Déploiement efficace des Transformers sur FPGA : des stratégies de quantification à l'architecture adaptée au matériel

07/04/2026

(OASIS) Lubin GAUTHIER
lubin.gauthier@toulouse-inp.fr

Thesis director
(OASIS) Julien PERCHOUX
Co-director
(OASIS) Francis BONY

Sommaire

- *Edge AI*
- *Transformer sur FPGA*
- *FloPoCo*
- *Quantification entier 8 bits*
- *Discussion*

Edge AI

Edge AI

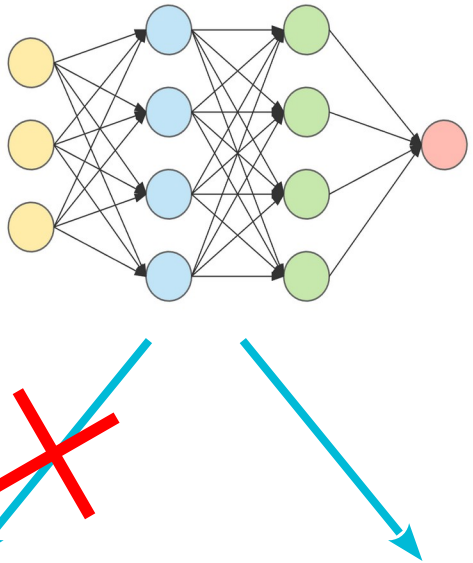
Notre objectif

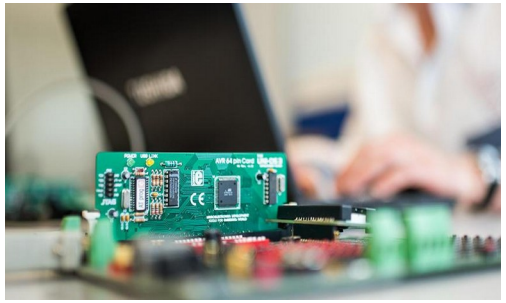
Notre objectif :

Exécuter des modèles de deep learning à l'état de l'art en périphérie (edge AI), sans connexion cloud

→ L'entraînement est sur GPU classique, mais l'inférence doit être embarquable et donc doit consommer peu d'énergie

Notre Mission : Maîtriser les opérations fondamentales de modèles de deep learning pour les adapter aux contraintes matérielles des FPGAs





Quantification

La quantification, à quoi ça sert ?

Le principal problème des modèles de deep learning en edge AI :
→ le nombre élevé de paramètres

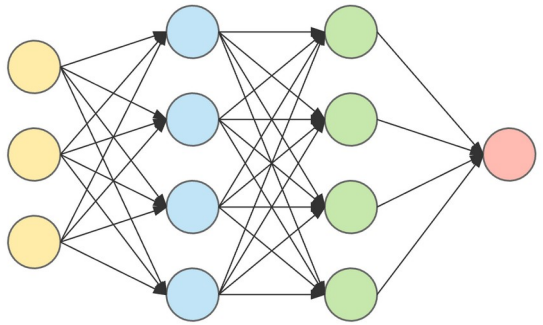
- **Empreinte mémoire élevée**

- Nombre important de paramètres = **beaucoup de calculs**, mais **parallélisation** possible

- **Calculs complexes avec des flottants**

Quantification

Qu’est-ce qu’un réseau de neurone ?



Qu’est-ce qu’un réseau de neurone ?

On peut résumer simplement un réseau de neurone à des **Multiplication-Accumulation (MAC)**

=

$$y = \sum_{i=0}^{n-1} w_i \cdot x_i + b$$

=

```
y = 0
pour i de 0 à n-1 :
    y = y + w[i] * x[i]
y = y + b
```

Quantification

Résumé

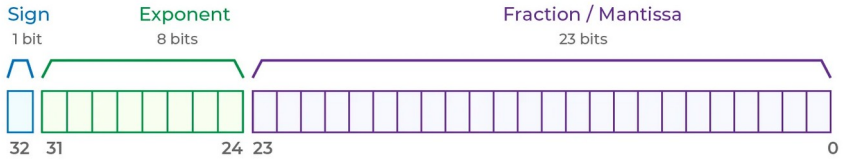
Accéléré un réseau de neurones = accélérer des MACs en parallèle

