooling2D(pool_size=(2, 2)),
keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
keras.layers.MaxPooling2D(pool_size=(2, 2)),
keras.layers.Flatten(),
keras.layers.Dropout(0.5),
keras.layers.Dense(100, activation = "relu"),
keras.layers.Dense(num_class, activation="softmax")
])

model.summary()
```

EXEMPLE COMPLET TFL : TRAINING

```
# Configure and train the baseline model
model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])

model.fit(X_train, y_train, batch_size=128, epochs=1)

# Evaluate on test set
score = model.evaluate(X_test, y_test)
print('accuracy on test data:', score[1])

# Save the TF model
model.save('model.h5')
```

EXEMPLE COMPLET TFL : TRAINING

- Résultats à l'exécution :

```
shape of X_train  (60000, 28, 28)
shape of X_test  (10000, 28, 28)
shape of X_train after  (60000, 28, 28, 1)
shape of X_test after  (10000, 28, 28, 1)

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320
 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496
 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 flatten (Flatten)           (None, 1600)              0
 dropout (Dropout)           (None, 1600)              0
 dense (Dense)               (None, 100)               160100
 dense_1 (Dense)             (None, 10)                1010
=================================================================
Total params: 179926 (702.84 KB)
Trainable params: 179926 (702.84 KB)
Non-trainable params: 0 (0.00 Byte)

469/469 [==============================] - 12s 20ms/step - loss: 0.5970 - accuracy: 0.7841
313/313 [==============================] - 1s 2ms/step - loss: 0.4108 - accuracy: 0.8519
accuracy on test data: 0.8518999814987183
```

EXEMPLE COMPLET TFL : PAS DE QUANTIFICATION

```
# Convert it into an Equivalent TFLite model
converter = tf.lite.TFLiteConverter.from_keras_model(baseline_model)
tflite_model = converter.convert()

# Save the TFLite model in your workspace
tflite_models_dir = pathlib.Path("./.")
tflite_models_dir.mkdir(exist_ok=True, parents=True)
tflite_model_file = tflite_models_dir/"model.tflite"
tflite_model_file.write_bytes(tflite_model)

# Prediction on the test set using the TFLite model
tflite_model_file = 'model.tflite'
interpreter = tf.lite.Interpreter(model_path=tflite_model_file)
interpreter.allocate_tensors()

input_index = interpreter.get_input_details()[0]["index"]
output_index = interpreter.get_output_details()[0]["index"]
```

EXEMPLE COMPLET TFL : PAS DE QUANTIFICATION

```
pred_list = []
for images in X_test:
  input_data = np.array(images, dtype=np.float32)

  input_data = input_data.reshape(1, input_data.shape[0], input_data.shape[1], 1)

  interpreter.set_tensor(input_index, input_data)
  interpreter.invoke()
  prediction = interpreter.get_tensor(output_index)
  prediction = np.argmax(prediction)
  pred_list.append(prediction)

accurate_count = 0
for index in range(len(pred_list)):
  if pred_list[index] == np.argmax(y_test[index]):
      accurate_count += 1
accuracy = accurate_count * 1.0 / len(pred_list)

print('\n\n')
print('Accuracy tflite = ', accuracy)
print('\n\n')
```

EXEMPLE COMPLET TFL : QUANTIFICATION PTQ

```
# Post training quantization PTQ
converter = tf.lite.TFLiteConverter.from_keras_model(baseline_model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model_ptq = converter.convert()

# Save the model after post-training quantization
tflite_models_dir = pathlib.Path("./.")
tflite_models_dir.mkdir(exist_ok=True, parents=True)
tflite_model_file = tflite_models_dir/"model_ptq.tflite"
tflite_model_file.write_bytes(tflite_model_ptq)

# Evaluate the model performance
tflite_model_file = 'model_ptq.tflite'
interpreter = tf.lite.Interpreter(model_path=tflite_model_file)
interpreter.allocate_tensors()

input_index = interpreter.get_input_details()[0]["index"]
output_index = interpreter.get_output_details()[0]["index"]
```

EXEMPLE COMPLET TFL : QUANTIFICATION PTQ

