Deep Learning Optimisé - Jean Zay

Les parallélismes des gros modèles

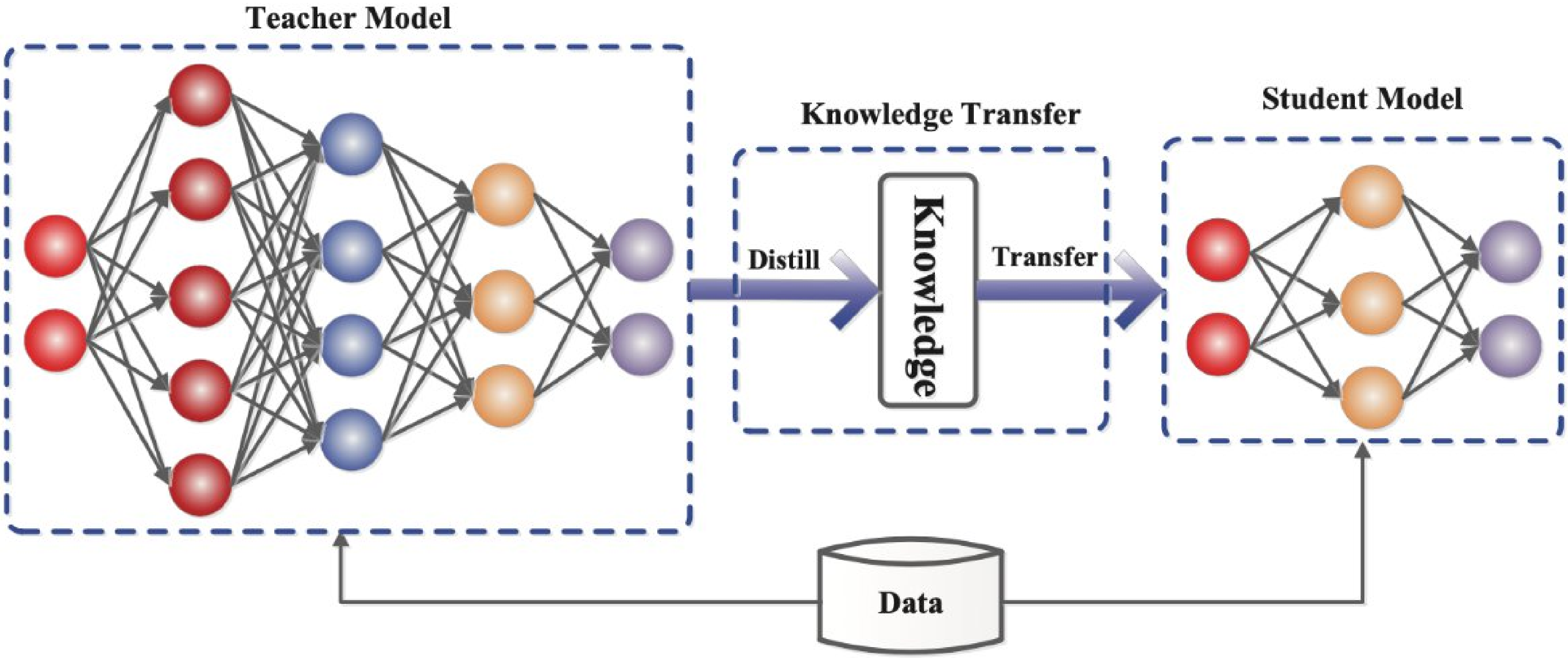
Institut du
Développement et des
Ressources en
Informatique
Scientifique

cc BY NC SA

Inférence et fine-tuning

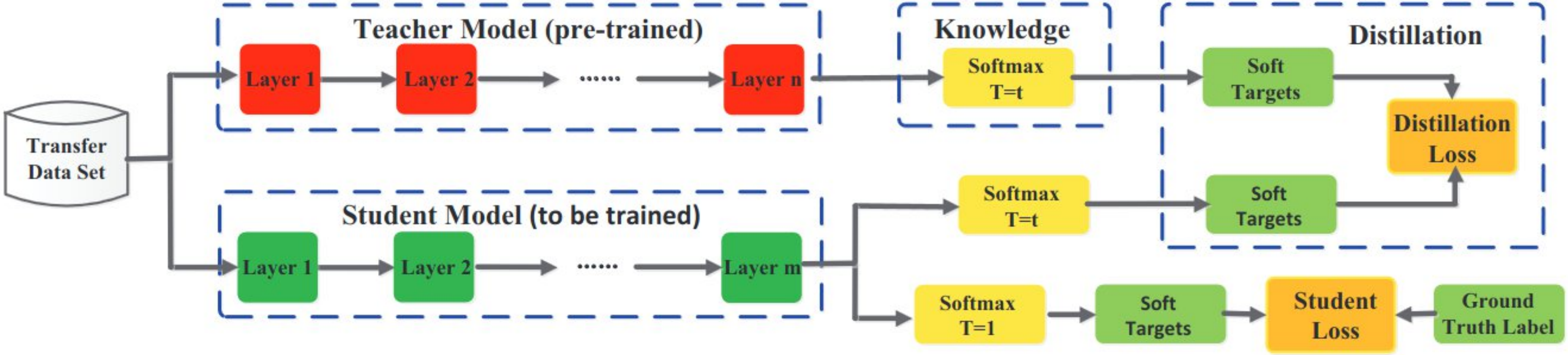
- Distillation ◂
- Quantification ◂
- Pruning ◂

Distillation



[18]

Distillation



$$\mathcal{L}_{\text{tot}} = \underbrace{\mathcal{L}_{\text{distil}}}_{\text{Cross-entropy, Divergence KL, Wasserstein, ...}} \left(y_{\text{teacher}}, y_{\text{student}}\right) + \lambda \mathcal{L}_{\text{CE}} \left(y_{\text{target}}, y_{\text{student}}\right)$$

DistilBERT a 40% moins de paramètres, mais ne perd que 3% de performance.

