Deep Learning Optimisé - Jean Zay

Introduction – Jean Zay – GPU

cnrs

INSTITUT DU
DÉVELOPPEMENT ET DES
RESSOURCES EN
INFORMATIQUE
SCIENTIFIQUE

Présentation de DLO-JZ

Plan ◂

Imagenet / Resnet-50 ◂

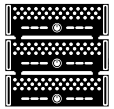
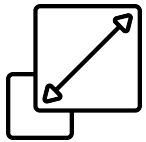
Présentation des participants ◂

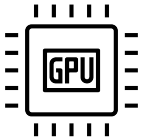
Présentation - Sujets traités

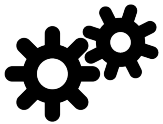
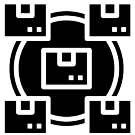
Jour 1
- Jean Zay
- Les enjeux de la montée à l'échelle
- GPU computing
- **Distribution - Data Parallelism**
- Dataloader

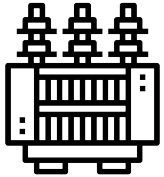
Jour 2
- Profiler
- Data Augmentation
- ***Optimizer* et larges *batch***
- Résultat sur Weight & Biases
- Sources et astuces

Jour 3
- **Les parallélismes de modèle**
- Deepspeed

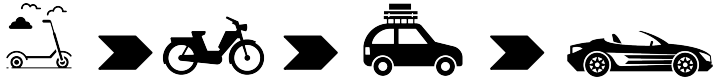



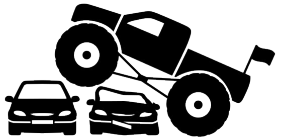


Imagenet Race - Déroulé des TP

- Les TP des Jours 1 et 2:
  - Optimisations système : GPU, Mixed Precision, Data Parallelism
  - DataLoader
  - Profiler
  - Data Augmentation
  - *Optimizer* et Hyperparamètres avec des larges *batch*
  - Course de *job* sur 32 GPU pendant les nuits



- Les TP du Jour 3 :
  - Résultats de la course : Meilleur *Top-1 validation accuracy*
  - *Model parallelisms* avec un modèle CoAtNet-7 (gros Vision Transformer SOTA)

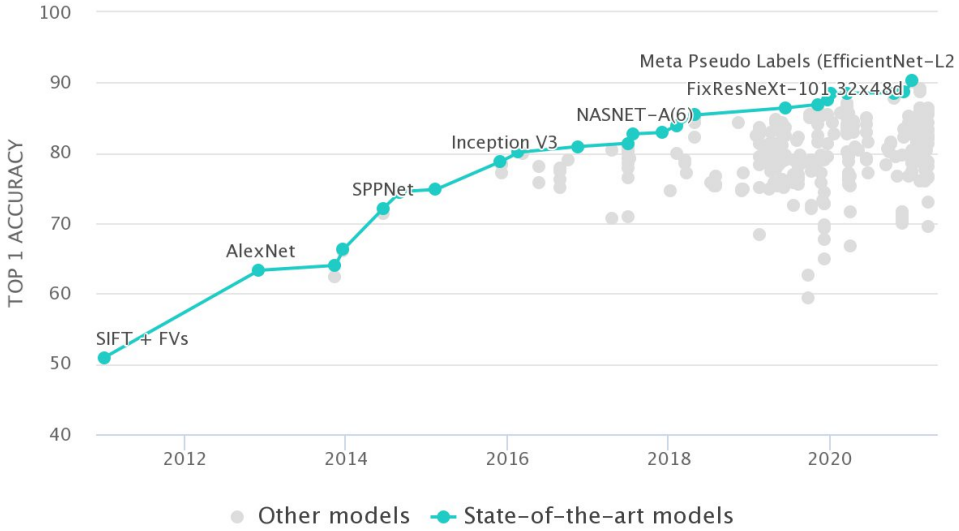


- Mini Jean Zay réservé : 32 GPU A100 sur 8 noeuds

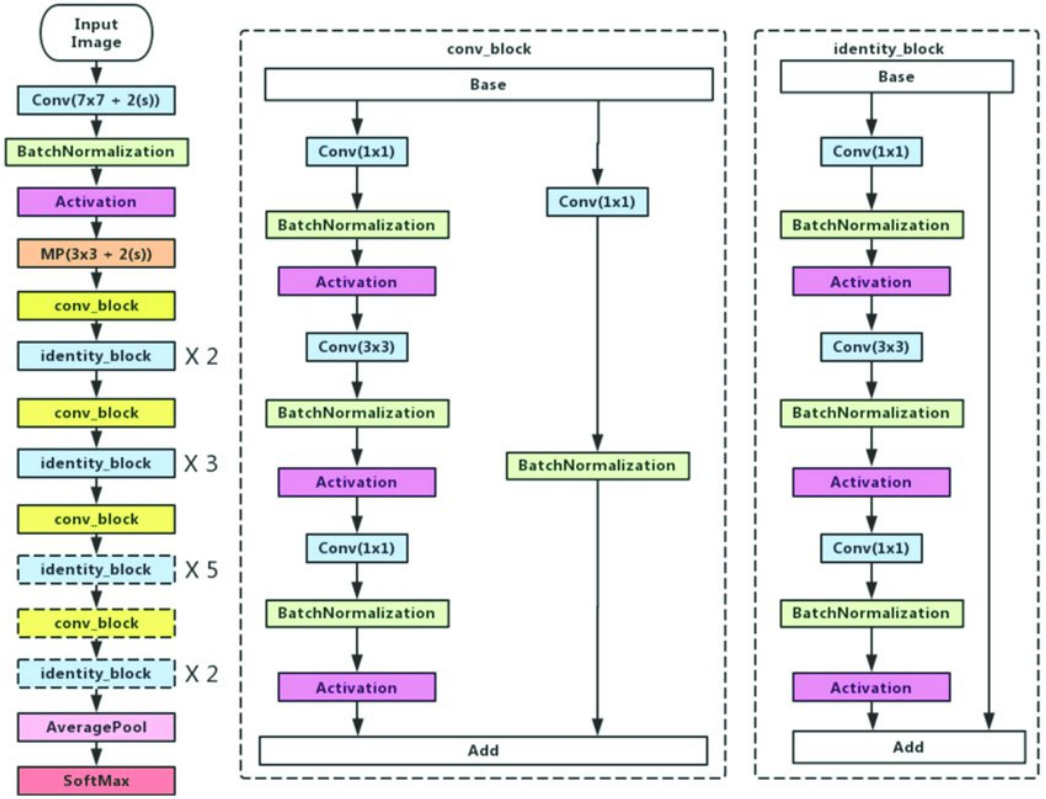
Données - Imagenet

But:
Classification (1000 classes)

Dataset:
Train dataset: **1,2 Millions** d'images labellisées
Validation dataset: **50 000** images labellisées
http://www.image-net.org/



Imagenet - Resnet-50



Resnet :

- Residual Learning
- BatchNorm layer
  - Remplace les *dropouts*
- Average Pooling
  - Rend le modèle indépendant de la taille des images d'entrée

Imagenet - Resnet-50

How long does it take to train Resnet-50 on ImageNet?

14 days

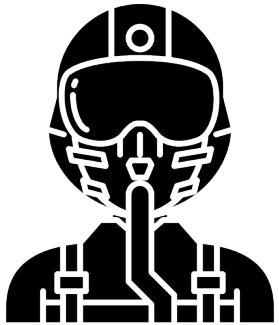
NVIDIA M40 GPU

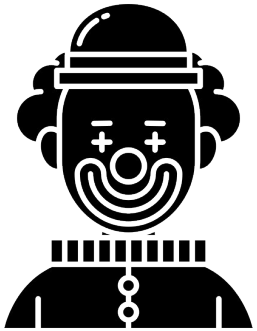
Imagenet - Resnet-50

Training Resnet-50 on Imagenet

Facebook Caffe2	UC Berkeley, TACC, UC Davis Tensorflow	Preferred Network ChainerMN	Tencent TensorFlow	Sony Neural Network Library (NNL)	Fujitsu MXNet
1 hour	31 mins	15 mins	6.6 mins	2.0 mins	1.2 mins
Tesla P100 x 256	1,600 CPUs	Tesla P100 x 1,024	Tesla P40 x 2,048	Tesla V100 x 3,456	Tesla V100 x 2,048
Apr	Sept	Nov	July	Nov	Apr
2017				2018	2019

Présentation des participants





Jean Zay

Supercalculateur ◂

Jean Zay ◂

Environnements de travail ◂

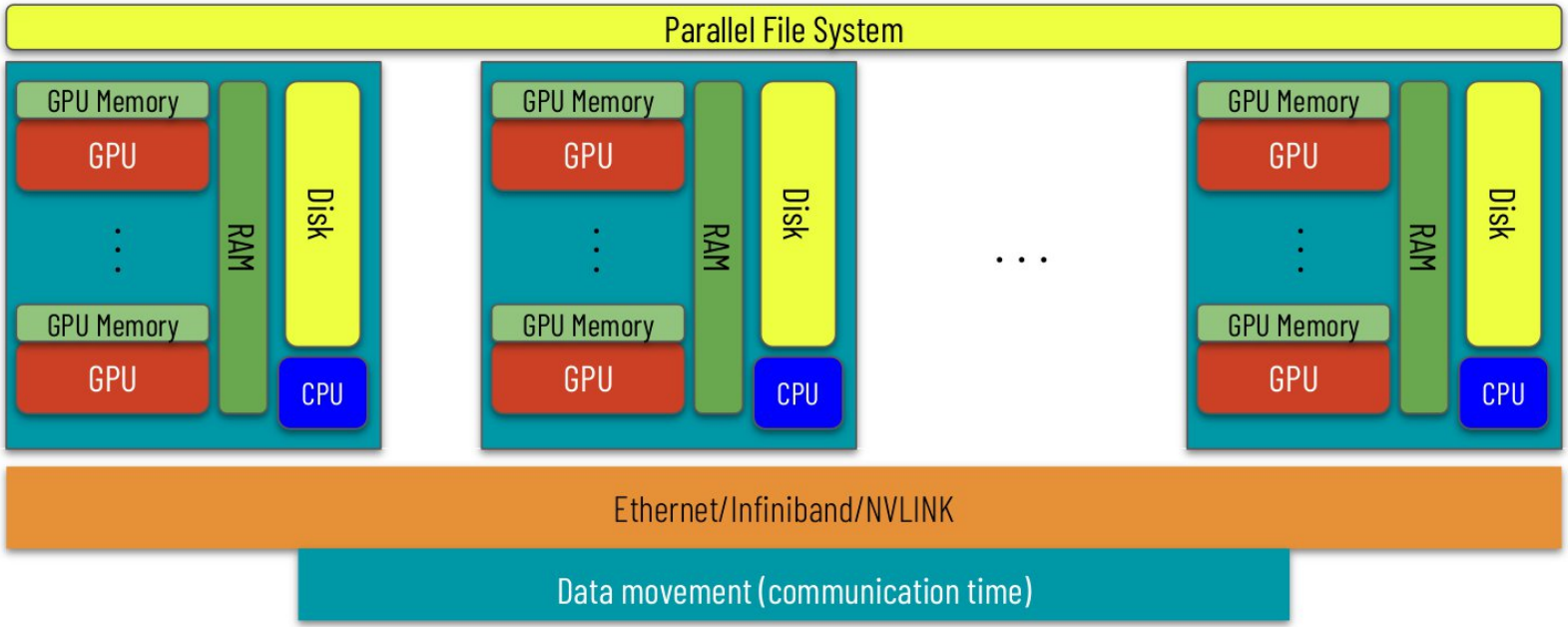
Supercalculateur

Multi-node multi-GPU HPC clusters

Système sécurisé

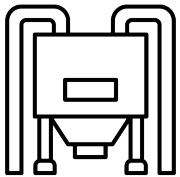
Stockage partagé / Stockage distribué

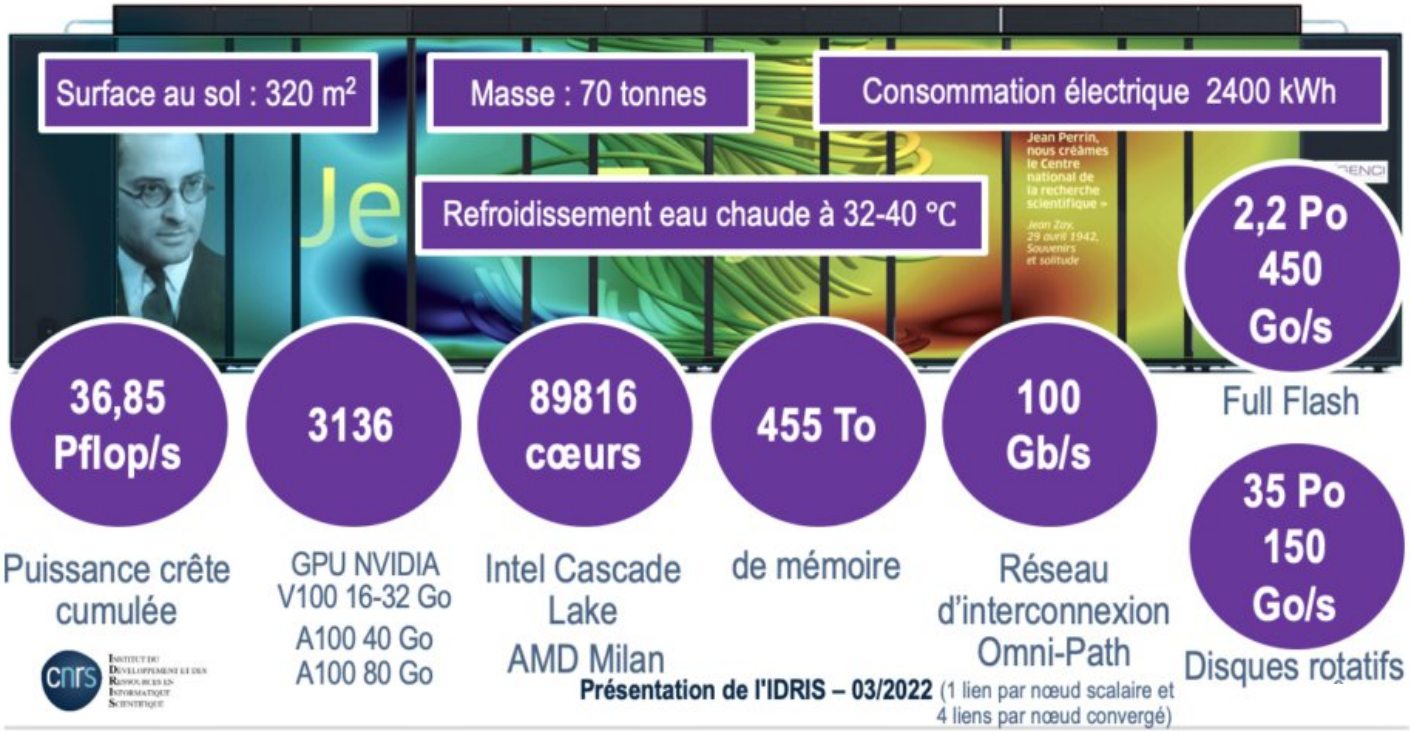
Architecture la plus répandue pour un Supercalculateur

