Clusters de calcul adaptatifs & Algorithmes MCMC

Matthieu François & Jordy Palafox

GreenAI U.P.P.A.

Journées MAS 2022

31-08-2022

En collaboration avec P. Gay, S. Lebeaud, S. Loustau, F.K. Sow, N. Tirel, Y.Chaigneau.

Notre point de départ

Basé sur les travaux de S. Loustau
[Chee and Loustau, 2021][Chee et al., 2021], nous voulons :

- écrire des **réseaux faiblement consommateurs**,
- utiliser du **matériel optimisé** pour diminuer la consommation du cycle de vie d'un algorithme.

Théorie
(Stats Pac-Bayésiennes)
$\Downarrow$
Implémentation
+
Choix de l'hardware

Notre approche

Au niveau du réseau :

1. **Architecture** : Pruning & Sparsité [Chaigneau and Tirel, 2021],
2. **Choix de la nature des coefficients** :
   - Quantisation : format de stockage $int8$, $int16$...,
   - Binarisation [M.Courbariaux et al., 2016]: jusqu'à $7\times$ plus rapide avec Larq sur des processeurs ARM,
3. **Early Exit** : $50\times$ plus rapide dans certains cas favorables de vidéos surveillances [Teerapittayanon et al., 2016],
4. Choix d'un **optimiseur** : SGD ou ... optimiseurs MCMC (cf. exposé de S.Loustau).

Notre approche

Processeur Reduced Instruction Set Computing : RISC,
Processeur Complex Instruction Set Computers : CISC.

RISC ARM VS CISC x86.

- Processeurs CISC : classiques, utilisent des instructions adaptées pour l'exécution de tâches complexes,
- Processeurs RISC : préférables pour la manipulation de tâches très bas niveau (binaire).

[Jamil and Tariq, 1995]

Notre approche

Autres motivations

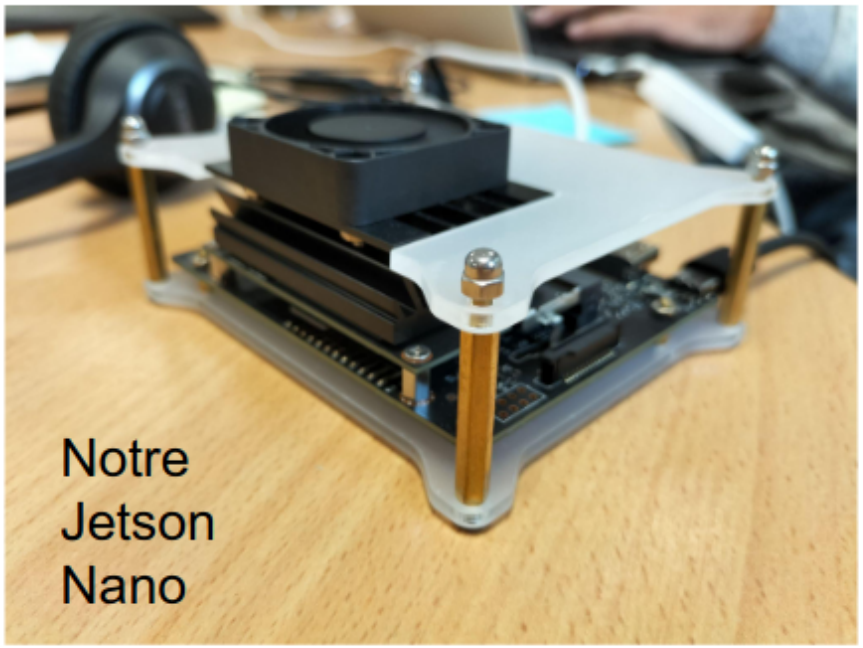
- Le **parallélisme de modèle** pour entraîner de gros modèles sur le cloud [Huang et al., 2019].
- Avantage d'une inférence en plusieurs fois dans un contexte d'embarqué et de edge / cloud computing.
- Profiter de la puissance de calcul de capteurs embarqués.