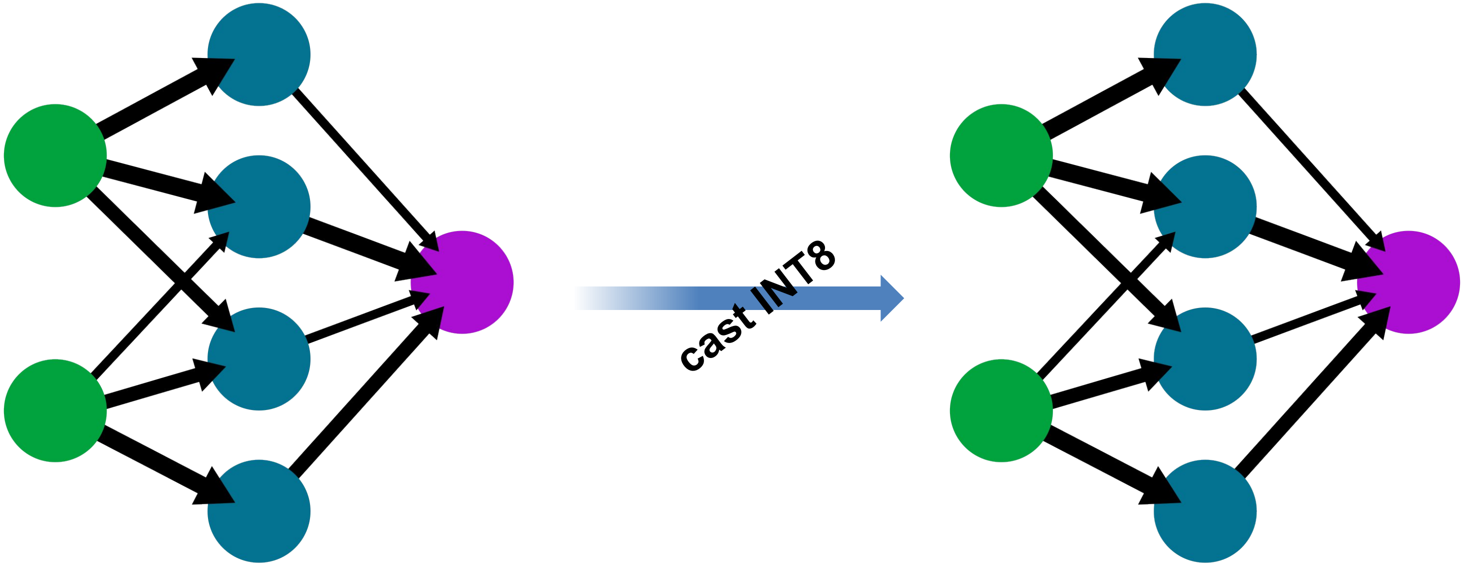
[18, 27]

Quantification

	A100 80 Go PCIe	A100 80 Go SXM
FP64	9,7 TFlops	
FP64 Tensor Core	19,5 TFlops	
FP32	19,5 TFlops	
Tensor Float 32 (TF32)	156 TFlops | 312 TFlops*	
BFLOAT16 Tensor Core	312 TFlops | 624 TFlops*	
FP16 Tensor Core	312 TFlops | 624 TFlops*	
INT8 Tensor Core	624 TOPs | 1248 TOPs*	