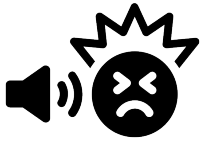


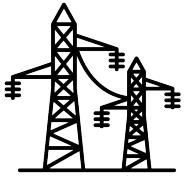
Jean Zay

Tier1 - Premier supercalculateur convergé français pour l'Intelligence Artificielle (IA) et le calcul de haute performance (HPC)

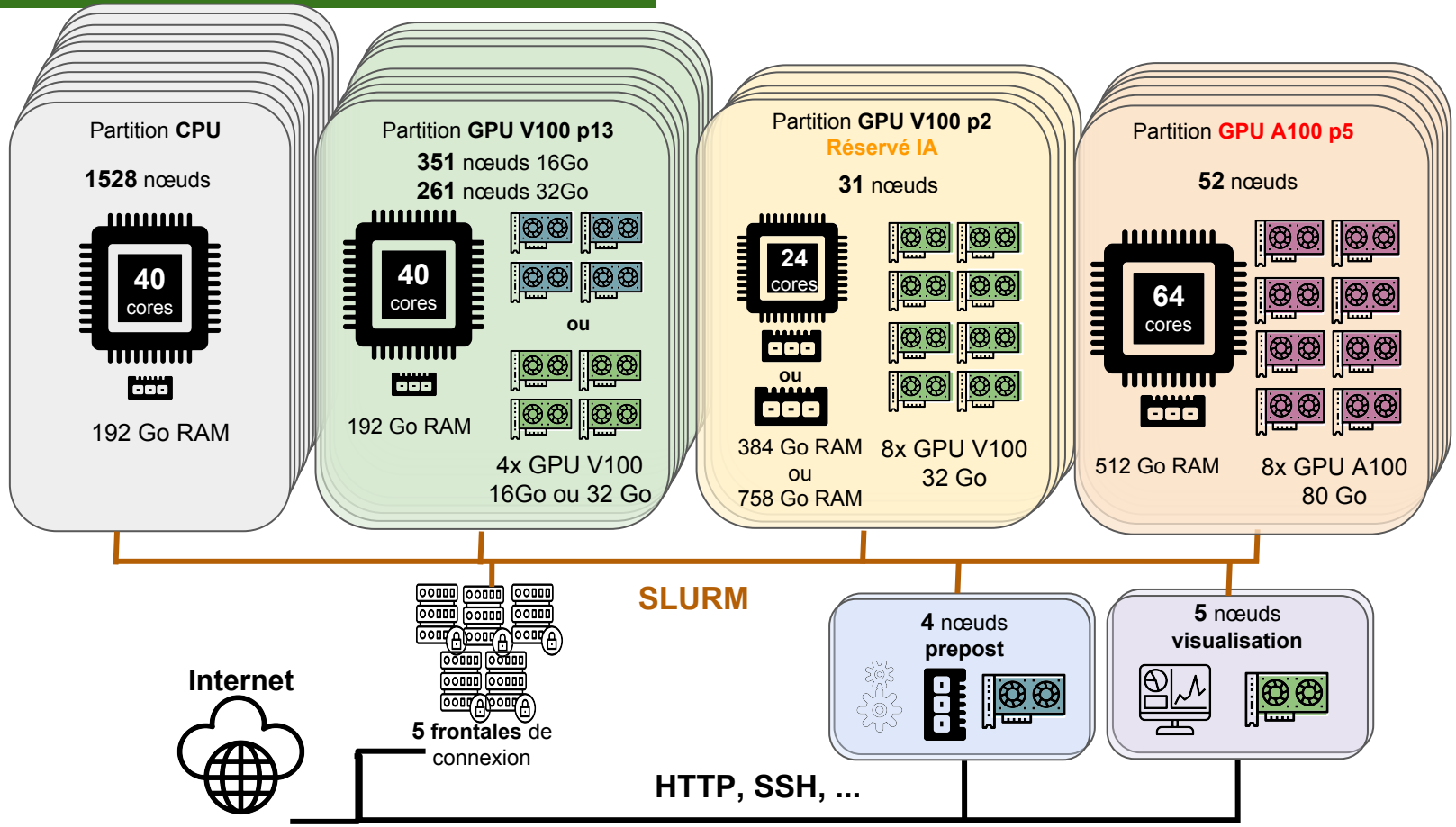




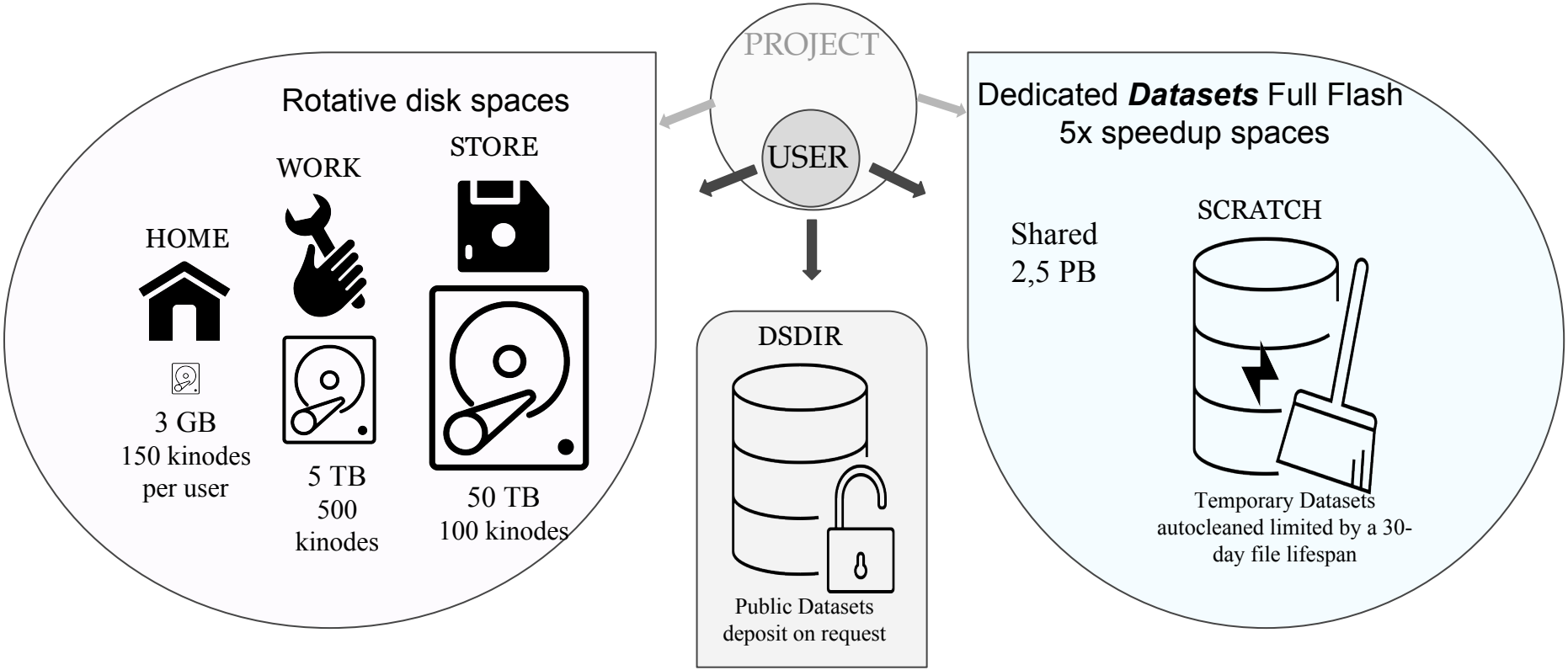




Jean Zay: Partitions



Jean Zay: Espaces de stockage



Jean Zay: Environnements

Modules
Embarqués sur Jean Zay

Catalogue de modules mutualisés

- Maintenus et optimisés par l'IDRIS

```
$ module avail caffe pytorch tensorflow mxnet

$ module purge
$ module load tensorflow-gpu/py3/2.4.1
```

Personnalisez votre environnement en ajoutant des librairies (attention à la compatibilité)

```
$ pip install --user --no-cache-dir <paquet>
```

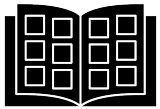
```
(my_environment)$ conda install <paquet>
```

Conteneur Singularity
A partir de dépôts publiques

Build a read-only secure SIF Image on Jean Zay

Jean Zay: Jupyter

1. Charger un module IA (avec Jupyter)



2. **idrjup** pour Jupyter Notebook / **idrlab** pour Jupyter Lab

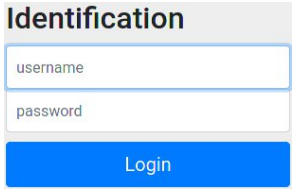
A partir d'une frontale :

[**frontal**]**$ srun** --ntasks=1 --cpus-per-task=10 --gres=gpu:1 --time=01:00:00 **--pty bash**

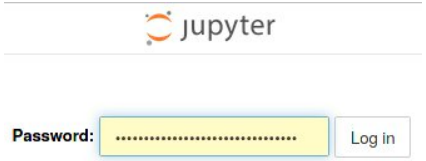
ou d'un nœud de calcul :

[**node**]**$ idrjup** $WORK
INFO 2019-11-22 12:10:08,916 Starting Jupyter server. Please wait...:
INFO 2019-11-22 12:10:08,933 --Launching Jupyter server. Please wait before attempting to connect...
INFO 2019-11-22 12:10:14,070 --Jupyter server launched. Please connect.
URL de connexion : **https://idrvprox.idris.fr**
Mot de passe URL : <mot de passe utilisateur>
Mot de passe jupyter : **xxxxpasswordjupyterxxxxx**

3. Double authentification

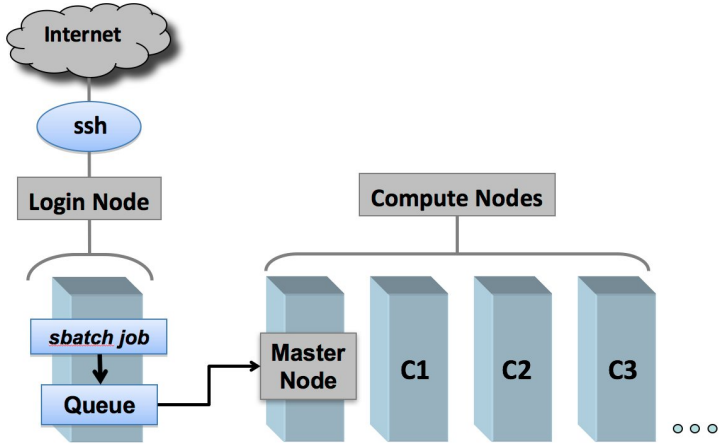


○ au serveur



○ au notebook

Soumission Jobs - SLURM




```
$ sbatch script.slurm
```

```
$ squeue -u $USER

JOBID  PARTITION  NAME  USER  ST   TIME  NODES  NODELIST(REASON)
  235  part_name  test   abc   R  00:02      1  r6i3n1


$ scontrol show job $JOBID
```

script.slurm

```
#!/bin/bash

#SBATCH --job-name="GTSRB Full conv."      # number of job
#SBATCH --node=2                           # number of node
#SBATCH --gres=gpu:4                       # number of GPU per node
#SBATCH --ntasks-per-nodes=4               # number of task per node
#SBATCH --cpus-per-task=10                 # nbr of cores
#SBATCH --hint=nomultithread               # no hyper threading
#SBATCH --time=03:00:00                    # max execution time
#SBATCH --output="_batch/G.out"            # out file
#SBATCH --error="_batch/G.err"             # error file
#SBATCH --mail-user=mail@domaine
#SBATCH --mail-type=ALL

MODULE_ENV="tensorflow-gpu/py3/2.2.0"
RUN_DIR="$WORK/fidle/GTSRB"
RUN_SCRIPT="./run/full_convolutions.py"

# ---- Module
module purge
module load "$MODULE_ENV"

# ---- Run it...
cd "$RUN_DIR"
srun python "$RUN_SCRIPT"
```

Jean Zay: Soumission de Jobs automatisée pour notebook python

```
from idr_pytools import gpu_jobs_submitter, display_slurm_queue, search_log
```

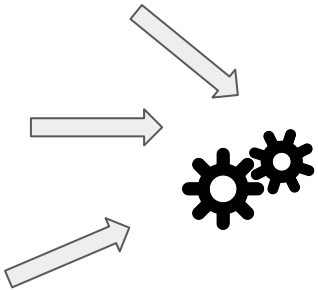
```
jobid = gpu_jobs_submitter(command, n_gpu, MODULE, name=name,
                           account='user@gpu', time_max='00:10:00', qos='qos_gpu-dev')
```

srun python + 'dlo_jz.py -b 128 --image-size 176 --test'

Commande

Nombre de GPU

MODULE



./slurm

batch.slurm

```
#!/bin/bash


#SBATCH --job-name="GTSRB Full conv."          # number of job
#SBATCH --node=2                               # number of node
#SBATCH --gres=gpu:4                           # number of GPU per node
#SBATCH --ntasks-per-nodes=4                   # number of task per node
#SBATCH --cpus-per-task=10                     # nbr of cores
#SBATCH --hint=nomultithread                   # no hyper threading
#SBATCH --time=03:00:00                        # max execution time
#SBATCH --output="_batch/G.out"                # out file
#SBATCH --error="_batch/G.err"                 # error file
#SBATCH --mail-user=mail@domaine
#SBATCH --mail-type=ALL

MODULE_ENV="tensorflow-gpu/py3/2.2.0"
RUN_DIR="$WORK/fidle/GTSRB"
RUN_SCRIPT="./run/full_convolutions.py"

# ---- Module
module purge
module load "$MODULE_ENV"


# ---- Run it...
cd "$RUN_DIR"
srun python "$RUN_SCRIPT"
```

sbatch

TP0 : Accès et prise en main du notebook

```
local:~$ ssh jean-zay

jz:~$ cd $WORK
jz:~$ cp -r $ALL_CCFRWORK/DLO-JZ .
jz:~$ module load pytorch-gpu/py3/1.11.0
jz:~$ idrjup $WORK
```