Notre approche

D'autres travaux dans cette direction

- **Division des tâches** (apprentissage asynchrone) : un premier modèle apprend des données et trie les cas faciles et rejète les difficiles vers un serveur plus puissant. (Similaire avec le principe d'une architecture en cascade type early exit)
- L'apprentissage asychrone pour le cloud reste un problème difficile malgré des résultats encourageants.
  [Oyallon et al., 2021]

Choix du matériel



Raspberry pi

Choix du matériel

	Poids (g)	Dim. (mm)	Mémoire	Opération / sec.	Conso.
Serveur (RTX 3090)	-	313x138	(N x) 24Gb	(N x) 285 TFLOPS	350W
Jetson Nano	138g	45x70	2Gb	472 GFLOPS	10W
Raspberry pi	40g	56x85	0.5Gb	~700MFLOPS	3W

Choix du matériel

	Poids (g)	Dim. (mm)	Mémoire	Opération / sec.	Conso.
Serveur (RTX 3090)	-	313x138	(N x) 24Gb	(N x) 285 TFLOPS	350W
Jetson Nano	138g	45x70	2Gb	472 GFLOPS	10W
Raspberry pi	40g	56x85	0.5Gb	~700MFLOPS	3W

Choix du matériel

	Poids (g)	Dim. (mm)	Mémoire	Opération / sec.	Conso.
Serveur (RTX 3090)	-	313x138	(N x) 24Gb	(N x) 285 TFLOPS	350W
Jetson Nano	138g	45x70	2Gb	472 GFLOPS	10W
Raspberry pi	40g	56x85	0.5Gb	~700MFLOPS	3W

Un optimiseur par type de carte

Optimiseur MCMC

Initialisation : $\lambda > 0$, $L \geq 1$, $k = 1$ et un prior π.

Générer $\hat{w}^{(1)}$ à partir de π.

Pour $k = 1, \ldots, N$:

- **Choix uniforme** d'une couche $\hat{l} \in \{1, \ldots, L\}$,
- **Choix uniforme** d'un voisinage $\mathcal{V}_{\hat{l}}(\hat{w}^{(k)})$
- **Générer** $\tilde{w}_{\hat{l}} \sim \tilde{p}_{\hat{w}^{(k)}}$ qui dépend de la couche et du voisinage.
- **Calcul de ratio d'acceptation :**

$$\rho(\tilde{w}, \hat{w}^{(k)}) = \frac{exp(-\lambda \mathcal{R}_n(\tilde{w}))\pi(\tilde{w})\tilde{p}(\hat{w}^{(k)})}{exp(-\lambda \mathcal{R}_n(\hat{w}^{(k)}))\pi(\hat{w}^{(k)})\tilde{p}(\tilde{w})}$$

$$\text{où } \mathcal{R}_n(w) = \sum_{i=1}^{n} \ell(y_i, g_w(x_i))$$

- **Mise à jour des poids :**

$$(\hat{w}^{(k+1)}, \hat{l}^{(k+1)}) = \begin{cases} (\tilde{w},\ \tilde{l}) \text{ avec proba } \rho(\tilde{w},\ \hat{w}^{(k)}), \\ (\hat{w}^{(k)}, \hat{l}^{(l)}) \text{ sinon.} \end{cases}$$

Loi de Student
$Stud(\hat{w}_{\hat{l}}, \mathcal{V}_{\hat{l}})$
Jetson :
Sparsité et Pruning

Loi de binarisation
$Bin(\hat{w}_{\hat{l}}, \mathcal{V}_{\hat{l}})$
Raspberry :
Binarisation et Quantisation

Choix des optimiseurs

Poids du réseau et chaines de Markov

Chaque optimiseur :

- est défini par un algorithme de type Metropolis-Hasting,
- définit une chaîne de Markov.

