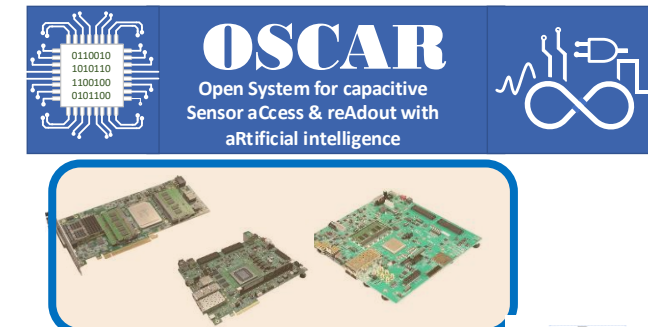
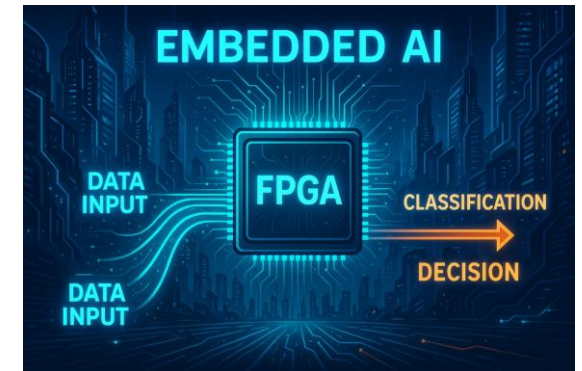
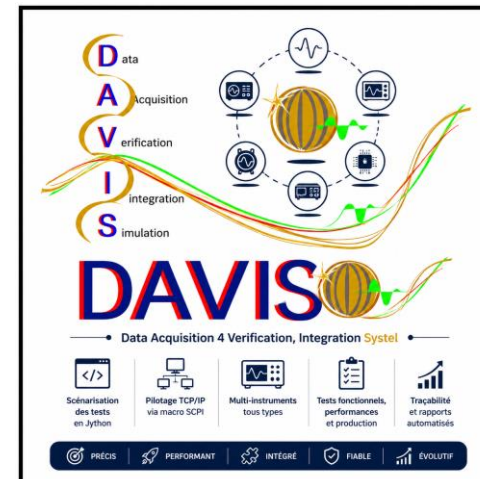
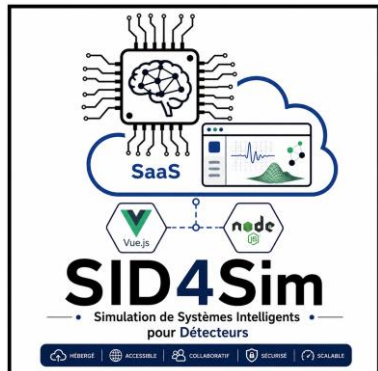
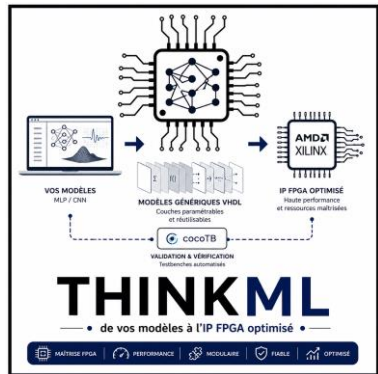
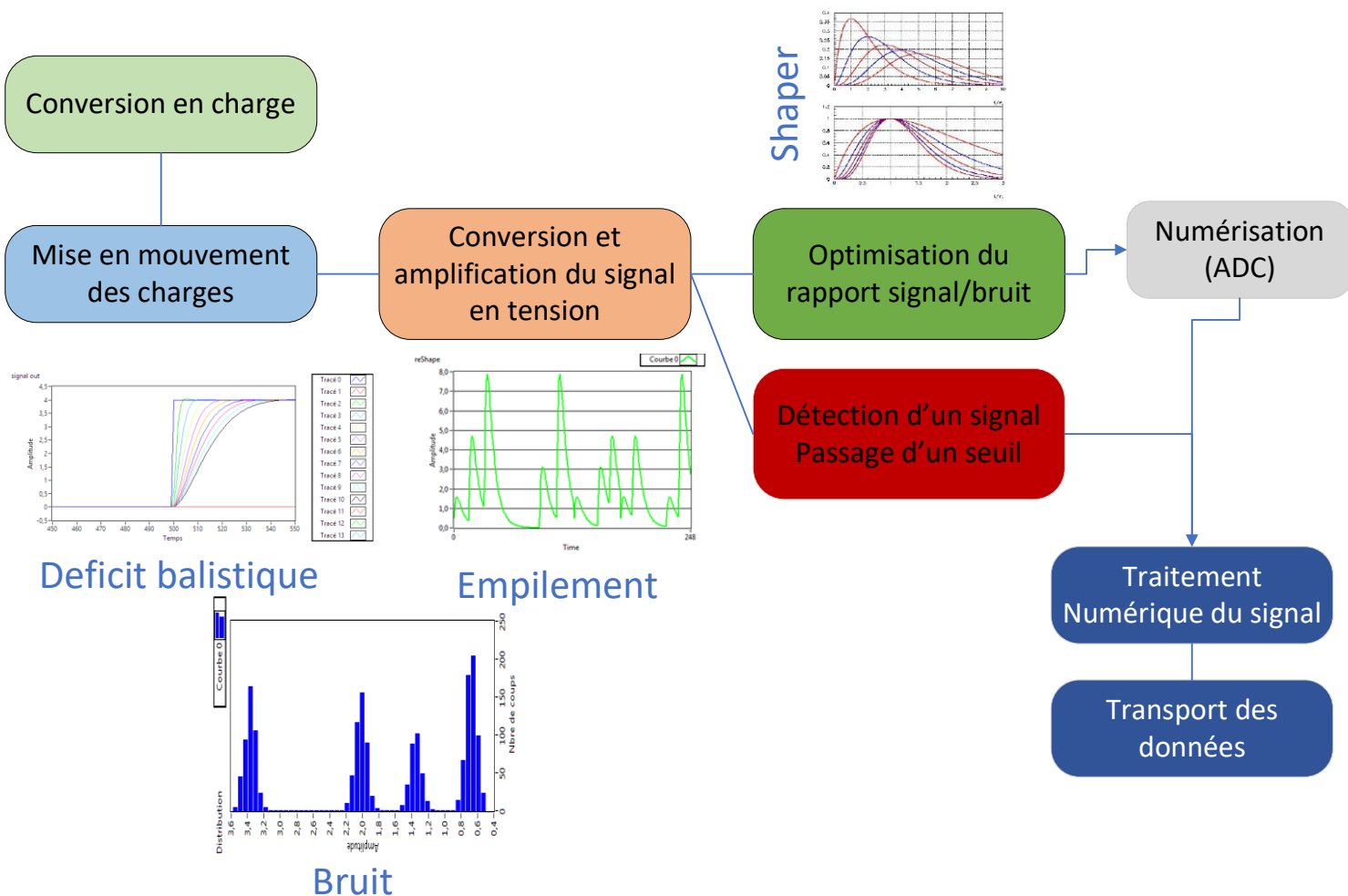


Inférer des Modèles de Réseaux de Neurones sur Systèmes Embarqués – – Applications sur AMD Xilinx – *Traitement de signaux de détecteurs pour la physique*



1. *Problématique: Electronique front-end*
2. *Objectifs de la R&T THINK*
3. *Etats des lieux de l'IA embarqué*
4. *L'inférence optimisée avec THINKML*
5. *Modélisations systèmes et générations de données*
6. *Test automatisé du modèle embarqué sur FPGA*
7. *Perspectives*

➔ DEPUIS 50 ans, on traite les signaux de détecteurs capacitifs de la même façon.



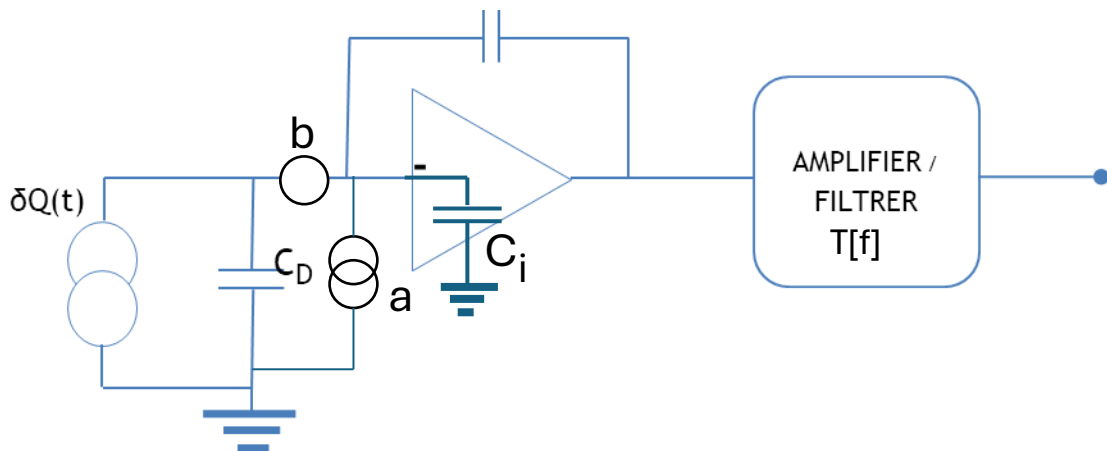
Les erreurs de mesures irrésolues

- L'empilement
- Le déficit balistique
- Le cross-talk
- Le bruit du détecteur ou du capteur
- Le bruit de la chaîne électronique de mesure

$$\sigma_{\text{total}}^2 = \sigma_{\text{det}}^2 + \sigma_{\text{pu}}^2 + \sigma_{\text{ba}}^2 + \sigma_{\text{elec}}^2$$



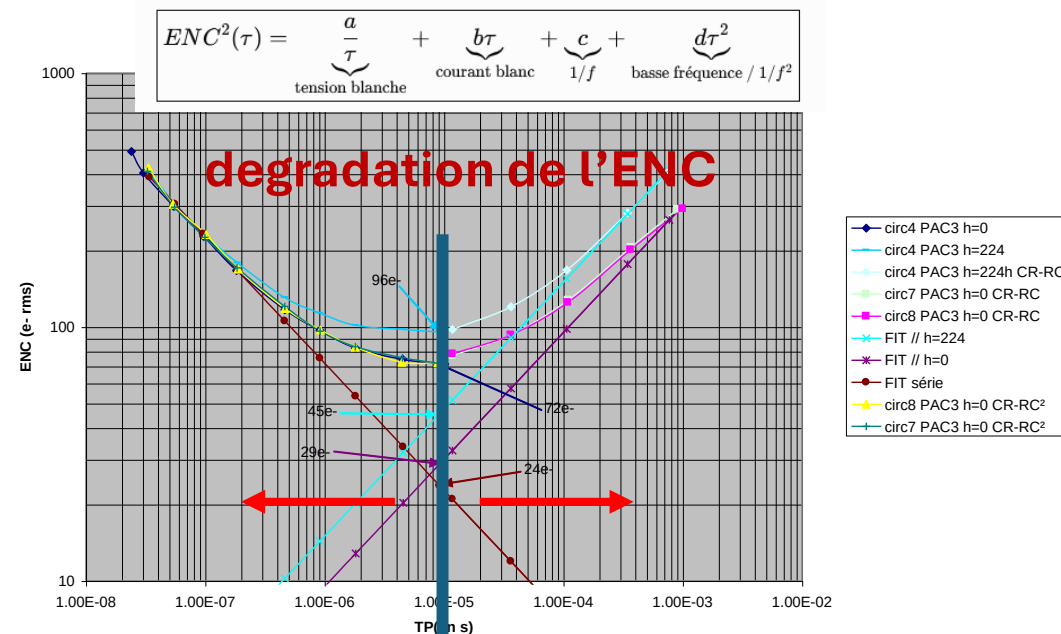
L'électronique est alors optimisée pour une seule application, un type de détecteur (Semi-conducteur, Det. Gazeux, PMT,..)



$$ENC = \frac{\left[\frac{1}{2\pi} \int_{-\infty}^{+\infty} \left(a(C_d + C_i)^2 + \frac{b}{w^2} \right) \cdot |T(w)|^2 \cdot dw \right]^{1/2}}{\text{MAX} \left\{ L^{-1} \left[\frac{T(s)}{s} \right] \right\}}$$

L'objectif est de déterminer $T(s)$ afin de minimiser ENC.

Utilisation de fonctions linéaires ou légèrement non linéaires (discriminateur)
 ➔ Limité en terme de solutions



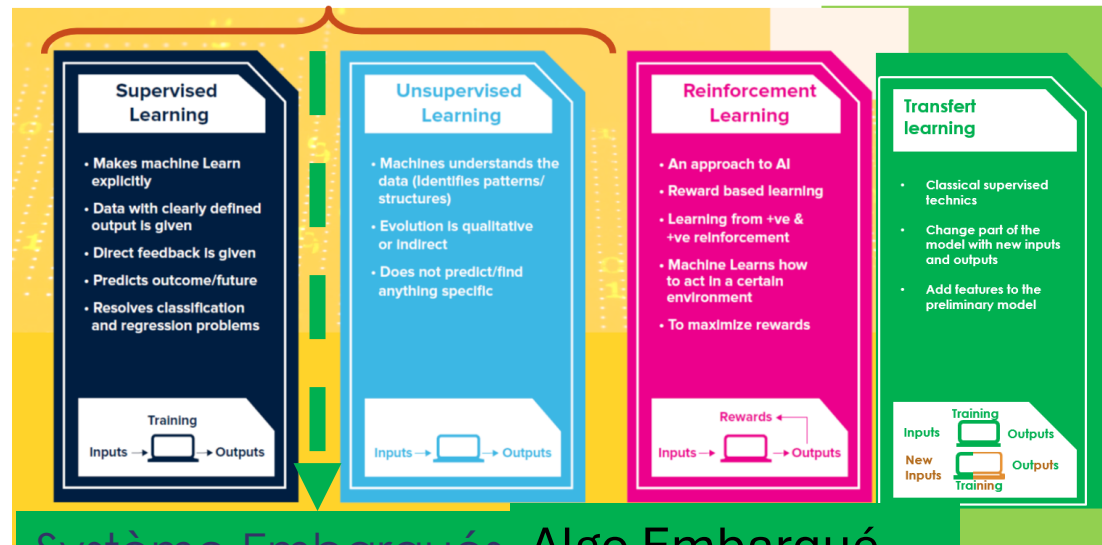
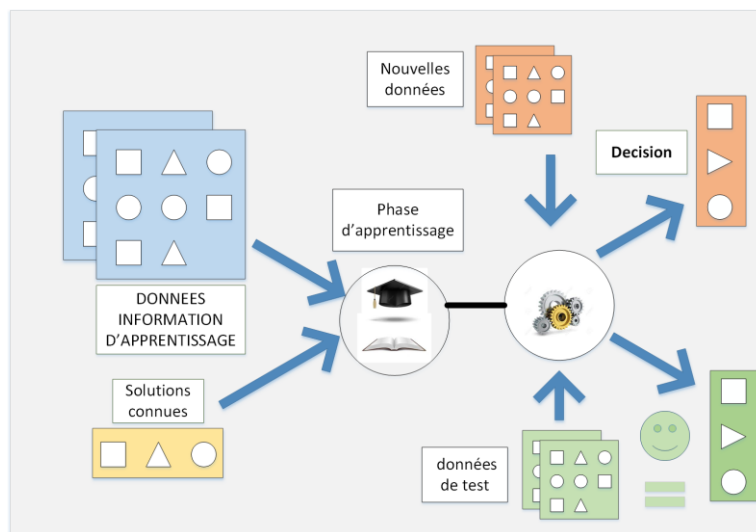
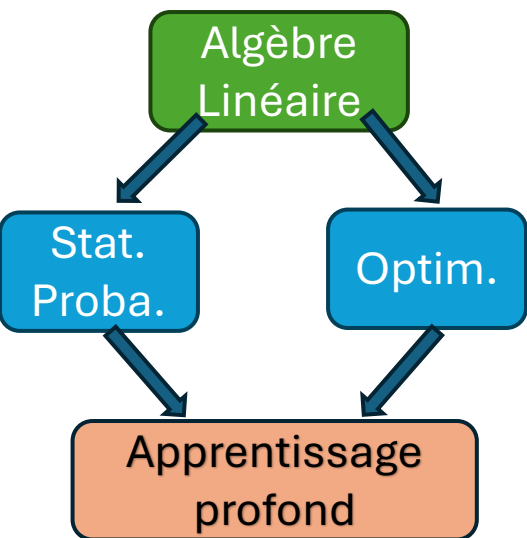
SCIENCES
EXPERIMENTALES

SCIENCES
THEORIQUES

SCIENCES
CALCULATOIRES

SCIENCES PILOTEES PAR LES DONNEES

SCIENCES PILOTEES PAR
L'INTELLIGENCE ARTIFICIELLE



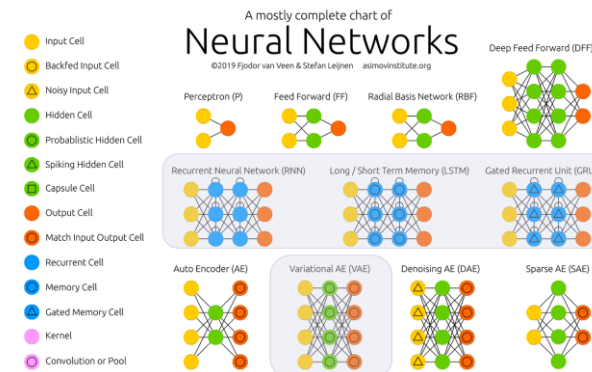
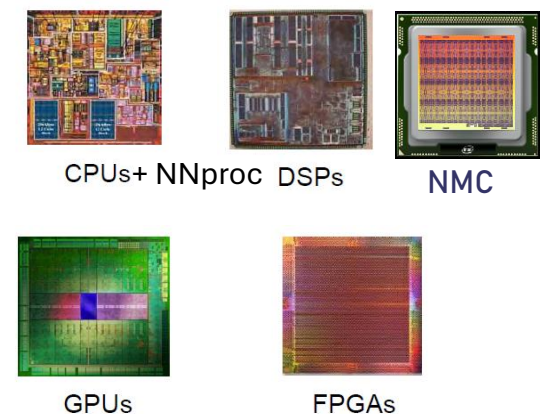
Système Embarqués Algo Embarqué

L'intelligence Artificielle dans les systèmes embarqués

Nouvelles architectures matérielles

Outils de générations de données

Réseaux de neurones et descente de gradient



2020-2026: Test Hardware Infering Neural Network

Phase 1: 2020-2023

- Formation des ingénieurs à ces techniques
- Faisabilité de portage sur plusieurs types de hardware
- Exploration des problématiques solvables par Réseau de neurones
- Exploration des performances
- Exploration des outils existants
- Restitution d'informations à la communauté

LP2I Bordeaux

LLR Palaiseau

CPPM Marseille

IP2I Lyon

IPHC Strasbourg

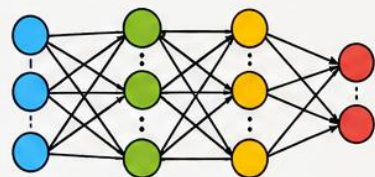
Phase 2: 2024-2026

- ☐ ***Recherche des modèles de DNN pour répondre au besoin d'expériences dans nos instruments.***
- ☐ ***Déploiement d'exemples en openSource sur Git***
- ☐ ***Mise en commun des connaissances et de l'expérience de chacun.***
- ☐ ***Développer méthodes et outils adaptés à nos problématiques (améliorer la mesure)***

Base de connaissances

Expériences Pratiques

APPRENTISSAGE PROFOND

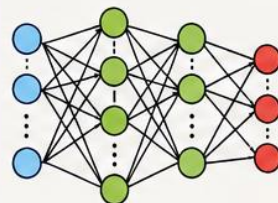


MODÈLES BASÉS SUR LES RÉSEAUX DE NEURONES

- Apprend des représentations hiérarchiques
- Gère des motifs complexes et non linéaires
- Approche guidée par les données

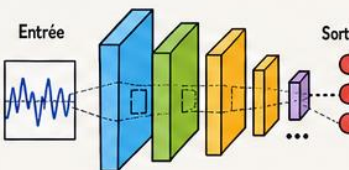
TYPES DE RÉSEAUX DE NEURONES

MLP (Perceptron Multicouche)



- Couches entièrement connectées
- Utilisé pour des données tabulaires, régression, classification

CNN (Réseau de Neurons Convolutif)



- Les couches convolutives extraient des caractéristiques locales
- Partage des poids et pooling réduisent le nombre de paramètres
- Idéal pour les signaux, images, séries temporelles

ALGORITHME DE DESCENTE DE GRADIENT



- Initialiser les paramètres θ
- Calculer le gradient $\nabla L(\theta)$
- Mettre à jour les paramètres
- Répéter jusqu'à convergence

FONCTION DE PERTE

Mesure l'erreur entre la prédiction $\hat{y}$ et la cible y

Choix courants :

- Erreur quadratique moyenne (MSE)

$$L = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

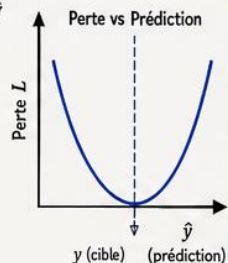
- Erreur absolue moyenne (MAE)

$$L = \frac{1}{N} \sum_{i=1}^N |\hat{y}_i - y_i|$$

- Perte de Huber (robuste aux valeurs aberrantes)

$$L_{\delta}(e) = \begin{cases} \frac{1}{2} e^2 & \text{si } |e| \leq \delta \\ \delta(|e| - \frac{1}{2} \delta) & \text{sinon} \end{cases}$$

où $e = \hat{y} - y$



Objectif : minimiser la perte en ajustant les paramètres du modèle θ

LE NEURONE

Équation

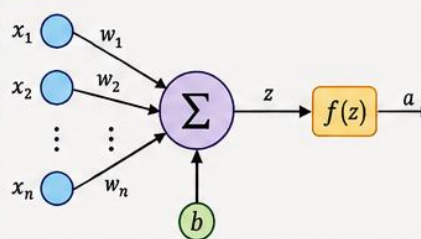
$$z = \sum_{i=1}^n w_i x_i + b$$

$$a = f(z)$$

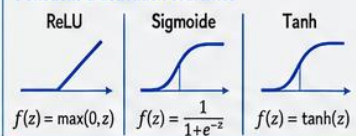
où :

x_i : entrées
 w_i : poids
 b : biais
 z : combinaison linéaire
 a : sortie (activation)
 f : fonction d'activation

Schéma



Fonctions d'activation courantes



Exemple

$$z = 0.5x_1 - 1.2x_2 + 0.3x_3 + 0.7$$

$$a = \text{ReLU}(z) = \max(0, z)$$

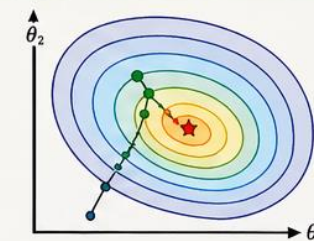
DESCENTE DE GRADIENT : DÉTAILS

1. Calculer le gradient

Trouver le gradient de la perte par rapport aux paramètres θ

$$g_t = \nabla_{\theta} L(\theta_t)$$

Indique la direction de la plus grande augmentation de la perte



2. Mettre à jour les paramètres

Se déplacer dans la direction opposée au gradient

$$\theta_{t+1} = \theta_t - \eta g_t$$

η : taux d'apprentissage (taille de pas)

3. Répéter

Itérer jusqu'à convergence ou nombre maximal d'itérations



L'apprentissage profond combine des modèles puissants (MLP, CNN) avec des techniques d'optimisation (comme la descente de gradient) pour apprendre à partir des données en minimisant une fonction de perte.