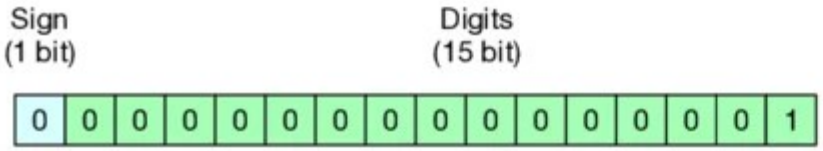
La quantification :

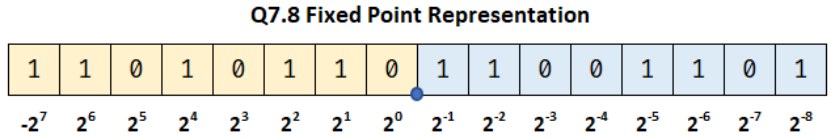
En quel format faire les MACs ? En flottant ? En entier ? En virgule fixe ?

Quantification

Les différents formats numériques







Format	Min	Max	Résolution
Float32	$\pm 1.17549435 \times 10^{-38}$	$\pm 3.402823466 \times 10^{38}$	$\sim 10^{-7}$
Float16	$\pm 6.103515625 \times 10^{-5}$	±65504	$\sim 10^{-3}$
Int8	-128	127	1
Fixed Q8.8	-128	127.99609375	$2^{-8} = 0.00390625$

Le float est très précis mais complexe à manipuler en embarqué,
contrairement à l'entier et à la virgule fixe qui sont beaucoup plus simple, mais sont moins précis

Un cas particulier : le Transformer sur FPGA

Le Transformer sur FPGA

C'est quoi un FPGA ?

Un **FPGA** est une ressource matérielle, une puce, donc juste une matrice de transistors

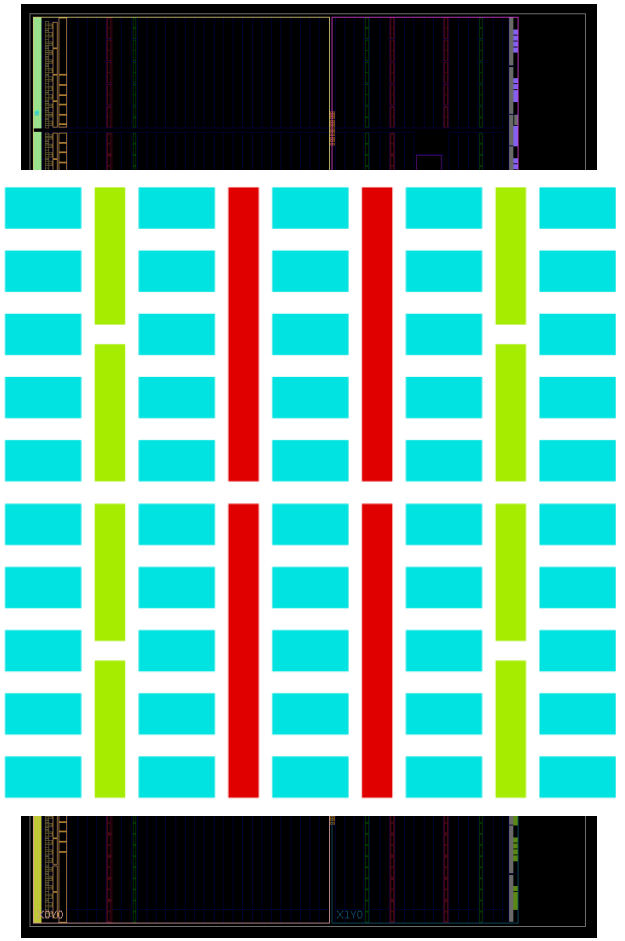
Les transistors sont organisés et structurés par ***bloc logique***

Les principaux blocs logiques sont :

- **DSP** → multiplieur, additionneur
- **BRAM** → mémoire au plus proche de la logique
- **Slice** → LUT, FlipFlops, Multiplexeur et Carry Logic

Comment on les connecte et assemble entre eux ?