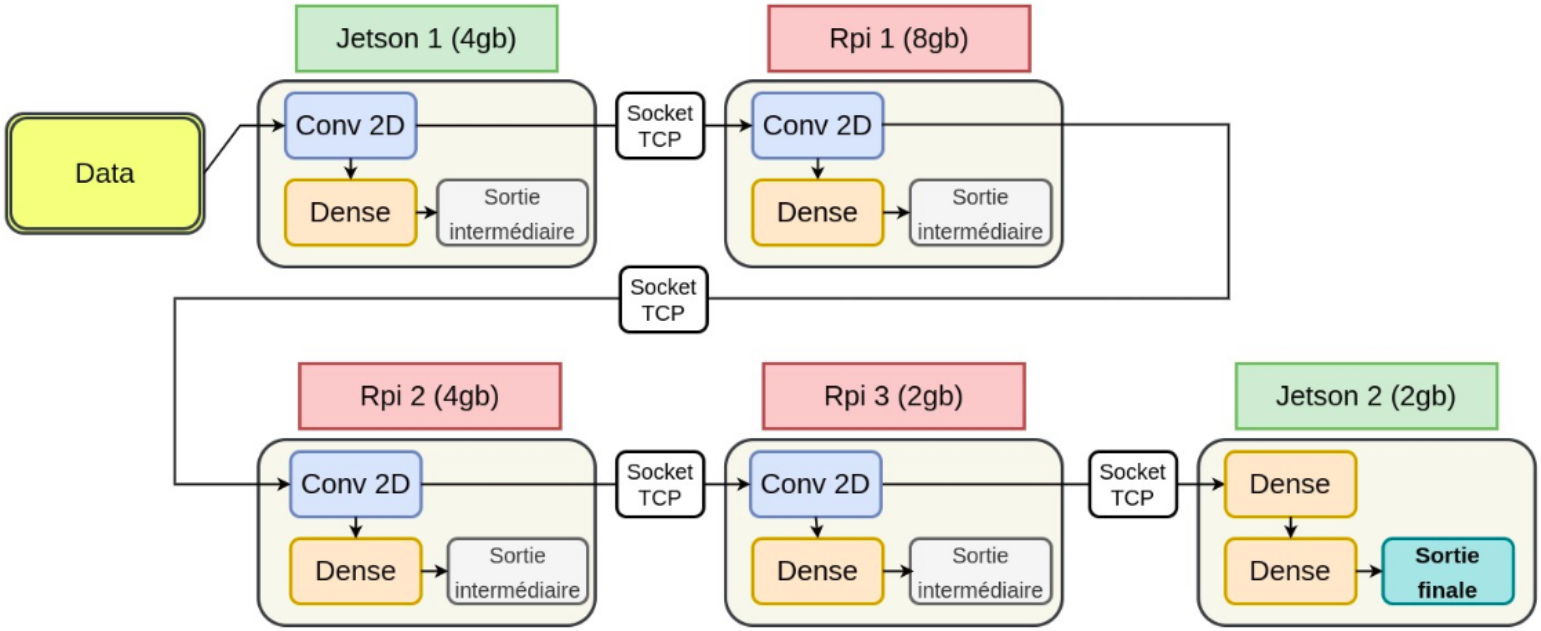
Propriété

Soit $(X_n)_{n\in\mathbb{N}}$ une chaîne de Markov qui est soit :

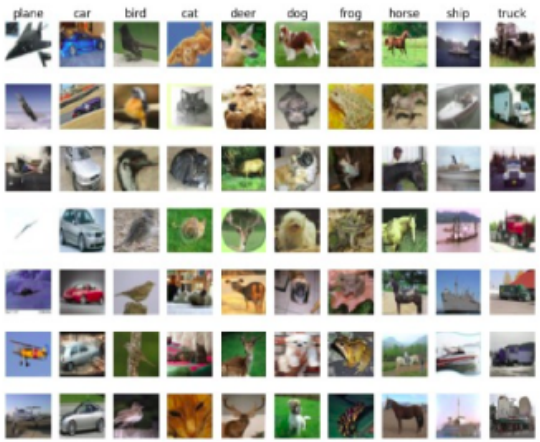
- générée par l'optimiseur Jetson (espace d'états continu),
- générée par l'optimiseur RaspBerry (espace d'états discret).