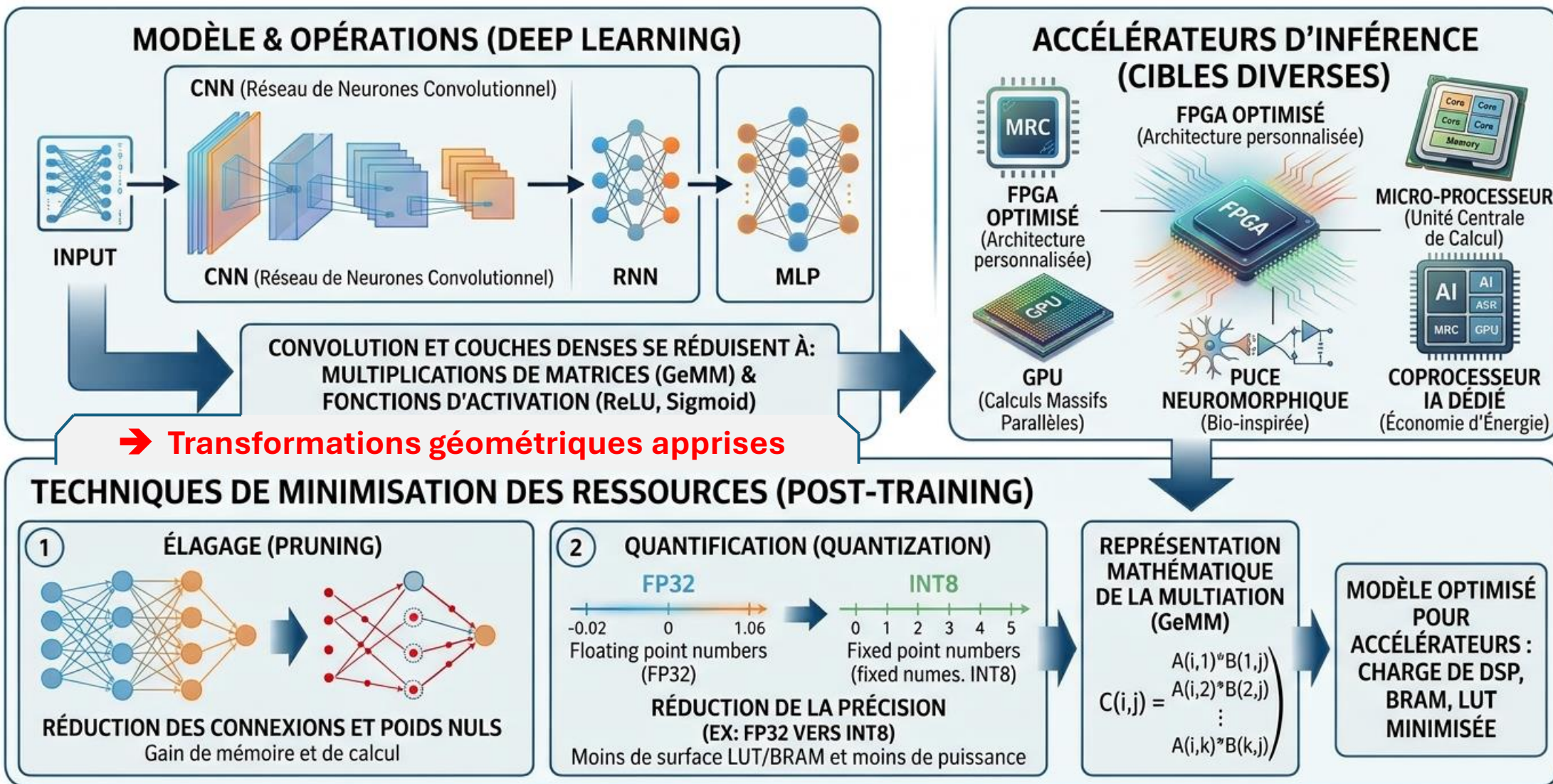


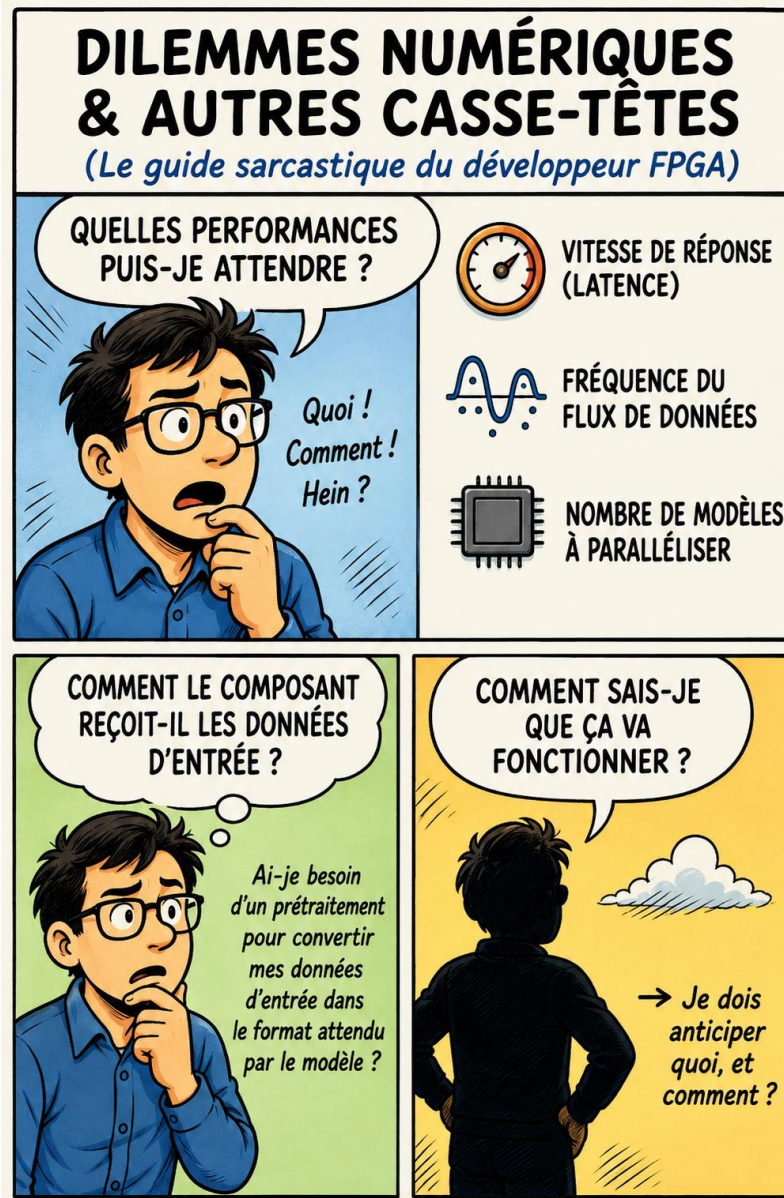
Objectif : trouver les paramètres θ^* qui minimisent la perte

$$\theta^* = \arg \min_{\theta} L(\theta)$$


Bon modèle = faible perte sur les données d'entraînement
 Bonne généralisation = faible perte sur de nouvelles données

OPTIMISATION DE L'INFÉRENCE IA POUR UN MATÉRIEL ÉTENDU (CNN -> MLP/Matrice -> ACCÉLÉRATEURS -> RESSOURCES)





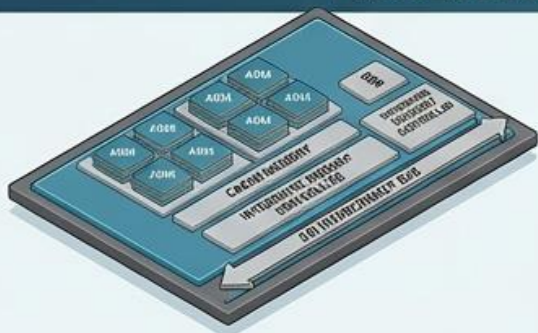
Objectifs de la R&T THINK (SID-MIT)



COMPARISON DES ARCHITECTURES DE CALCUL : PROCESSEUR À PUCE NEUROMORPHIQUE

UNITÉS DE TRAITEMENT GÉNÉRALISTES & DÉDIÉES

PROCESSEUR (CPU)



UNITÉ CENTRALE DE CALCUL (CALCULS SÉRIELS ET COMPLEXES)

Traitement de l'OS et des applications générales, flexible mais moins efficace pour les calculs massifs.

GPU (GRAPHICS PROCESSING UNIT)

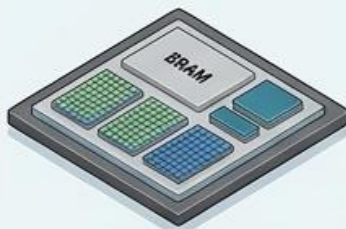


CALCULS MASSIFS PARALLÈLES (GRAPHISMES ET CALCUL SCIENTIFIQUE)

Optimisé pour exécuter la même opération sur de grandes quantités de données simultanément, efficace pour le deep learning.

ACCÉLÉRATEURS IA & ARCHITECTURES SPÉCIALISÉES (POST-TRAINING ET INFÉRENCE)

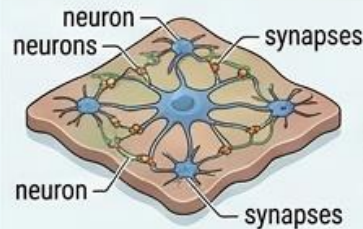
1 COPROCESSEUR IA DÉDIÉ (PUCE IA / NPU)



ACCÉLÉRATION SPÉCIFIQUE DE L'INFÉRENCE IA

Bancs de registres massifs, multiplication de matrices (GeMM) à ultra-haute vitesse. Moins de consommation, plus de performance par Watt.

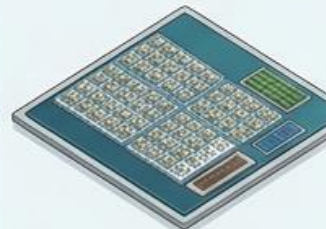
2 PUCE NEUROMORPHIQUE (BIO-INSPIRÉE)



TRAITEMENT ÉVÉNEMENTIEL (SPIKING NEURAL NETWORKS)

Simule le fonctionnement du cerveau, très faible consommation, efficace pour les applications temps réel et la détection d'événements.

3 FPGA (FIELD-PROGRAMMABLE GATE ARRAY)



MATÉRIEL RECONFIGURABLE (PROGRAMMATION DE PORTES LOGIQUES)



Permet de concevoir des pipelines matériels dédiés pour des algorithmes spécifiques, très faible latence, haute performance, reconfigurable après fabrication.

MODÈLES OPTIMISÉS : CHOISIR L'ARCHITECTURE LA PLUS ADAPTÉE (PERFORMANCE VS CONSOMMATION VS FLEXIBILITÉ)

2

In development

~~CPU~~

AI Model inferred in
programmation logic
(DSP, BRAM, LUT)

Spatial Accelerators

Based on hardware functions dedicated to
calculus without software

ASIC, Neuromorphic engines, FPGA

VIVADO HLS

HLS4ML

FiNN (Xilinx R&D)

ChaiDNN

HADDOC2

DNNWeaver

...

Spatial Accelerators

1 ns

1 μ s

10 μ s

1 ms

100 ms

10 s

AI Edge computing

1

Standard design

CPU

Co-processeur de
calcul annex (PE)

Edge computing

Based on software

MMPA, GPU, FPGA
PU redesigned for AI (TPU,
KPU...)

TensorFlowLite

Ax8i (max78000)

ST Microelectronics

CubeIDE + Cube-AI

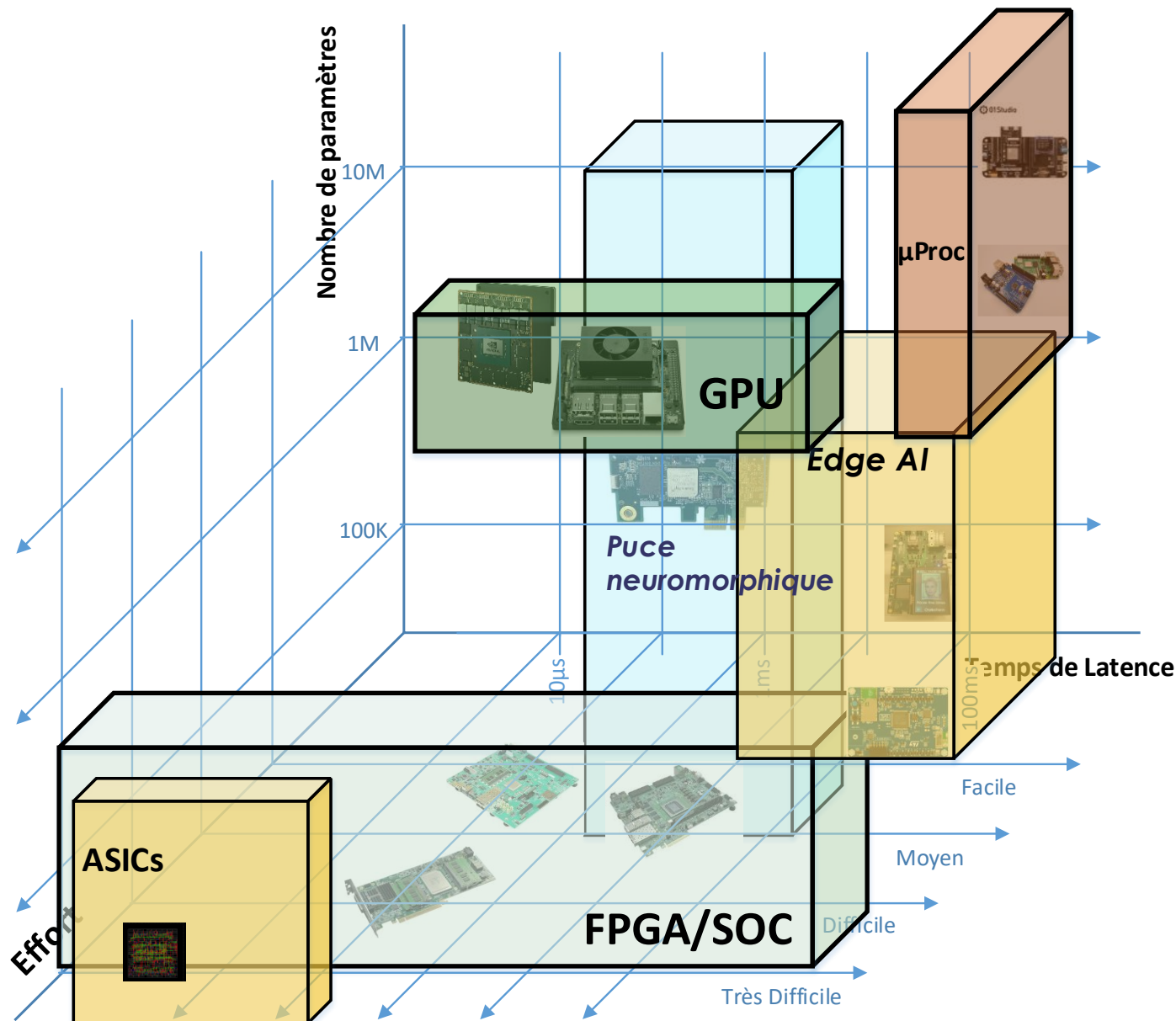
VITIS AI (AMD)

OpenVino (Intel)

N2D2 – DeepGreen

Eclipse Aidge

Edge Impulse (société)



1. FPGA le plus versatile pour les petits modèles
2. GPU utile en backend (calcul matriciel)
3. ASIC pour nos applications
4. Les autres pour applications video et sons.

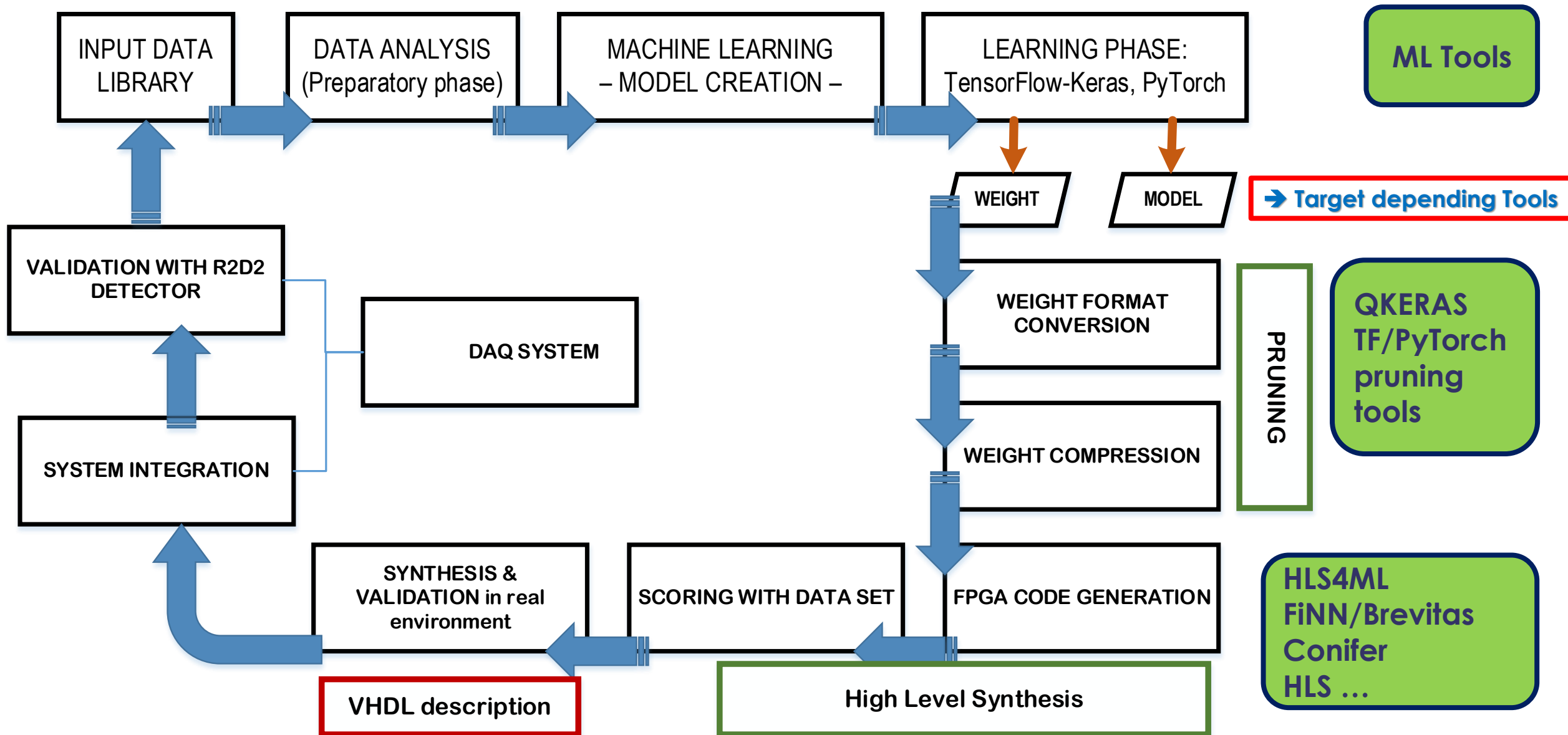
Critères	CPU	CPU+AI-CoPro	GPU	FPGA	ASIC
Adaptabilité (à différents modèles)					
Puissance de calcul					
Latence			(μ s) 	(100 ns) 	
Flux d'entrées					
Parallelisme					
Consommation électrique					
Mise en œuvre facile					
Densité de modèles (plusieurs canaux)					

-

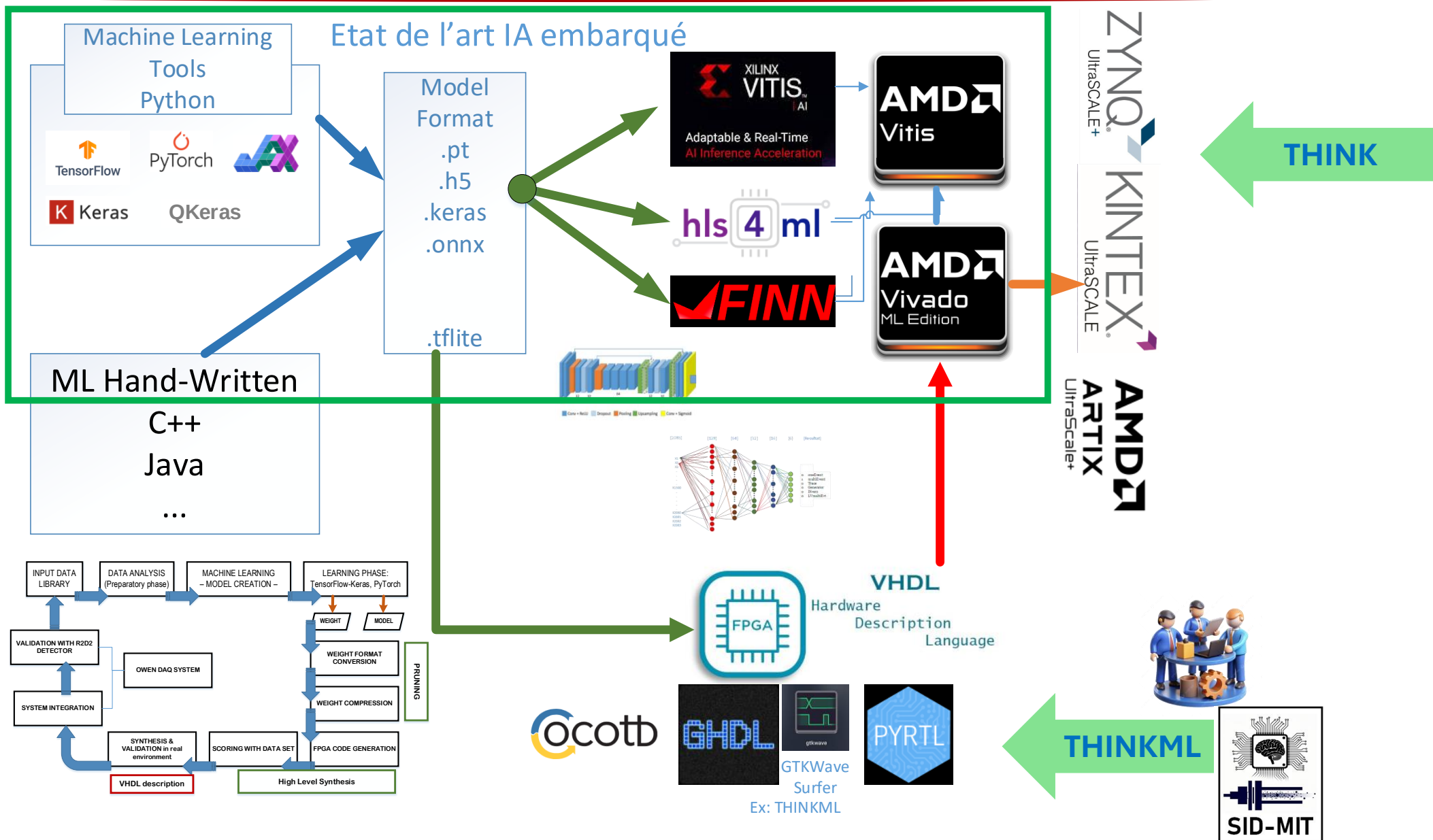
Complexité

+

++



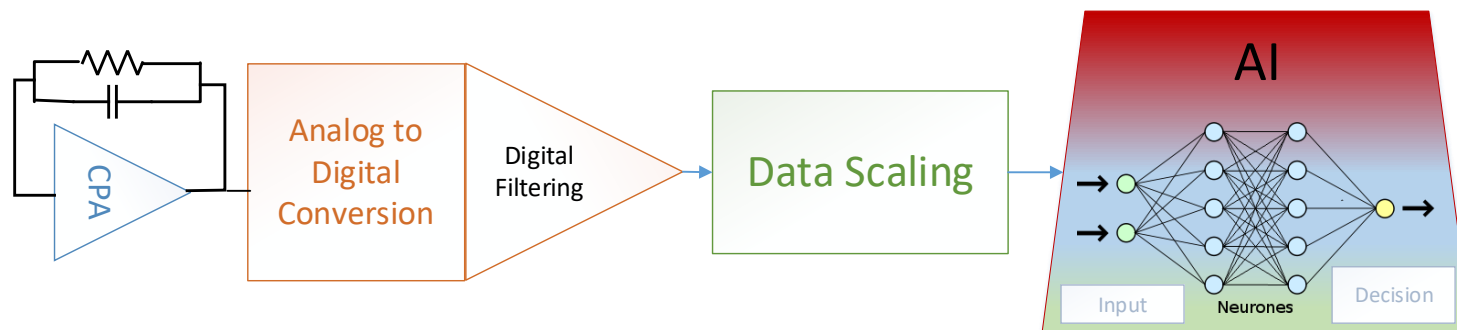
L'inférence optimisée sur cible embarquée FPGA Xilinx





- Jocelyn Domange (DéTECTeurs)
- Bruno Mansoux (Logiciel embarqué)
- Thomas Chabaud (Mesures et câblages)
- Frédéric Druillole (FPGA, Analyse et modélisation)
- stagiaires

→ TNS pour avoir une mesure parfaite → Utilisation de modèles de Réseaux de Neurones (MLP, CNN, RNN)

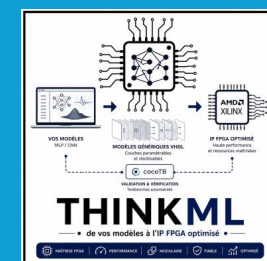


- **Gestion des Ressources FPGA versus Temps de calcul:**
 - Nbre de DSP
 - Parallelisme des operations
 - **Gestion de la Latence versus Ressources FPGA**
- Modèles sur petit FPGA Artix-7, Zynq-xcz7020
- Temps de calcul < 1 ms (objectifs < qq 10 µs)