- Ouvrir le notebook DLO-JZ_Jour1.ipynb
- Choisir un pseudonyme
- Lancer un job
- Prendre en main le script de référence et les différentes fonctionnalités

Les enjeux de la montée à l'échelle

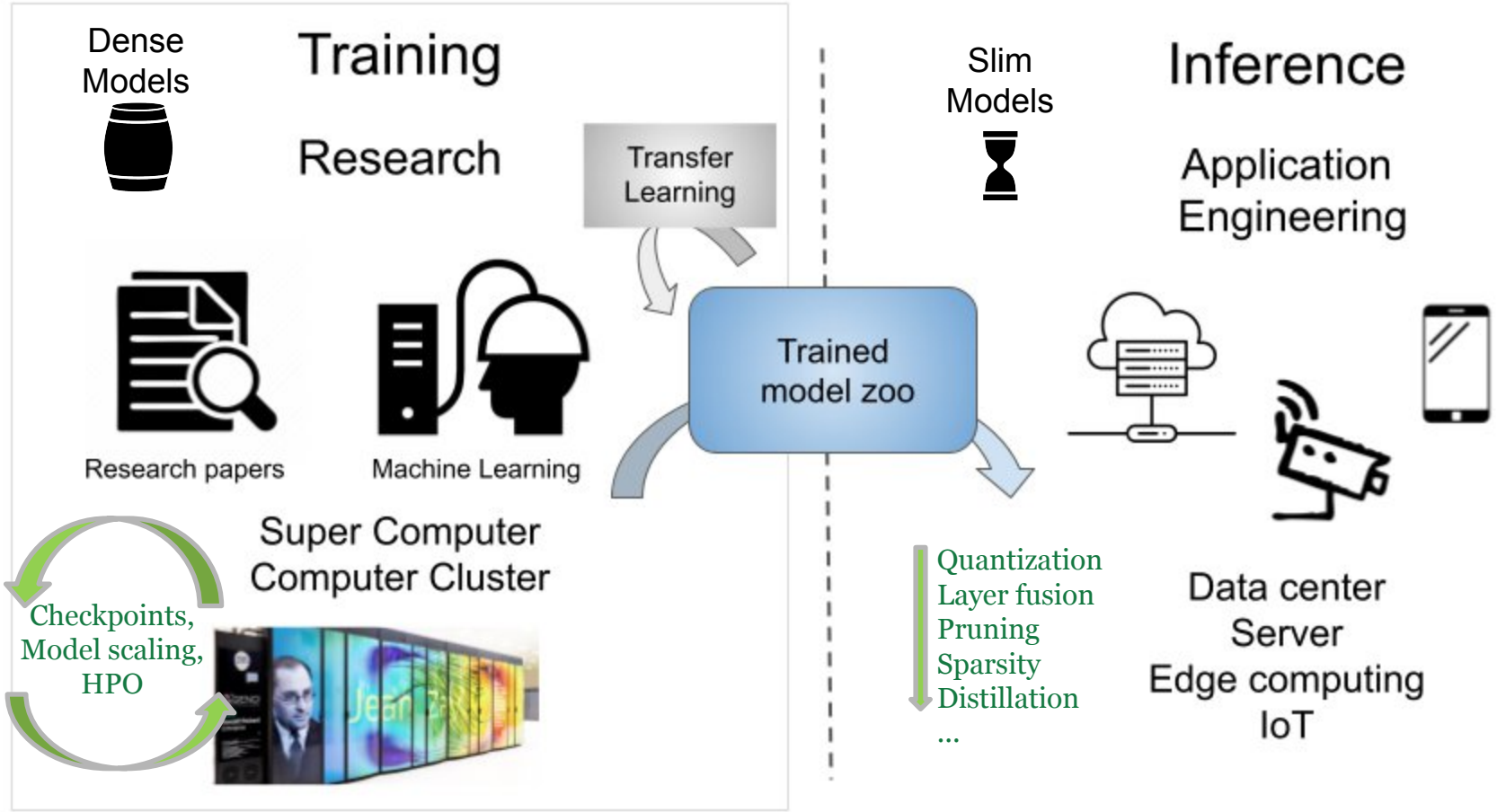
Temps d’apprentissage ◂

Empreinte Mémoire ◂

Solutions ◂

Economie énergétique ◂

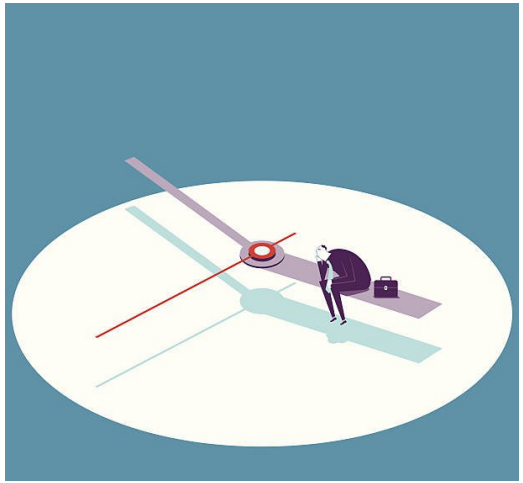
Apprentissage / Inférence



Contraintes du Deep Learning

2 problèmes à traiter:

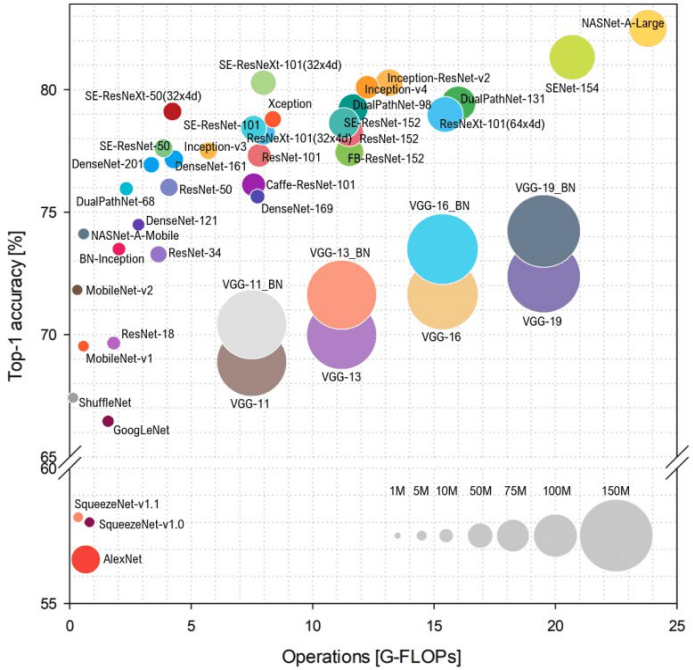
Temps d’apprentissage



Surconsommation mémoire (OOM)

Les Gros Modèles

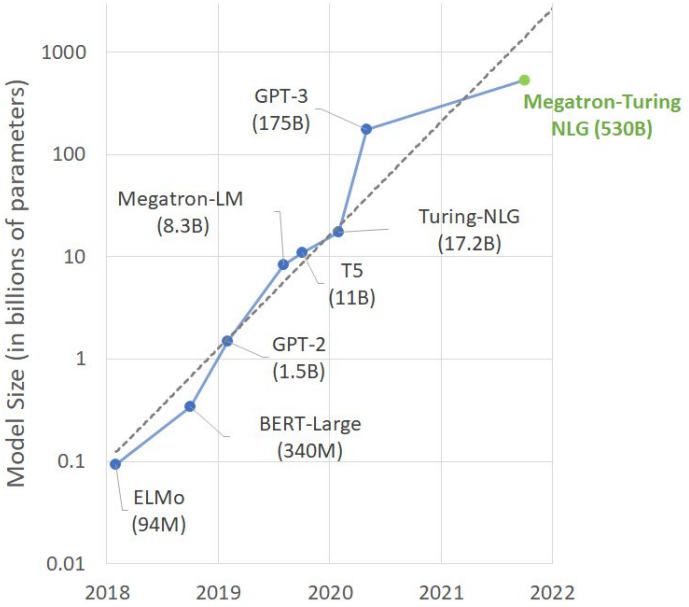
Convolutional Neural Network



Les modèles gros, et profonds permettent d'obtenir de meilleurs métriques d'*accuracy*.

Les énormes modèles provoquent de très coûteux temps de calcul et de larges empreintes mémoire (4 Go pour un modèle d'1 milliard de paramètres).

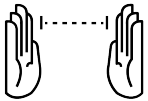
Transformers



Le temps de calcul

Le temps de calcul augmente avec le **nombre de FLOP nécessaire**, dépendant de :

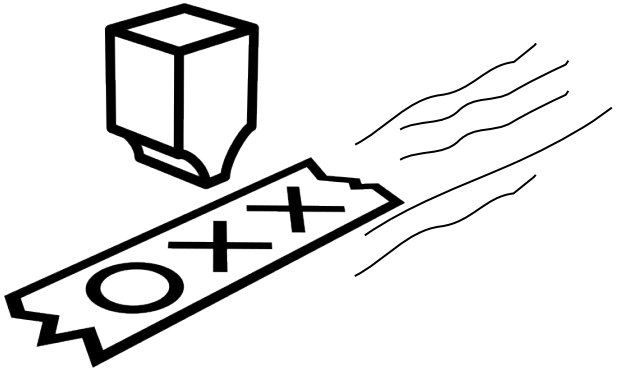
- La taille du modèle

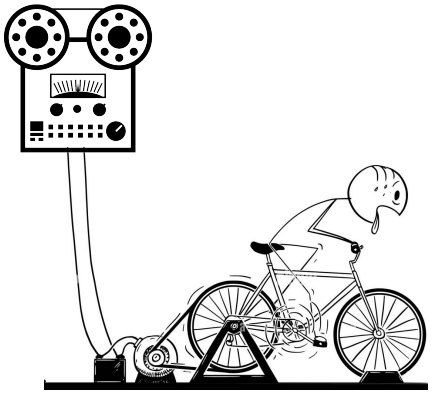


- La profondeur du modèle
- La taille des données d'entrée (Résolution des images, longueur de la séquence, …*)*

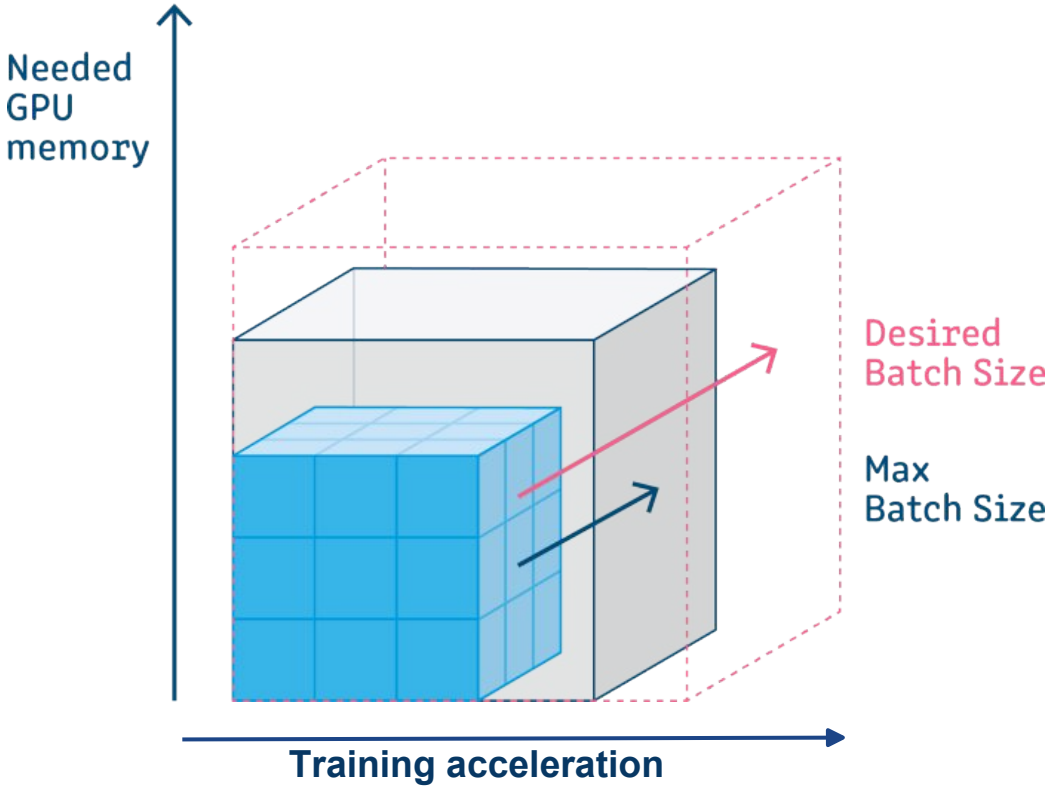
- La taille du *dataset*

- Nombre d'*epochs* nécessaire

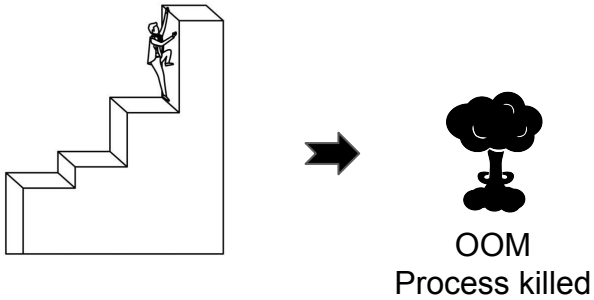




Taille de batch et Mémoire



Augmenter la taille du batch et ainsi augmenter le pas d'itération permet d'accélérer l'apprentissage.