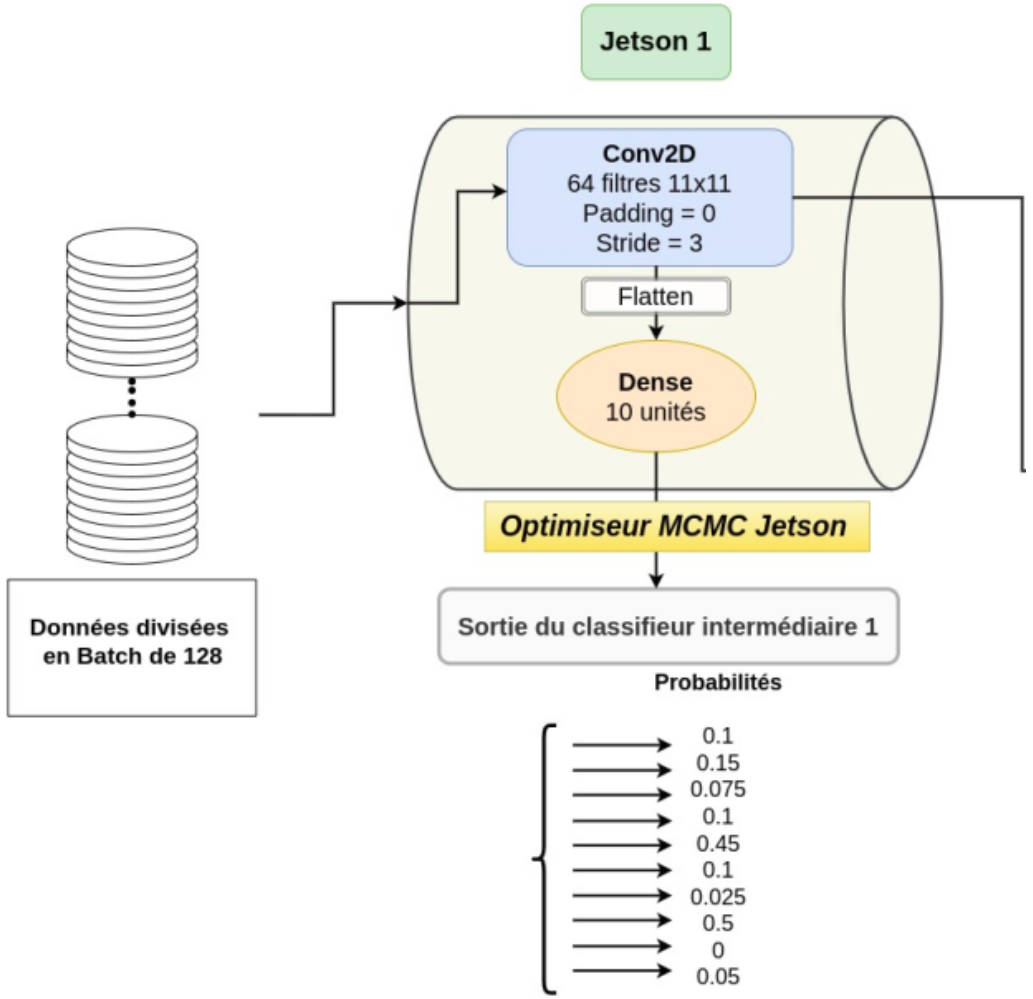
Alors (X_n) converge en variation totale.