Via un langage de description (VHDL)



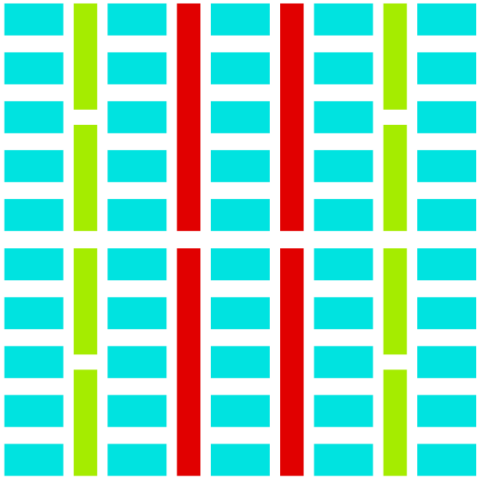
Le Transformer sur FPGA

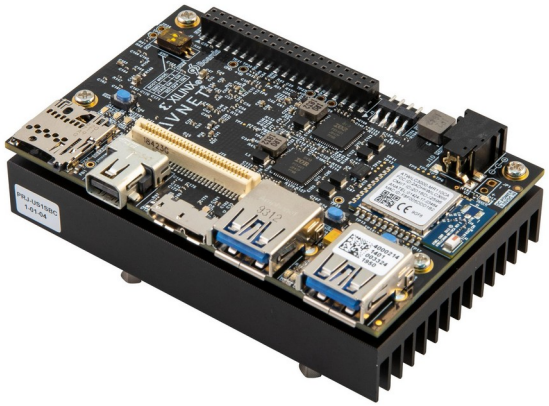
Pourquoi les FPGAs ?

Faible consommation : on utilise seulement les ressources dont on a besoin

Parallélisation : on peut paralléliser les calculs

Flexibilité : on peut le reprogrammer





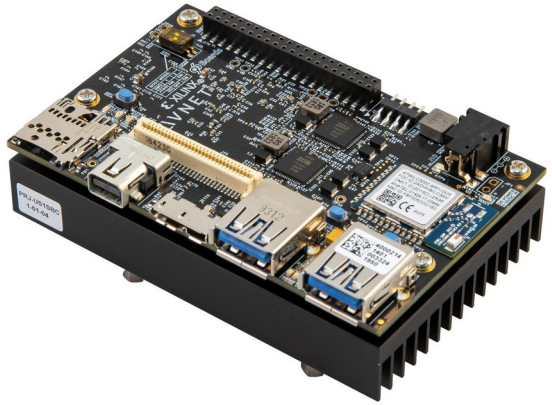
Le Transformer sur FPGA

Les challenge pour déployer des réseaux de neurones sur FPGA

Mémoire

- ZU3EG = 0,95 Mo

solution : compression ou mémoire DDR externe



Complexité des calculs

FPGA = arithmétique de base en entier
- Pas de FPU (Floating Point Unit) sur FPGA (DSP seulement en entier et virgule fixe)

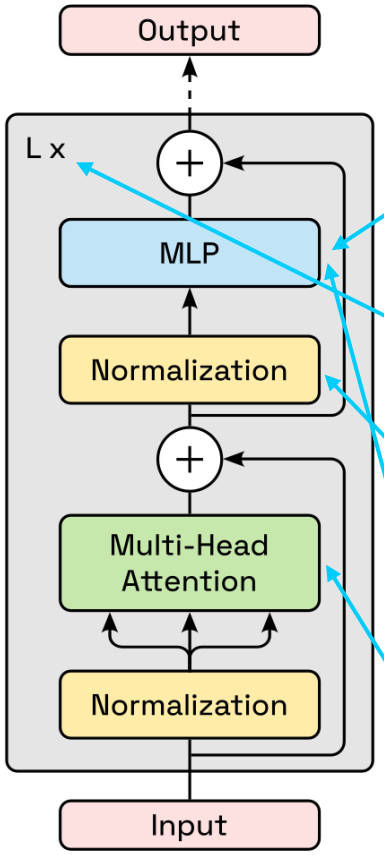
Mais les modèles sont en flottant

solution : quantification en entier ou concevoir ses propre FPUs avec les blocs existant