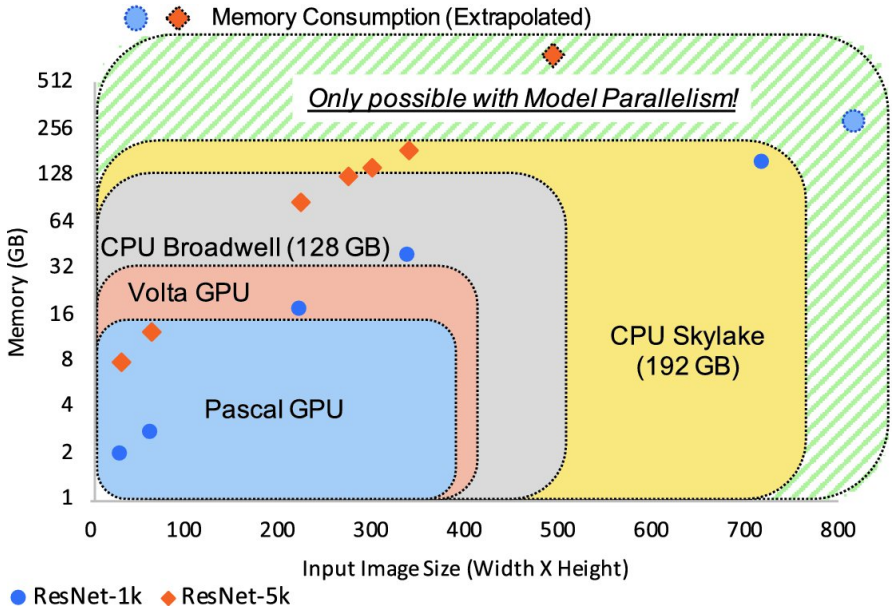


Cependant cela augmente d'autant l'empreinte mémoire risquant d'atteindre la limite du système.

Données à haute dimension

Les données à haute dimension provoquent de sérieux **problèmes d'occupation de mémoire** pendant l'apprentissage, accentués par **la profondeur du modèle**.

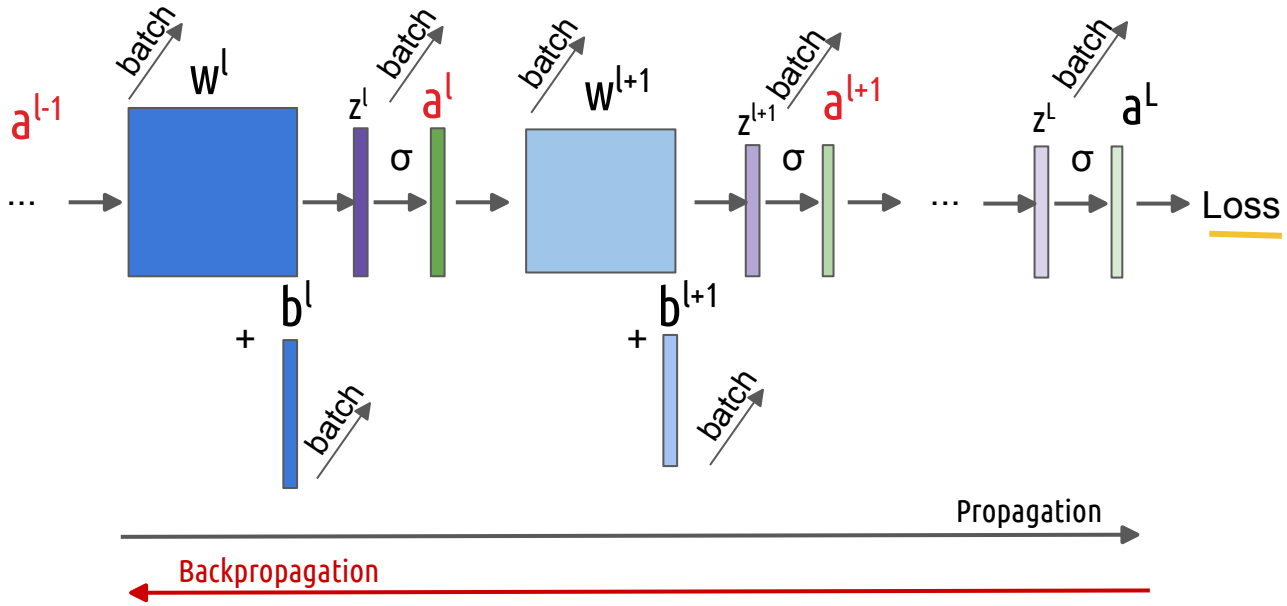
- ➔ Texte (N, 100, 500) ***~x1***
- ➔ Image 2D (N, 226, 226, 3) ***~x3***
- ➔ Image 3D (N, 226, 226, 100, 3) ***~x300***
- ➔ Video (N, 100, 226, 226, 3) ***~x300***



(GNN : Graph de petit à très très gros !!)

Source : HyPar-Flow

Forward / Backward - problème des activations



Propagation

$$a^l = \sigma(w^l a^{l-1} + b^l) = \sigma z^l$$

Backpropagation

$$\delta^l = \frac{\partial C}{\partial z^l} \qquad w^l \to w^l - \frac{\eta}{m} \cdot \frac{\partial C}{\partial w^l}$$

$$b^l \to b^l - \frac{\eta}{m} \cdot \frac{\partial C}{\partial b^l}$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

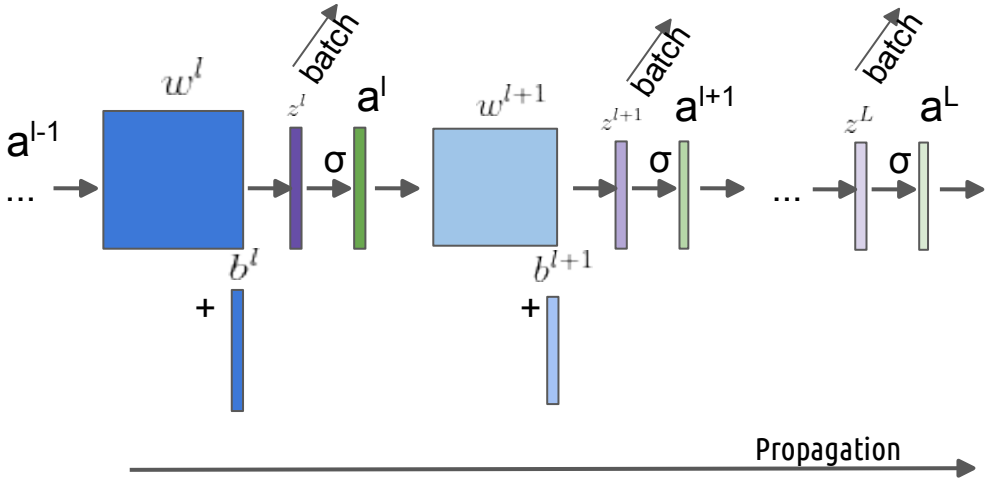
$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial w^l} = \delta^l (a^{l-1})^T$$

$$\frac{\partial C}{\partial b^l} = \delta^l$$

Note: Pour la *backpropagation*, il est nécessaire de garder en mémoire les activations intermédiaires.

Inférence et évaluation

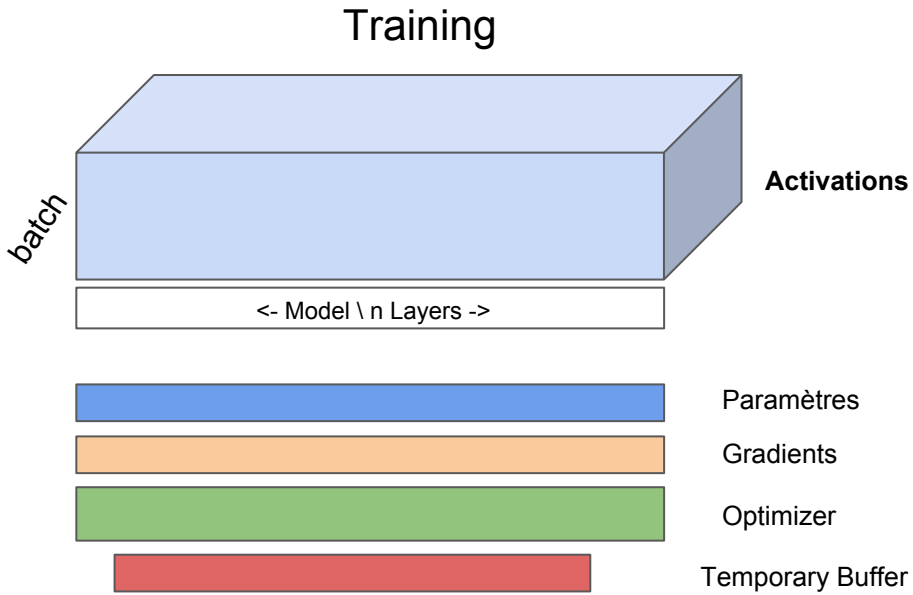


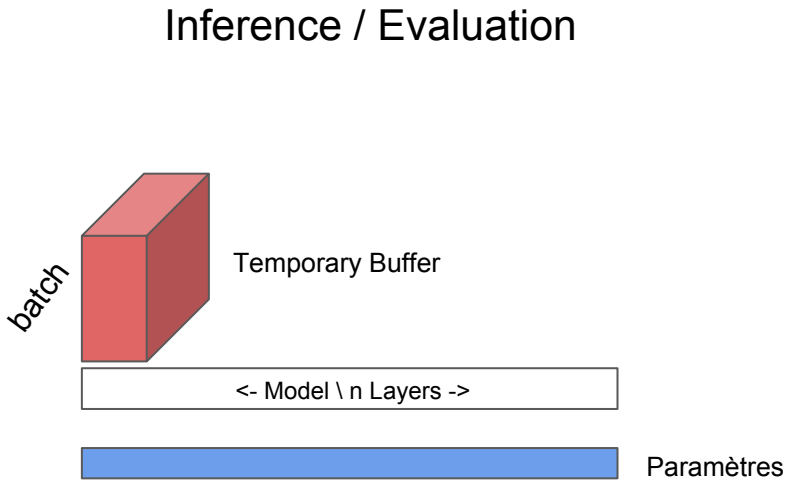
Propagation

$$a^l = \sigma(w^l a^{l-1} + b^l) = \sigma z^l$$

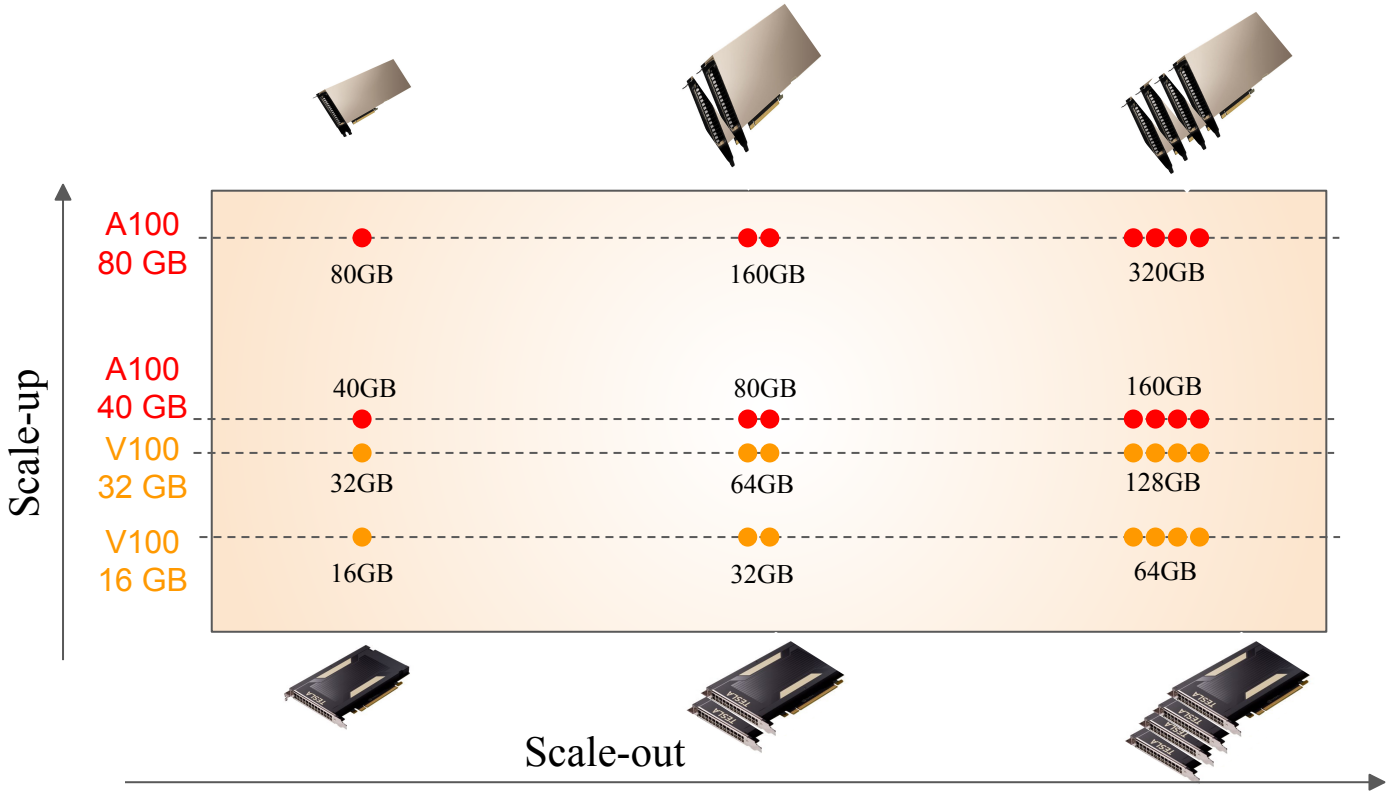
```
…
with torch.no_grad():
    val_outputs = model(val_images)
    loss = criterion(val_outputs, val_labels)
…
```