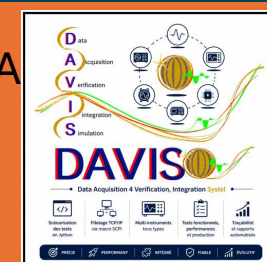
Modélisation
TNS –
Apprentissage –



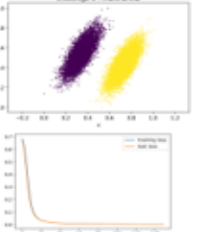
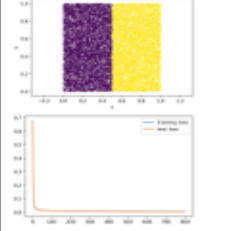
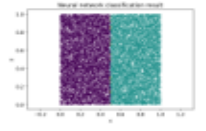
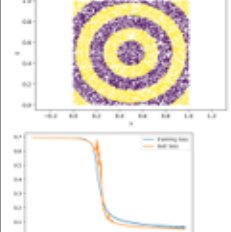
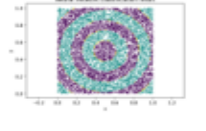
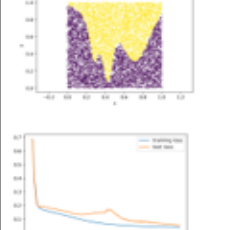
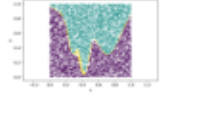
Inférence des
modèles IA



Test des modèles IA
Scénarii
Analyse



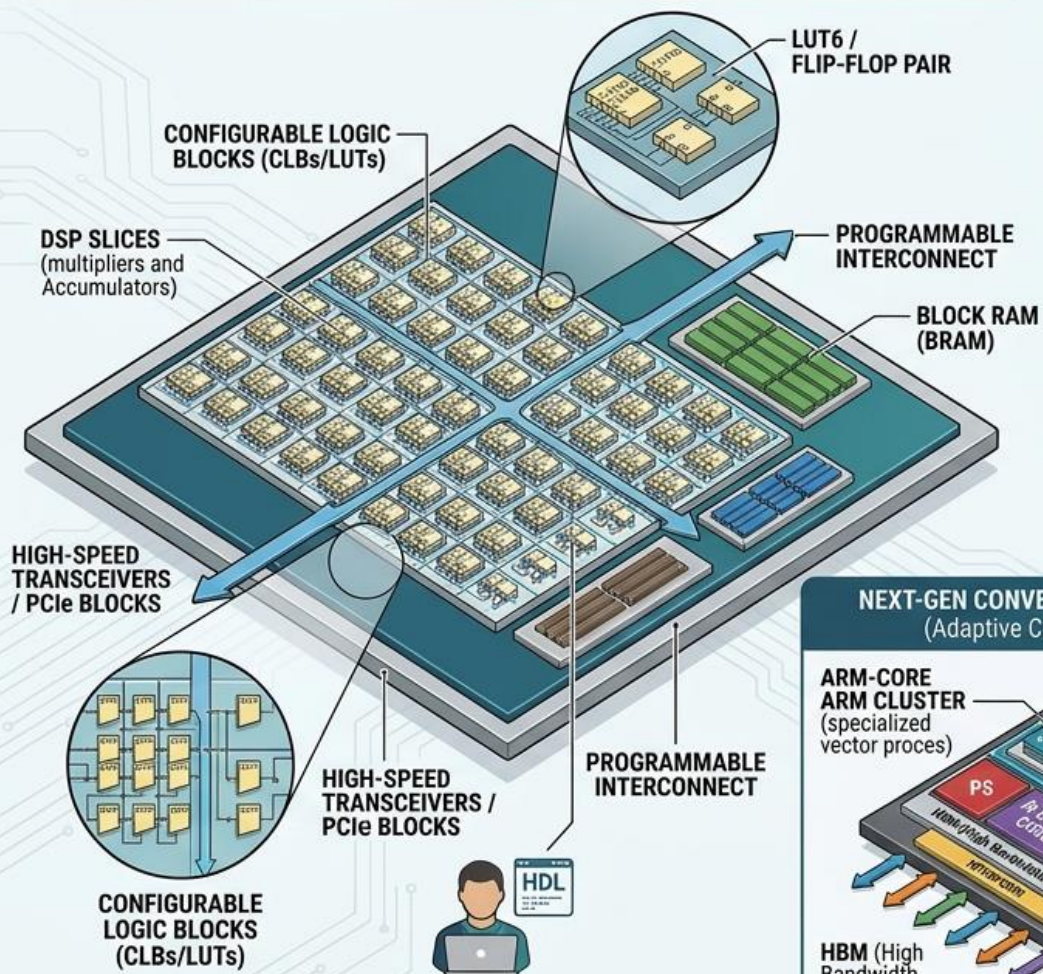
Resultats → Comparaison entre composants

Input/Output	Model	Performances
Challenge 1 (CH1) 	Epoch 149/150: train loss: 0.002179 test loss: 0.002273 Layers: 2x8x1 Parameters: 33	
Challenge 3 (CH3) 	Epoch 799/800: train loss: 0.003657 test loss: 0.001825 Layers: 2x200x100x50x1 Parameters: 24582	Bad: 17/10000 
Challenge 4 (CH4) 	Epoch 1199/1200: train loss: 0.065118 test loss: 0.053723 Layers: 2x1000x500x250x100x50 Parameters: 659002	Bad: 252/10000 
Challenge 5 (CH5) 	Epoch 499/500: train loss: 0.039056 test loss: 0.054641 Layers: 2x500x250x100x50x1 Parameters: 156101	Bad: 330/10000 

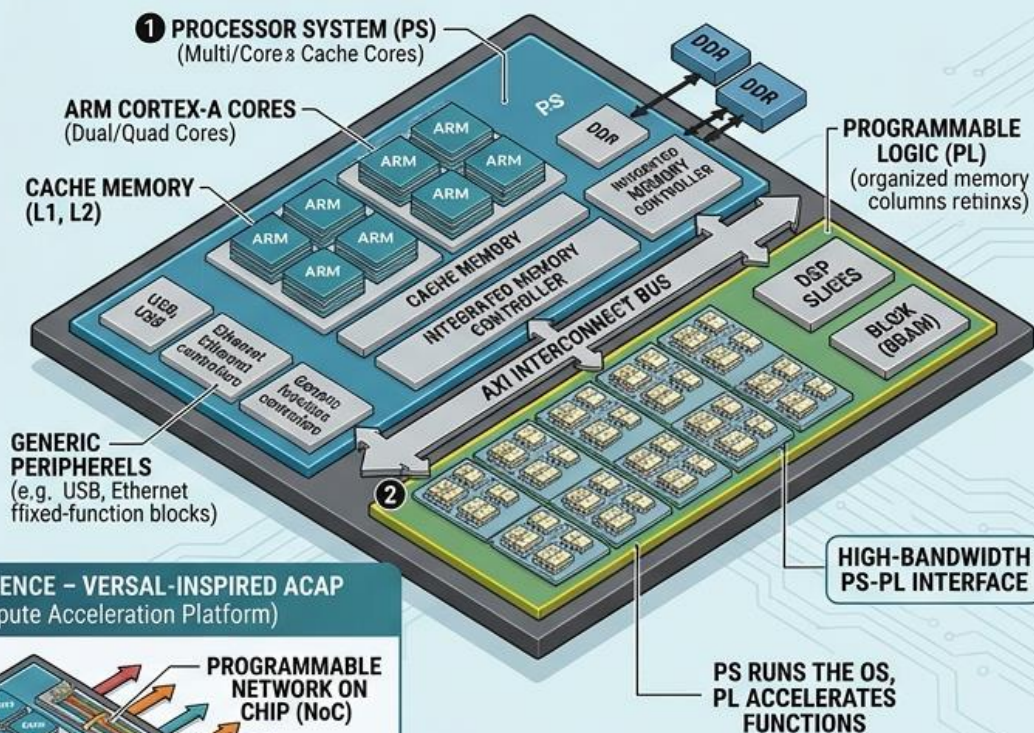
		CH1	CH3	CH4	CH5
Components	Inference Tools	BATCH Size = 1	BATCH Size = 1	BATCH Size = 1	BATCH Size = 1
GPU Nano-2GB-4-CPU-1.5GHz	onnx runtime	15710 FPS	11192 FPS	1587 FPS	5264 FPS
GPU Nano-2GB-4-CPU-1.5GHz	TORCH	2844 FPS	1375 FPS	310 FPS	792 FPS
GPU Nano-2GB-4-CPU-1.5GHz	Tensor-RT	419 FPS	490 FPS	319 FPS	956 FPS
Soc Zynq ZCU102	Tensorflow + HLS4ML + VITIS HLS + VIVADO		398406 FPS		999001 FPS
Soc Zynq ARTY Z7	Tensorflow + HLS4ML + VITIS HLS + VIVADO	14285714 FPS	50495 FPS		
FPGA ARTY Nexys 100T	Tensorflow + HLS4ML + VITIS HLS + VIVADO		10158 FPS		
STM32 Nucleo-F411	Tensorflow Lite + Cube-AI		17 FPS		
DE-10- AGILEX AI Edge	Intel FPGA AI Suite Just In Time Compilation	10535 FPS	18489 FPS	14053 FPS	18662 FPS
DE-10- AGILEX AI Edge	Intel FPGA AI Suite Ahead in Time compilation	11128 FPS	10294 FPS	8400 FPS	9596 FPS
DE-10- AGILEX AI Edge	Tensorflow + Intel HLS non optimisé	5454545 FPS	9383 FPS	365 FPS	
DE-10- AGILEX AI Edge	Tensorflow + Intel HLS optimisé	240000000 FPS	2376238 FPS		

VISUALIZING ARCHITECTURES: FPGA vs. SOC

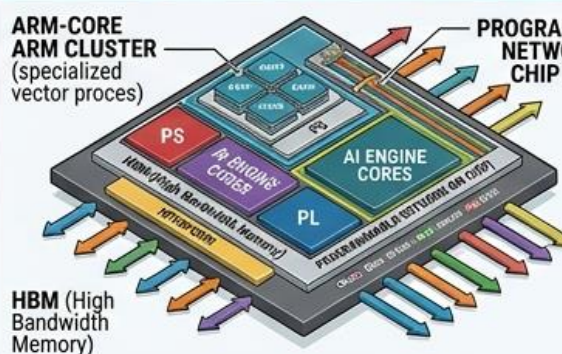
FPGA (Field-Programmable Gate Array) - INSPIRED BY KINTEX



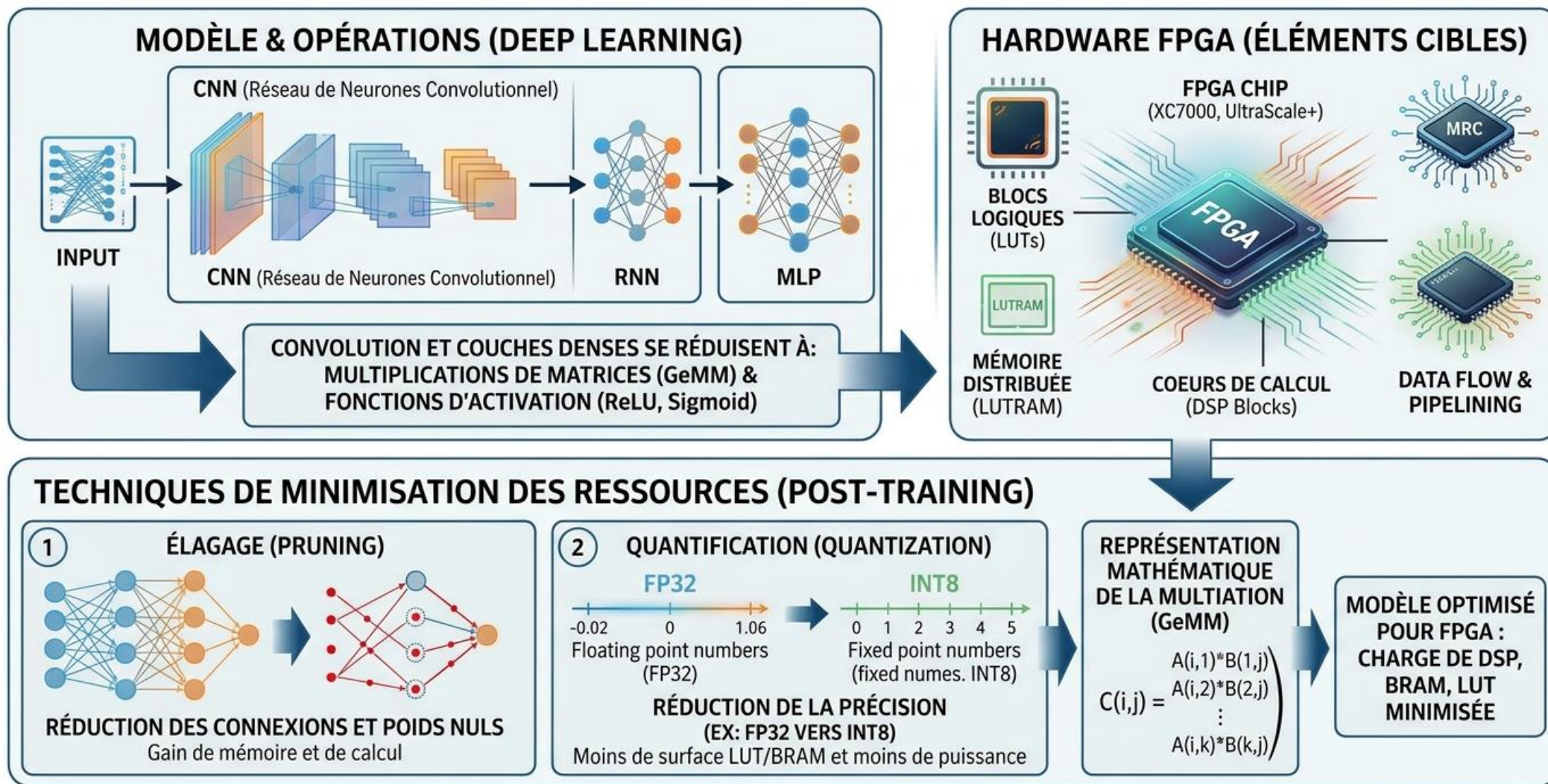
SoC (System-on-Chip) - INSPIRED BY ZYNQ-7000



NEXT-GEN CONVERGENCE - VERSAL-INSPIRED ACAP (Adaptive Compute Acceleration Platform)



OPTIMISATION DE L'INFÉRENCE IA POUR FPGA (CNN -> MLP/Matrice -> FPGA -> RESSOURCES)



- Zynq + HLS4ML:** io_parallel / io_stream / Reusefactor / Resource / Latency

	ARTY-Z7 CH1	ARTY Z7 CH3	ARTY Z7 CH3 Optimisé	ZCU102 CH3	CH4	ZCU102 CH4 Qbit<16,2>	ZCU102 CH5	ZCU102 CH7 Optim Ressource	ZCU102 CH7 Optim Latence	ARTY Z7 CH7 Optim HLS Stream
Nombre de cellules	16	25801	25801	25801	658951	658951	156951	156951	156951	156951
Horloge de reference	4,166ns	9,408ns	9,408ns	4,396ns		4,369ns	4,369ns	4,369ns	4,028ns	9,410ns
Temps de latence	70ns	19,804us	19,804us	2,510us		5,510us	5,510us	10,010us	10,015us	141us
BRAM	0%	41%	8%	6%		36%	18%	9%	15%	72%
DSP48E	21%	115%	11%	10%		59%	59%	12%	24%	6%
FF	2%	85%	28%	10%		28%	22%	32%	53%	167%
LUT	1%	280%	68%	46%		103%	113%	81%	104%	423%
URAM	0%	0%	0%	0%		0%	0%	0%	0%	0%

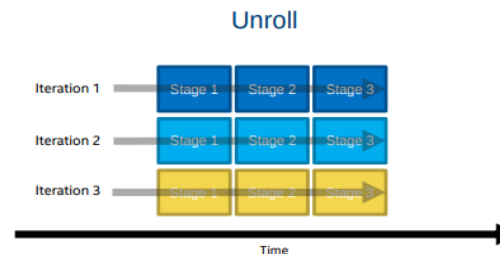


ARTY Z7
ZCU104

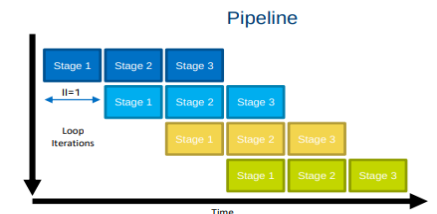


Tensorflow/Keras
+ HLS4ML
VITIS HLS

depending on VITIS-HLS #pragma
The way HLS handles vector/matrix before DSP

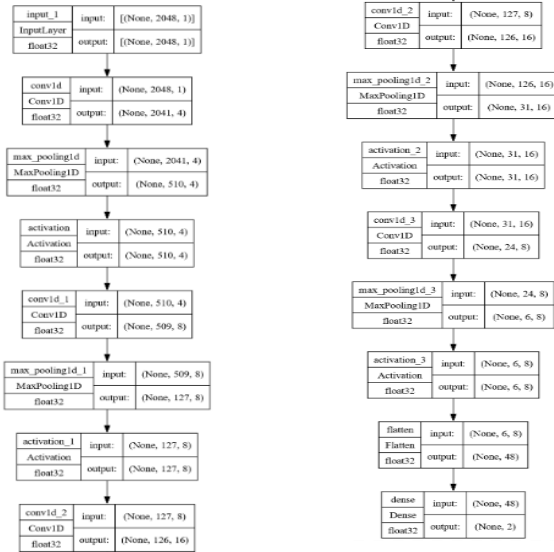


Question of HLS
optimisation

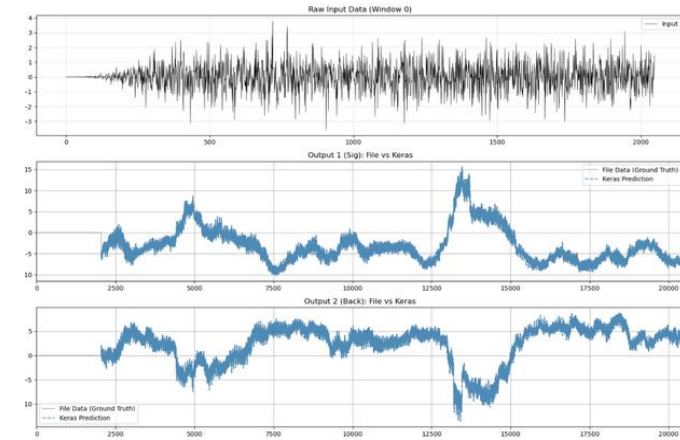


What if directly
written in VHDL ?

1- Modèle CNN de débruitage des ondes gravitationnelles



- 4 couches de convolution
- 1 couche dense
- 37094 paramètres



← Input

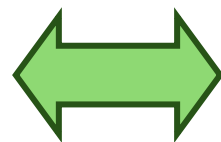
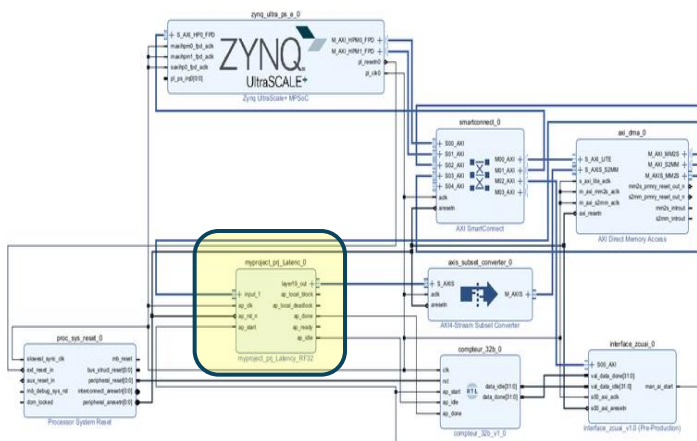
← Output



(cible ZCU104)

Client DAVIS

(Data Acquisition Verification Integration system)

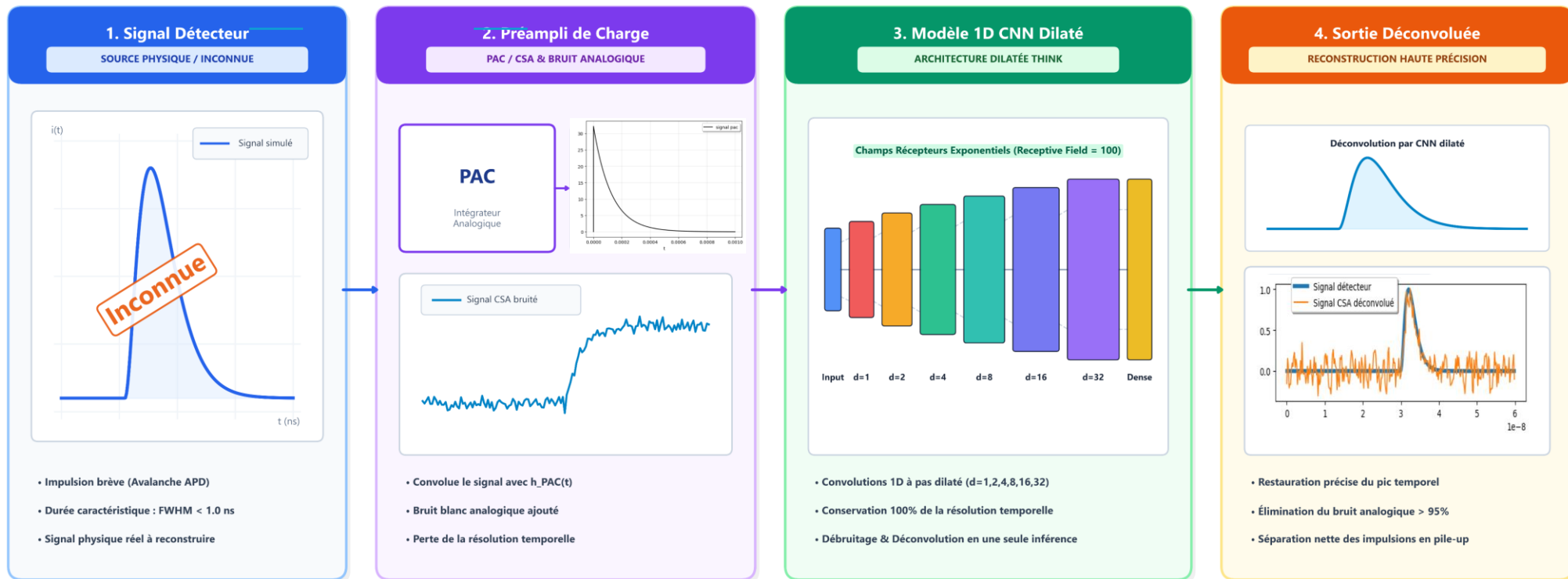


RF= Reuse Factor Optimisation en Ressource ou latence	Utilisation des Ressources				Performance du modèle	
	DSP	LUT	FF	BRAM	Nombre de cycles	Temps inférence
Resource_RF_32	48	70374	26599	165	N/A	N/A
Latency_RF_32	46	69728	33402	137	88248	265.01 us
Resource_RF_2	736	36650	63653	466	39066	117.32 us
Latency_RF_2	736	67301	20274	137	32896	98.79 us
Ressources maximales disponibles	1728	230400	460800	312 (*)		

- 1- HLS4ML optimise lui-même la description du réseau en HLS
- 2- HLS synthétise le code VHDL de l'IP avec peu d'influence du concepteur
- 3- Les ressources utilisées et le temps d'inférence sont constatés après coup
- 4- Le concepteur ne maîtrise pas le code VHDL créé

2- MODÈLE 1D CNN DILATÉ DE DÉBRUITAGE ET DÉCONVOLUTION

Chaîne Électronique avec Préampli de Charge (PAC / CSA) & Restauration Temps-Réel du Signal Détecteur



➔ Problème mal posé au sens de Hadamard (solutions multiples)

Layer (type)	Output Shape	Param #
signal_input (InputLayer)	(None, 100, 1)	0
dilated_1_pruned (Conv1D)	(None, 100, 9)	36
dilated_2_pruned (Conv1D)	(None, 100, 64)	1,792
dilated_4_pruned (Conv1D)	(None, 100, 64)	12,352
dilated_8_pruned (Conv1D)	(None, 100, 64)	12,352
dilated_16_pruned (Conv1D)	(None, 100, 64)	12,352
dilated_32_pruned (Conv1D)	(None, 100, 64)	12,352
final_output (Conv1D)	(None, 100, 1)	65

Total params: 51,301 (200.39 KB)

Trainable params: 51,301 (200.39 KB)

Modèle de base:

- 6 couches de CNN
- 0 couche dense
- 51301 paramètres



Echec de l'implémentation HLS
Indétermination de la sortie
➔ **Absence MLP**

■ HLS4ML OK sur FPGA puissant

- **Difficile de maîtriser la latence** (analyse à posteriori)
- Maîtrise de la quantization
- **Echec de l'implémentation possible (Pb ressources et absence de MLP)**
- **Synthèse HLS peut durer plusieurs heures (voir jours)**