Architecture du réseau

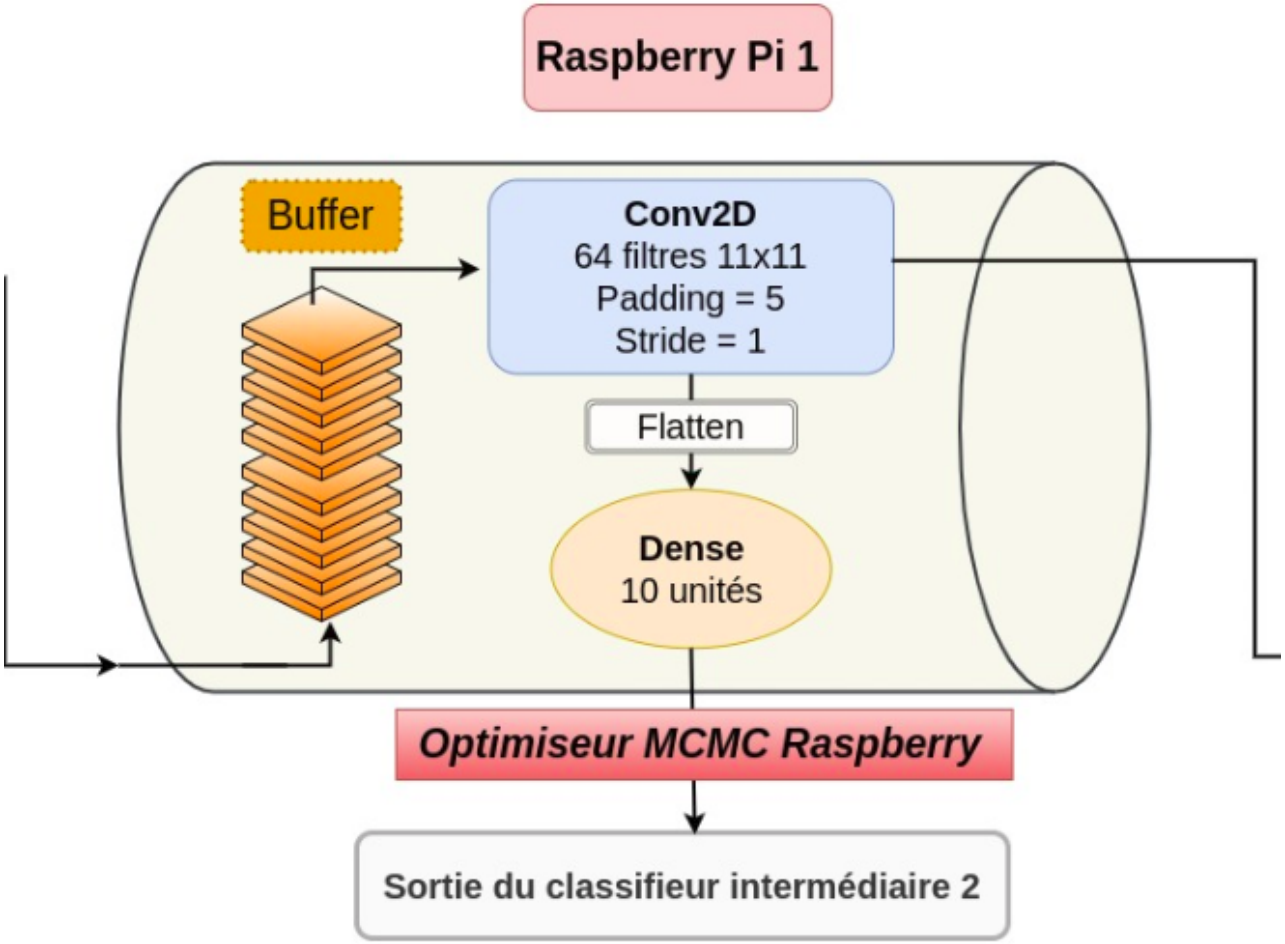


Début du réseau

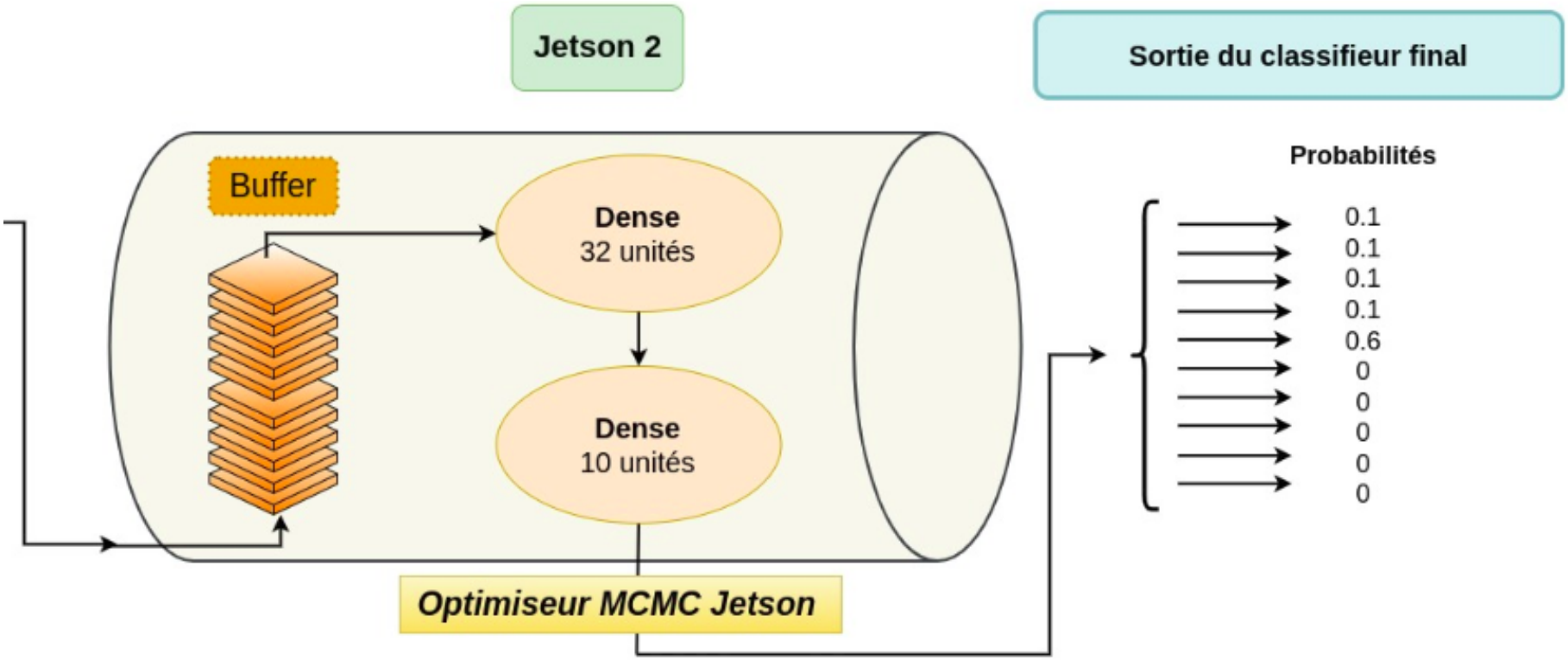




Milieu du réseau