Le Transformer sur FPGA

Déploiement d'un Transformer, le Vision Transformer, les difficultés

Notre but, déployer un Vision Transformer (ViT)



Les 5 grandes difficultés :

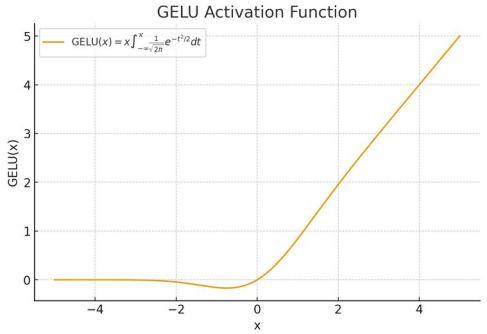
Grande multiplication matricielle → (200x3000) * (3000x800)

Des dizaines de million de paramètres → ViT-B = 80M

LayerNorm : fonction globale + normalisation en "temps réel" + racine carrée + division

GELU : forme analytique très complexe

Softmax : fonction globale + exponentielle + division



$$\mathrm{GELU}(x) = x \cdot \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-t^2/2}\, dt$$

$$\mathrm{LayerNorm}(x) = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta$$

$$\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

Le Transformer sur FPGA

Résumé

Comment faire pour déployer un ViT sur FPGA ?

2 approches :

- entrainement en float16 puis déploiement en float16 sur FPGA avec MACs
  → nécessite un opérateur MAC en float16 sur le FPGA,
- quantification en entier après entrainement, puis déploiement sur FPGA avec MAC en entier
  → DSP déjà là, approximation/quantification des autres fonctions

FloPoCo

FloPoCo

L'idée

FloPoCo (Floating-Point Cores) est une librairie pour générer des opérateurs flottants en VHDL de façon optimisé pour les FPGAs

C'est **open-source**, développé par Florent de Dinechin, professeur à l'INSA Lyon

```
flopoco frequency=200 FPMult wE=8 wF=23
```

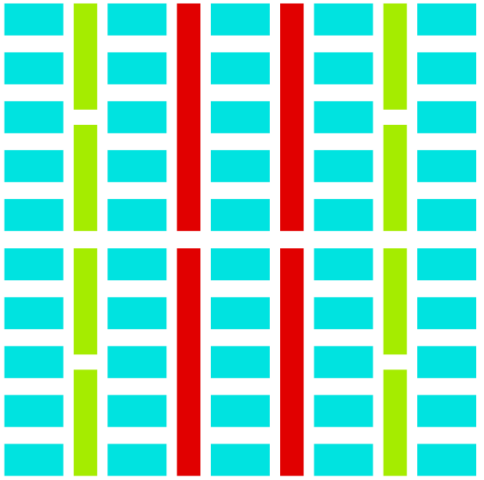
```
entity FPMult_8_23_uid2_Freq200_uid3 is
    port (clk : in std_logic;
          X : in  std_logic_vector(8+23+2 downto 0);
          Y : in  std_logic_vector(8+23+2 downto 0);
          R : out  std_logic_vector(8+23+2 downto 0)    );
end entity;
```

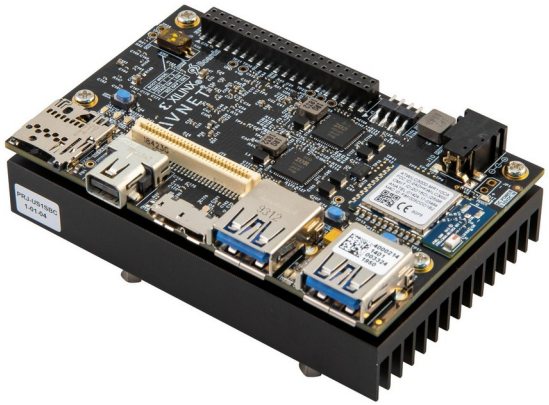
https://flopoco.org/

Opérateur FMA (Fused Multiply Add)