```
pred_list = []
for images in X_test:
  input_data = np.array(images, dtype=np.float32)

  input_data = input_data.reshape(1, input_data.shape[0], input_data.shape[1], 1)

  interpreter.set_tensor(input_index, input_data)
  interpreter.invoke()
  prediction = interpreter.get_tensor(output_index)
  prediction = np.argmax(prediction)
  pred_list.append(prediction)

accurate_count = 0
for index in range(len(pred_list)):
  if pred_list[index] == np.argmax(y_test[index]):
      accurate_count += 1
accuracy = accurate_count * 1.0 / len(pred_list)

print('\n\n')
print('Accuracy Post-Training Quantization = ', accuracy)
print('\n\n')
```

EXEMPLE COMPLET TFL : QUANTIFICATION PTQ

- Résultats à l'exécution :

```
Accuracy tflite =  0.8514
Accuracy Post-Training Quantization =  0.8513
```

- La précision du modèle TFL sans quantification (*float*) ou avec quantification PTQ est équivalente.

- On a donc intérêt à quantifier sachant que l'on peut corriger les erreurs de quantification avec la quantification QAT.

- La mise en œuvre de la quantification QAT ne sera pas abordée (voir [4] ou [8]).

EXEMPLE COMPLET TFL : RESULTATS

- On peut regarder la taille des différents modèles :
  - Modèle de base (fichier *model.h5*) : 2 203 472 octets.
  - Modèle TFL sans quantification (fichier *model.tflite*) : 723 084 octets.
  - Modèle TFL avec quantification PTQ (fichier *model_ptq.tflite*) : 189 736 octets.
  - Modèle TFL avec quantification QAT (fichier *model_qat.tflite*) : 187 472 octets.

- Si l'on utilise la bibliothèque *tftlite-runtime*, on écrira alors :