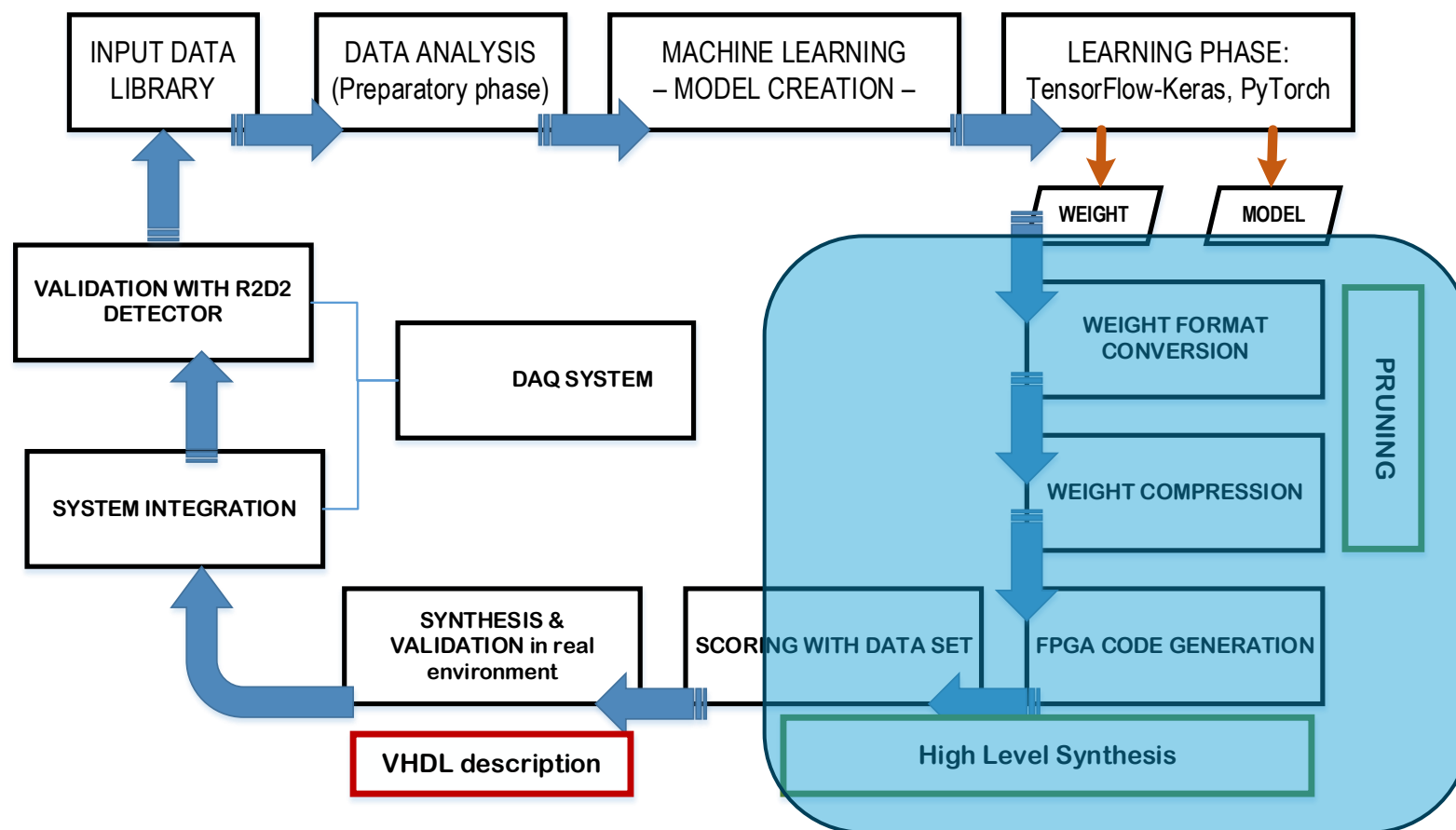
HLS4ML: HLS ne permet pas de maîtriser le design VHDL

Echec de l'implémentation HLS
Indétermination de la sortie
→ Absence MLP

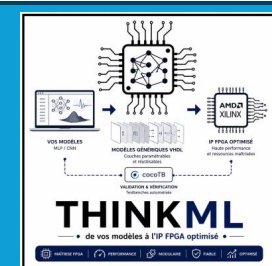


Peut-on faire mieux ?

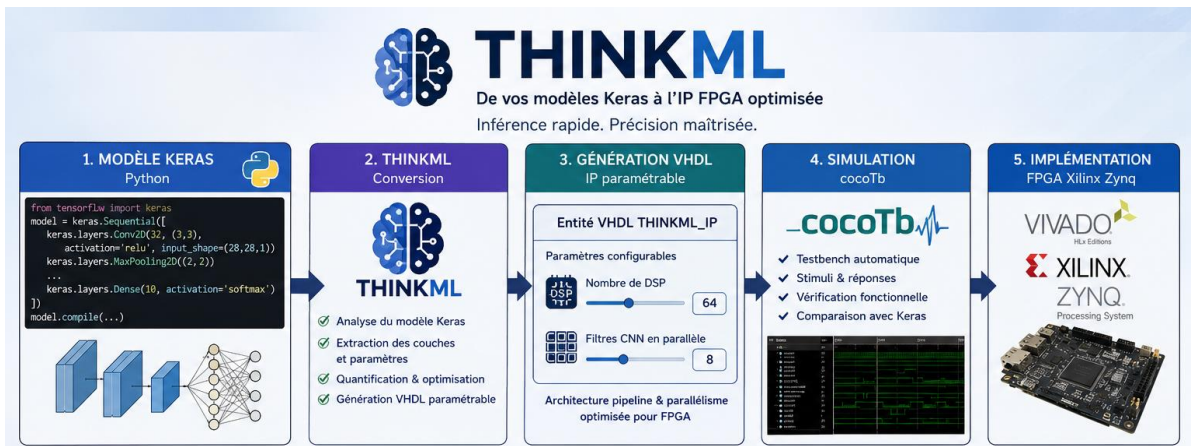
CREATION DE MES PROPRES OUTILS POUR CONTRÔLER MES MODÈLES



Inférence des modèles IA Sur FPGA AMD



- Choix des ressources matériels
- - DSP vs parallélisme
- Conversion en VHDL (IP)
- Simulation matérielle (cocoTB)
- Inférence sur carte
- Evaluation des performances



CHAÎNE DE VALIDATION CONJOINTE

THINK AI Framework : Du Modèle Python au Silicium FPGA

1. Modèle & Référence Logicielle (Python)

SOFTWARE

Frameworks : Keras / TensorFlow / QKeras

- Extraction de l'AST & Topologie du Réseau**
Parsing automatique des couches Conv2D, Dense, MaxPooling, ReLU
- Quantification en Virgule Fixe (Fixed-16 Q6.10)**
Conversion normalisée des poids, biais et données d'activation
- Génération des Stimuli et Références de Test**
Génération de mnist_in.csv (Stimuli) et mnist_out.csv (Golden Reference)

Stimuli AXI & Référence

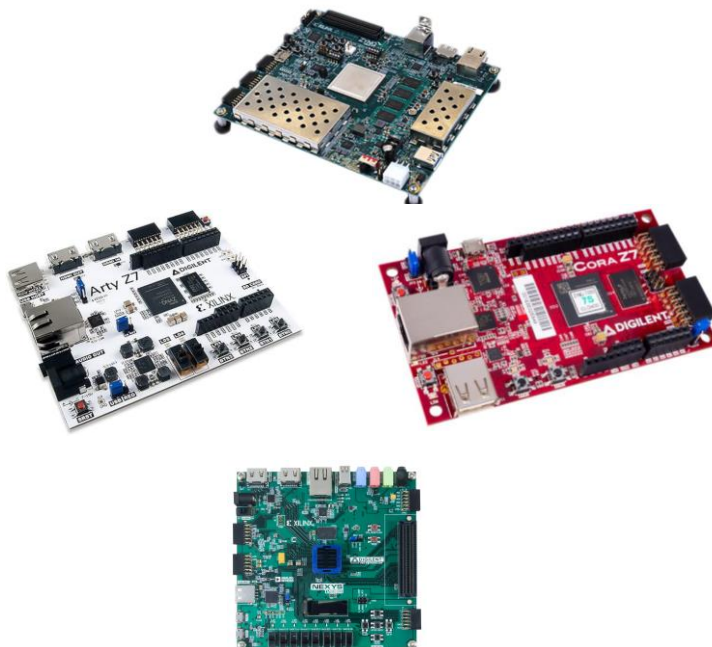
Vecteurs de test quantifiés

2. Co-Simulation RTL Cycle-Exacte (Cocotb)

SIMULATION RTL

Moteur : Cocotb (Python) <-> VPI <-> Simulateur GHDL

- Testbench Asynchrone Python (tb_network.py)**
Contrôle cycle-exact des signaux AXI4-Stream (s_axis, m_axis)
- Émulation Matérielle sous GHDL (--std=08)**
Exécution intégrale du code VHDL structuré (Datapath V3 & Contrôleurs)
- Vérification Bit-Exacte & Calcul MSE**
Capture à la volée sur TVALID/TREADY & validation MSE < 10⁻⁶



3. Synthèse & Packaging IP Vivado

HARDWARE IP

Automatisation Vivado : Scripts TCL & IP-XACT Catalog

- Emballage IP-XACT Clé en Main**
Création de l'IP think_ia.ai_cores avec bus AXI-Stream et registres AXI-Lite
- Allocation Granulaire des Ressources FPGA**
Mapping optimisé des 454 DSPs et synthèse en Distributed RAM (0 BRAM)
- Génération du Bitstream SoC (Zynq UltraScale+)**
Interconnexion AXI DMA 32 bits & Zynq PS sur carte ZCU104

Bitstream FPGA (.xsa)

Déploiement sur cible ZCU104

4. Inférence Temps Réel sur Carte (ZCU104)

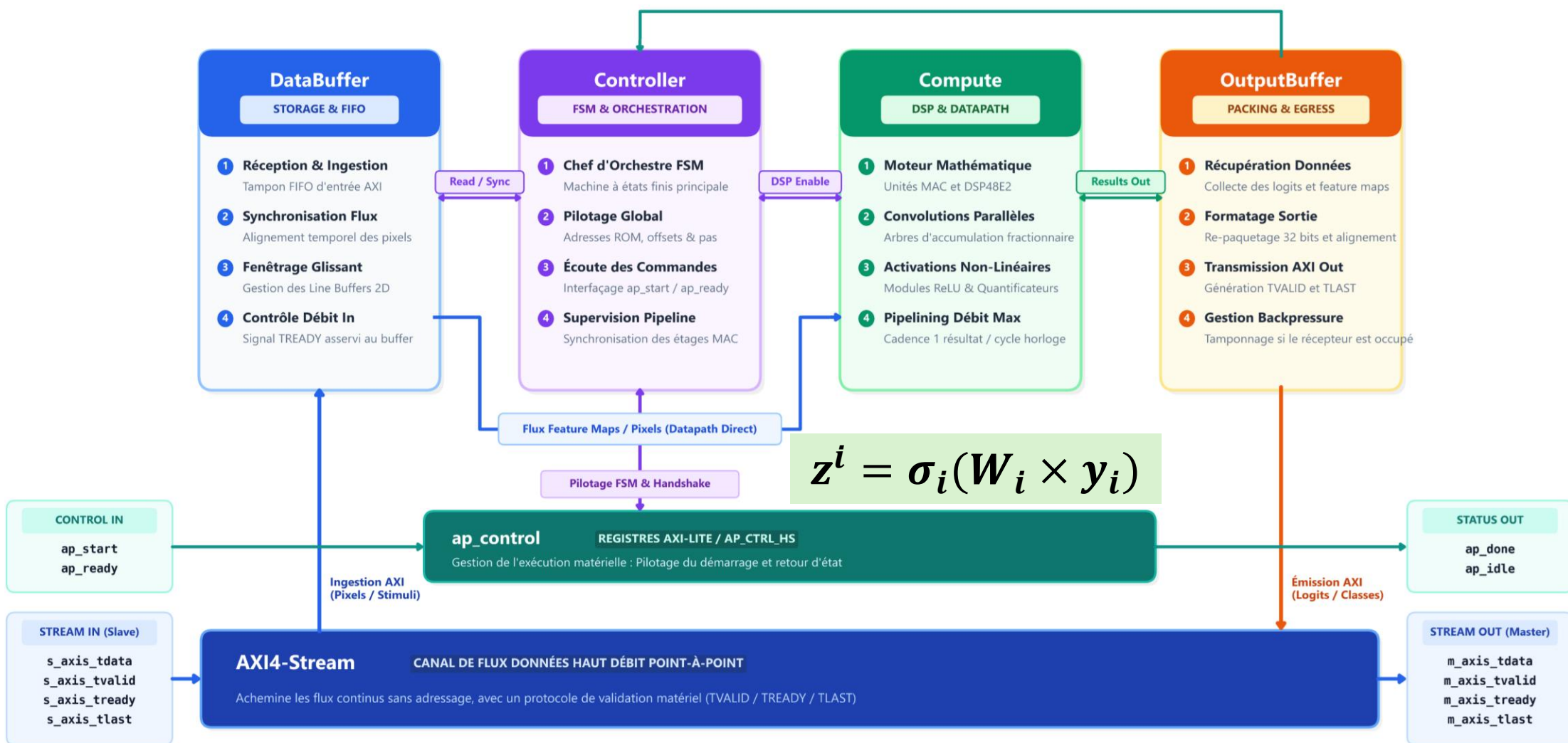
ON-BOARD RUN

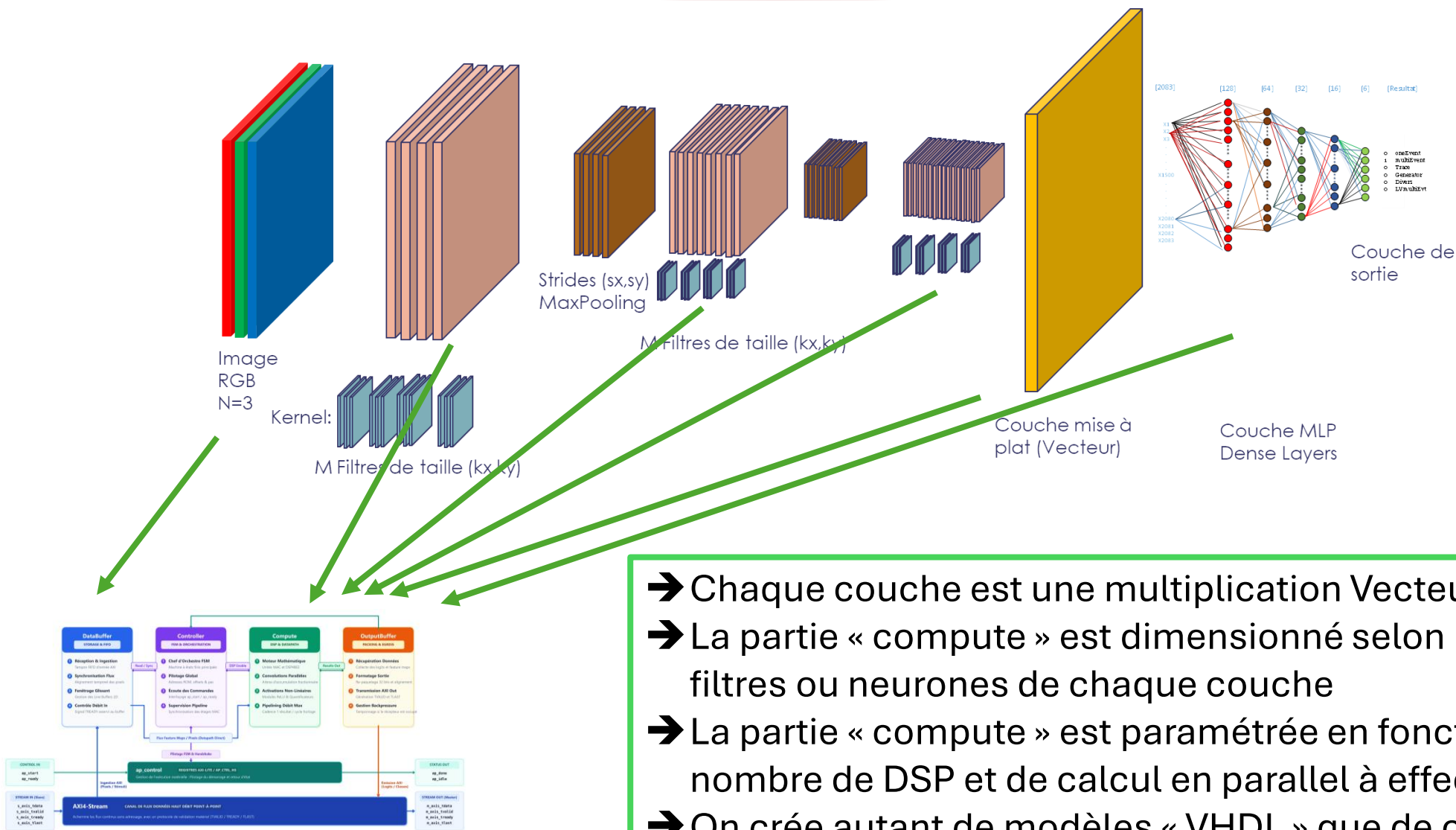
Plateforme : AMD Xilinx Zynq UltraScale+ ZCU104

- Serveur SCPI / TCP-IP Embarqué (FreeRTOS)**
Liaison Ethernet 1 Gbps pilotée en direct depuis les Notebooks Jupyter
- Transferts DMA Haut Débit (Scatter-Gather / Simple)**
Injection directe des images 784 px et récupération des logits 32 bits
- Métrie & Moniteur Dual_Perf_Monitor**
Mesure physique des cycles d'horloge (IA.TIME) et précision hardware 100%

ARCHITECTURE MATÉRIELLE MODULAIRE D'UN BLOC IP IA

Datapath Haute Performance THINK V3 : Découplage Contrôle / Calcul / Flux AXI-Stream





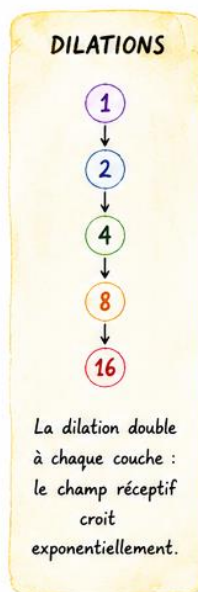
- ➔ Chaque couche est une multiplication Vecteur x Matrice
- ➔ La partie « compute » est dimensionné selon le nombre de filtres ou neurones de chaque couche
- ➔ La partie « compute » est paramétrée en fonction du nombre de DSP et de calcul en parallèle à effectuer
- ➔ On crée autant de modèles « VHDL » que de couches du réseau

CNN 1D DILATÉ - 5 COUCHES DE 16 FILTRES

Entrée : vecteur de longueur 100 → Sortie : vecteur de longueur 100

Paramètres

- Taille des entrées
- Taille du noyau
- Nbre de DSP
- Nombre de Filtres
- Déplacement du noyau



ENTRÉE
Longueur = 100

COUCHE 1
16 filtres
dilation = 1

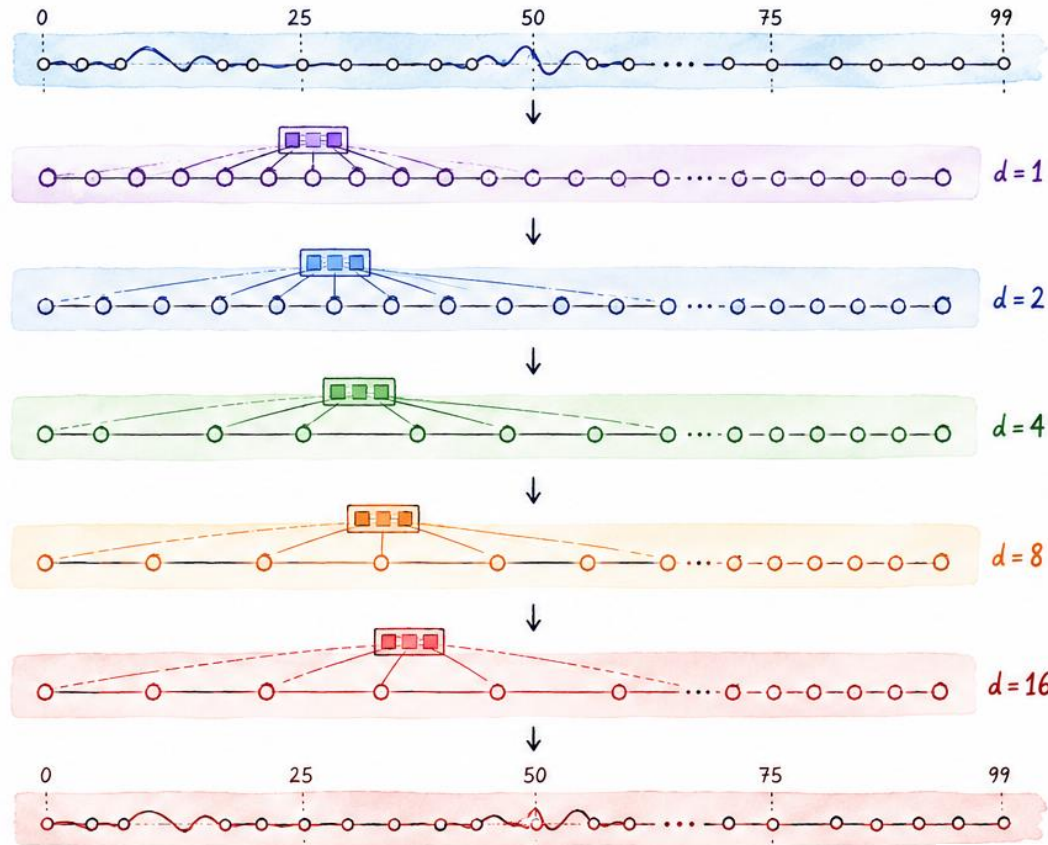
COUCHE 2
16 filtres
dilation = 2

COUCHE 3
16 filtres
dilation = 4

COUCHE 4
16 filtres
dilation = 8

COUCHE 5
16 filtres
dilation = 16

SORTIE
Longueur = 100



RÉCEPTIF FIELD AU CENTRE (pos. 50)

Couche 1 (d=1) → 3
Couche 2 (d=2) → 5
Couche 3 (d=4) → 9
Couche 4 (d=8) → 17
Couche 5 (d=16) → **33**

Formule générale (k=3) :

$$R_L = 1 + \sum_{i=0}^{L-1} (k-1) d_i$$

Ici : $R_5 = 33$

PARAMÈTRES

- Noyau (taille k) = 3
- 16 filtres par couche
- Padding «same» → longueur conservée (100)
- Activation (ReLU) après chaque couche
- Sortie à la même longueur que l'entrée

ARCHITECTURE MATÉRIELLE MODULAIRE D'UN BLOC IP IA

LÉGENDE

- Fenêtre du noyau (k=3)
- Connexions utilisées
- Connexions des positions voisines

CONV1D DILATÉE

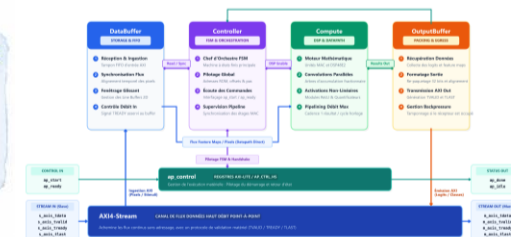
(à la position t)

$$y_t^{(l)} = \phi \left(\sum_{i=0}^{k-1} w_i^{(l)} x_{t-i \cdot d_l} + b^{(l)} \right)$$

k : taille noyau d_l : dilation couche l ϕ : ReLU

RÉCEPTEUR EFFECTIF APRÈS L COUCHES

$$R_L = 1 + \sum_{i=0}^{L-1} (k-1) d_i$$



CNN 1D DILATÉ : ÉMULER UN MLP OU IN RNN 1D

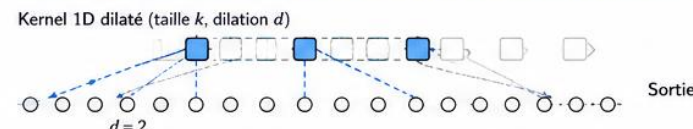
La convolution 1D dilatée est un opérateur universel pour modéliser des dépendances locales ou séquentielles

IDÉE CLÉ : En ajustant la taille du noyau (k), la dilation (d) et le stride (s), un CNN 1D dilaté peut reproduire le comportement d'un MLP (contexte global) ou d'un RNN 1D (dépendances séquentielles).



Entrée : $x = [x_1, x_2, \dots, x_T]$

k : taille du noyau (nombre d'éléments utilisés)
 d : dilation (espacement entre les éléments du noyau)
 s : stride (pas de déplacement du noyau)



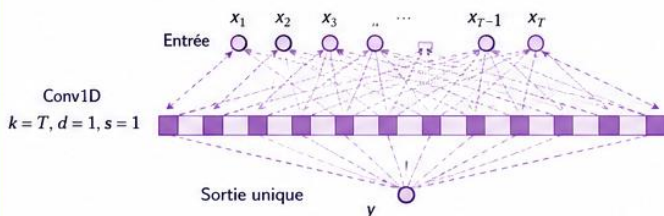
1) ÉMULER UN MLP AVEC UN CNN 1D DILATÉ

Objectif : Chaque sortie dépend de tous les éléments de l'entrée, comme dans un MLP.