Opérateur flottant pour faire une **MAC** plus rapidement et avec moins de ressources qu'une simple multiplication puis addition

→ L'idée c'est de générer les FMA avec FloPoCo, puis de les instancier dans le FPGA pour faire des MAC en parallèle





Dans un premier temps nous allons instancier une FMA en **float32**

Format	LUT	DSP	FF
Float32	2400	2	450
ZU3EG	70560	360	141120

Problème : 30 MACs → ressources insuffisantes
→ plus gros FPGA, ou quantification plus basse

FMA en **float16**

Format	LUT	DSP	FF
Float16	600	1	250
ZU3EG	70560	360	141120

Problème : La multiplication et l'accumulation sont en float16, mais on a une grosse perte de précision (99.9921 pour MAC de [1….100] et [1….1/100] au lieu de 100]

→ en réalité, il faut toujours accumuler en précision supérieur (float32, float24)

FMA avec multiplication en float16 et accumulation en float32

Format	LUT	DSP	FF
Mixte	1100	1	300
ZU3EG	70560	360	141120

Problème : On utilise toujours beaucoup de ressources (118 MAC max), il reste beaucoup de DSP inutilisés

→ réduire la précision vers float8 ou float4 → nécessite un processus de quantification

FloPoCo

Résumé

Les opérateurs flottants, même optimisés pour les FPGAs, utilisent beaucoup de ressources

→ quantification en entier pour réduire au minimum l'utilisation des ressources

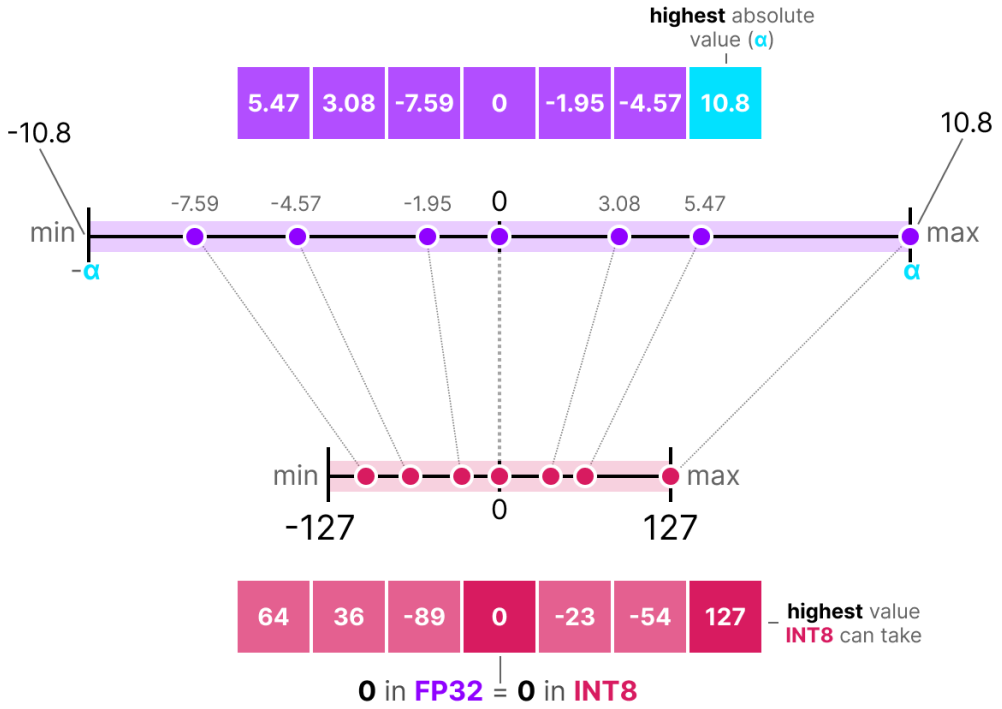
Quantification en entier 8 bits

Quantification en entier 8 bits

Principe

Autre schéma de quantification :
on passe de la simple réduction de précision à un tout nouvel espace de nombre : **l'entier**

Le principe :
- on choisit un format/nombre de bits
- ***on reproduit la dynamique de l'espace flottant dans l'espace entier***



Quantification en entier 8 bits

Avantage pour les FPGA

Avantages :

- stockage des poids divisé par 4
- multiplication et addition sur DSP

Inconvénients :

- étape de quantification, du travail supplémentaire
- perte de précision du modèle

Est-ce que le travail et la perte de précision vallent le coup ?