```
from tflite_runtime.interpreter import Interpreter

interpreter = Interpreter('model.tflite')
```

CONCLUSION

- Nous avons pu voir la mise en œuvre de TensorFlow Lite (TFL).
- TFL est adaptée aux systèmes embarqués disposant d'un système d'exploitation.
- Le développement se fait en langage Python.
- Le modèle classique est transformé en modèle TFL.
- On peut utiliser la quantification ou le *pruning* pour diminuer la taille d'un facteur 4 du modèle TFL sans pertes significatives de précision.

CHAPITRE 5 :
CANEVAS DE PROGRAMMATION TENSORFLOW LITE FOR MICRO

INTRODUCTION

- TensorFlow Lite for Micro (TFLM) est un *framework open source* pour le *Machine Learning* développé par Google.
- TFLM est fait pour les microcontrôleurs ayant peu de mémoire et peu de puissance de calcul.
- TFLM n'a pas besoin d'OS ni d'allocation mémoire dynamique.
- La consommation électrique du système embarqué mis en œuvre est de quelques dizaines de mW.

INTRODUCTION

- Comme il n'y a pas d'OS, TFLM n'utilise pas le langage Python mais le langage C/C++.
- TFLM n'utilise pas les bibliothèques standards C++ pour limiter l'empreinte mémoire.
- TFLM a son dépôt :

  https://github.com/tensorflow/tflite-micro/tree/main
- et ses exemples :

  https://github.com/tensorflow/tflite-micro/tree/main/tensorflow/lite/micro/examples

INTRODUCTION

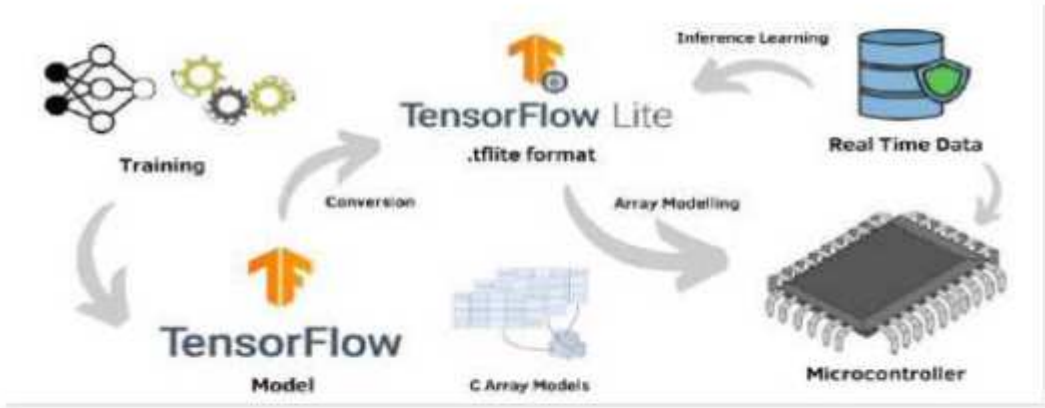
- TFLM a des besoins limités en mémoire :
  - *Runtime* : 16 Ko.
  - Exemple : modèle de reconnaissance de la voix (oui/non) : 22 Ko
  - Exemple : modèle de détection de personne : 250 Ko

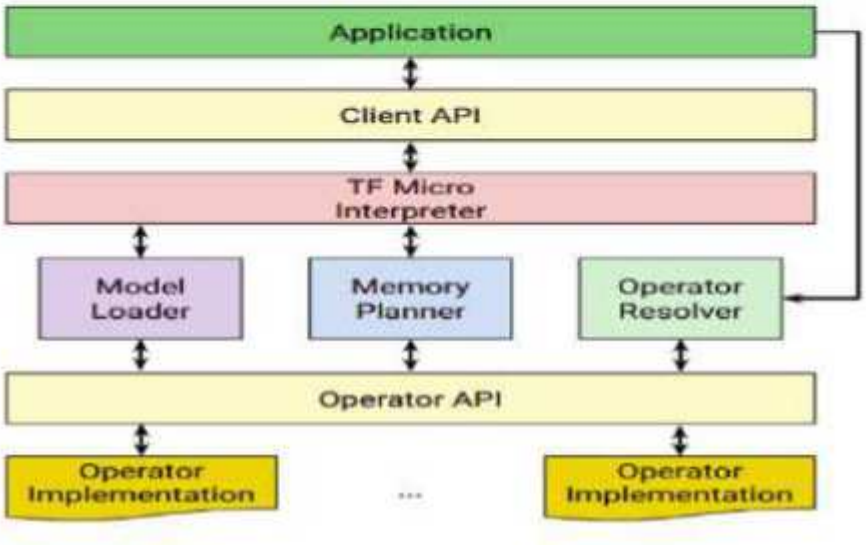
- TFLM supporte un certain nombre d'opérateurs d'IA (100 ops) de TF mais pas tous pour l'instant :

  https://github.com/tensorflow/tflite-micro/blob/main/tensorflow/lite/micro/kernels/micro_ops.h

- TFLM utilise comme TFL le format *FlatBuffer*.

MISE EN OEUVRE





Source [4]

MISE EN OEUVRE

- La mise en œuvre de TFLM est la même que TFL. On distingue différentes étapes :
  - 1. Entraînement d'un modèle TF.
  - 2. Conversion et optimisation du modèle TF: le modèle TF est converti en modèle TFL.
  - 2bis. Le modèle TFL est ensuite converti en modèle TFLM sous forme d'un tableau C/C++.
  - 3. Déploiement et exécution du modèle TFLM.

- Les étapes 1 et 2 sont identiques à celle pour TFL. On ne va détailler que les étapes 2bis et 3 pour TFLM.

MISE EN ŒUVRE : ETAPE 2BIS MODELE TFLM

- Le modèle TFL est transformé en modèle TFLM qui est un tableau C++

- On utilise pour cela la commande `xxd` :

```
host% xxd -i model.tflite > model.cc
```

MISE EN ŒUVRE : ETAPE 2BIS MODELE TFLM

- Le fichier C++ model.cpp ressemble à :

```
#include "model.h"

// Keep model aligned to 8 bytes to guarantee aligned 64-bit accesses.
alignas(8) const unsigned char g_model[] = {

  0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x14, 0x00, 0x20, 0x00,
  0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x00, 0x00,
  0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
  . . .
  0x01, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x0c, 0x00,
  0x0b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0x00, 0x0c, 0x00, 0x00, 0x00,
  0x09, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x09
};
const int g_model_len = 2576;
```

- Le tableau C++ s'appelle `g_model`. Sa taille en octets est `g_model_len`.

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- L'exécution d'un modèle TFLM se fait à l'aide d'un interpréteur.

- Les différentes étapes sont :
  - 0. Déclaration des fichiers *include* :
    - Fichier *all_ops_resolver.h* : classe C++ qui autorise l'interpréteur à charger des opérations *ops*.
    - Fichier *micro_error_reporter.h* : classe C++ pour faire le *log* de la sortie standard et des erreurs.
    - Fichier *micro_interpreter.h* : interpréteur qui exécute le modèle.
    - Fichier *schema_generated.h* : définition de la structure TFLM *FlatBuffer.*
    - Fichier *version.h* : version courante.
    - Fichier *model.h* : fichier qui définit les constantes de notre modèle :