COMMENT FAIRE ?

- Utiliser un seul Conv1D avec $k = T$ (longueur de la séquence)
- $d = 1$ (pas de saut)
- $s = 1$ (produit une seule sortie globale)

Réceptif = $k + (k-1)(d-1) = T$
 $\Rightarrow$ La convolution voit toute la séquence



MLP (1 couche)

Équation : $y = \phi(Wx + b)$
 où : $x \in \mathbb{R}^T$, $W \in \mathbb{R}^{m \times T}$, $b \in \mathbb{R}^m$
 ϕ : fonction d'activation

Équivalent

CNN 1D dilaté

Paramètres : $k = T$, $d = 1$, $s = 1$
 Équation : $y = \phi(W_{conv} * x + b)$
 où : $W_{conv} \in \mathbb{R}^{m \times T}$ (m filtres)

$\Rightarrow$ Le Conv1D avec $k = T$, $d = 1$, $s = 1$ est équivalent à un MLP à une couche cachée (sans biais temporel).

2) ÉMULER UN RNN 1D AVEC UN CNN 1D DILATÉ

Objectif : Modéliser des dépendances séquentielles et causales (passé uniquement).

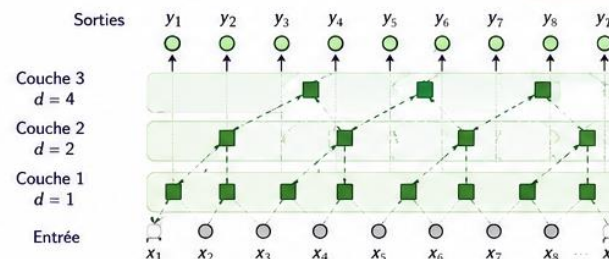
COMMENT FAIRE ?

- Utiliser un Conv1D causal (zéro-padding à gauche)
- $k = 2$, $d = 1$, $s = 1$
- Empiler plusieurs couches Conv1D dilatées avec d croissant : 1, 2, 4, 8, ...

Réceptif après L couches :

$$R_L = 1 + \sum_{i=0}^{L-1} (k_i - 1) \prod_{i=0}^{L-1} d_i$$

(avec $k_i = 2 \Rightarrow R_L = 2^L$)



Chaque sortie y_t dépend du passé jusqu'à t .
 La taille du contexte croît exponentiellement avec la profondeur.

RNN 1D (simple)

Équations : $h_t = \phi(W_h h_{t-1} + W_x x_t + b)$
 $y_t = \psi(W_y h_t + b_y)$
 où h_t : état caché, y_t : sortie

Équivalent

CNN 1D dilaté causal

Paramètres : $k = 2$, $s = 1$, $d_l = 2^l$
 Équation (couche l) : $h_t^{(l)} = \phi(W^{(l)} * h_t^{(l-1)} + b^{(l)})$
 (concatenation / somme des couches)
 $\Rightarrow y_t = \psi(h_t^{(L)})$

RÉSUMÉ COMPARATIF



	Objectif	k (taille noyau)	d (dilation)	s (stride)	Padding	Réceptif	Comportement
Émuler un MLP		$k = T$	$d = 1$	$s = 1$	Valid	T (global)	Chaque sortie dépend de tous les entrées (contexte global).
Émuler un RNN 1D		$k = 2$	$d = 1, 2, 4, \dots$	$s = 1$	Causal (gauche)	$R_L = 2^L$ (exponentiel)	Chaque sortie dépend du passé jusqu'à t (dépendances séquentielles).



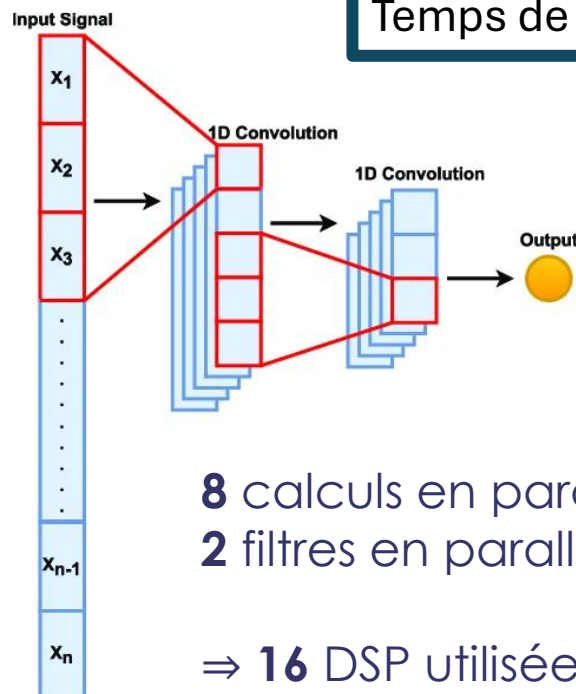
POURQUOI C'EST PUISSANT ?

- ✓ Parallélisable (contrairement aux RNN)
- ✓ Contexte long avec peu de couches (dilatation)
- ✓ Unifie MLP et RNN dans une même famille (CNN 1D)

Couche	Arch	(K, C_IN, NB_FILTRE)	DSP Réels	PARALLEL_CH	DSP/CH
conv1d	v3	(8, 1, 4)	16	2	8
max_pooling1d	common	(_, 4, 4)	0	-	-
activation	common	(_, 4, 4)	0	-	-
conv1d_1	v3	(2, 4, 8)	32	4	8
max_pooling1d_1	common	(_, 8, 8)	0	-	-
activation_1	common	(_, 8, 8)	0	-	-
conv1d_2	v3	(2, 8, 16)	32	2	16
max_pooling1d_2	common	(_, 16, 16)	0	-	-
activation_2	common	(_, 16, 16)	0	-	-
conv1d_3	v3	(8, 16, 8)	128	2	64
max_pooling1d_3	common	(_, 8, 8)	0	-	-
activation_3	common	(_, 8, 8)	0	-	-
dense	v3	(6, 8, 2)	64	2	32

>> TOTAL DSP CONSOMMÉS : 272 / 1728

THINKML vs HLS4ML:
Nbre de DSP divisé par 3
Temps de latence 100x plus rapide



- 1- **Contrôle total du nombre de DSP**
 - 2- **Optimisation des latences par couche**
 - 3- **Minimisation du nombre des LUT et FF dans le code**
- ➔ **Aucun contrôle des LUT et FF (Optimisation Vivado en minimisant la taille des boucles dans le code VHDL)**

8 calculs en parallèle sur les $K \cdot C_{in}$ à faire
2 filtres en parallèle sur les NB_FILTRE à faire
⇒ **16** DSP utilisées pour la première

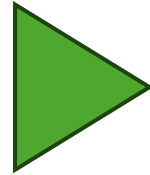
Optimisation nbre DSP et calcul paralleles	DSP	LUT	FF	BRAM	Nombre de cycles	Temps inférence
Modele OG	272	30168	24126	11,5	115	1,15μs
Ressources max. disponibles	1728	230400	460800	312		



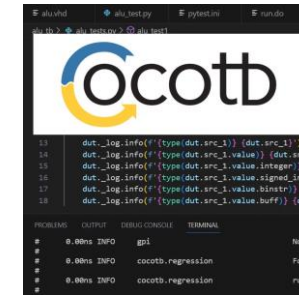
Modèle de base:

- 6 couches de CNN
- 0 couche dense
- 51301 paramètres

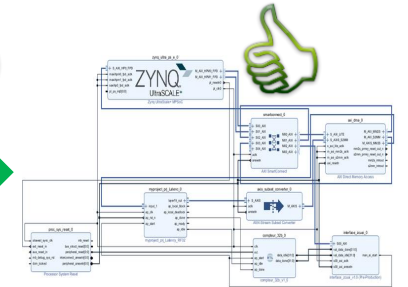
Echec de l'implémentation HLS
Indétermination de la sortie
→ Absence MLP



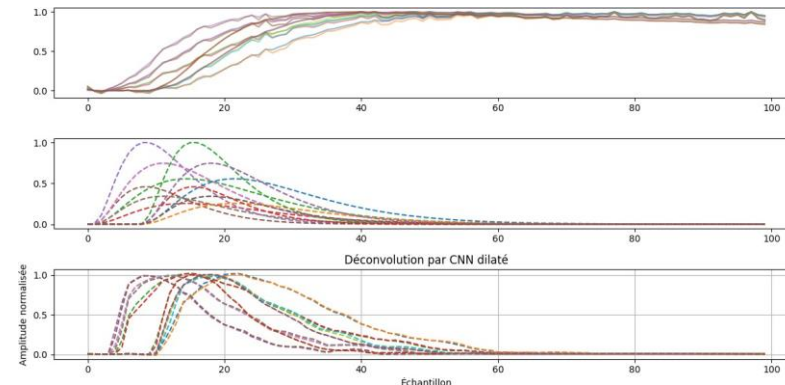
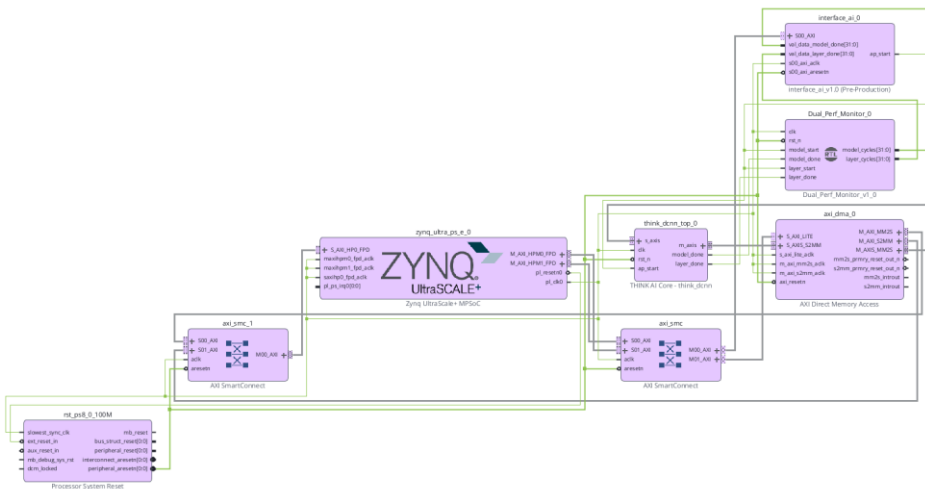
Modèle VHDL



Simulation avec cocotb



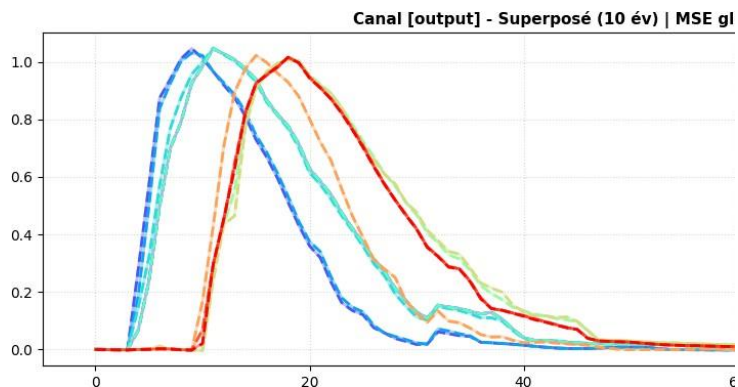
Projet Vivado OK



- Maîtrise du nombre de DSP
- Maîtrise du nombre de calcul en parallèle
- Maîtrise des buffers d'entrée et du modèle
- **Modèle: maîtrise de l'utilisation des LUT**

Rmq: 2x Nbre DSP / OG → 2 x plus lent.

Optimisation nbre DSP et calcul paralleles	DSP	LUT	FF	BRAM	Nombre de cycles	Temps inférence
Modele OG	423	161387	49621	18,5	213	2,13µs
Ressources max. disponibles	1728	230400	460800	312		



ThinkML — Récapitulatif des Couches & Ressources (Modèle 1ddcnn)

Horloge cible : 100 MHz | Précision : Fixed-16 bits (ap_fixed<16, 3>) | II = Initiation Interval (Cycles d'horloge entre 2 données d'entrée)

Couche	Géométrie (K, C _{in} , N _f)	Paramètres ThinkML	DSP	BRAM	FF	LUT	II	HW Timing Réel (Carte)	Statut Timing
dilated_1	K=3, C _{in} =1, N _f =16	//ch=8, dsp/ch=3, rom=block	26	2	2 152	4 674	2	10.100 µs (1010 c)	Standard
dilated_2	K=3, C _{in} =16, N _f =16	//ch=4, dsp/ch=24, rom=block	98	96	5 706	10 815	8	6.180 µs (618 c)	BOTTLENECK
dilated_4	K=3, C _{in} =16, N _f =16	//ch=4, dsp/ch=24, rom=block	98	48	5 606	10 037	8	0.190 µs (19 c)	FASTER
dilated_8	K=3, C _{in} =16, N _f =16	//ch=4, dsp/ch=24, rom=block	98	48	5 614	10 095	8	0.210 µs (21 c)	FASTER
dilated_16	K=3, C _{in} =16, N _f =16	//ch=4, dsp/ch=24, rom=block	98	48	5 620	10 222	8	0.200 µs (20 c)	FASTER
final_output	K=1, C _{in} =16, N _f =1	//ch=1, dsp/ch=8, rom=block	8	7	753	1 548	2	0.130 µs (13 c)	FASTER
TOTAL SYNTHÈSE	6 Couches	Synthèse Vivado ZCU104	426	265	25 453	47 559	—	100% Timing Met @ 100 MHz	

Ressource matérielle	Version initiale (Grenoble)	Version finale optimisée	Gain mesuré / Impact opérationnel
Tables logiques (LUT)	161 387	39 752	-75% (-121 635 LUTs libérées) — Décongestion totale du routage
Bascules mémoires (FF)	49 621	25 453	-49% (-24 168 FFs libérées) — Division par deux des registres
Multiplieurs matériels (DSP)	426	426	100% Efficacité arithmétique préservée (saturation optimale sans gaspillage)
Blocs mémoire (BRAM 36K)	18.5	265	+246.5 — Déport dédié des buffers circulaires vers les blocs RAM synchrones
Qualité de fit (MSE vs Keras)	1.42 × 10 ⁻³ (Défaut de fit)	< 10 ⁻⁶	Bit-True Superposition rigoureusement parfaite (bug +1 cycle résolu)

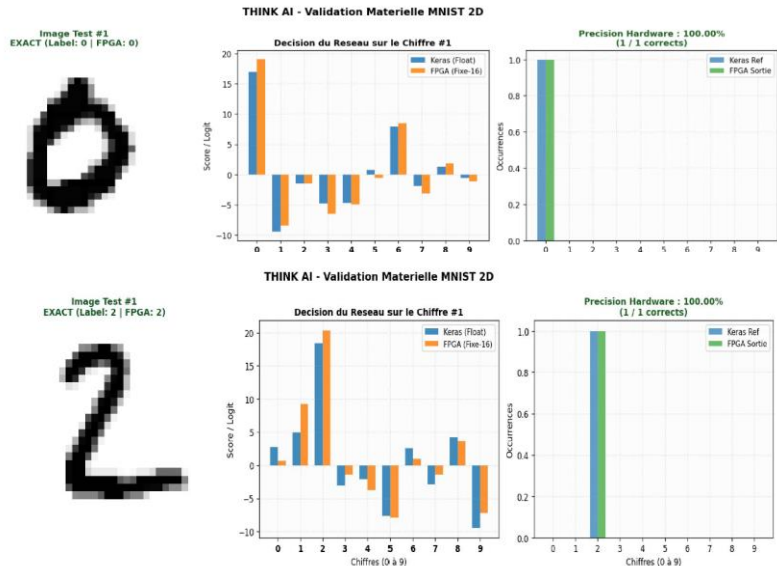
Modèle v2.0

- ➔ Maîtrise du nombre de DSP
- ➔ Maîtrise du nombre de calcul en parallèle
- ➔ Maîtrise des buffers d'entrée et du modèle
- ➔ Utilisation BRAM au lieu de LUT
- ➔ Modèle plus lent mais moins de ressources utilisées

Latence Matérielle Pure (Pipeline après TLAST) : 7.090 µs (709 cycles)

Temps Matériel Streaming (IP complète) : 17.010 µs (1701 cycles)

Taux de Traitement Soutenu : > 58 000 inférences / seconde



THINK AI — Allocation des DSP & Stratégie d'Architecture

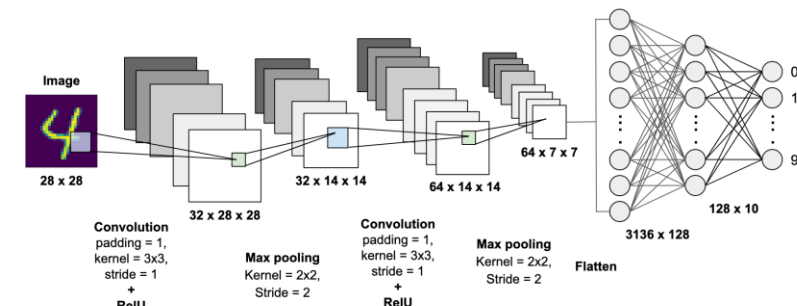
Modèle : MNIST 2D CNN | Horloge cible : 50 MHz | Précision : Fixed-16 bits

Couche	Arch	Géométrie (K, C_IN, NB_FILTRE)	DSP Réels	PARALLEL_CH	DSP/CH
conv2d_1	v3	(3, 1, 16)	144	16	9
conv2d_1_relu	common	(_, 16, 16)	0	-	-
pool2d_1	common	(_, 16, 16)	0	-	-
conv2d_2	v3	(3, 16, 32)	64	16	4
conv2d_2_relu	common	(_, 32, 32)	0	-	-
pool2d_2	common	(_, 32, 32)	0	-	-
dense_1	v3	(25, 32, 64)	160	8	20
dense_1_relu	common	(_, 64, 64)	0	-	-
final_output	v3	(1, 64, 10)	80	10	8
>> TOTAL DSP CONSOMMÉS :			448 / 1728 DSPs	25.9 % du FPGA (ZCU104)	

Estimation des Ressources par Couche (Synthèse Vivado)

Rapport d'utilisation des primitives matérielles | Cible : AMD Zynq UltraScale+ ZCU104

Couche / Entité RTL	LUTs	Flip-Flops (FF)	Block RAM (BRAM)	DSPs	Part LUTs	Part DSPs
conv2d_1	19 103	11 310	0	146	20.0 %	32.2 %
conv2d_2	16 638	12 576	0	65	17.4 %	14.3 %
dense_1	45 770	11 089	302	163	47.8 %	35.9 %
final_output	8 038	4 914	8	80	8.4 %	17.6 %
TOP IP (Total Synthèse Vivado)	95 748	47 670	309	454		



Rapport de Co-Simulation RTL Cycle-Exacte (Cocotb & GHDL)

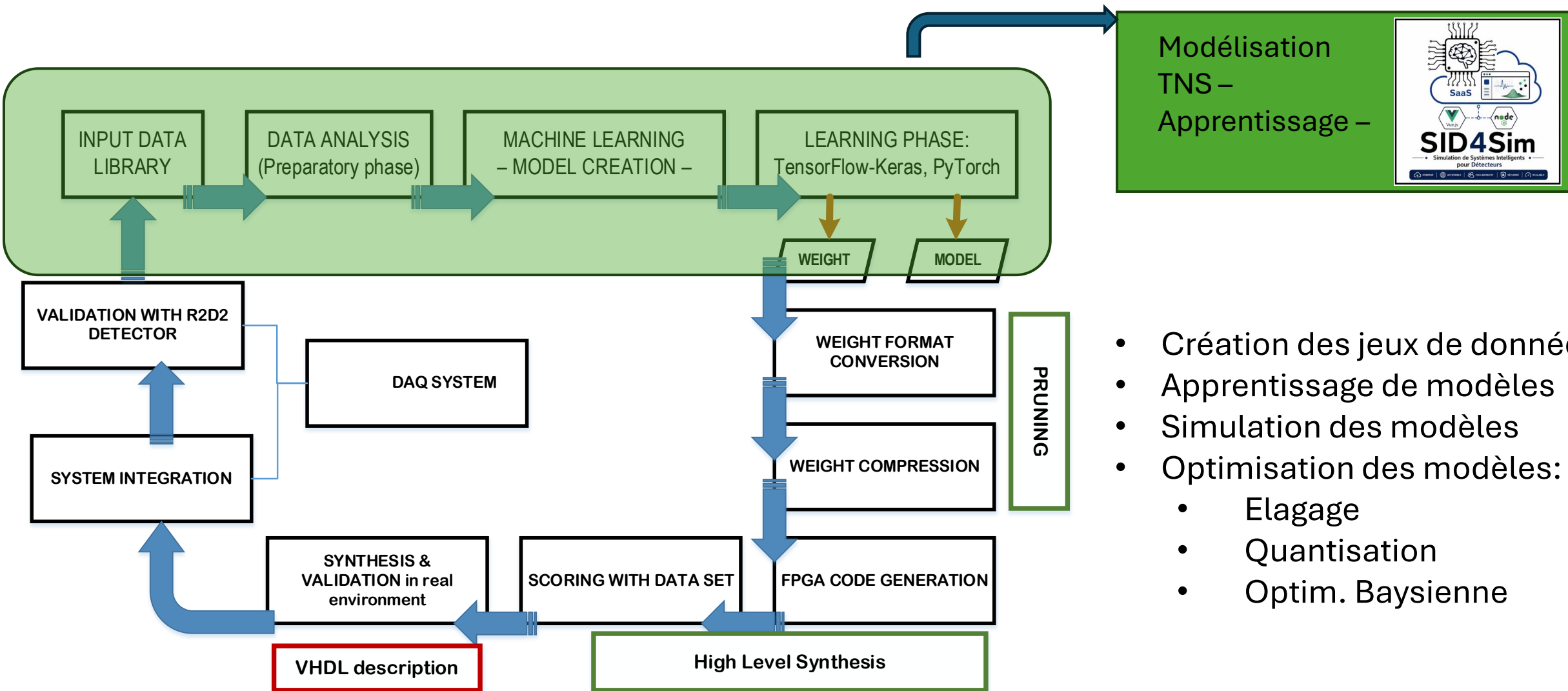
Framework : Cocotb v2.0.1 | Simulateur VHDL : GHDL v6.0 | Bus AXI4-Stream

Module Testbench	Test Case	Statut	Sim Time (ns)	Real Time (s)	Vitesse (ns/s)
thinkml.tb_network	test_network_axi	PASS (100%)	45 430.00 ns	153.16 s	296.62 ns/s

[OK] Résultat global : 1/1 test réussi (0 échec) | Erreur MSE < 10⁻⁶ | TVALID et TLAST parfaitement synchronisés

➔ Modèle RTL CNN 1DD ➔ transformation en CNN 2D

CREATION DE MES PROPRES OUTILS POUR CONTRÔLER MES MODÈLES



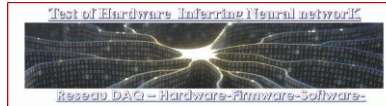
- Création des jeux de données
- Apprentissage de modèles
- Simulation des modèles
- Optimisation des modèles:
 - Elagage
 - Quantisation
 - Optim. Bayésienne



SID4SIM Beta



La modélisation avec SID4Sim (SaaS)



Reinitialiser

Ajouter script

Sauvegarder

Executer

Espaces de travail

Voir les librairies importees

24 bloc(s) disponible(s)

SID4SIM System SYSTEME

Spectre simple (FFT)

system_fft.py

Bode systeme entree/sortie

system_bode.py

Comparaison de signaux

system_compare.py

SID4SIM Math SYSTEME

Somme

system_sum.py

Soustraction

system_subtract.py

Multiplication

system_multiply.py

Division

system_divide.py

SNR

system_snr.py

Erreur entre deux signaux

system_error.py

Diagramme

Signaux

Logs

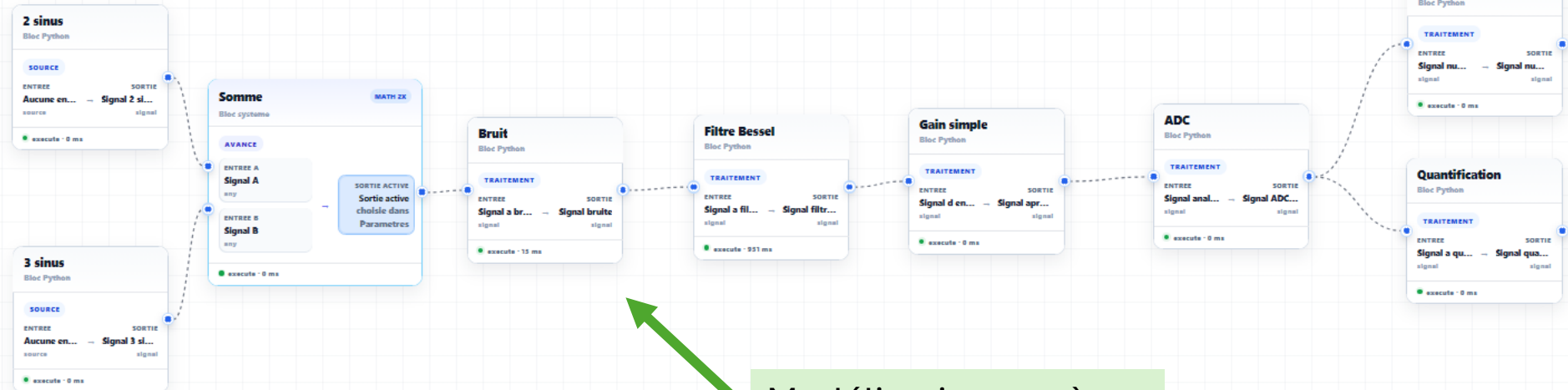
Parametres

Editeur

Classes python:
-systemes
-projets

Ajouts de scripts
utilisateurs

Execution du
modèle



Modélisation système



La modélisation avec SID4Sim

Espaces de travail

Voir les bibliothèques importées

24 bloc(s) disponible(s)

SID4SIM System

SYSTEME

Spectre simple (FFT)

system_fft.py

Bode système entrée/sortie

system_bode.py

Comparaison de signaux

system_compare.py

SID4SIM Math

SYSTEME

Somme

system_sum.py

Soustraction

system_subtract.py

Multiplication

system_multiply.py

Division

system_divide.py

SNR

system_snr.py

Execution en cours

Diagramme

Signaux

Logs

Paramètres

Editeur

EXECUTION

Logs d execution

Execution en cours...