Empreinte Mémoire



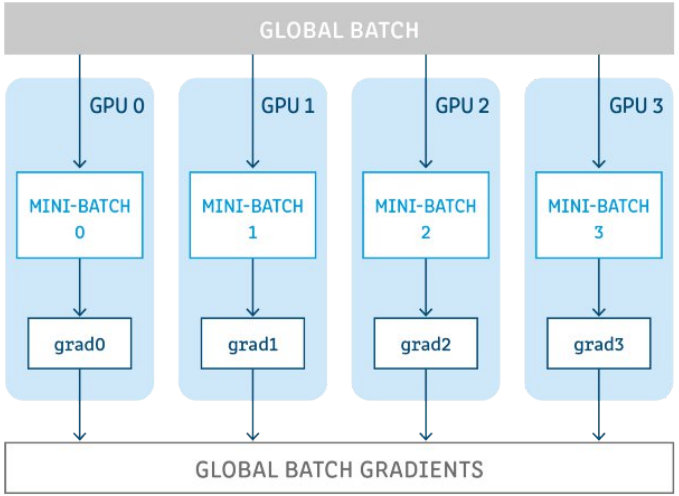


Solutions système

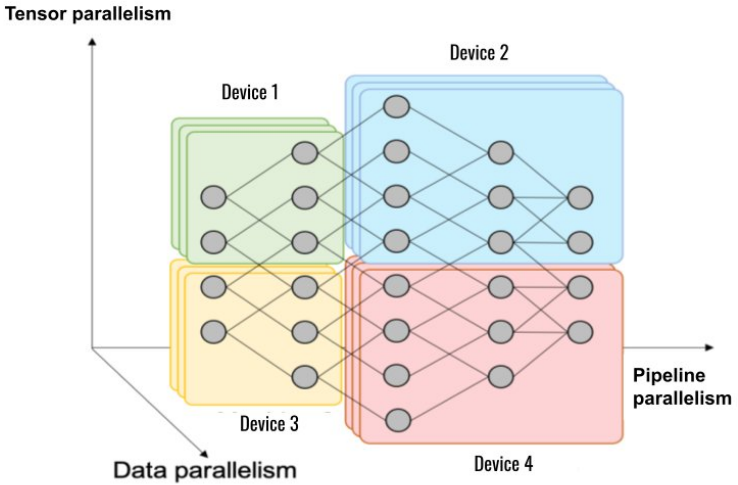


Solutions: Distribution – Scale-out

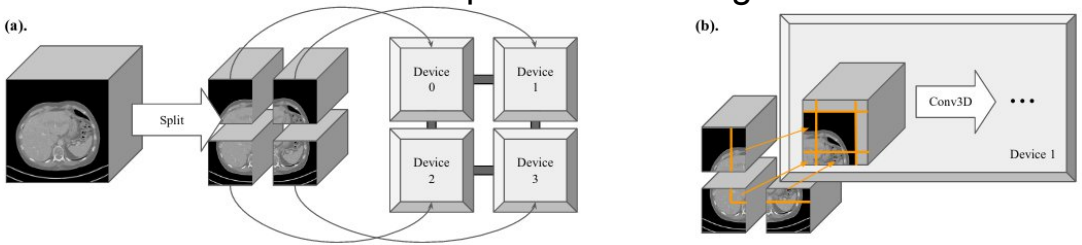
Data Parallelism



Model Parallelism

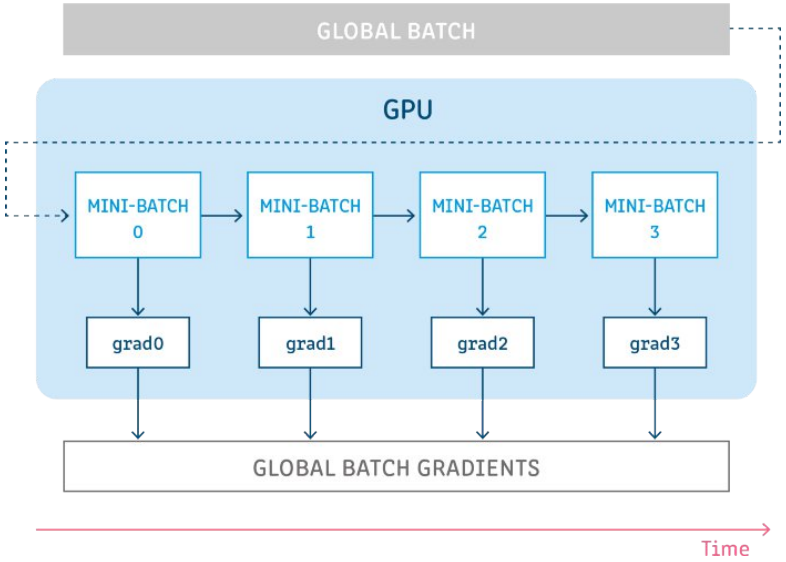


Spatial Partitioning

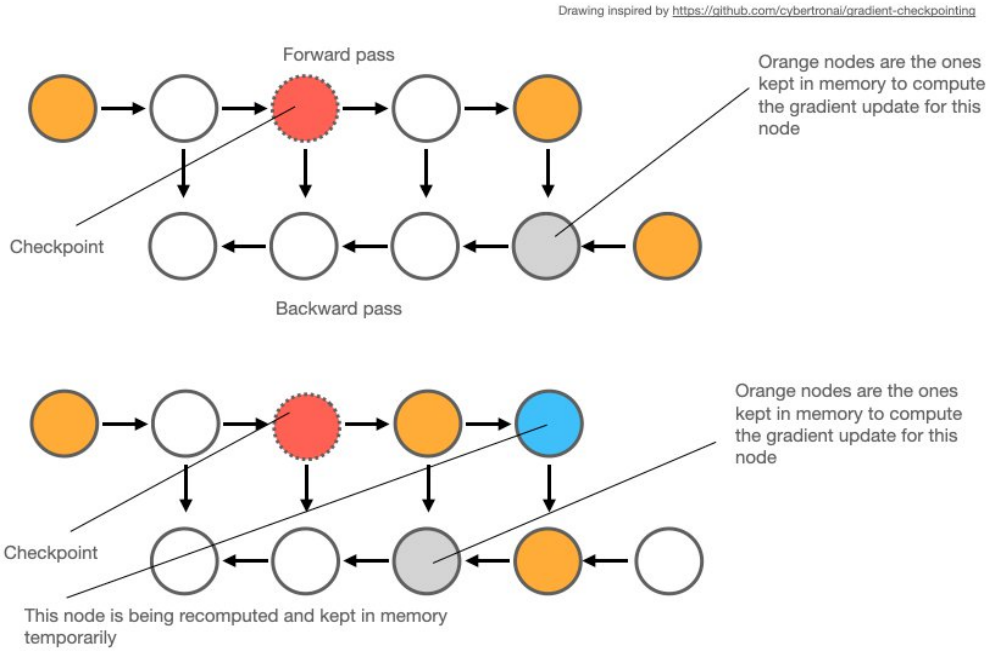


Solutions de contournement

Gradient aggregation



Gradient/activation checkpointing

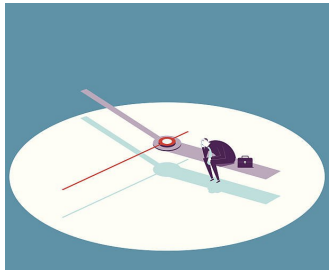


Un 3e problème à traiter ...

La consommation électrique !!

2 problèmes à traiter:

Temps d'apprentissage



Surconsommation mémoire (OOM)

Consommation énergétique

	A100 PCIe	A100 SXM2	V100 PCIe	V100 SXM2
Max Power	250W	400W	250W	300W
Idle Power	~30W	~60W	~40W	~45W
Performance	90%	100%	45%	50%

Pour un nœud : Le CPU (souvent 2 processeurs) consomme ce que consomme à peu près 1 GPU.

La consommation électrique varie selon l'utilisation partielle ou globale du GPU.

Cependant le rapport performance énergétique est en faveur d'une pleine utilisation du GPU.

Économie énergétique / Heures GPU

Économie énergétique

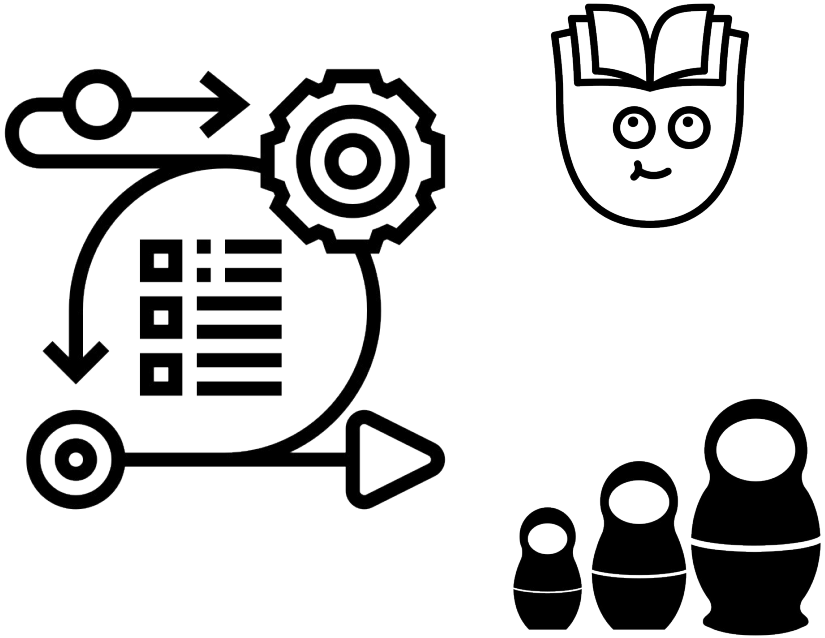
≅

Économie d'heures GPU

Optimisation du système (DLO-JZ)

- Chercher le *throughput* le plus important
- Optimiser le chargement de données pour éliminer les temps vides du GPU
- Paralléliser l'apprentissage à la bonne mesure du modèle : ni trop, ni pas assez

Économie énergétique / Heures GPU

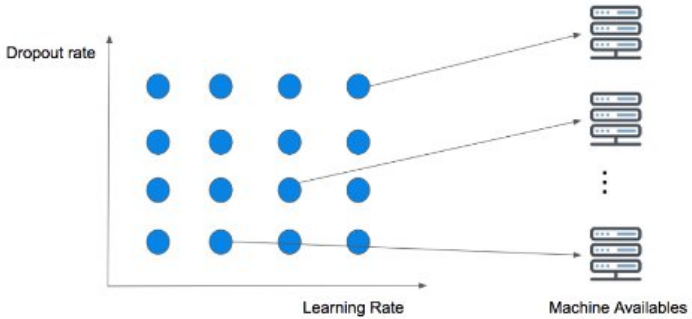


Méthodologie (économiser la recherche, ne répéter pas les apprentissages inutilement)

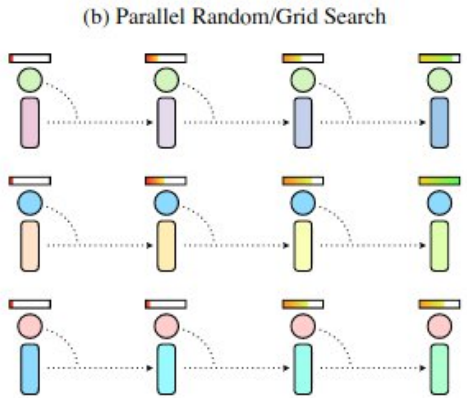
- Chercher les hypers paramètres dans les publications et reproduire l'état de l'art
- Chercher les bons hypers paramètres sur des plus petits modèles, puis appliquer à l'échelle
- Techniques d'*Hyper-Parameter Optimization* (HPO)

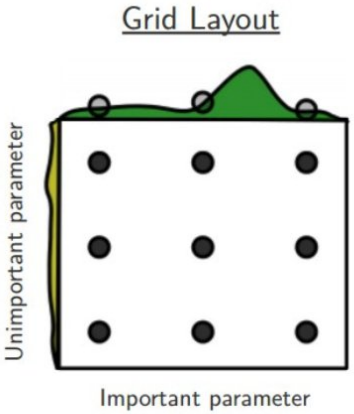
Optimisation des Hypers paramètres (HPO)

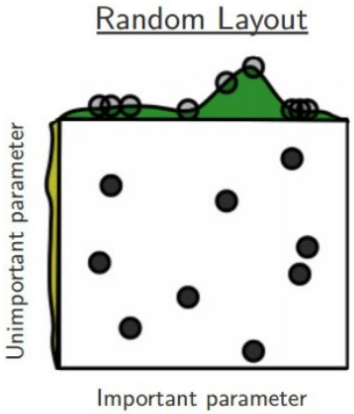
Grid/Random Search



Pour les essais massivement parallélisables sur beaucoup de machines



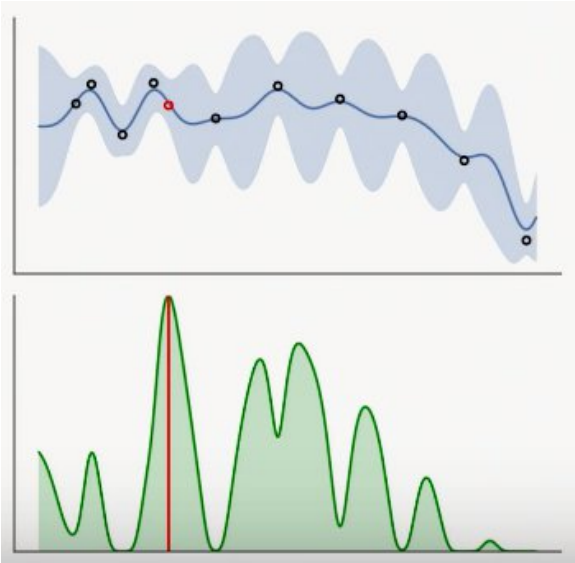




Random Search plus efficace et plus économe !!

Optimisation des Hypers paramètres (HPO)

Bayesian Optimization

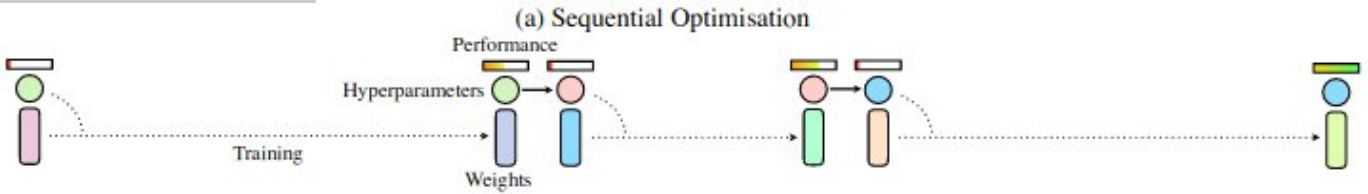


Expected metric score according to Hyper-parameters

Maximize Acquisition function
e.g. Expected Improvement

Propose un nouvel ensemble d'hypers paramètres en fonction des scores obtenus par les précédents testés.