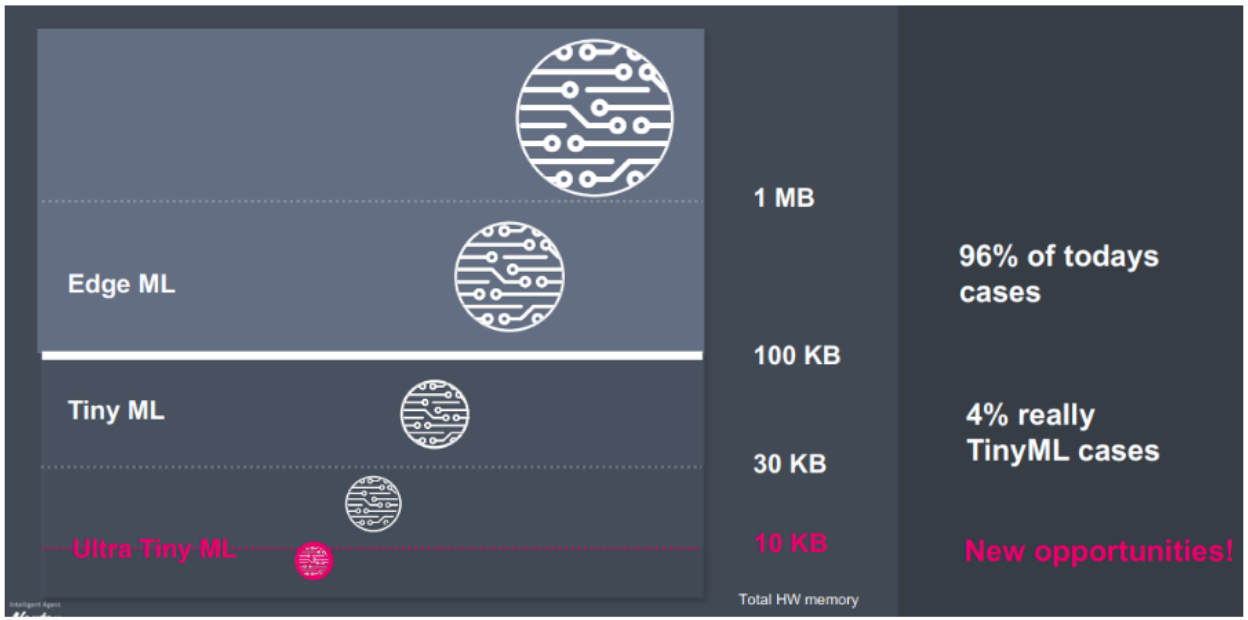
Fin du réseau



Démonstration

Vers plus de contrainte sur la consommation

Peut-on imposer des puissances encore plus faibles sur le hardware ?