Validation avant execution : OK.

Envoi du workflow v4 au serveur...

Resultat serveur

Status : ok

Message : Workflow execute avec SID4SIM vBeta

Blocs : 9

Connexions : 8

Ordre : 2 sinus -> 3 sinus -> Somme -> Bruit -> Filtre Bessel -> Gain simple -> ADC -> Filtre numerique -> Quantification

Branches detectees

Source commune : ADC

Branche 1 : Filtre numerique

Branche 2 : Quantification

Rapport scientifique

1. 2 sinus

SID4SIM Beta

Reinitialiser

Espaces de travail

Voir les bibliothèques importées

24 bloc(s) disponible(s)

SID4SIM System

SYSTEME

Spectre simple (FFT)

system_fft.py

Bode système entrée/sortie

system_bode.py

Comparaison de signaux

system_compare.py

SID4SIM Math

SYSTEME

Somme

system_sum.py

Soustraction

system_subtract.py

Multiplication

system_multiply.py

Division

system_divide.py

SNR

system_snr.py

Erreur entre deux signaux

system_error.py

Délai circulaire

system_delay.py

Diagramme

Signaux

Logs

Paramètres

Editeur

Source indépendante : Signal 2 sinus

Source indépendante : Signal 3 sinus

Amplitude complexe (V)

4.0465

2.0404

0.0342

0.9719

-3.9718

0.005

0.025

0.05

0.075

0.105

Temps (s)

Source indépendante : Signal 2 sinus (V)

Source indépendante : Signal 3 sinus (V)

EXEMPLE 2

Signal bruité

Glissez dans le graphe pour zoomer. Survoler pour lire les valeurs.

Branches : Avant branchement

Unités : Avant branchement : Signal bruité : V

Affichage allégé pour garder l'interface fluide. Les exports conservent les données sélectionnées.

1 sortie affichée

Type

Courbes Y=f(X)

Axe X

Lineaire

Axe Y

Lineaire

Normaliser

Selection auto

Reinitialiser zoom

Avant branchement : Signal bruité

Signal bruité (V)

5.4209

2.0943

0.0302

-2.7588

-5.4853

0.005

0.025

0.05

0.075

0.105

Temps (s)

Avant branchement : Signal bruité (V)

SID4SIM Beta

Reinitialiser

Espaces de travail

Voir les bibliothèques importées

24 bloc(s) disponible(s)

SID4SIM System

SYSTEME

Spectre simple (FFT)

system_fft.py

Bode système entrée/sortie

system_bode.py

Comparaison de signaux

system_compare.py

SID4SIM Math

SYSTEME

Somme

system_sum.py

Soustraction

system_subtract.py

Multiplication

system_multiply.py

Division

system_divide.py

SNR

system_snr.py

Erreur entre deux signaux

system_error.py

Diagramme

Signaux

Logs

Paramètres

Editeur

Affichage allégé pour garder l'interface fluide. Les exports conservent les données sélectionnées.

2 sorties affichées

Type

Courbes Y=f(X)

Axe X

Lineaire

Axe Y

Lineaire

Normaliser

Selection auto

Reinitialiser zoom

Branche 1 : Filtre numerique - Signal numerique filtre

Avant branchement - Signal filtre Bessel

Amplitude complexe (V)

0.1192

0.0591

0.71e-4

-0.0609

-0.1209

0.0223

0.0281

0.0299

0.0337

0.0375

Temps (s)

Zoom actif : cliquez sur "Reinitialiser zoom" pour revenir à l'affichage complet.

Branche 1 : Filtre numerique - Signal numerique filtre (V)

Avant branchement - Signal filtre Bessel (V)

Ajouter un graphique

SID4SIM Beta

Reinitialiser

Espaces de travail

Voir les bibliothèques importées

24 bloc(s) disponible(s)

SID4SIM System

SYSTEME

Spectre simple (FFT)

system_fft.py

Bode système entrée/sortie

system_bode.py

Comparaison de signaux

system_compare.py

SID4SIM Math

SYSTEME

Somme

system_sum.py

Soustraction

system_subtract.py

Multiplication

system_multiply.py

Division

system_divide.py

SNR

system_snr.py

Erreur entre deux signaux

system_error.py

Diagramme

Signaux

Logs

Paramètres

Editeur

GRAPHIQUE 1

Comparaison de branches

Glissez dans le graphe pour zoomer. Survoler pour lire les valeurs.

Branches : Avant branchement - Source commune : ADC

Unités : Avant branchement - Signal après gain : V - Source commune : ADC - Signal ADC ZOH : V - Source commune : ADC - Echantillons ADC : V

Affichage allégé pour garder l'interface fluide. Les exports conservent les données sélectionnées.

3 sorties affichées

Type

Courbes Y=f(X)

Axe X

Lineaire

Axe Y

Lineaire

Normaliser

Selection auto

Reinitialiser zoom

Avant branchement - Signal après gain

Source commune : ADC - Signal ADC ZOH

Source commune : ADC - Echantillons ADC

Amplitude complexe (V)

0.3287

0.1559

-0.0169

-0.1897

-0.3625

0.0364

0.0376

0.0388

0.04

0.0412

Temps (s)

IA (qwen) pour vous aider dans la création de scripts

ASSISTANT IA LOCAL

Creer un bloc SID4SIM

Modele configure

qwen2.5-coder:7b

GPU demande · num_gpu=999 · contexte=2048

Modele Ollama

qwen2.5-coder:7b · installe

Nom souhaite du script genere

ex : filtre_notch_rf.py

Optionnel : tu peux aussi modifier ce nom juste avant l'import dans la fenetre Ajouter script.

Description

Ex : crée un bloc filtre notch RF, un lissage Savitzky-Golay, une source AM, ou un convertisseur impedance R/X vers Gamma.

Fichier Python non fonctionnel

Parcourir... Aucun fichier sélectionné.

Templates locaux enrichis

Annuler

Assistance IA

```
import numpy as np
from sid4sim import sid4sim_block, series

@sid4sim_block(
    label="3 sinus",
    category="TNS reel",
    description="Genere un signal compose de trois sinusoides.",
    inputs={},
    outputs={"signal": {"label": "Signal 3 sinus", "type": "signal", "unit": "v"}},
    parameter_metadata={
        "f1_hz": {"label": "Frequence 1", "unit": "Hz", "type": "number", "min": 0, "step": 1},
        "f2_hz": {"label": "Frequence 2", "unit": "Hz", "type": "number", "min": 0, "step": 1},
        "f3_hz": {"label": "Frequence 3", "unit": "Hz", "type": "number", "min": 0, "step": 1},
        "amplitude1": {"label": "Amplitude 1", "unit": "v", "type": "number", "step": 0.1},
        "amplitude2": {"label": "Amplitude 2", "unit": "v", "type": "number", "step": 0.1},
        "amplitude3": {"label": "Amplitude 3", "unit": "v", "type": "number", "step": 0.1},
        "phase1_rad": {"label": "Phase 1", "unit": "rad", "type": "number", "step": 0.1},
        "phase2_rad": {"label": "Phase 2", "unit": "rad", "type": "number", "step": 0.1},
        "phase3_rad": {"label": "Phase 3", "unit": "rad", "type": "number", "step": 0.1},
        "t_start_s": {"label": "Temps initial", "unit": "s", "type": "number", "step": 0.001},
        "t_end_s": {"label": "Temps final", "unit": "s", "type": "number", "step": 0.001},
        "num_points": {"label": "Nombre de points", "type": "number", "min": 8, "step": 1},
    },
)

class Generateur3Sinus:
    def __init__(self, f1_hz=5.0, f2_hz=20.0, f3_hz=50.0,
                 amplitude1=1.0, amplitude2=0.5, amplitude3=2.0,
                 phase1_rad=0.0, phase2_rad=1.57079632679, phase3_rad=3.14159265359,
                 t_start_s=0.0, t_end_s=1.0, num_points=1000):
        self.frequencies_hz = [float(f1_hz), float(f2_hz), float(f3_hz)]
        self.amplitudes = [float(amplitude1), float(amplitude2), float(amplitude3)]
        self.phases_rad = [float(phase1_rad), float(phase2_rad), float(phase3_rad)]
        self.t_start_s = float(t_start_s)
        self.t_end_s = float(t_end_s)
        self.num_points = int(num_points)

    def apply(self, inputs):
        if self.num_points < 2:
            raise ValueError("num_points doit etre superieur ou egal a 2.")
        if self.t_end_s <= self.t_start_s:
            raise ValueError("t_end_s doit etre strictement superieur a t_start_s.")
        time = np.linspace(self.t_start_s, self.t_end_s, self.num_points)
        values = np.zeros_like(time)
        for frequency_hz, amplitude, phase_rad in zip(self.frequencies_hz, self.amplitudes, self.phases_rad):
            values += amplitude * np.sin(2 * np.pi * frequency_hz * time + phase_rad)
        fs = (self.num_points - 1) / (self.t_end_s - self.t_start_s)
        return {"signal": series(values, x=time, semantic_type="signal", label="Signal 3 sinus", unit="v",
                                x_label="Temps", x_unit="s", sampling_freq_hz=fs)}
```

Décorateur de bloc

Paramètres

Fonction d'exécution

Espaces de travail

Voir les bibliothèques importées

32 bloc(s) disponible(s)

Spectre simple (FFT)

system_fft.py

Bode systeme entree/sortie

system_bode.py

Comparaison de signaux

system_compare.py

SID4SIM Math SYSTEME

Somme

system_sum.py

Soustraction

system_subtract.py

Multiplication

system_multiply.py

Division

system_divide.py

SNR

system_snr.py

Erreur entre deux signaux

system_error.py

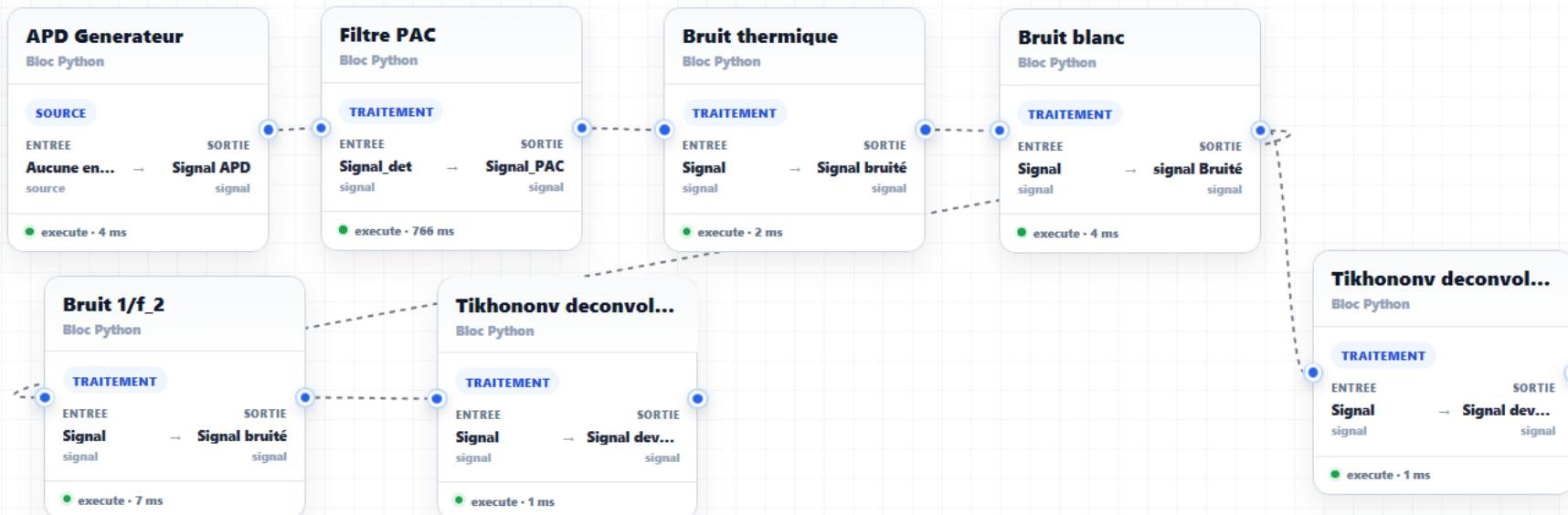
Diagramme

Signaux

Logs

Parametres

Editeur



SID4SIM Beta

Reinitialiser

Ajouter script

Sauvegarder

Executer

Espaces de travail

Voir les bibliothèques importées

32 bloc(s) disponible(s)

Spectre simple (FFT)

system_fft.py

Bode systeme entree/sortie

system_bode.py

Comparaison de signaux

system_compare.py

SID4SIM Math SYSTEME

Somme

system_sum.py

Soustraction

system_subtract.py

Multiplication

system_multiply.py

Division

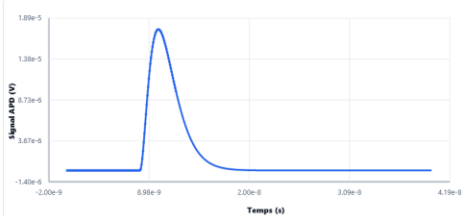
system_divide.py

SNR

system_snr.py

Erreur entre deux si

system_error.py



Source indépendante - Signal APD (V)

APD Générateur

Bloc Python

SOURCE

ENTREE

Aucune en...

source

execute • 4 ms

Bruit 1/f_2

Bloc Python

TRAITEMENT

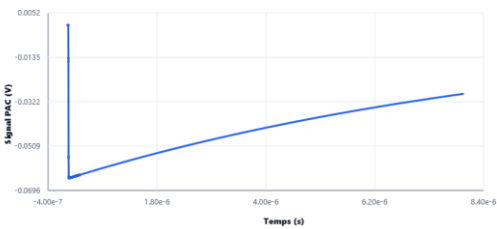
ENTREE

Signal

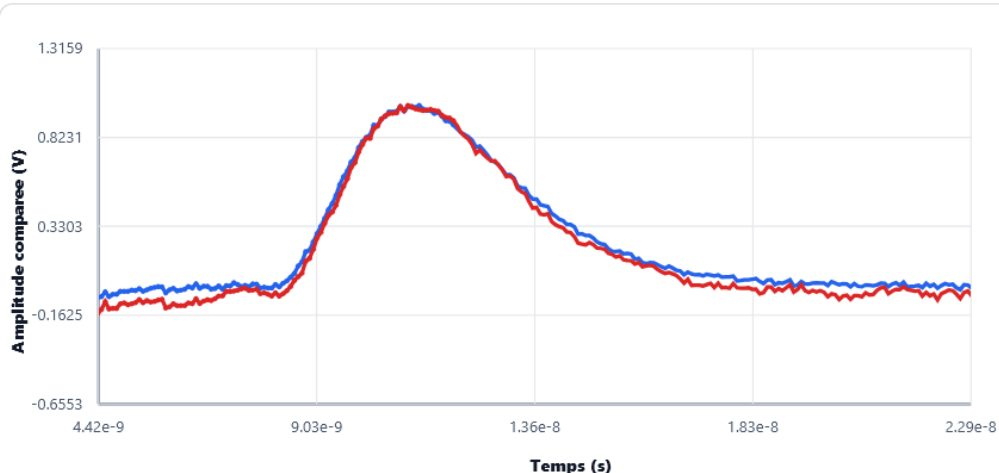
signal

execute • 7 ms

Editeur

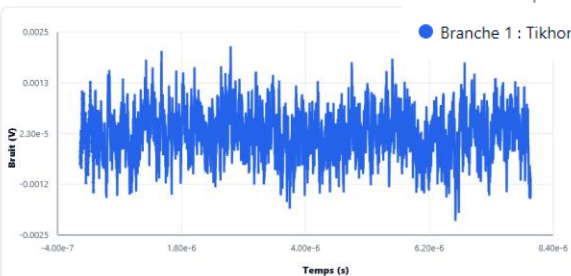


Avant branchement - Signal PAC (V)

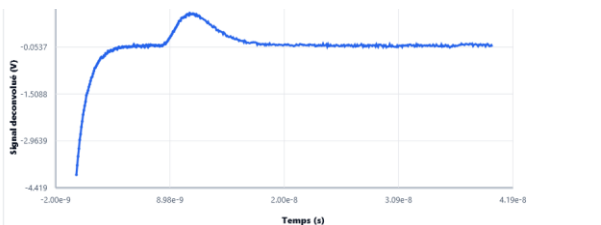


Zoom actif : clique sur "Reinitialiser zoom" pour revenir à l'affichage complet.

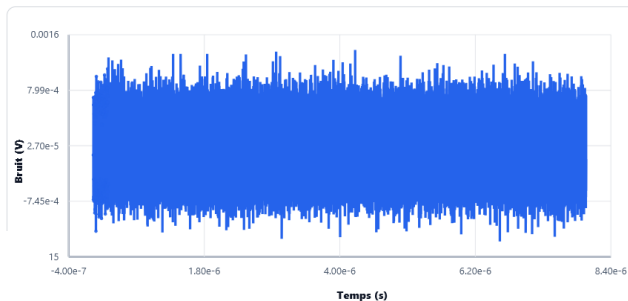
● Branche 1 : Tikhonov deconvolution - Signal deconvolué (V) ● Branche 2 : Bruit 1/f_2 - Signal deconvolué (V)



● Branche 2 : Bruit 1/f_2 - Bruit (V)



● Branche 2 : Bruit 1/f_2 - Signal deconvolué (V)



commune : Bruit blanc - Bruit (V)

Tikhonov deconvol...

Bloc Python

TRAITEMENT

ENTREE

Signal

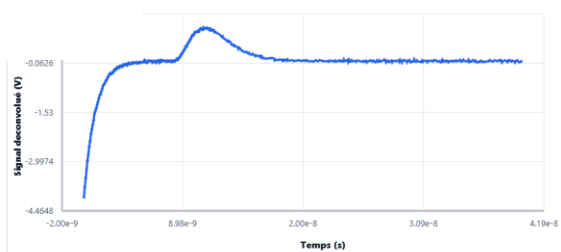
signal

SORTIE

Signal dev...

signal

execute • 1 ms



● Branche 1 : Tikhonov deconvolution - Signal deconvolué (V)



Reinitialiser

Ajouter script

Sauvegarder

Executer

Espaces de travail

Diagramme

Signaux

Logs

Parametres

Editeur[Voir les librairies importees](#)

36 bloc(s) disponible(s)

Delai circulaire
system_delay.py

Redimensionnement signal

SID4SIM RF SYSTEME

Spectre amplitude/phase
(FFT)

Smith depuis signaux RF
system_smith.py

TNS reel

2 sinus
sid4sim_2sinus.py

3 sinus

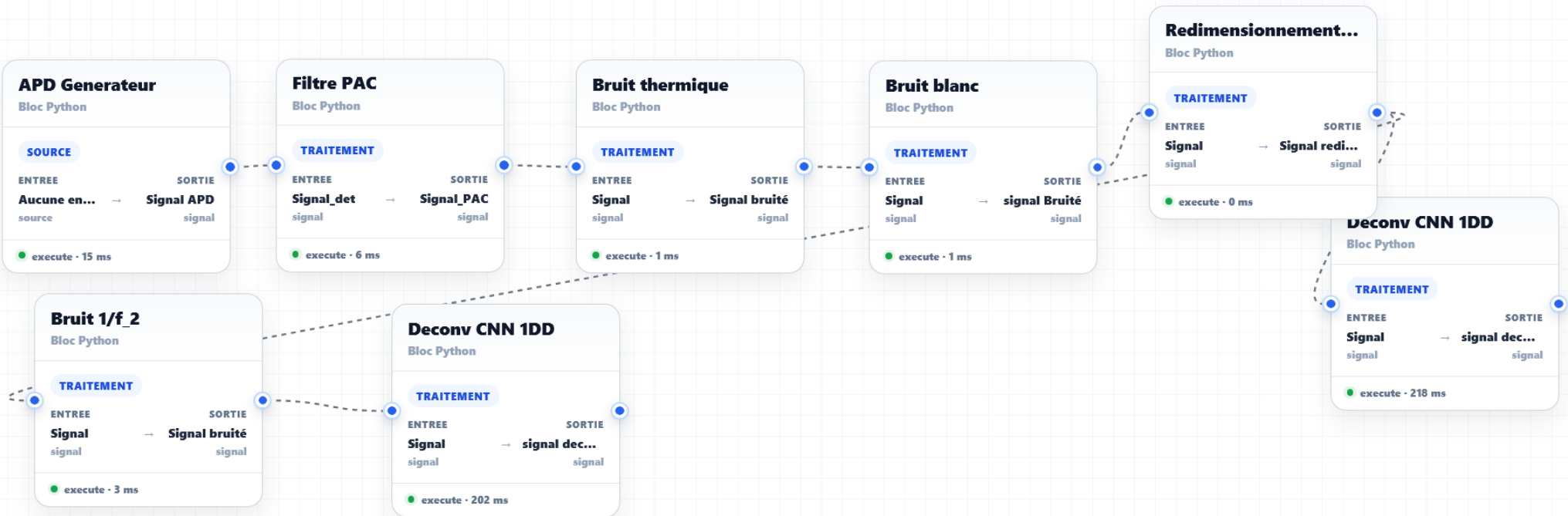
ADC
sid4sim_adc.py

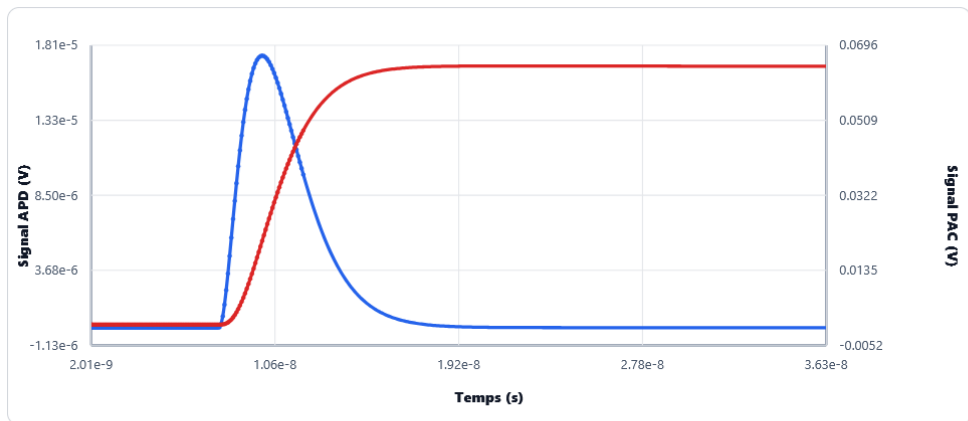
Filtre Bessel
sid4sim_bessel.py

Bruit
sid4sim_bruit.py

Filtre numérique
sid4sim_filtre_numerique.py

Gain simple

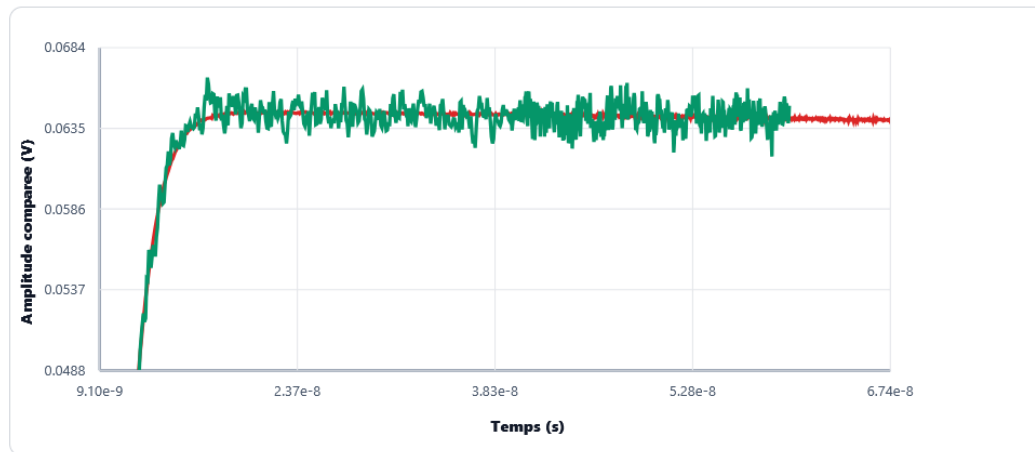




Deux échelles Y sont utilisées : la première courbe est lue à gauche, la deuxième à droite.