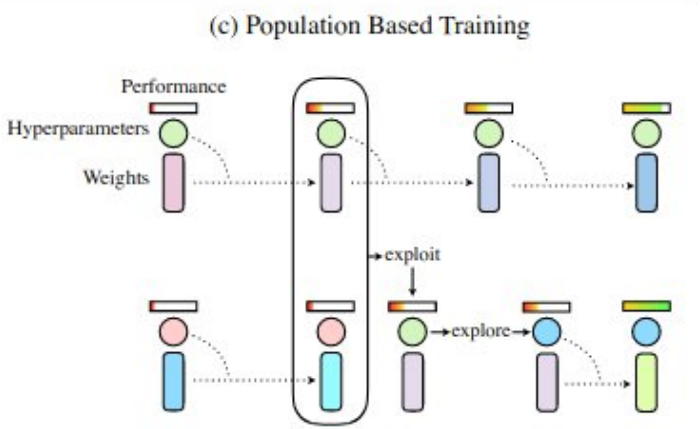
Séquentiel mais permet de trouver rapidement l'optimum global.



Optimisation des Hypers paramètres (HPO)

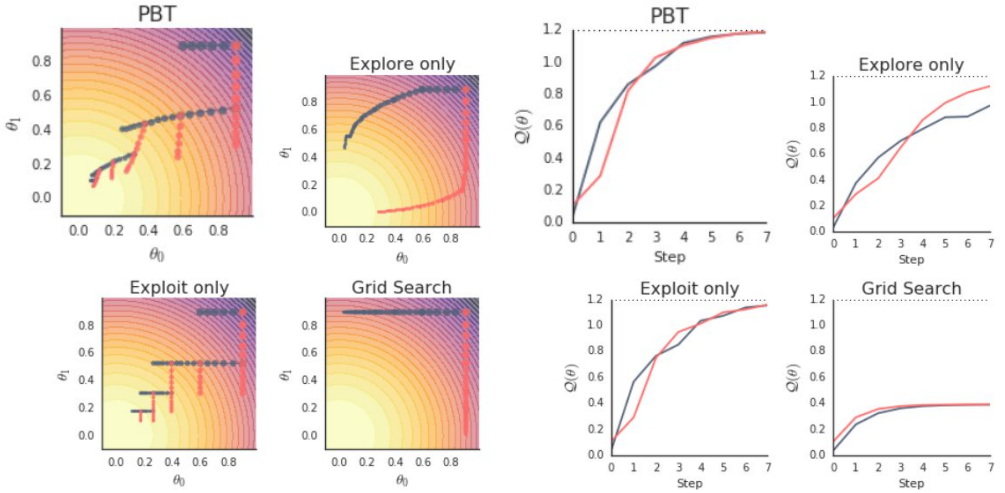
Population Based Training

Recherche et optimisations des hypers paramètres durant l'apprentissage.



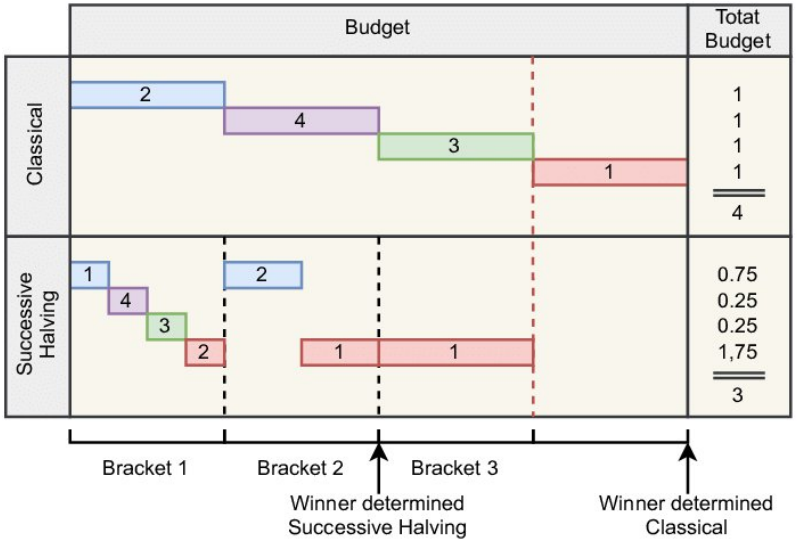
Pour les gros modèles avec des essais longs et faiblement parallélisables sur quelques machines.

- Exploit = copie des poids du meilleur modèle
- Explore = *Bayesian Optimization*



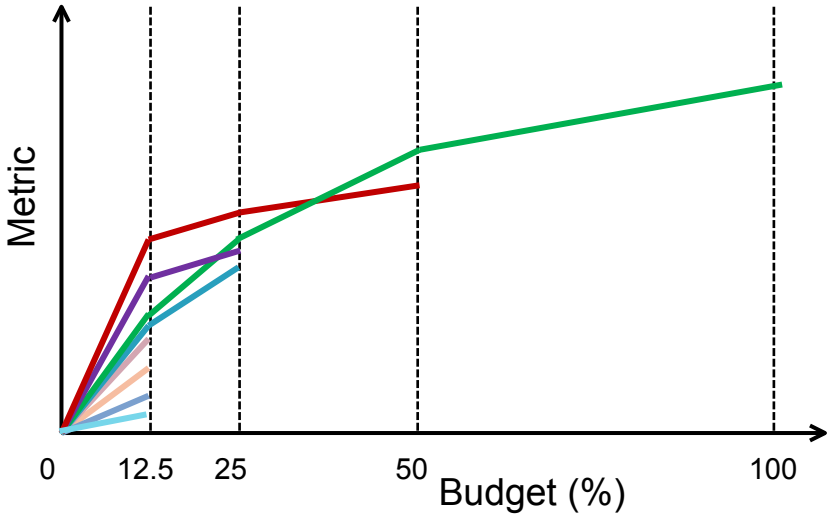
Optimisation des Hypers paramètres (HPO)

Successive Halving Algorithm



- Pour des essais séquentielles
- ou **pour des essais massivement parallélisables : ASHA** (Asynchronous Successive Halving Algorithm)

Pour les petits ou moyens modèles avec des essais assez rapides.



GPU computing

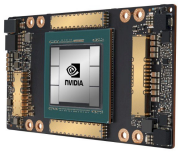
V100, A100 ◂

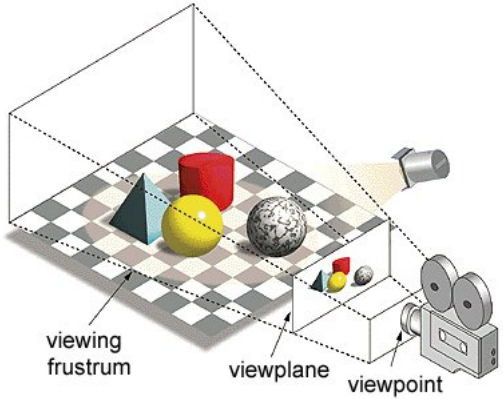
CUDA ◂

CuDNN ◂

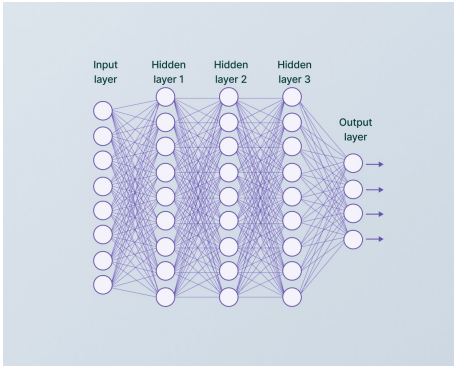
AMP ◂

GPU computing

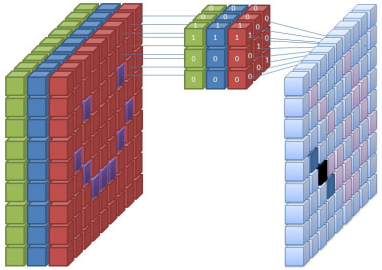




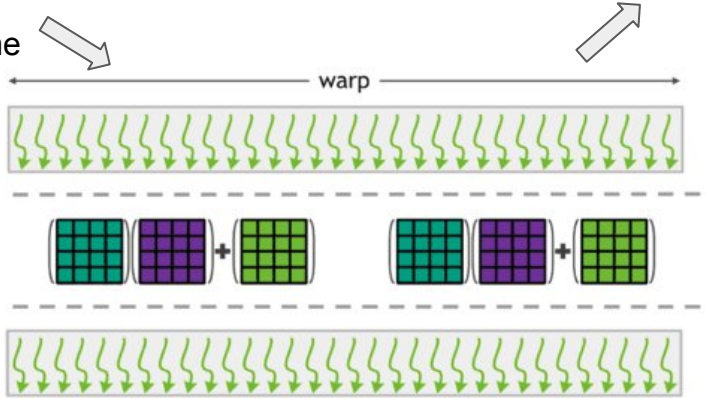
GPU Rendering & Game Graphics Pipeline



NN

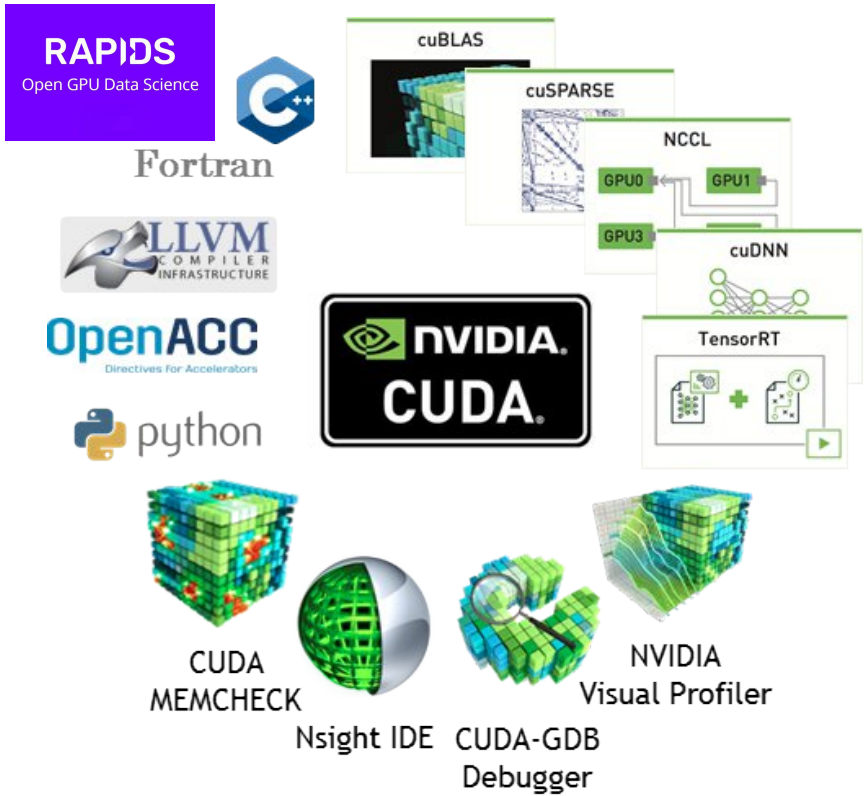


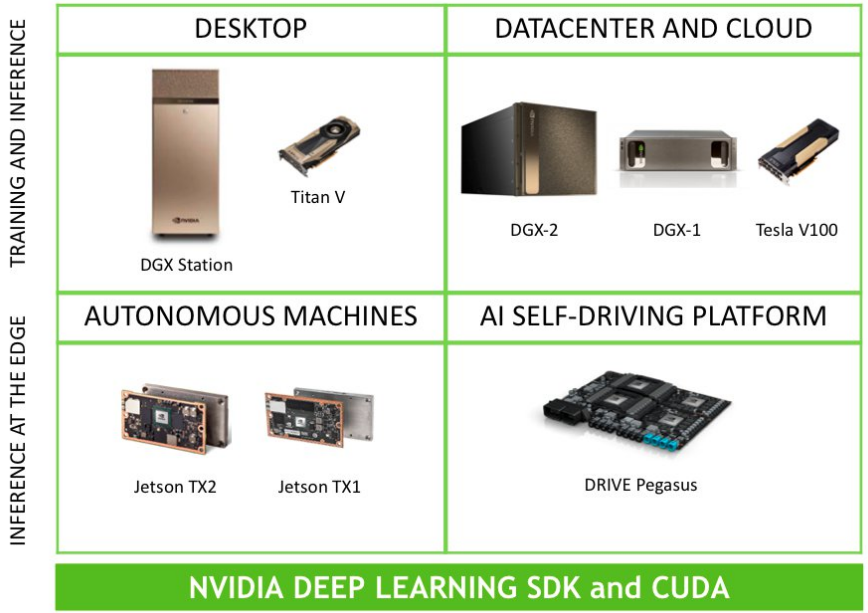
CNN



Matrix Multiply-accumulate operations

Galaxie NVIDIA



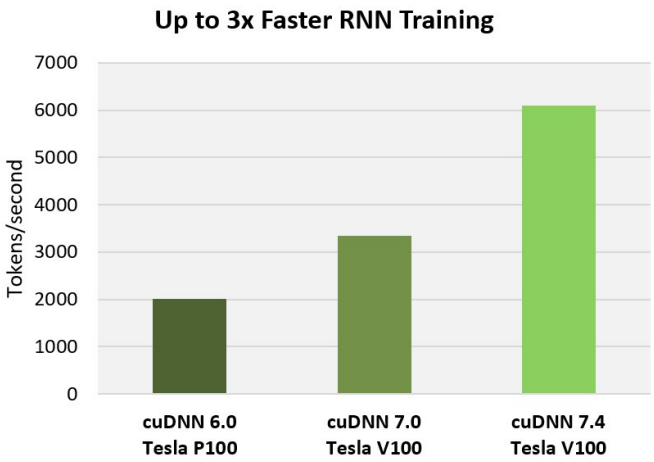


Source : NVidia

CuDNN

PyTorch

NVIDIA CUDA cuDNN



TensorFlow performance (tokens/sec), Tesla P100 + cuDNN 6 (FP32) on 17.12 NGC container, Tesla V100 + cuDNN 7.0 (Mixed) on 18.02 NGC container, Telsa V100 + cuDNN 7.4 (Mixed) on 18.10 NGC container, OpenSeq2Seq (GNMT), Batch Size: 64