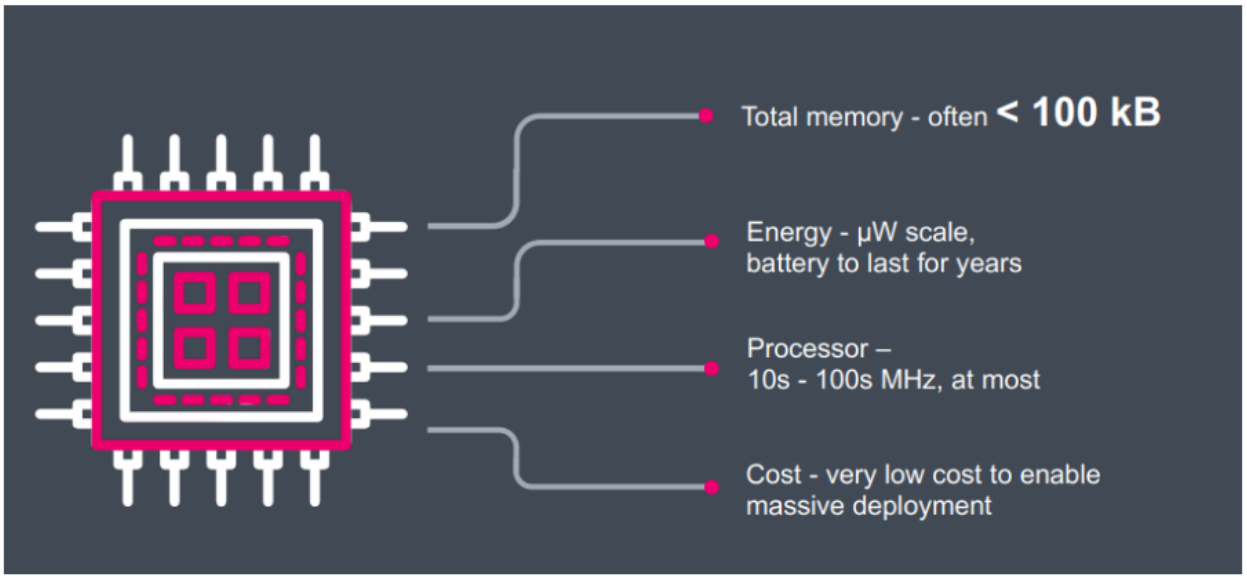


[NEUTON.AI, 2022]