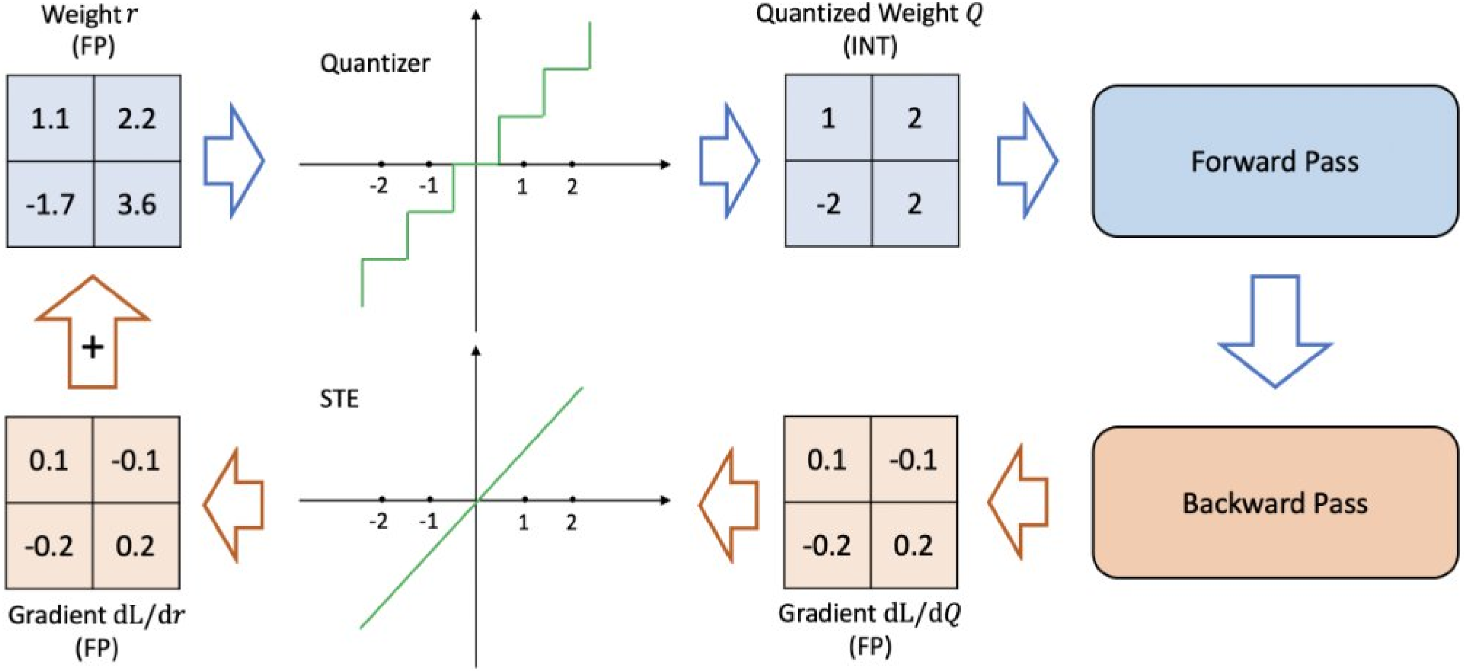
x2

Quantification



Il est possible de rencontrer une perte en performance

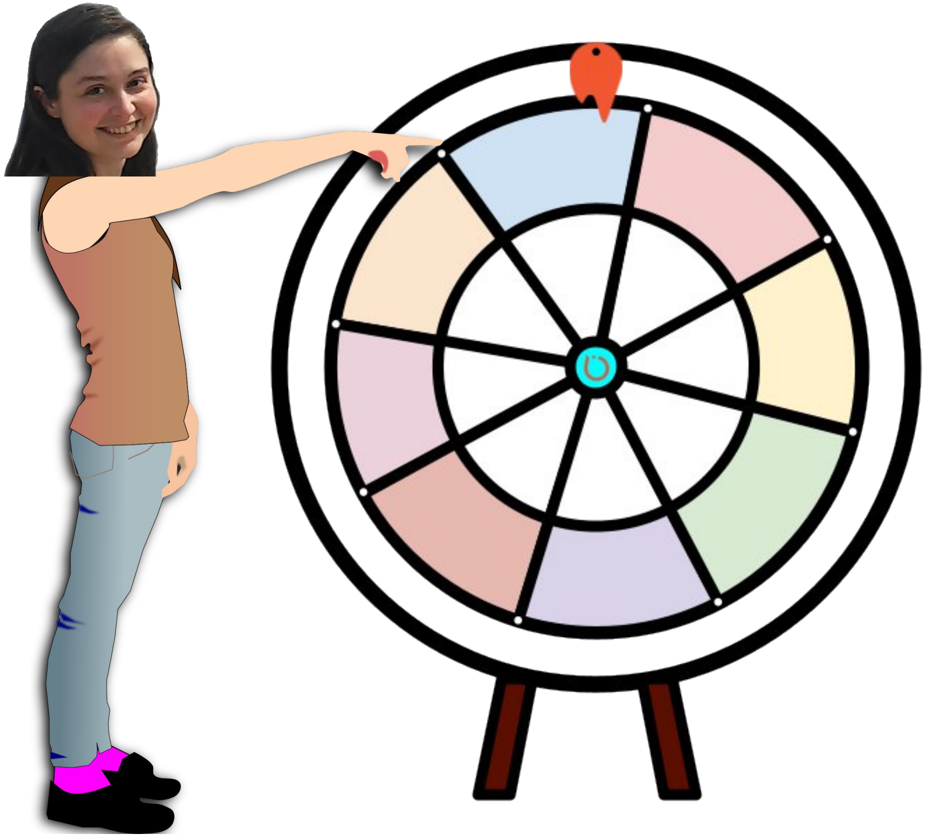
Quantification



Exemple d’entraînement post-quantisation

[19, 23]

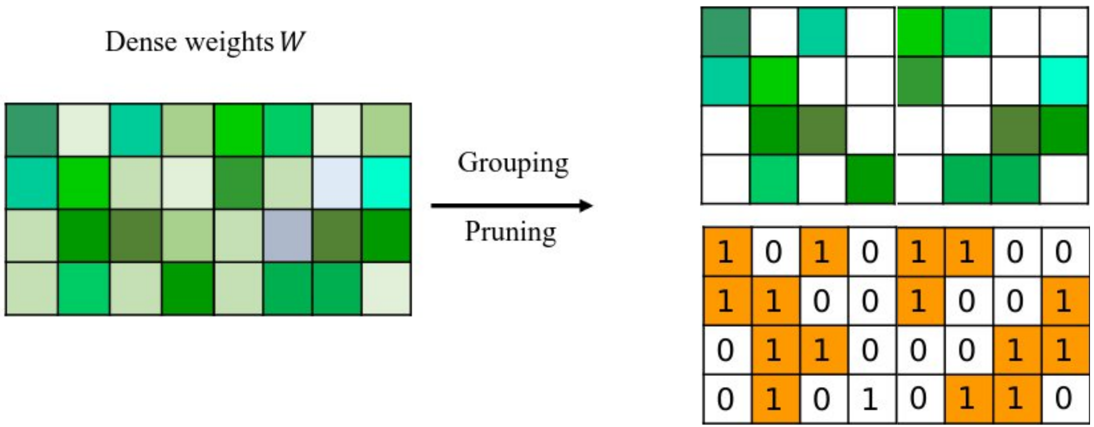
Pruning



The Lottery Ticket Hypothesis.
A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

[24, 25, 26]