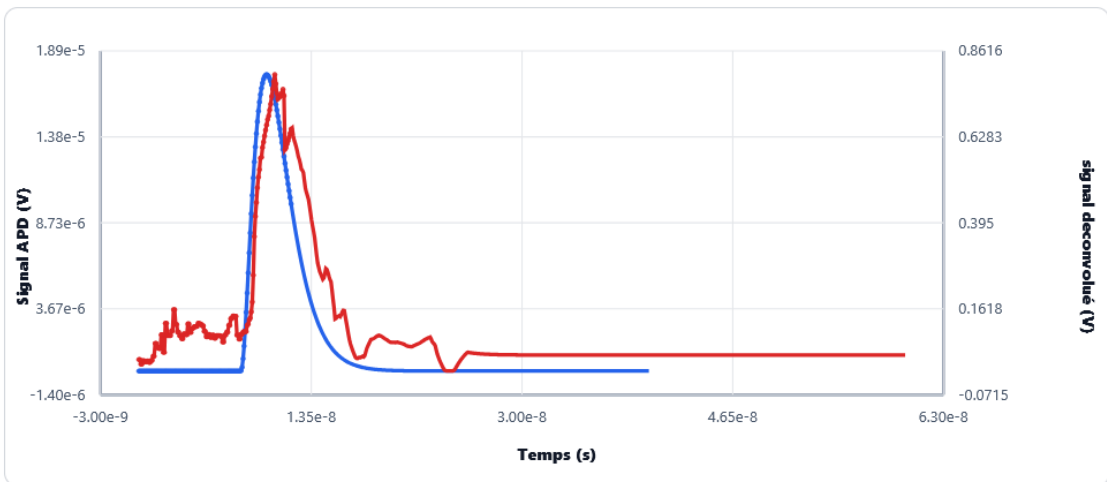
Zoom actif : cliquez sur "Reinitialiser zoom" pour revenir à l'affichage complet.

● Source indépendante · Signal APD (V) ● Avant branchement · Signal PAC (V)



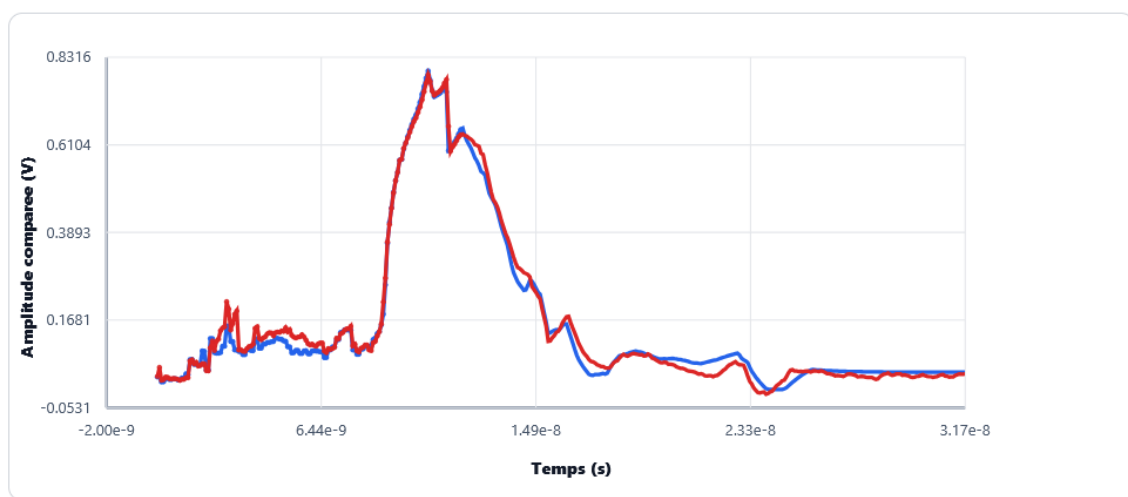
Zoom actif : cliquez sur "Reinitialiser zoom" pour revenir à l'affichage complet.

● Source indépendante · Signal APD (V) ● Avant branchement · signal bruité (V) ● Branche 2 : Bruit 1/f_2 · Signal bruité (V)



Deux échelles Y sont utilisées : la première courbe est lue à gauche, la deuxième à droite.

● Source indépendante · Signal APD (V) ● Branche 1 : Redimensionnement signal · signal deconvolué (V)



Zoom actif : cliquez sur "Reinitialiser zoom" pour revenir à l'affichage complet.

● Branche 1 : Deconv CNN 1DD · signal deconvolué (V) ● Branche 2 : Bruit 1/f_2 · signal deconvolué (V)

SID4SIM

Reinitialiser

Ajouter script

Sauvegarder

Executer

Espaces de travail

Librairies importées

53 bloc(s) disponible(s)

Soustraction

system_subtract.py

Somme

system_sum.py

Redimensionnement signal

resize_signal.py

SID4SIM Fichiers

6 bloc(s) - système

Créer données CSV demo

system_demo_csv_signal_source.py

Référence fichier

system_file_reference.py

Lire signal CSV

system_signal_csv_reader.py

Ecrire signal CSV

system_signal_csv_writer.py

Exporter paquet CSV

system_wave_bundle_csv_export.py

Lire paquet CSV

Diagramme

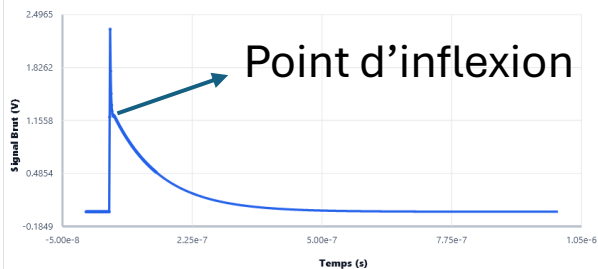
Signaux

Logs

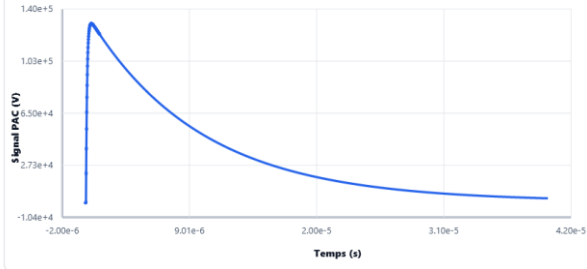
Paramètres

Editeur

Projets



Signal Brut (V)



Signal PAC (V)

Deconv CNN 1DD n'a pas appris les signaux de SiPM

SiPM Signal Simulator

Bloc Python

SOURCE

ENTREE

Aucune en...

source

SORTIE

Signal brut...

signal

execute - 22 ms

Filtre PAC

Bloc Python

TRAITEMENT

ENTREE

Signal det

signal

SORTIE

Signal PAC

signal

execute - 164 ms

Redimensionnement...

Bloc Python

TRAITEMENT

ENTREE

Signal

signal

SORTIE

Signal redi...

signal

execute - 0 ms

Deconv CNN 1DD

Bloc Python

TRAITEMENT

ENTREE

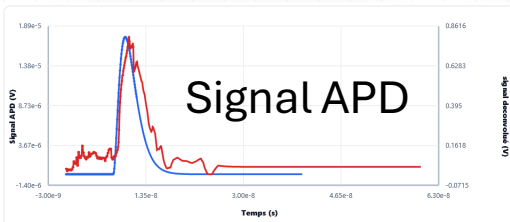
Signal

signal

SORTIE

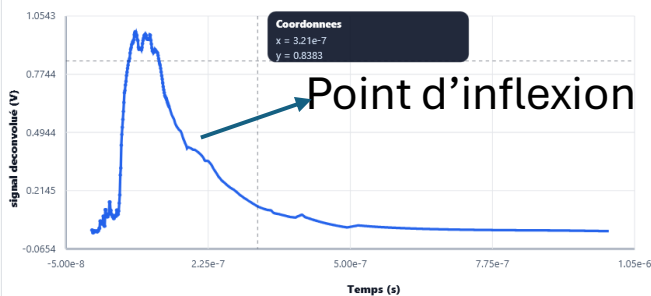
signal dec...

signal



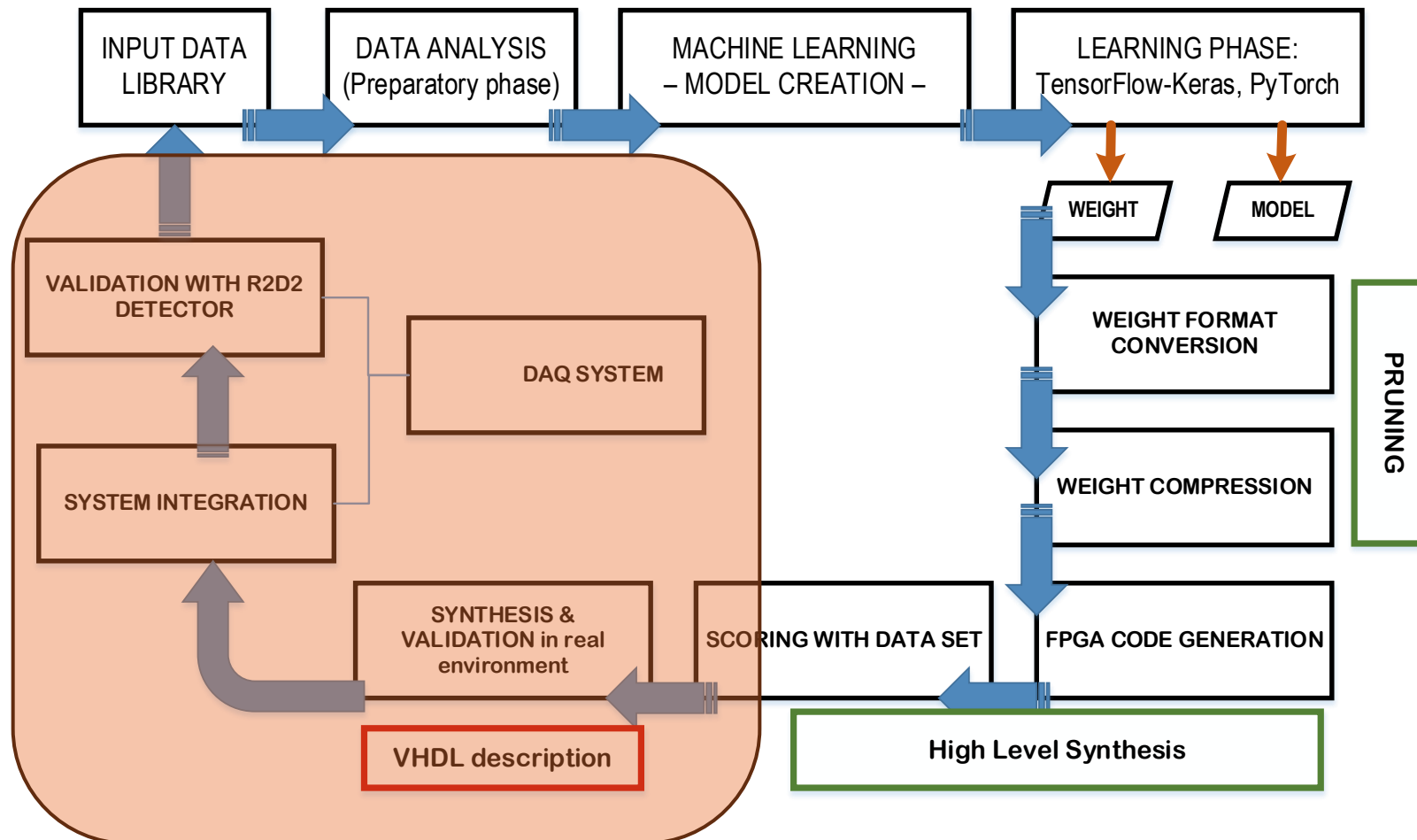
Deux échelles Y sont utilisées : la première courbe est lue à gauche, la deuxième à droite.

Signal APD (V) : Source indépendante - Signal APD (V) ; Signal APD (V) : Branche 1: Redimensionnement signal - signal deconvolué (V)

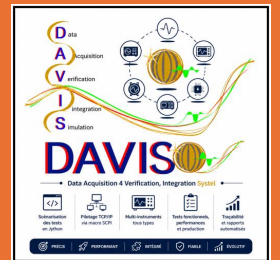


signal deconvolué (V)

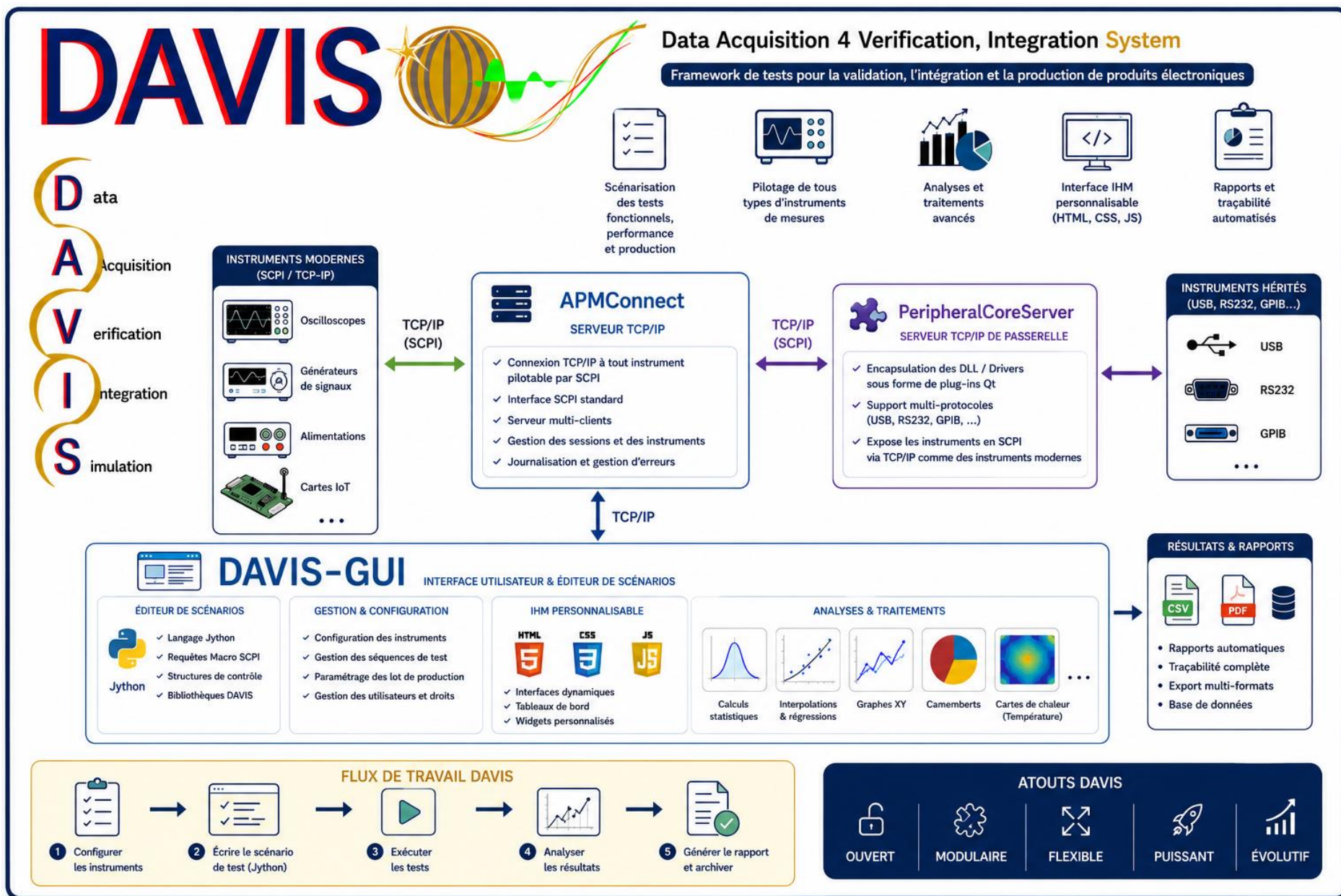
CREATION DE MES PROPRES OUTILS POUR CONTRÔLER MES MODÈLES



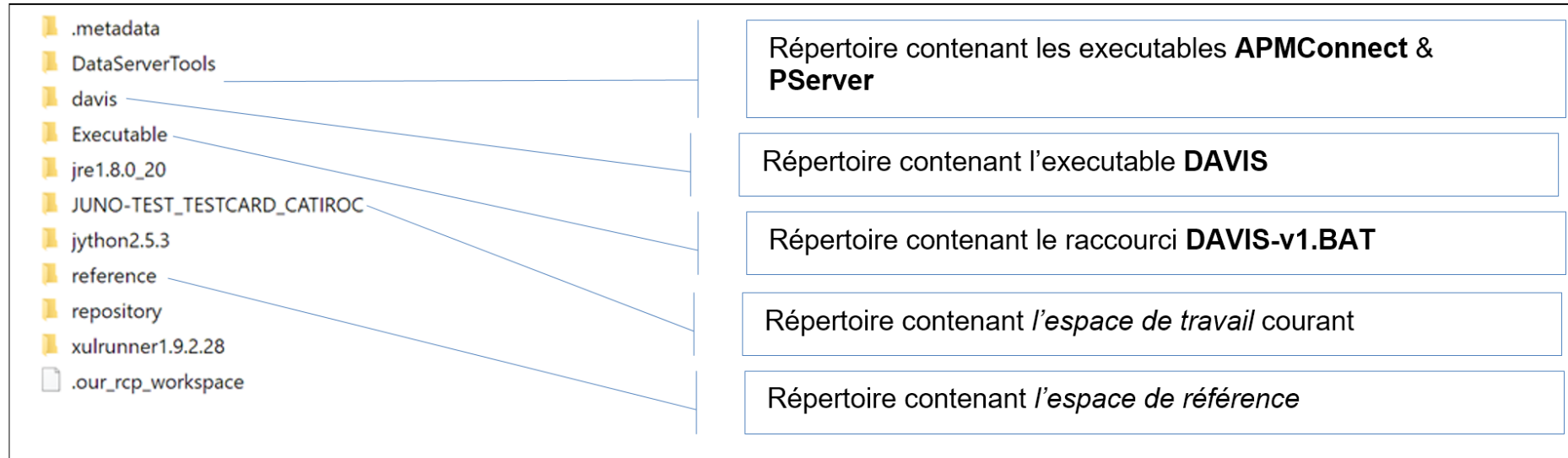
Modèles IA sur cartes
Scénarii de test
Analyses
Performances



Intégration de l'IP dans un système
Communication TCP/IP
Elaboration des scénarii de test
Evaluation des performances réelles

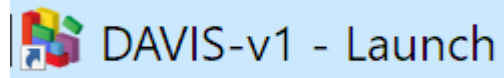


Copie/Coller d'un répertoire



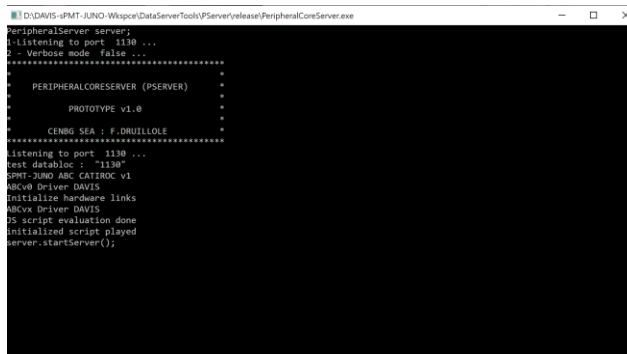
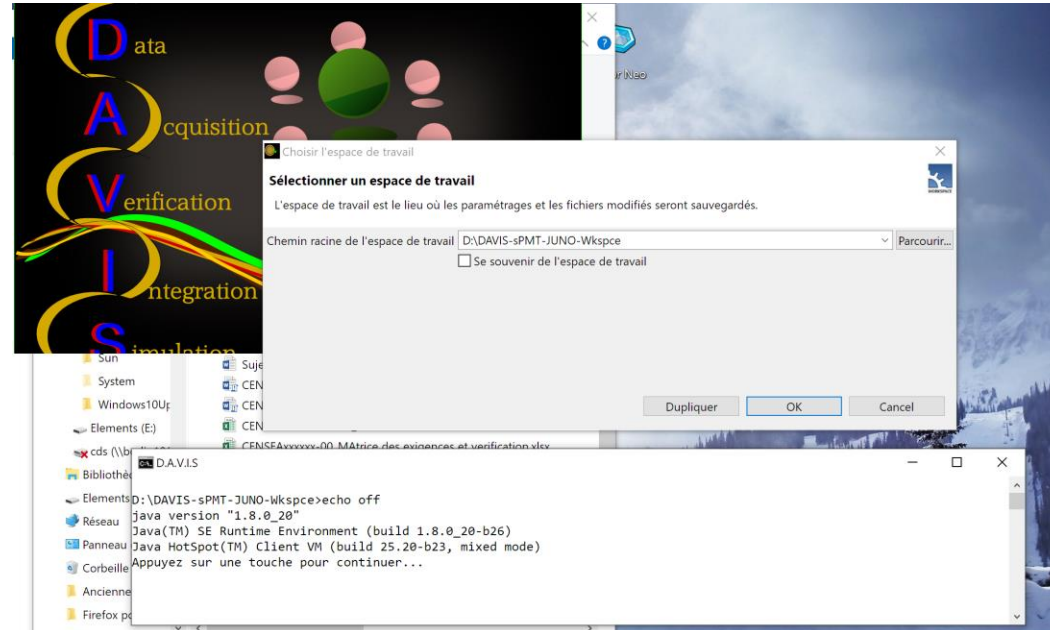
1

LANCEMENT DU LOGICIEL : D:\DAVIS-sPMT-JUNO-Wkspce



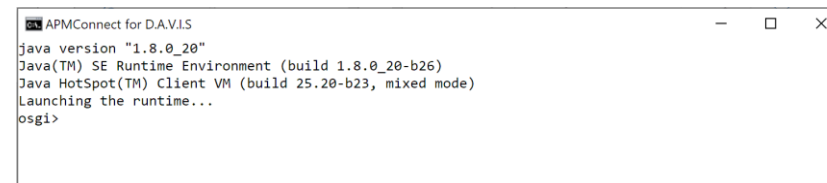
Exécution des 3 Logiciels formant le framework DAVIS

DAVIS-GUI : Contrôle les Tests Serie CATIROC à travers les scripts Jythons et APMConnect



PeripheralCoreServer : Pilote les Drivers pour :