En général quand on fait du hardware, oui

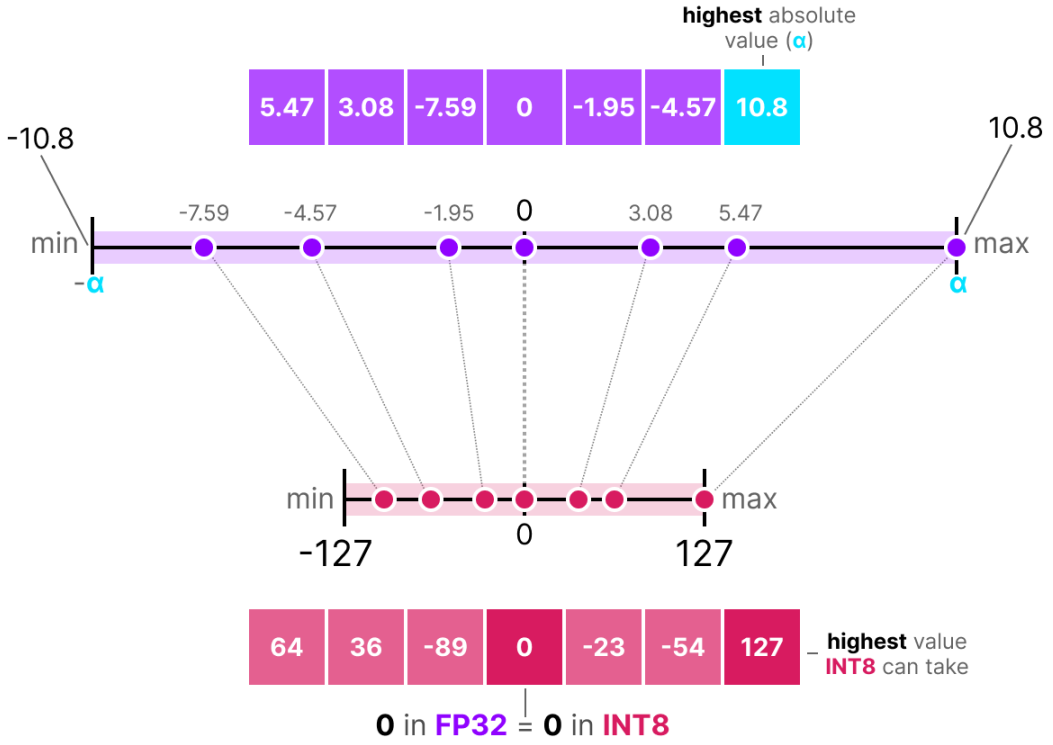
Quantification en entier 8 bits

Quantification statique symétrique après entrainement

Statique : poids et biais quantifiés et facteur d'échelle calculé pré inférence, mêmes poids et facteurs d'échelle quelque soit l'input

Symétrique : quantification autour de 0

Après entrainement (post training) : pas de backward



Quantification en entier 8 bits

Principe

$$A = \begin{pmatrix} 1.5 & -2.3 \\ -0.7 & 3.1 \end{pmatrix}, \quad B = \begin{pmatrix} -3.8 & 0.9 \\ 1.4 & -2.6 \end{pmatrix}$$

$$\begin{pmatrix} 61 & -94 \\ -29 & 127 \end{pmatrix}_{\text{int8}} \qquad \begin{pmatrix} -127 & 30 \\ 47 & -87 \end{pmatrix}_{\text{int8}}$$

$$s_A = \frac{\max|A|}{127} = \frac{3.1}{127} \approx 0.02441$$

$$s_B = \frac{\max|B|}{127} = \frac{3.8}{127} \approx 0.02992$$

$$\text{acc}_{\text{int32}} = \begin{pmatrix} -12165 & 10008 \\ 9652 & -11919 \end{pmatrix}$$

$$Y = \text{acc}_{\text{int32}} \times s_A \times s_B$$

$$Y \approx \begin{pmatrix} -8.89 & 7.31 \\ 7.05 & -8.71 \end{pmatrix}_{\text{float32}}$$

$$A \times B = \begin{pmatrix} -8.93 & 7.33 \\ 7.00 & -8.69 \end{pmatrix}$$

Quantification en entier 8 bits

FMA précision mixte

Implémentation de MAC en int8 sur FPGA :