```
extern const unsigned char g_model[];
extern const int g_model_len;
```

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- Les différentes étapes sont :
  - 1. Déclaration des variables TFLM :
    - Instance `error_reporter` de la classe `ErrorReporter` : mécanisme pour le *log* et le *debug* durant l'exécution.
    - Instance `interpreter` de la classe `MicroInterpreter` : interpréteur pour l'exécution du modèle.
    - Instance `model` de la classe `Model` : modèle TFLM.
    - Instance `modele_input` de la classe `TfLiteTensor` : entrée de l'interpréteur. C'est un tenseur.
    - Instance `modele_output` de la classe `TfLiteTensor` : sortie de l'interpréteur. C'est un tenseur.
    - Tableau `tensor_arena` de taille `kTensorArenaSize` : mémoire de travail de l'interpréteur.

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- Les différentes étapes sont :
  - 2. Validation du log :
    - Instance `micro_error_reporter` de la classe `MicroErrorReporter` : sous-classe de la classe `ErrorReporter` qui surcharge la méthode de *log*.
    - Affection de l'instance `micro_error_reporter` à l'instance `error_reporter` car on veut utiliser la méthode `Report()` de la classe `ErrorReporter` et non de `MicroErrorReporter`.
  - 3. Chargement du modèle :
    - On charge le modèle `g_model` avec la méthode `GetModel()`.
    - On vérifie que la version du modèle chargé est compatible avec la version de TFLM.

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- Les différentes étapes sont :
  - 4. Création de l'interpréteur :
    - Instance `resolver` de la classe `AllOpsResolver` pour accéder aux *ops*.
    - Instance `static_interpreter` de la classe `MicroInterpreter`.
    - Affection de l'instance `static_interpreter` à l'instance `interpreter`.
  - 5. Allocation mémoire à l'interpréteur.

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- Les différentes étapes sont :
  - 6. Affection des variables d'entrée et de sortie de l'interpréteur pour l'accès aux tenseurs d'entrée et de sortie de l'interpréteur :

```
model_input = interpreter->input(0);
model_output = interpreter->output(0);
```

  - 7. Copie de la valeur d'entrée dans le tenseur d'entrée de l'interpréteur :

```
model_input->data.f[0] = x_val;
```

    - La notation `data.f[0]` est pour un flottant.
    - La notation `data.i32[0]` est pour un entier 32 bits.
    - Voir le fichier *c_api_internal.h*.

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- Les différentes étapes sont :
  - 7. Copie de la valeur d'entrée dans le tenseur d'entrée de l'interpréteur.
    - Le scalaire 1 donne :

```
model_input->data.f[0]=1.0
```

    - Le vecteur [1 2 3] donne :

```
model_input->data.f[0]=1.0 model_input->data.f[1]=2.0 model_input->data.f[2]=3.0
```

    - La matrice [1 2 3]
      [4 5 6] donne :

```
model_input->data.f[0]=1.0 model_input->data.f[1]=2.0 model_input->data.f[2]=3.0
model_input->data.f[3]=5.0 model_input->data.f[4]=5.0 model_input->data.f[5]=6.0
```

    - Idem pour un tenseur.

    - Tout est finalement mis à plat.

MISE EN ŒUVRE : ETAPE 3 EXECUTION

- Les différentes étapes suivies sont :
  - 8. Exécution de l'interpréteur par la méthode `Invoke()`.
  - 9. Récupération de la valeur de sortie du tenseur de sortie de l'interpréteur :

```
y_val = model_output->data.f[0];
```

MISE EN ŒUVRE : ETAPE 3 EXECUTION

```
// ETAPE 0 : liste des include
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"

#include "model.h"

// ETAPE 1 : declaration des variables TFLM
// TFLite globals, used for compatibility with Arduino-style sketches
tflite::ErrorReporter* error_reporter = nullptr;
const tflite::Model* model = nullptr;
tflite::MicroInterpreter* interpreter = nullptr;
TfLiteTensor* model_input = nullptr;
TfLiteTensor* model_output = nullptr;