Pruning

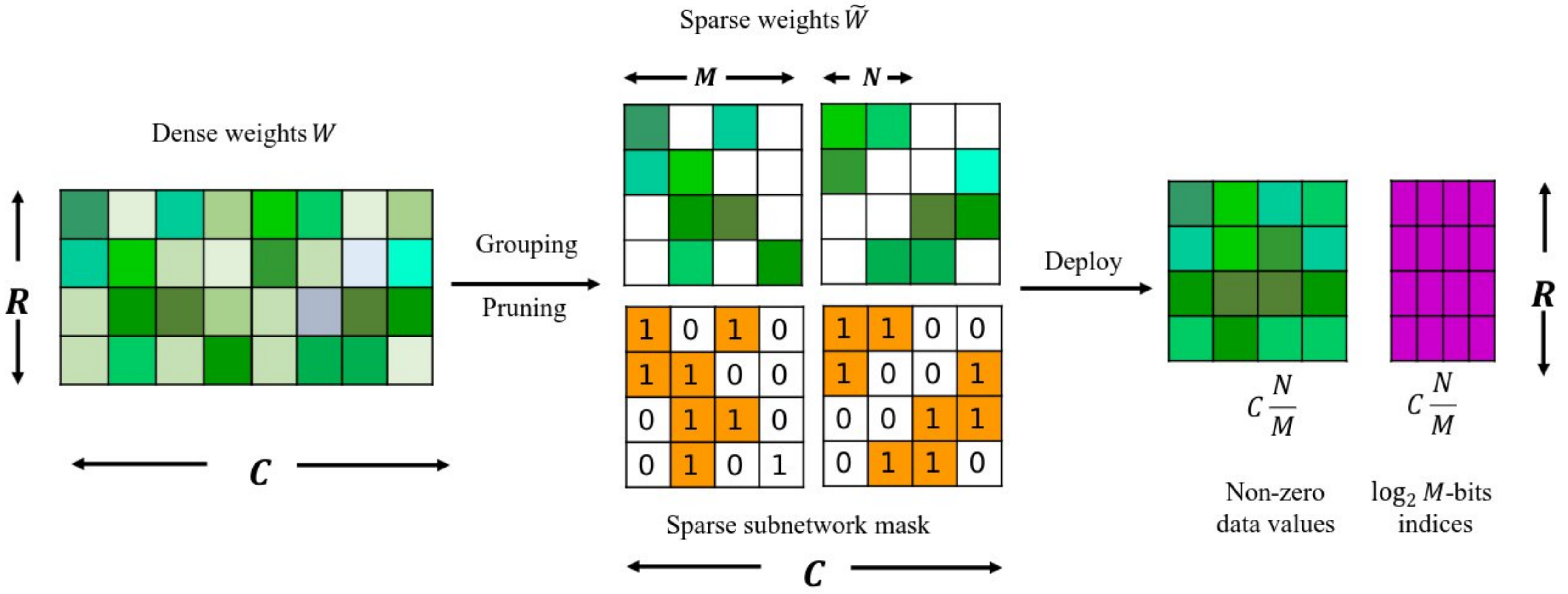


1	0	1	0	1	1	0	0
1	1	0	0	1	0	0	1
0	1	1	0	0	0	1	1
0	1	0	1	0	1	1	0

Les poids les plus petits sont mis à 0. Mais combien ?
Quel impact sur le temps de calcul ?

[20]

Pruning



1	0	1	0
1	1	0	0
0	1	1	0
0	1	0	1

1	1	0	0
1	0	0	1
0	0	1	1
0	1	1	0

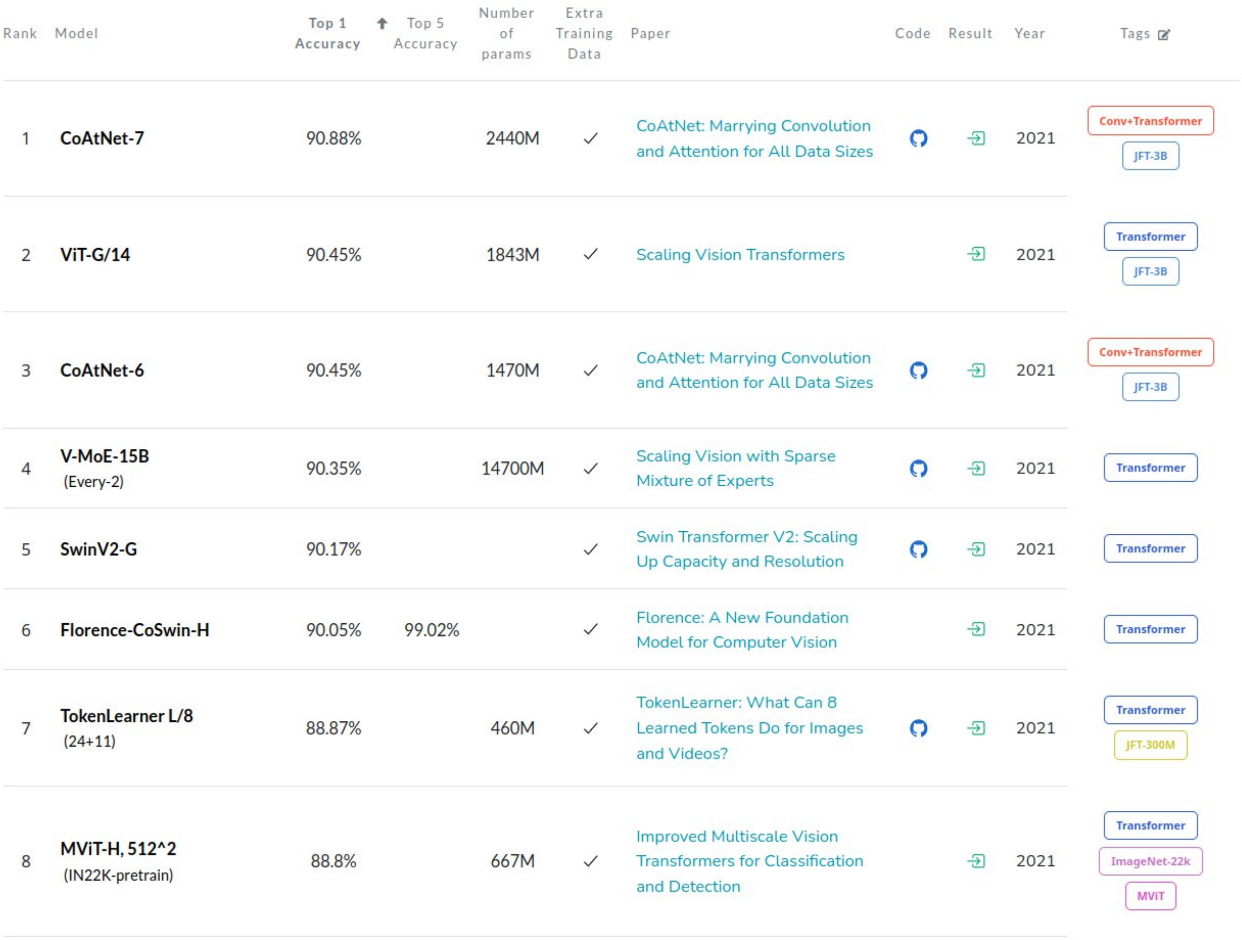
Les Tensor Cores des NVIDIA A100 supportent une dispersion 2:4.