- AFG3252C
- Keythley 2000

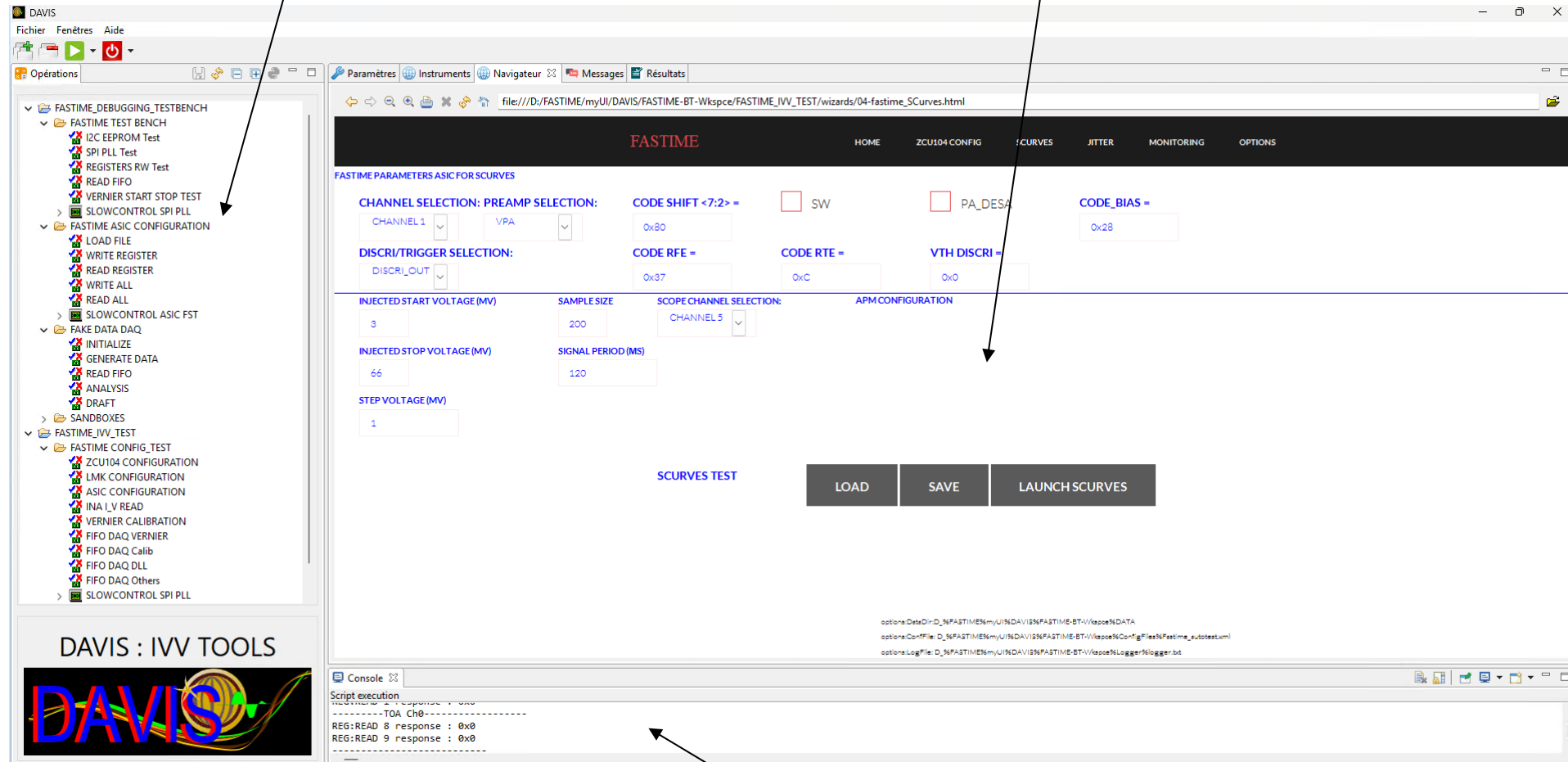


APMConnect : Pilote :

- PeripheralCoreServer
- QINJECTOR

Espace de travail:
Scénarii de test

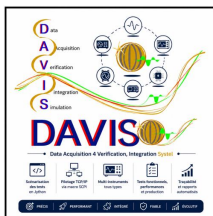
Interface utilisateur en
HTML



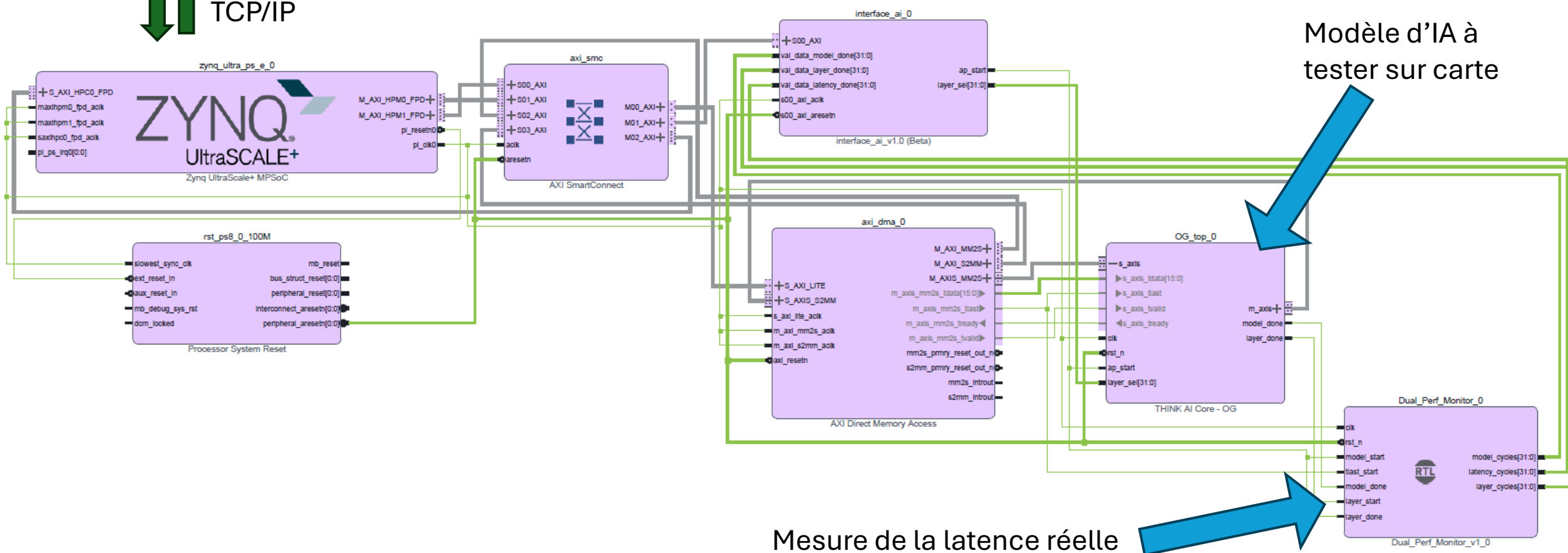
The screenshot displays the DAVIS software interface, which is divided into three main sections:

- Left Panel (Tree View):** A hierarchical tree structure showing the test scenario. The root is "FASTIME_DEBUGGING_TESTBENCH", which branches into "FASTIME TEST BENCH" and "FASTIME CONFIG TEST". Under "FASTIME TEST BENCH", there are sub-items like "I2C EEPROM Test", "SPI PLL Test", "REGISTERS RW Test", "READ FIFO", "VERNIER START STOP TEST", "SLOWCONTROL SPI PLL", and "FASTIME ASIC CONFIGURATION". Under "FASTIME CONFIG TEST", there are sub-items like "ZCU104 CONFIGURATION", "LMK CONFIGURATION", "ASIC CONFIGURATION", "INA LV READ", "VERNIER CALIBRATION", "FIFO DAQ VERNIER", "FIFO DAQ Calib", "FIFO DAQ DLL", "FIFO DAQ Others", and "SLOWCONTROL SPI PLL".
- Right Panel (HTML Interface):** A web-based configuration page titled "FASTIME". It contains various input fields and buttons for configuring the test. Key sections include:
 - FASTIME PARAMETERS ASIC FOR SCURVES:** Includes fields for "CHANNEL SELECTION: PREAMP SELECTION:" (CHANNEL 1, VPA), "CODE SHIFT <7:2> =" (0x80), "CODE BIAS =" (0x28), "DISCRI/TRIGGER SELECTION:" (DISCRI_OUT), "CODE RFE =" (0x37), "CODE RTE =" (0xC), and "VTH DISCRI =" (0x0).
 - INJECTED START VOLTAGE (MV):** 3
 - SAMPLE SIZE:** 200
 - SCOPE CHANNEL SELECTION:** CHANNEL 5
 - APM CONFIGURATION:**
 - INJECTED STOP VOLTAGE (MV):** 66
 - SIGNAL PERIOD (MS):** 120
 - STEP VOLTAGE (MV):** 1
 - Buttons:** "LOAD", "SAVE", and "LAUNCH SCURVES".
- Bottom Panel (Console / Logger):** A text area showing the output of the script execution. It displays "Script execution" and "REG:READ 8 response : 0x0" and "REG:READ 9 response : 0x0".

Console / Logger



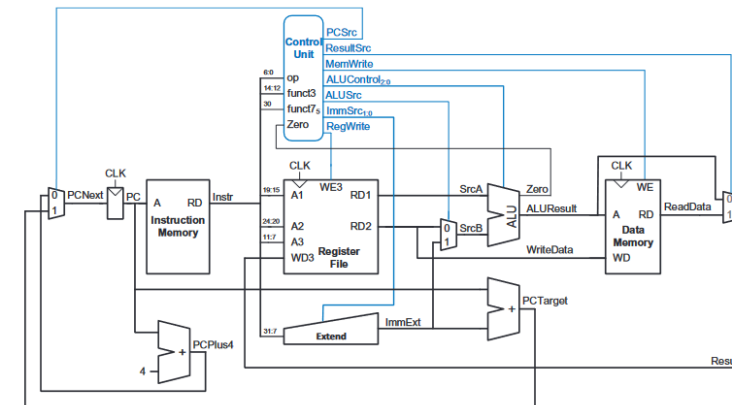
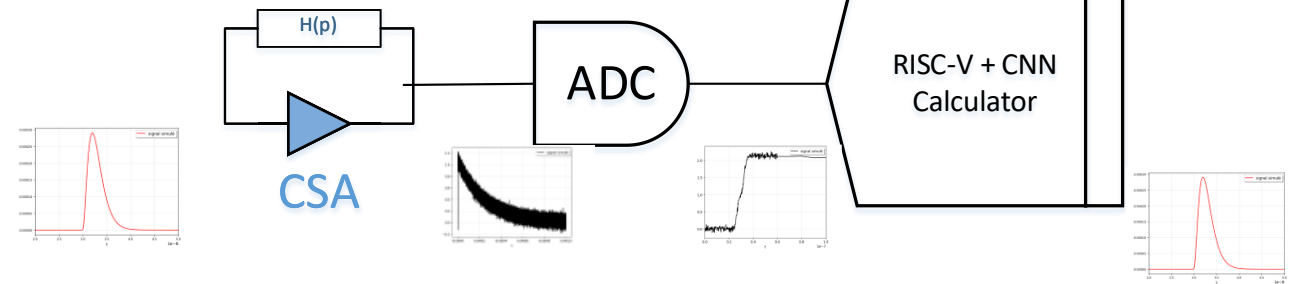
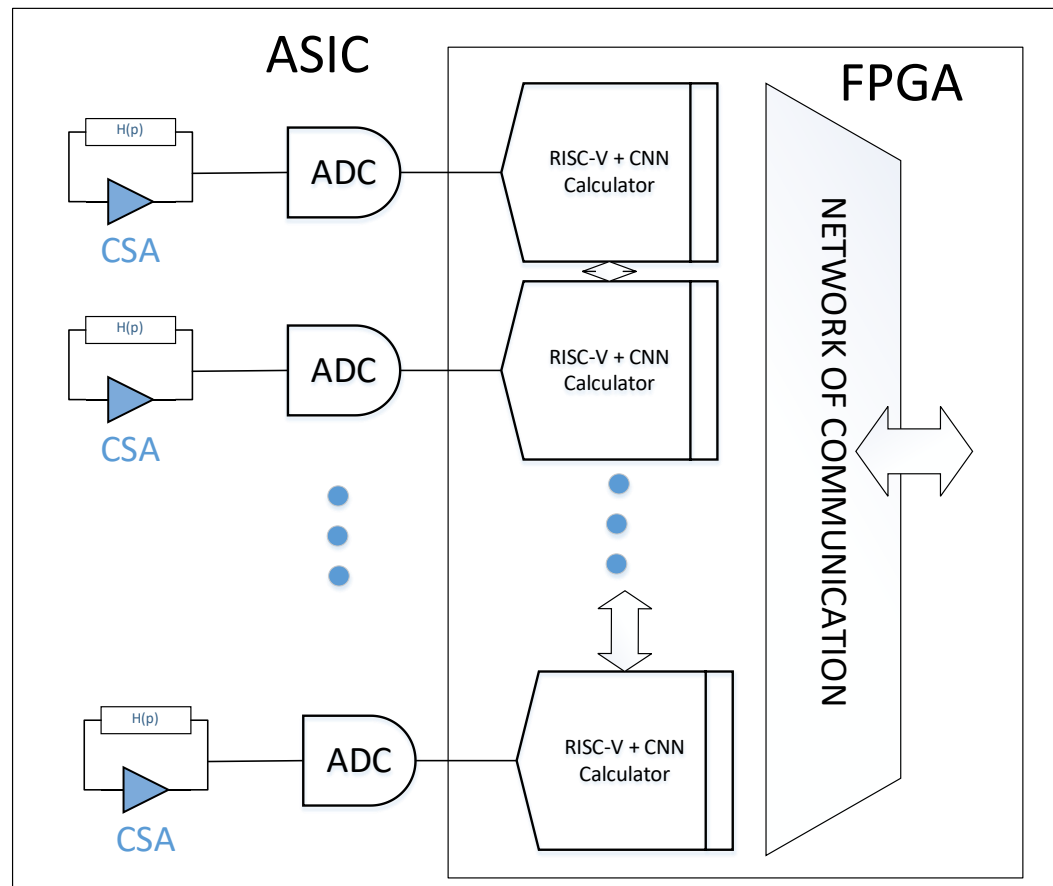
Communication Ethernet
TCP/IP





Aller vers un ASIC avec un processeur RISC-V
(facile à programmer)

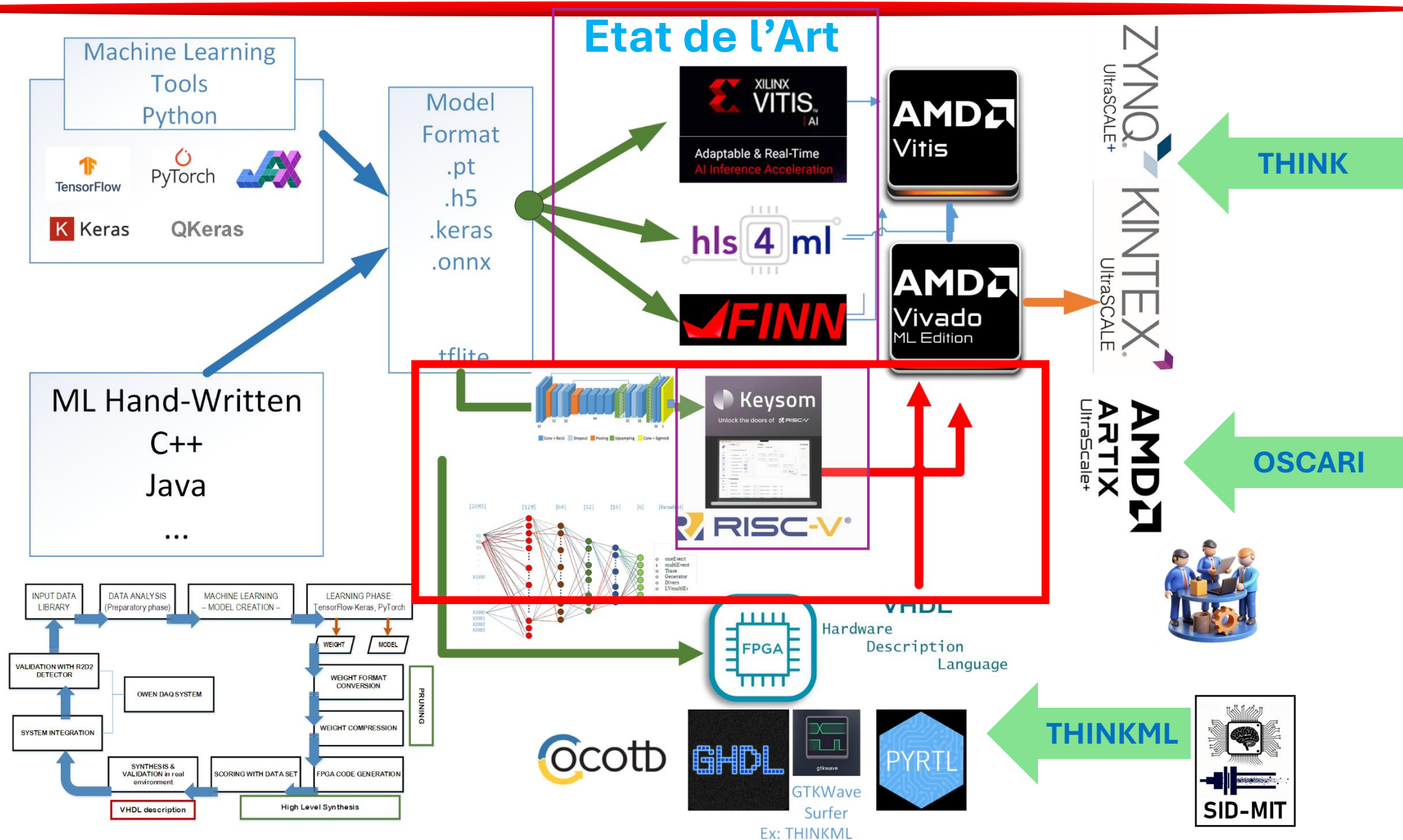
Absence de budget

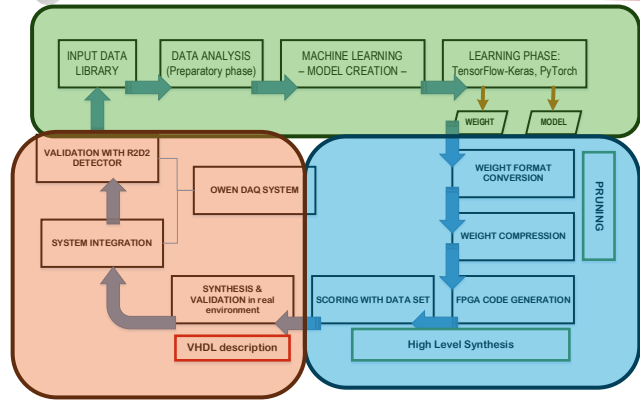


CATIE (CRT)
travaille sur
le sujet

→ Un processeur qui traite les informations suivant le flux des mesures

L'inférence optimisée

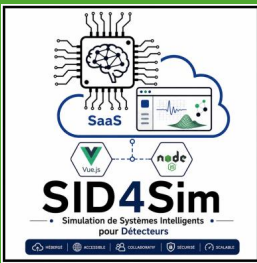




OUTILS COUVRANT TOUT LE PROCESSUS:

- OPTIMISATION DES MODELES EMBARQUES
- POUR CONTRÔLER MES MODÈLES

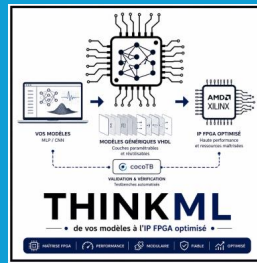
Modélisation
TNS –
Apprentissage –



Création des jeux de données
Apprentissage de modèles
Simulation des modèles
Optimisation des modèles:

- Elagage
- Quantisation
- Optim. Bayésienne

Inférence des
modèles IA
Sur FPGA AMD

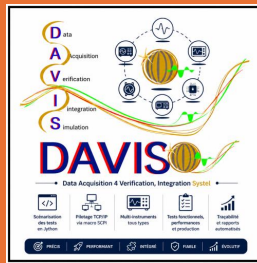


Choix des ressources matériels

- DSP vs parallélisme

Conversion en VHDL (IP)
Simulation matérielle (cocoTB)
Inférence sur carte
Evaluation des performances

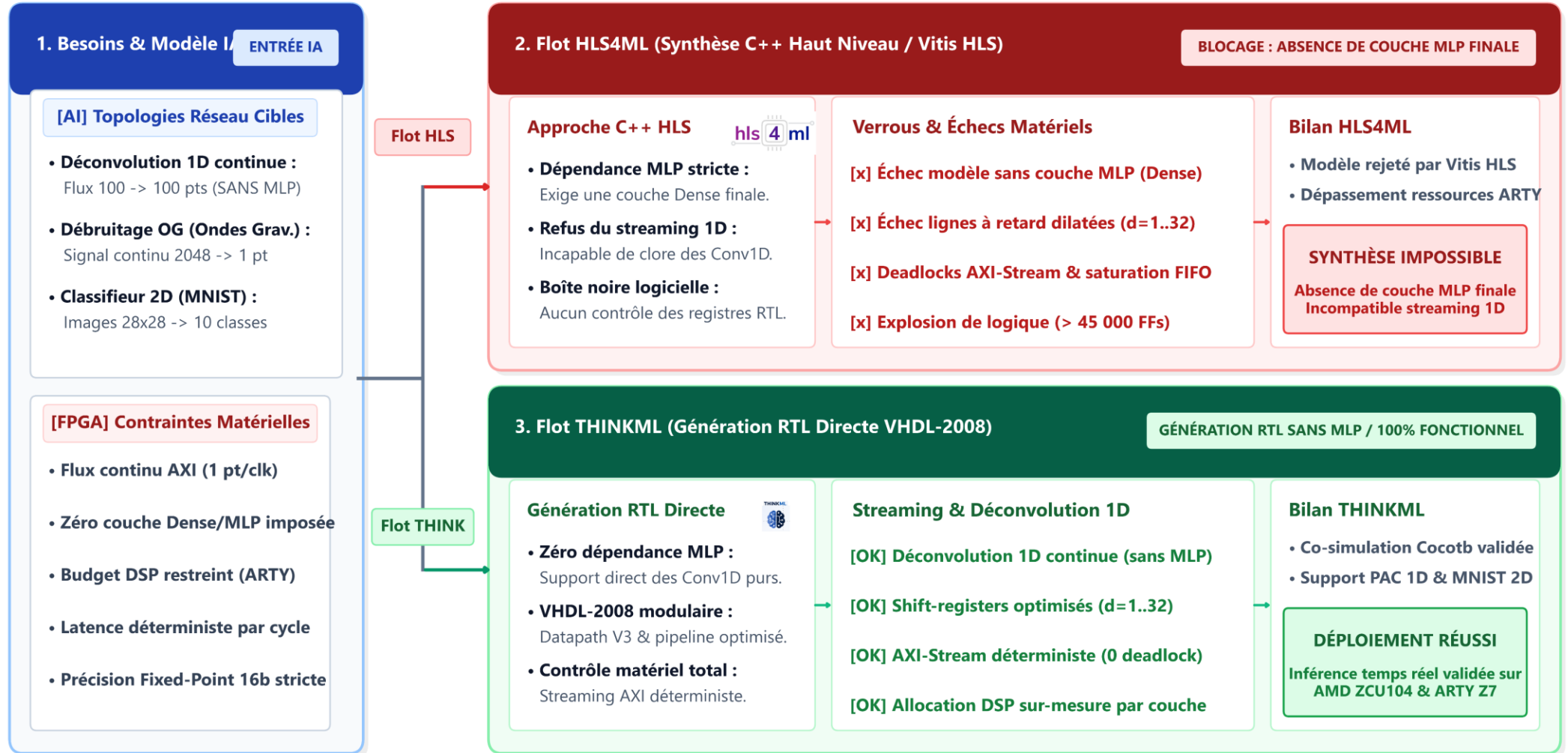
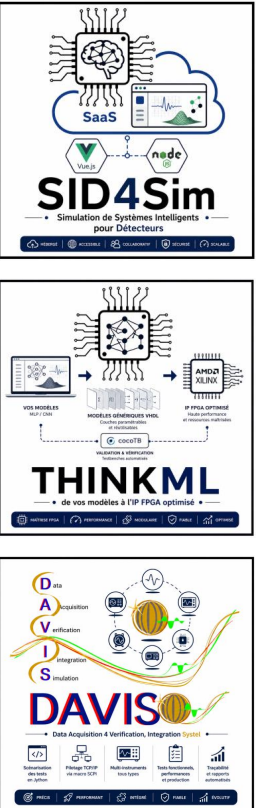
Modèles IA sur cartes
Scénarii de test
Analyses
Performances

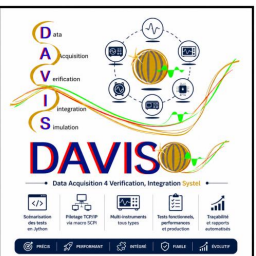
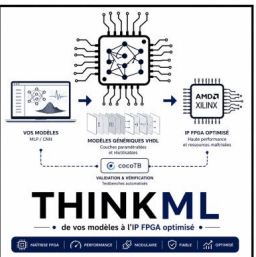
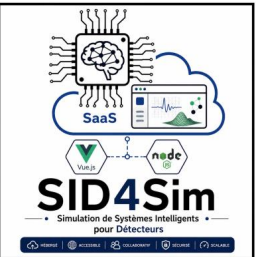


Intégration de l'IP dans un système
Communication TCP/IP
Elaboration des scénarii de test
Evaluation des performances réelles

POURQUOI PASSER DE HLS4ML À THINKML ?

Limites de la synthèse C++ HLS (obligation de couche MLP) & Avantages de la génération RTL Directe (VHDL)





- Continuer à explorer les possibilités des 3 frameworks et les améliorer.
- Développer une optimisation automatique du nombre de DSP et de la parallélisation des calculs du modèle VHDL
 - Optimisation Bayésienne (Thèse R. Bertrand au CPPM)
 - Exploration par algorithmie (algorithme glouton, programmation dynamique...) (IP2I Lyon)
- Explorer les modèles résolvant le TNS de nos détecteurs
- Création d'une base de données des signaux des détecteurs (SiPM, APD, CdTe)
- Travaux sur l'utilisation de RISC-V combiné avec l'inférence en VHDL
 - Recherche de signaux rares (Physique des neutrinos)
 - Reconstruction de traces (Physique nucléaire: radioactivité di-protons)
 - Contrôle de procédés de fabrication et maintenance préventive (Recherche appliquée pour l'industrie)



Continuer à hauteur de 20% de mon temps