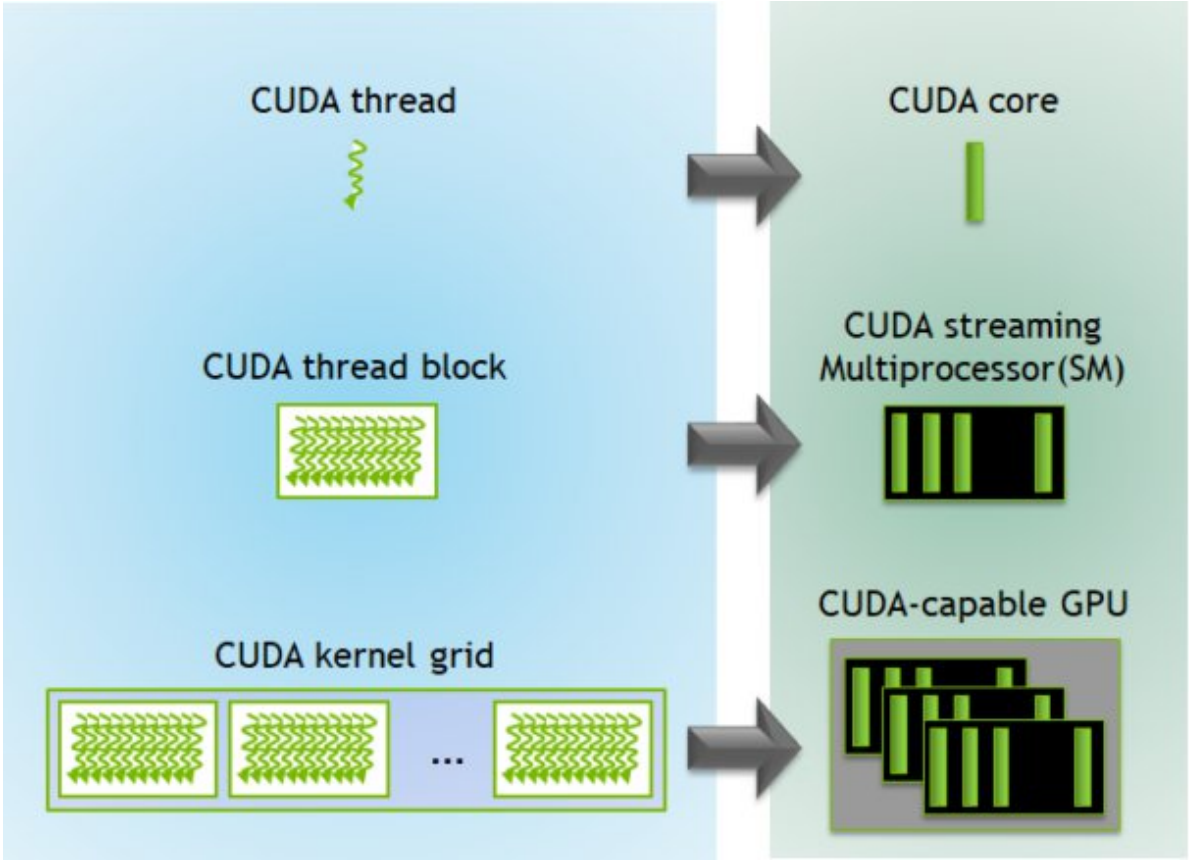
Caffe · Caffe2 · Chainer · Cognitive Toolkit · MATLAB · mxnet · PaddlePaddle · PYTORCH · TensorFlow · ...

NVIDIA DEEP LEARNING SDK and CUDA

L'ingénierie CUDA pour le deep learning sur GPU est gérée par cuDNN.
Merci cuDNN !!

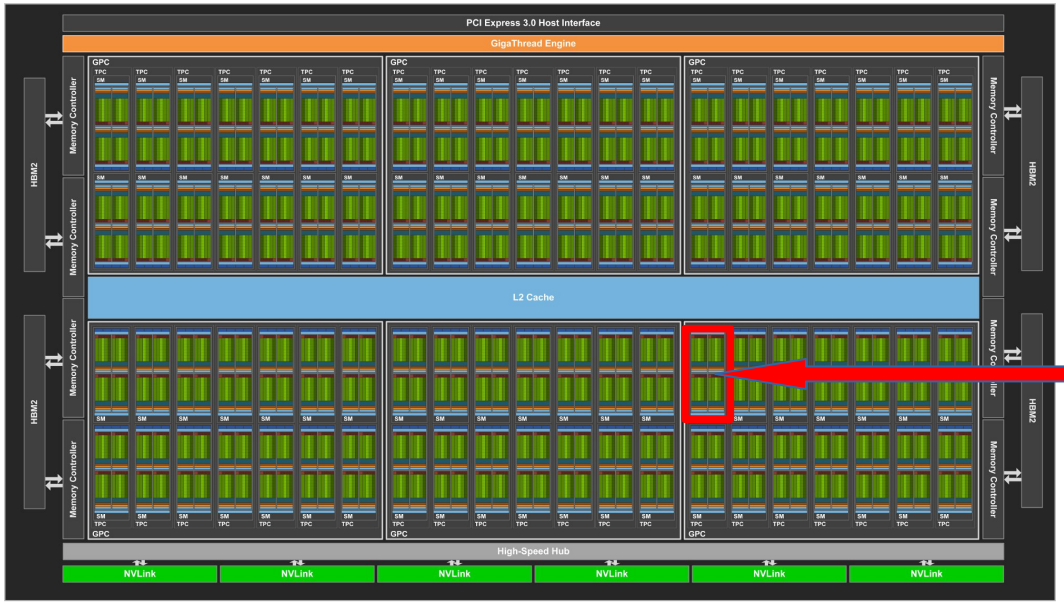
Recommandation: pour optimiser l'utilisation des *Tensor Cores* et des *Cuda Cores* : Utiliser des tenseurs aux dimensions (*batch size*, *sample size*, *channel*, etc …) **multiples de 8 !!**

GPU computing : CUDA



Source : NVidia

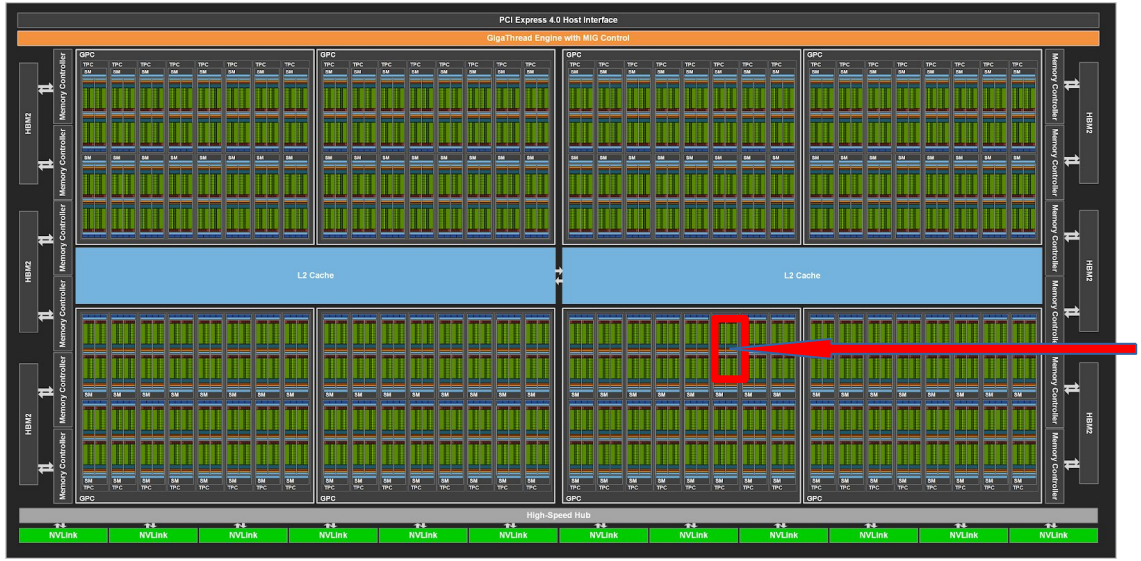
Architecture V100



- 6 GPC
- **84** Streaming Multiprocessors (SMs)
- 5376 CUDA Cores
- 672 Tensor Cores per full GPU

Source : NVidia

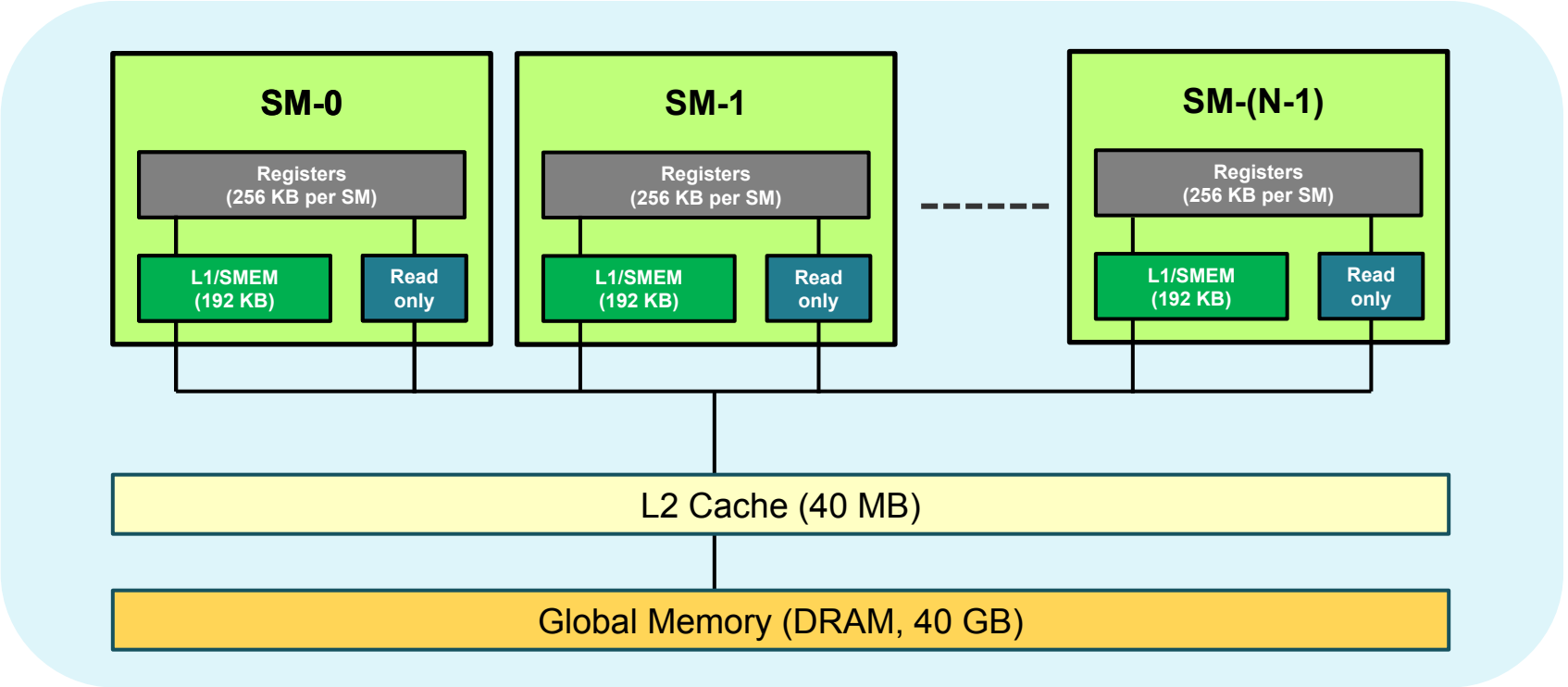
Architecture A100



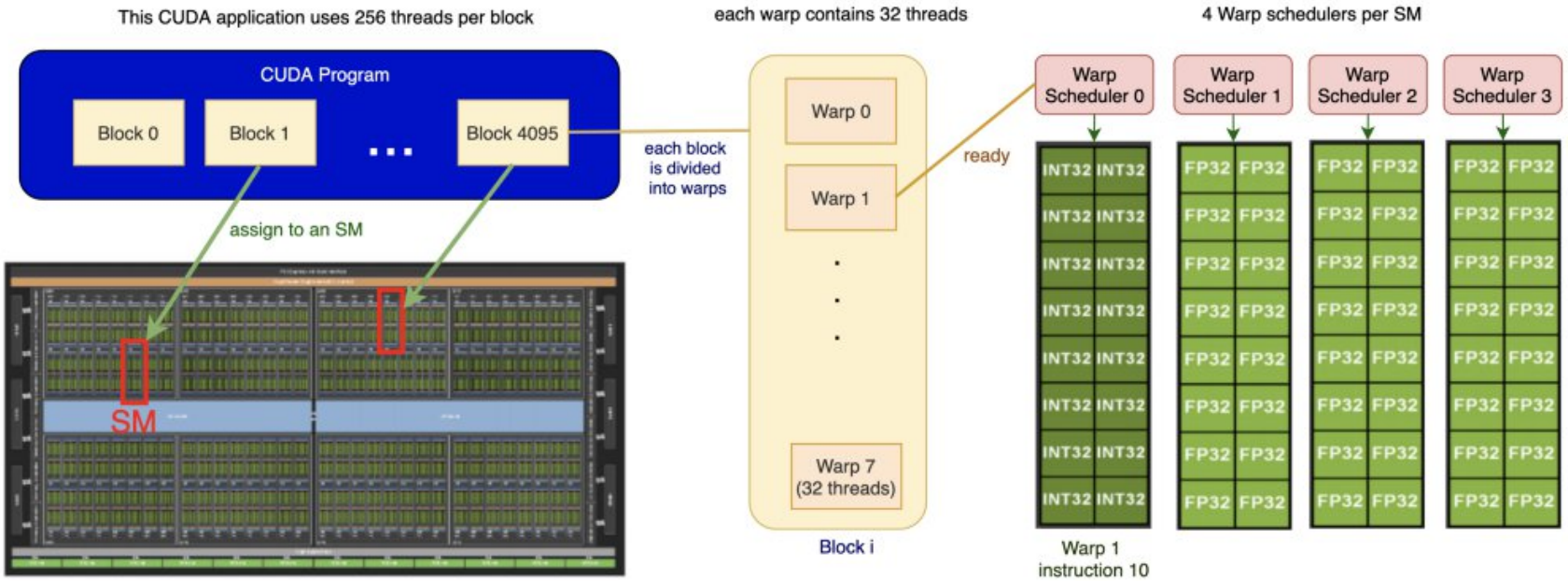
- 8 GPC
- **128** Streaming Multiprocessors (SMs)
- 8192 CUDA Cores
- 512 Tensor Cores per full GPU