[20, 21]

Vision Transformers

- Transformers ◀
- Vision Transformers ◀
- CoAtNet ◀

Vision Transformers >> Resnet-50: 25M



Rank	Model	Top 1 Accuracy	Top 5 Accuracy	Number of params	Extra Training Data	Paper	Code	Result	Year	Tags
1	**CoAtNet-7**	90.88%		2440M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes	●	⇨	2021	Conv+Transformer JFT-3B
2	**ViT-G/14**	90.45%		1843M	✓	Scaling Vision Transformers		⇨	2021	Transformer JFT-3B
3	**CoAtNet-6**	90.45%		1470M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes	●	⇨	2021	Conv+Transformer JFT-3B
4	**V-MoE-15B** (Every-2)	90.35%		14700M	✓	Scaling Vision with Sparse Mixture of Experts	●	⇨	2021	Transformer
5	**SwinV2-G**	90.17%			✓	Swin Transformer V2: Scaling Up Capacity and Resolution	●	⇨	2021	Transformer
6	**Florence-CoSwin-H**	90.05%	99.02%		✓	Florence: A New Foundation Model for Computer Vision		⇨	2021	Transformer
7	**TokenLearner L/8** (24+11)	88.87%		460M	✓	TokenLearner: What Can 8 Learned Tokens Do for Images and Videos?	●	⇨	2021	Transformer JFT-300M
8	**MViT-H, 512^2** (IN22K-pretrain)	88.8%		667M	✓	Improved Multiscale Vision Transformers for Classification and Detection		⇨	2021	Transformer ImageNet-22k MViT

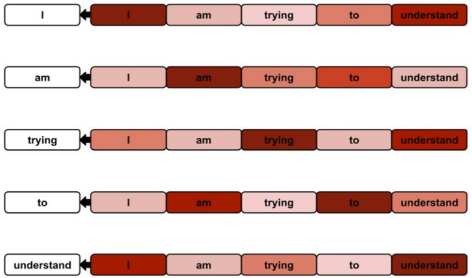
Paperswithcode

Le premier Transformer

Attention Is All You Need

Simple couche de réseaux de neurones

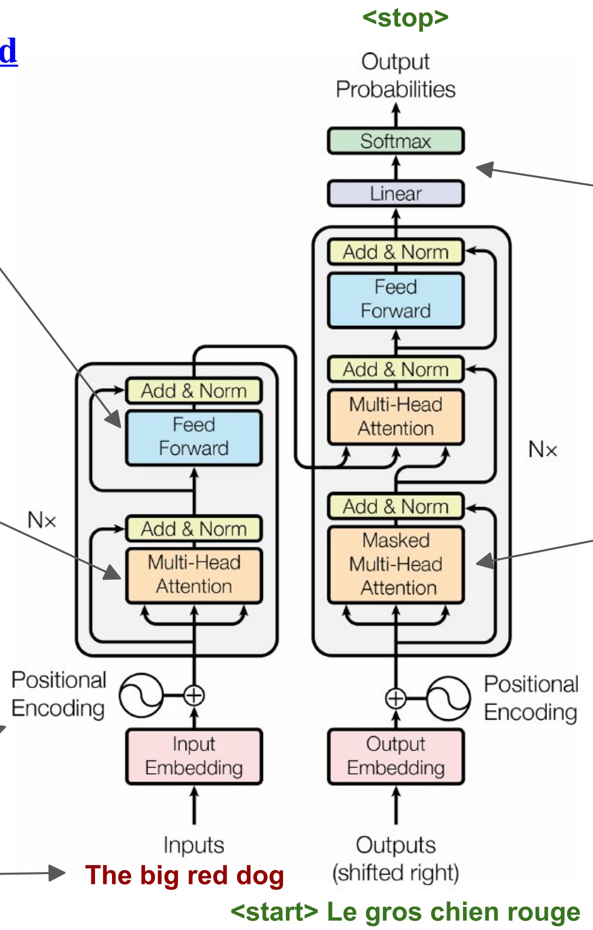
Mécanisme d'attention (encoder)



I	I	am	trying	to	understand
am	I	am	trying	to	understand
trying	I	am	trying	to	understand
to	I	am	trying	to	understand
understand	I	am	trying	to	understand

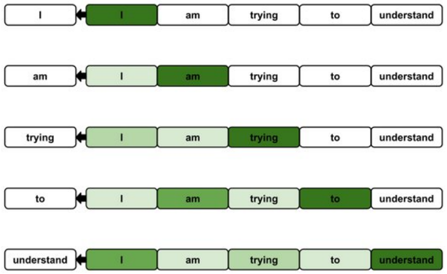
Ajout des vecteurs positions à la séquences

Séquence de vecteurs (word embeddings)