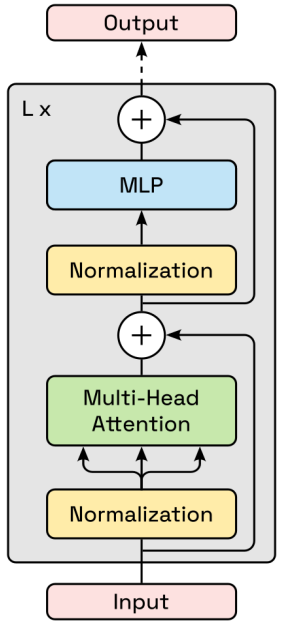
Format	LUT	DSP	FF
Int8	20	1	10
ZU3EG	70560	360	141120

→ On a résolu le problème des ressources (256 MACs largement possible), on arrive à maximiser les unités arithmétique du FPGA

Quantification en entier 8 bits

Un cas d'exemple : Vision Transformer on CIFAR10

Petit ViT (9,5M) + petit dataset (CIFAR10) = entrainement court = **développement plus rapide**
Baseline : 81,5% - 500 epochs → Quantification int8 : -0.02 % (Projection Linéaire seulement)



$$\mathrm{GELU}(x) = x \cdot \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-t^2/2}\, dt$$

$$\mathrm{LayerNorm}(x) = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta$$

$$\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

CIFAR10 : https://www.cs.toronto.edu/~kriz/cifar.html

Quantification en entier 8 bits

GELU

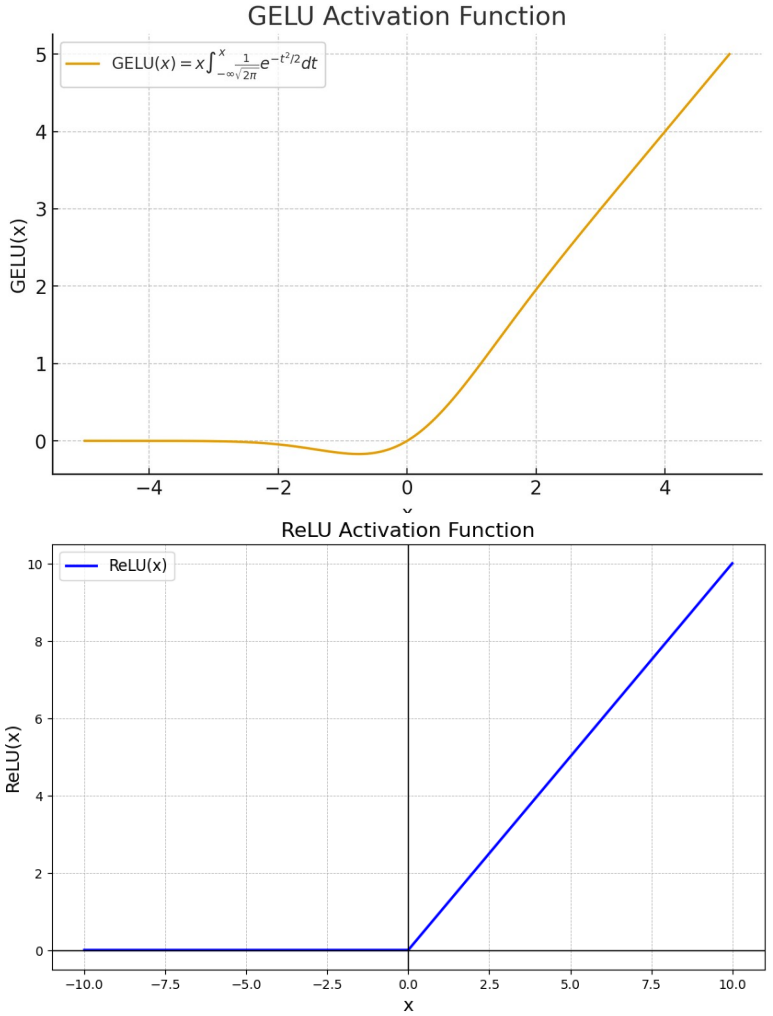
2 choix possibles :