Source : NVidia

Gestion de la mémoire optimisée



Ingénierie CUDA



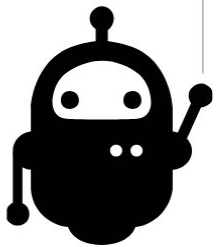
Optimisation :

- du remplissage d'un *block*
- de l'étalement sur le GPU

Optimisation avancée :

- Fusion des kernels pour économiser les temps d'initialisation

TP1 : Accélération GPU



- Envoyer le calcul sur le GPU
- Test Mémoire

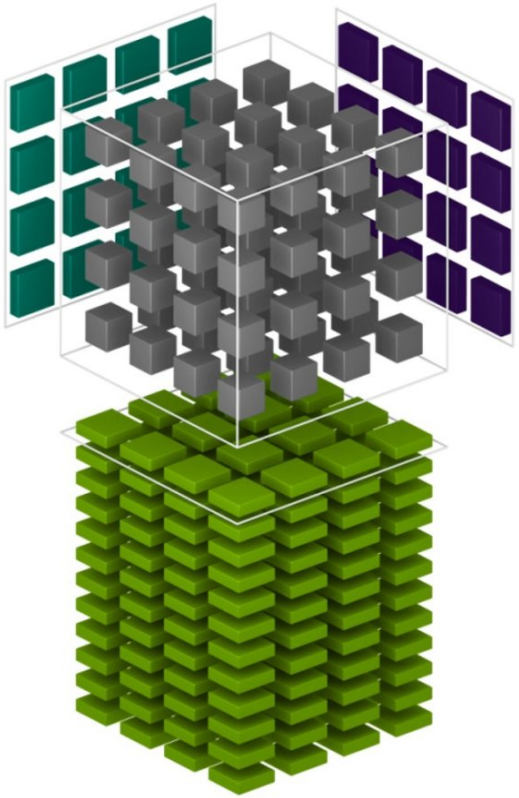
Mixed Precision

Tensor cores ◂

Précisions ◂

AMP ◂

Tensor Cores



Les *CUDA Core* sont spécialisés pour le calcul vectoriel.

Les *Tensor Core* sont spécialisés pour le **calcul matriciel.**

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

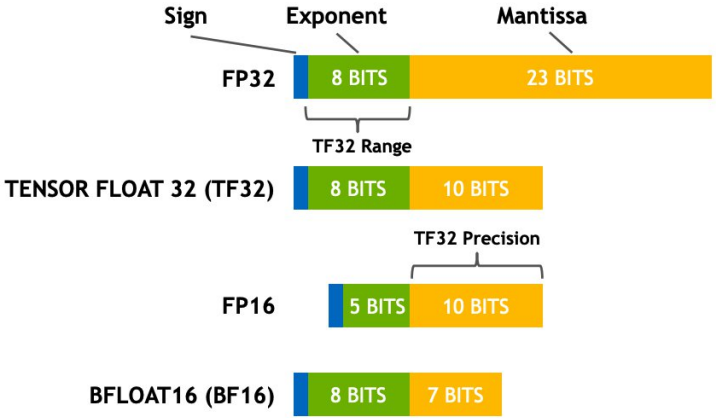
FP16 or FP32 — FP16 — FP16 — FP16 or FP32

Chaque *Tensor Core* est capable de traiter 64 opérations en 1 temps d'horloge.

Source : NVidia

Précisions & Tensor Cores

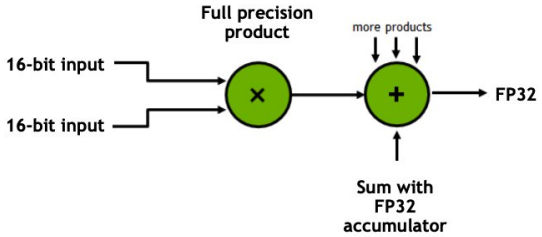
	NVIDIA A100	NVIDIA Volta
Supported Tensor Core Precisions	FP64, TF32, bfloat16, FP16, INT8, INT4, INT1	FP16
Supported CUDA® Core Precisions	FP64, FP32, FP16, bfloat16, INT8	FP64, FP32, FP16, INT8

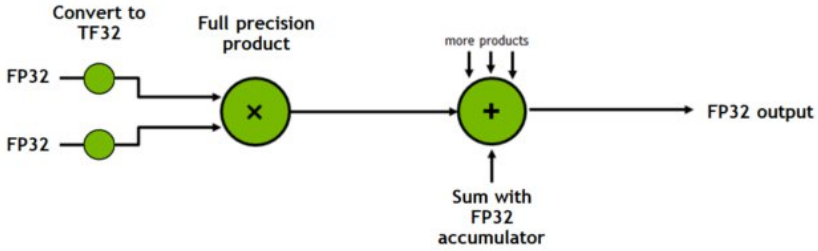


Source : NVidia

Précisions & Tensor Cores

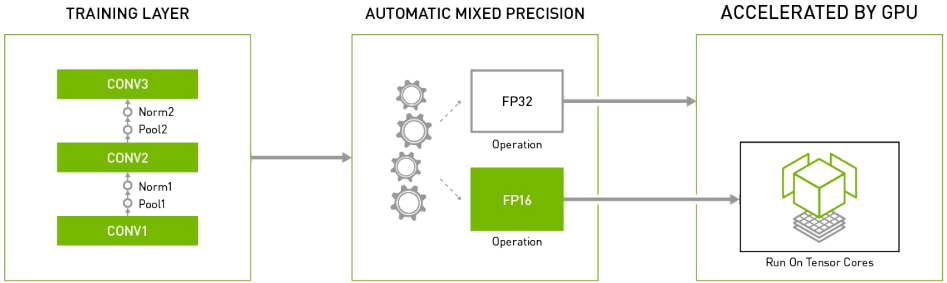
	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x	-	-
	IEEE FP64		19.5	1x	-	-



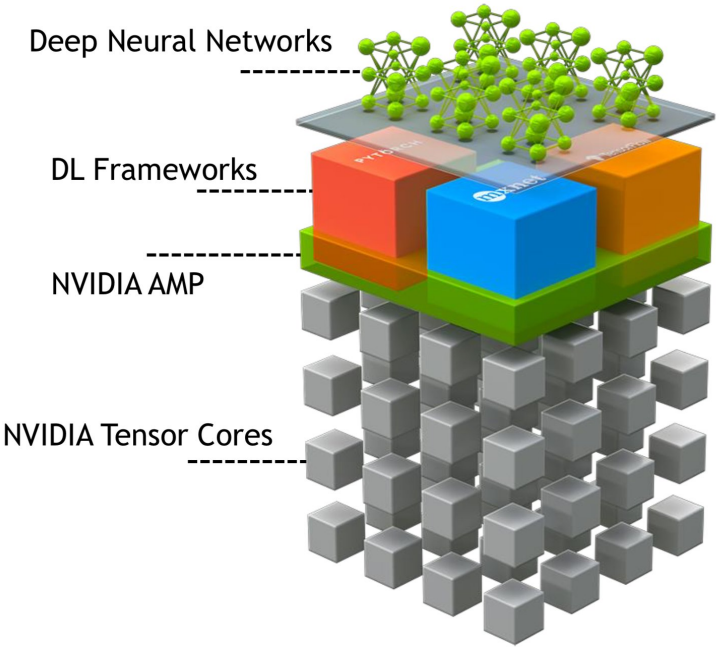


Automatic Mixed Precision

- Automatic Mixed Precision :
  - Nécessaire pour les V100 pour utiliser les *Tensor Core*
  - Les A100 utilisent les *Tensor Core* avec ou sans MP

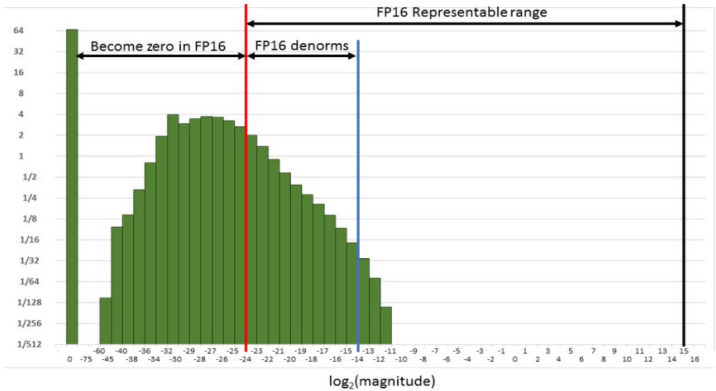


- Intérêts :
  - **Perte de précision non significative** pour l'apprentissage du modèle (gradient, loss, accuracy)
  - **Réduit** l'empreinte mémoire
  - **Accélère** les calculs
- 2 étapes à coder:
  - transformation des couches éligibles en FP16
  - Utilise un *scaling* pour le calcul des gradients

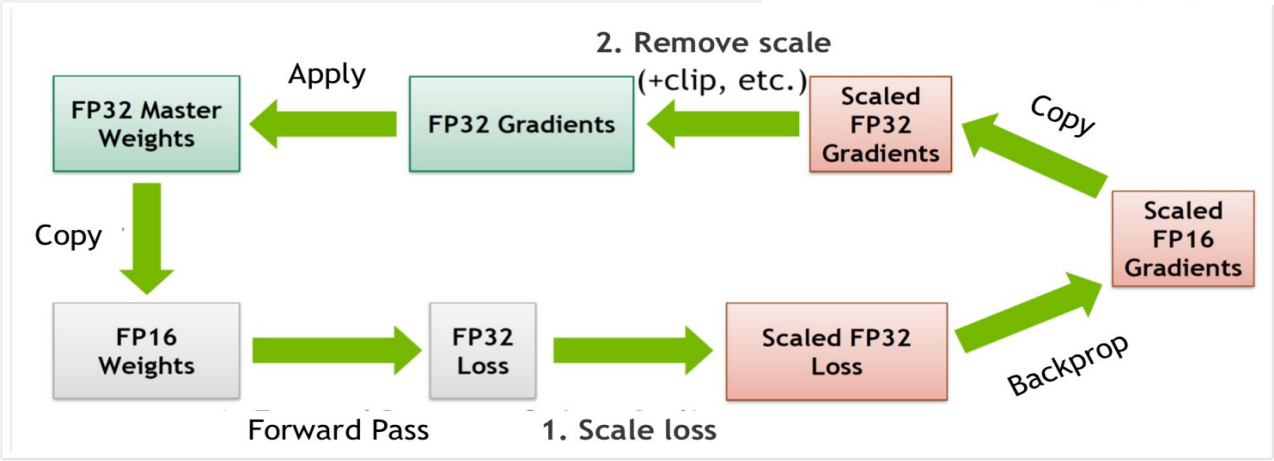


AMP Scaler

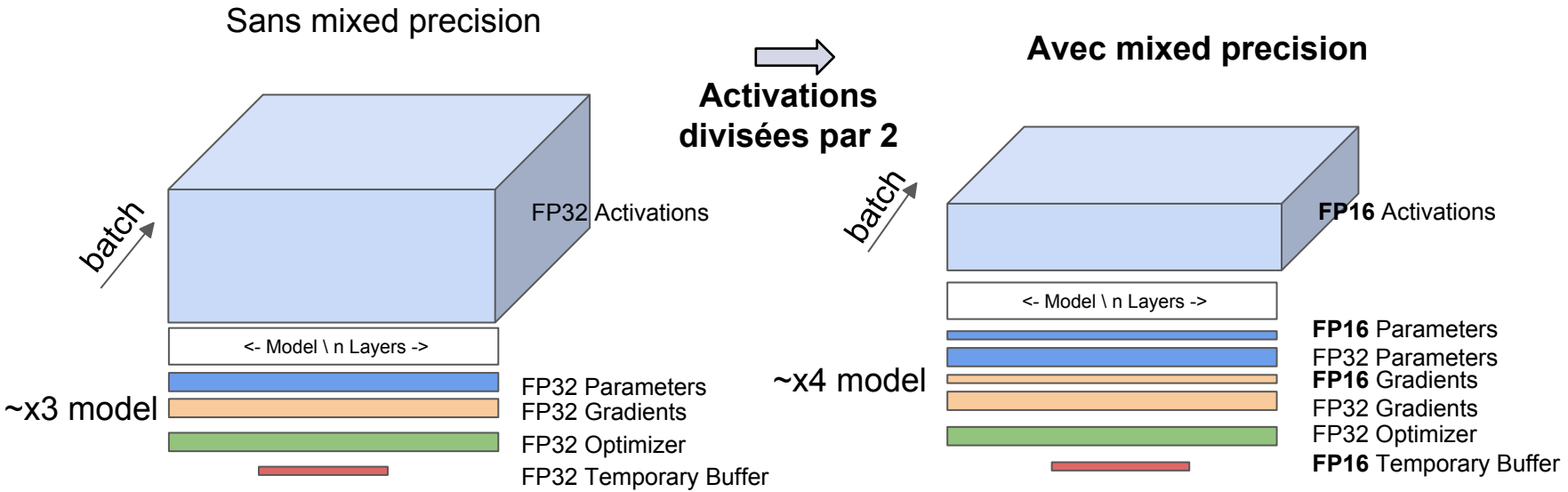
Distribution des gradients



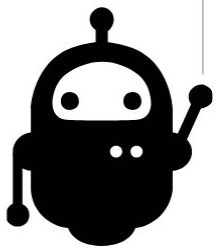
En FP16 les valeurs inférieures à 2^{-24} ($5.96e^{-8}$) sont considérées comme des 0.



Empreinte mémoire avec la Mixed Precision



TP2 : Automatic Mixed Precision



- Activer l’Automatic Mixed Precision
- Test Mémoire