Simple classification sur tout le vocabulaire

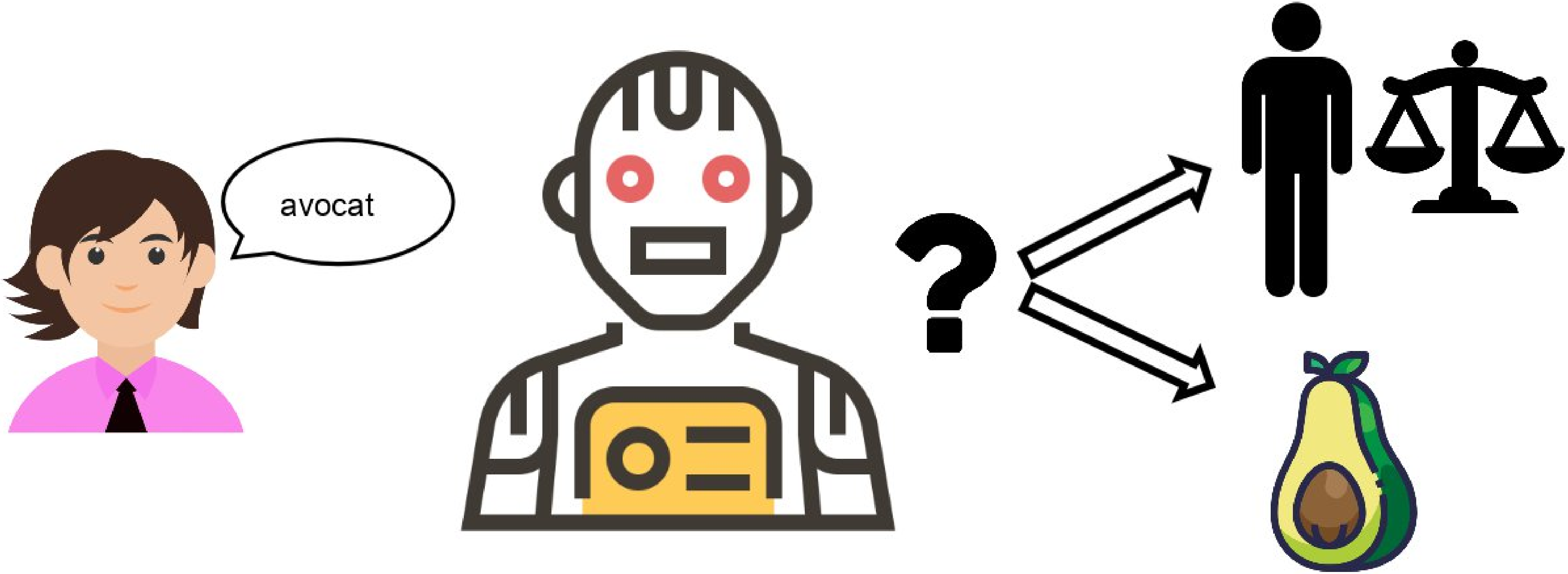
Mécanisme d'attention (decoder)



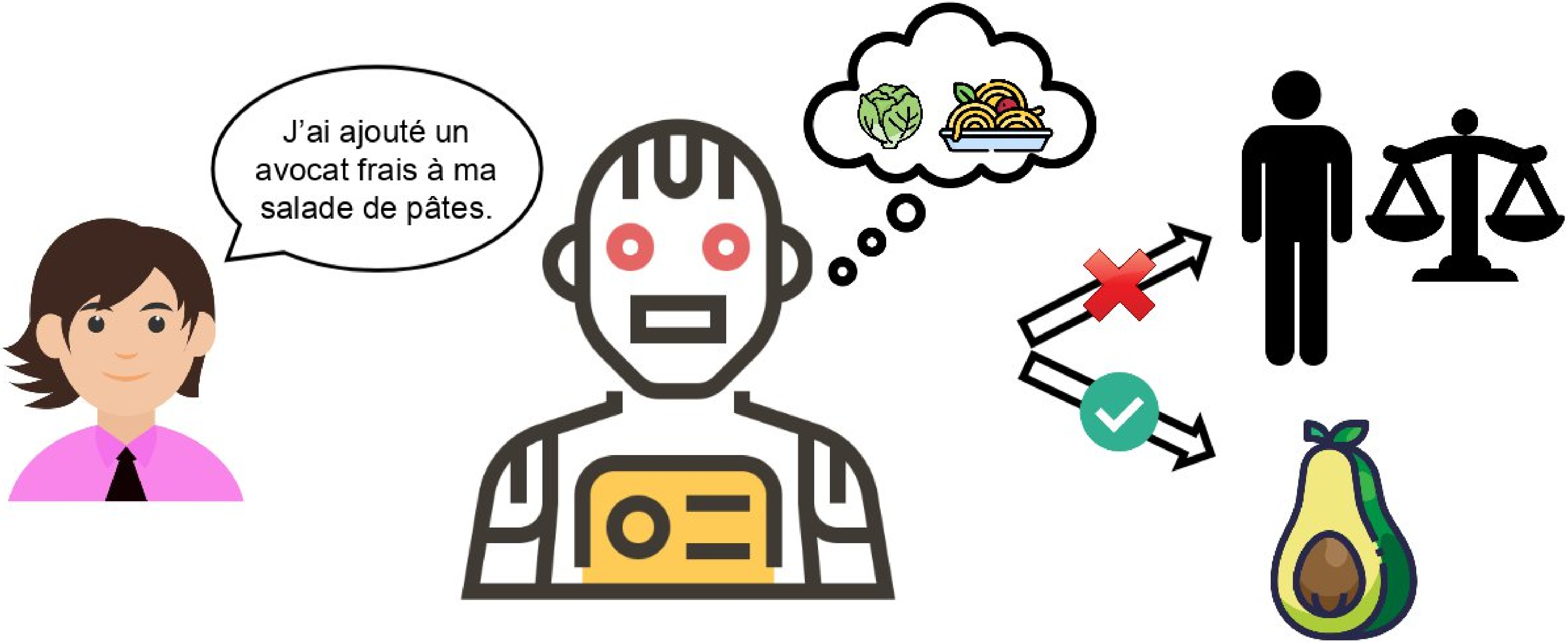
I	I	am	trying	to	understand
am	I	am	trying	to	understand
trying	I	am	trying	to	understand
to	I	am	trying	to	understand
understand	I	am	trying	to	understand