- remplacement par **ReLU** → nécessite un réentrainement léger
- approximation par **LUT**

LUT (Look up table) :

→ int8 : 256 valeurs possibles en entrée, 256 valeurs possibles en sortie

→ très léger à stocker et très rapide sur FPGA (BRAM avec addressage)



Quantification en entier 8 bits

LayerNorm and Softmax

Plusieurs problèmes :

- fonctions globales
- division et racine carrée pour LayerNorm
- modèles très sensibles

Softmax sur FPGA, mais latence élévée car globale

$$\text{LayerNorm}(x) = \frac{x - E[x]}{\sqrt{Var[x]} + \epsilon} * \gamma + \beta$$

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

<u>Solutions retenues :</u>

- remplacement par des **fonctions élément à élément** (comme ReLU)
- fonctions **facilement déployable sur FPGA et/ou approximable via LUT**

Quantification en entier 8 bits

LayerNorm → DyT/harDyT

Remplacement de LayerNorm par des fonctions élément à élément avec seulement quelques époques (5) et quelques échantillons du dataset originel (20000)

$$\mathrm{LayerNorm}(x) = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta$$

$$\mathrm{DyT}(x_i) = \gamma_i \cdot \tanh(\alpha_i x_i) + \beta_i$$

$$\mathrm{harDyT}(x_i) = \gamma_i \cdot \mathrm{hardtanh}(\alpha_i x_i) + \beta_i$$

Questions : comment quantifier ces nouvelles fonctions ?
- **harDyT** : très facile
- **DyT** : approche LUT de tanh

[1] L. Gauthier, F. Bony, and J. Perchoux, “harDyT: A Post-Training Hardware-Aware LayerNorm Replacement,” accepted ISCAS 2026, 2026

[2] J. Zhu, X. Chen, K. He, Y. LeCun, and Z. Liu, “Transformers without Normalization,” 2025

Quantification en entier 8 bits

Softmax → ConSmax

Même idée

MAIS nécessité d'entrainer le modèle de 0

+ Instabilité à l'entrainement
+ performances légérement inférieures

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

$$ConSmax(S_i) = \frac{e^{S_i - \beta}}{\gamma}$$

[3] S. Liu et al., "ConSmax: Hardware-Friendly Alternative Softmax with Learnable Parameters," IEEE/ACM International Conference on Computer-Aided Design, 2024

Quantification en entier 8 bits

hardViT

HardViT : hardware-aware ViT

Seulement 2 Opérations :

- MatMul (projections linéaires, attention et values)
- LUT (GELU, harDyT et ConSmax)

Quantification :

Statique symétrique int8, facteurs d'échelle flottants

Reste à :

- Facteur de scale flottant en virgule fixe ou en dyatic
- Déploiement sur FPGA

Discussions

Discussions

ConSmax :

- Améliorer les performances du modèle
- Remplacement après entrainement

hardViT :

- Tester sur les ViTs originaux (ImageNet)
- Valider sur d'autres architecture transformer et d'autre tâches et domaines (segmentation, time series)
- Améliorer le schéma de quantification
- Fiabilité de la quantification

Hardware :

- Finaliser l'architecture hardware
- Convertir les facteur d'échelle flottant en représentation hardware-aware

Time Series :

- HardViT pour l'analyse de time series pour la détection de micro-particules

Des questions ?

07/04/2026

-

Déploiement efficace des Transformers sur FPGA : des stratégies de quantification à l'architecture adaptée au matériel

(OASIS) Lubin GAUTHIER
lubin.gauthier@toulouse-inp.fr

Thesis director
(OASIS) Julien PERCHOUX
Co-director
(OASIS) Francis BONY