// Create an area of memory to use for input, output, and other TensorFlow
// arrays. You'll need to adjust this by combiling, running, and looking
// for errors.
constexpr int kTensorArenaSize = 5 * 1024;
uint8_t tensor_arena[kTensorArenaSize];
```

MISE EN ŒUVRE : ETAPE 3 EXECUTION

```
// ETAPE 2 : validation du log
// Set up logging (will report to Serial, even within TFLite functions)
static tflite::MicroErrorReporter micro_error_reporter;
error_reporter = &micro_error_reporter;

// ETAPE 3 : chargement du modele g_model
// Map the model into a usable data structure
model = tflite::GetModel(g_model);
if (model->version() != TFLITE_SCHEMA_VERSION) {
    error_reporter->Report("Model version does not match Schema");
}

// ETAPE 4 : creation de l'interpreteur
static tflite::AllOpsResolver resolver;
// Build an interpreter to run the model
static tflite::MicroInterpreter static_interpreter( model, resolver, tensor_arena,
    kTensorArenaSize, error_reporter);
interpreter = &static_interpreter;
```

MISE EN ŒUVRE : ETAPE 3 EXECUTION

```
// ETAPE 5 : allocation memoire pour tenseurs
// Allocate memory from the tensor_arena for the model's tensors
TfLiteStatus allocate_status = interpreter->AllocateTensors();
if (allocate_status != kTfLiteOk) {
    error_reporter->Report("AllocateTensors() failed");
}

// ETAPE 6 : affectation des variables d'entree et de sortie de l'interpreteur
// Assign model input and output buffers (tensors) to pointers
model_input = interpreter->input(0);
model_output = interpreter->output(0);

float x_val, y_val;

// ETAPE 7 : variable d'entree x_val pour interpreteur
// Copy x_val to input buffer (tensor)
model_input->data.f[0] = x_val;
```

MISE EN ŒUVRE : ETAPE 3 EXECUTION

```
// ETAPE 8 : execution interpreteur
// Run inference
TfLiteStatus invoke_status = interpreter->Invoke();
if (invoke_status != kTfLiteOk) {
    error_reporter->Report("Invoke failed on input: %f\n", x_val);
}

// ETAPE 9 : variable de sortie y_val de l'interpreteur
// Read predicted y value from output buffer (tensor)
y_val = model_output->data.f[0];
```

EXEMPLE COMPLET TFLM

- La mise en œuvre est faite sur un PC sous Linux avec l'environnement suivant :
  - IDE Arduino IDE
  - Bibliothèque Arduino TFLM.

- La mise en œuvre pratique sera vue en TP avec la carte Arduino Nano 33 BLE Sense.

EXEMPLE COMPLET TFLM

```
#include <TensorFlowLite.h>
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"

#include "model.h"

namespace {
  tflite::ErrorReporter* error_reporter = nullptr;
  const tflite::Model* model = nullptr;
  tflite::MicroInterpreter* interpreter = nullptr;
  TfLiteTensor* model_input = nullptr;
  TfLiteTensor* model_output = nullptr;

  // Create an area of memory to use for input, output, and other TensorFlow arrays
  constexpr int kTensorArenaSize = 5 * 1024;
  uint8_t tensor_arena[kTensorArenaSize];
}
```

EXEMPLE COMPLET TFLM

```
void setup() {

  Serial.begin(9600);

  // Set up logging (will report to Serial, even within TFLite functions)
  static tflite::MicroErrorReporter micro_error_reporter;
  error_reporter = &micro_error_reporter;

  // Map the model into a usable data structure
  model = tflite::GetModel(g_model);
  if (model->version() != TFLITE_SCHEMA_VERSION) {
    error_reporter->Report("Model version does not match Schema");
    while(1)
      ;
  }
```

EXEMPLE COMPLET TFLM

```
  static tflite::AllOpsResolver resolver;

  // Build an interpreter to run the model
  static tflite::MicroInterpreter static_interpreter(model, 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) {
    error_reporter->Report("AllocateTensors() failed");
    while(1)
      ;
  }

  // Assign model input and output buffers (tensors) to pointers
  model_input = interpreter->input(0);
  model_output = interpreter->output(0);
}
```

EXEMPLE COMPLET TFLM