[10]

Mécanisme d’attention (intuition)

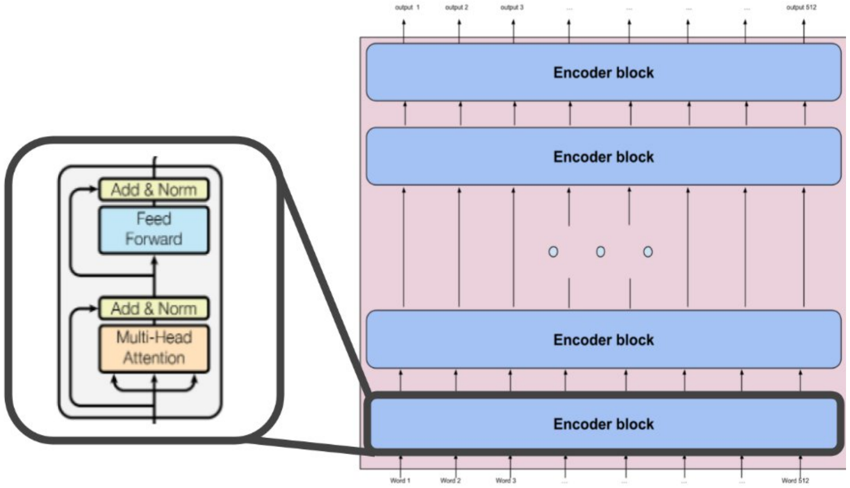


Mécanisme d’attention (intuition)

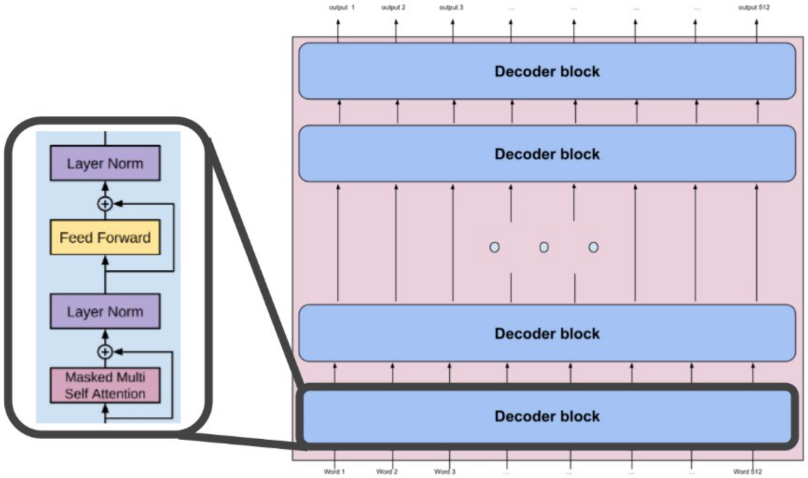


BERT vs GPT

BERT (Auto-encoding)



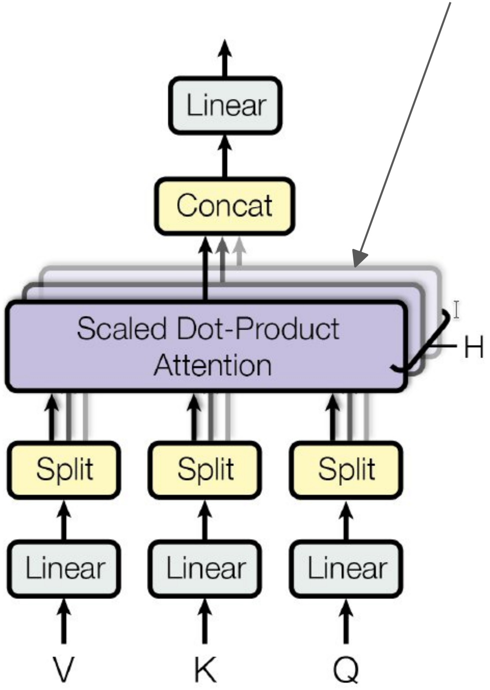
GPT (Auto-regressive)

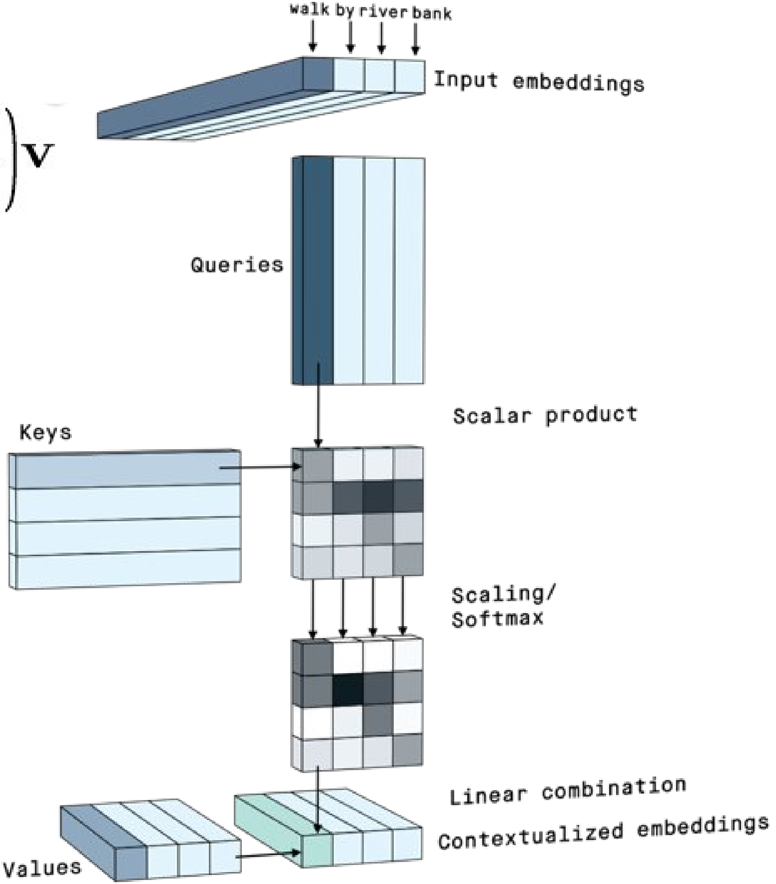


Le mécanisme de Self-Attention

Scaled Dot-Product Attention

$$\text{Attention}\,(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{d_{keys}}}\right)\mathbf{V}$$