Aller plus loin dans le matériel : le TinyML

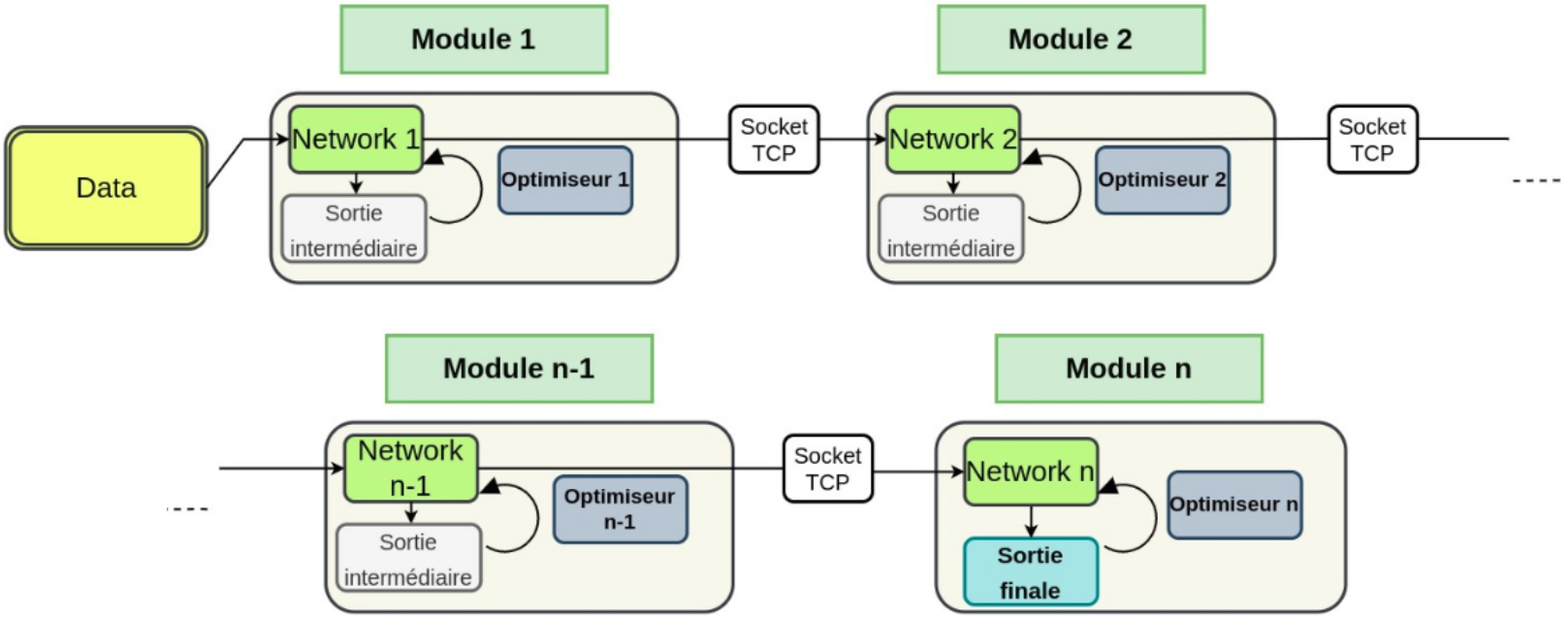
Défintion

Le **Tiny ML** permet d'utiliser des algorithmes de ML utilisant une puissance de moins de 1mW.



[NEUTON.AI, 2022]

Aller plus loin dans l'architecture



Conclusion et Perspectives

- Convolution Sparse et matrices sparses par block avec cuSparse : format de stockage rapide pour des multiplications avec des matrices sparses,
- Utiliser des modèles de forme différentes,
- Reversible Jump MCMC : permettre au modèle d'ajouter/supprimer des couches (quid de la convergence ?) [Green, 1995].

Conclusion et Perspectives

Merci de votre attention !

References I

[Chaigneau and Tirel, 2021] Chaigneau, Y. and Tirel, N. (2021).
Pruning in neural networks.

[Chee et al., 2021] Chee, A., Gay, P., and Loustau, S. (2021).
Sparsity regret bounds for xnor-nets++.

[Chee and Loustau, 2021] Chee, A. and Loustau, S. (2021).
Learning with bot - bregman and optimal transport divergences.

[Green, 1995] Green, P. (1995).
Reversible jump markov chain monte carlo computation and bayesian model determination.

[Huang et al., 2019] Huang, Y. et al. (2019).
Gpipe: Easy scaling with micro-batch pipeline parallelism.

References II

[Jamil and Tariq, 1995] Jamil and Tariq (1995).

Risc versus cisc.

[M.Courbariaux et al., 2016] M.Courbariaux et al. (2016).

Binarized neural networks: Training neural networks with weights and activations constrained to +1 or 1.

[NEUTON.AI, 2022] NEUTON.AI (2022).

A novel approach to building exceptionally tiny models without loss of accuracy.

[Oyallon et al., 2021] Oyallon, E. et al. (2021).

Decoupled greedy learning of cnns for synchronous and asynchronous distributed learning.

References III

[Teerapittayanon et al., 2016] Teerapittayanon, S., McDanel, B., and Kung, H.-T. (2016).
Branchynet: Fast inference via early exiting from deep neural networks.