```
void loop() {
  TfLiteStatus invoke_status;
  float x, y;

  model_input->data.f[0] = 0.0;
  invoke_status = interpreter->Invoke();
  y = model_output->data.f[0];

  . . .
}
```

CONCLUSION

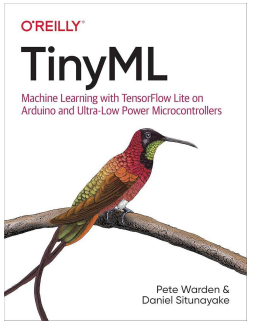
CONCLUSION

- Nous avons présenté TinyML ou l'IA embarquée pour les petits systèmes embarqués.
- Les principales caractéristiques de TinyML ont été présentées.
- Pour les processeurs supportant un système d'exploitation, nous avons étudié TensorFlow Lite.
- Pour les processeurs ne supportant pas un système d'exploitation, nous avons étudié TensorFlow Lite for Micro.

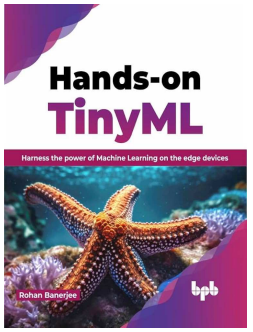
BIBLIOGRAPHIE

REFERENCES BIBLIOGRAPHIQUES

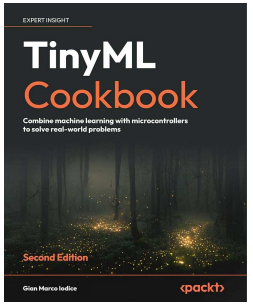
- [1] TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. P. Warden, D. Situnayake. Editions O'Reilly



- [2] Hands-on TinyML. R. Banerjee. Editions BPB Online



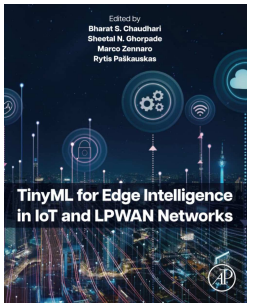
- [3] TinyML Cookbook. 2ème édition. G. Marco Iodice. Editions Packt



REFERENCES BIBLIOGRAPHIQUES

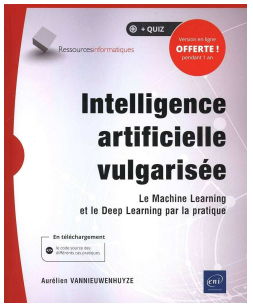
- [4] Deep Learning on Microcontrollers. A. Gupta, S. Nandyala. Editions BPB Online

- [5] TinyML for Edge Intelligence in IoT and LPWAN Networks. S. Chaudhari and al. Editions Academic Press

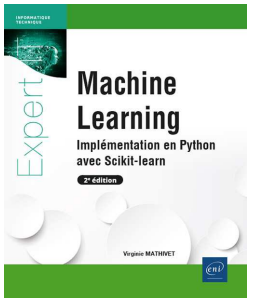


REFERENCES BIBLIOGRAPHIQUES

- [6] Intelligence artificielle vulgarisée. A. Vannieuwenhuyze. Editions ENI



- [7] Machine Learning. Implémentation en Python avec Scikit-learn. V. Mathivet. Editions ENI



REFERENCES BIBLIOGRAPHIQUES

- [8] LiteRT : https://ai.google.dev/edge/litert
- [9] Everything about TensorFlow Lite and start deploying your machine learning model. A. Ming. Mai 2022 https://www.seeedstudio.com/blog/2022/05/08/everything-about-tensorflow-lite-and-start-deploying-your-machine-learning-model/
- [10] TinyML: Machine Learning for Embedded System. Part 1. L. Cavagnis. Juillet 2021

  https://leonardocavagnis.medium.com/tinyml-machine-learning-for-embedded-system-part-i-92a34529e899
- L'IA. Du mythe à la réalité. N. Saby. Editions ENI. 2024
- CNRS. Le journal. L'héritage d'Alan Turing. Mai 2012
- Introduction à TinyML. J. Bagur. Elektor. Décembre 2022
- Let's talk aboutTinyML. Intelligence on Microcontrollers. S. Muhunyo. Juin 2023
- TensorFlow Lite for microcontrollers. D. Situnayake
- TensorFlow Lite for Microcontrollers: Recent Developments. A. Jain, D. Davis, J. Withers. 2022