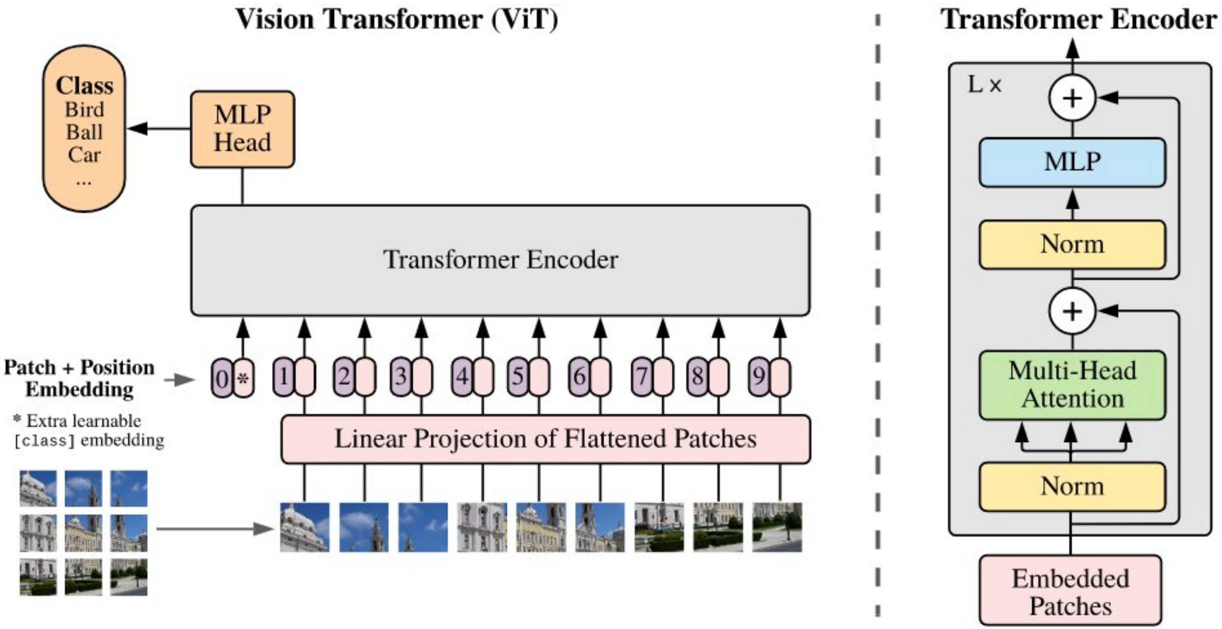


[10, 11]

Les Transformers

- Transforment la séquence entière (contrairement aux CNN et aux RNN)
- Possèdent un nombre conséquent de poids
- Nécessitent de gros *datasets*

Vision Transformer (ViT)



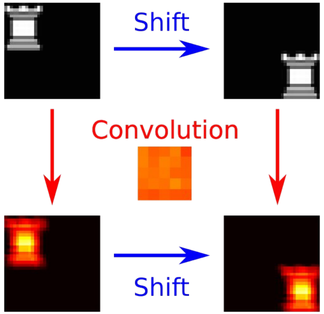
- Images découpées en *patch*
- *Patchs* séquencés avec un *Position embedding*
- Ajout d'un "classification token" pour réaliser la classification finale

[12]

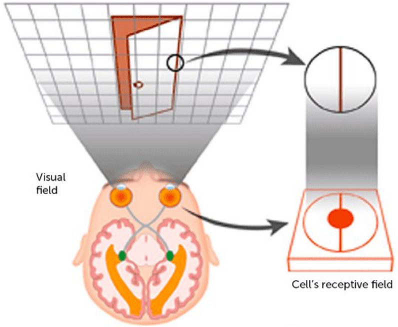
CoAtNet

“Marrying Convolution and Attention for All Data Sizes”

ConvNet
Translation Equivariance



Self-Attention Net
Global Receptive Field
Context comprehension



No patching !!

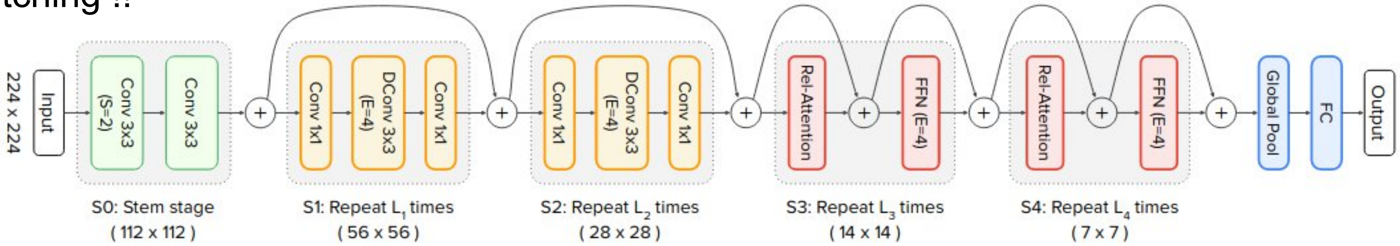


Figure 4: Overview of the proposed CoAtNet.

[13, 14, 15]

CoAtNet - Résultats

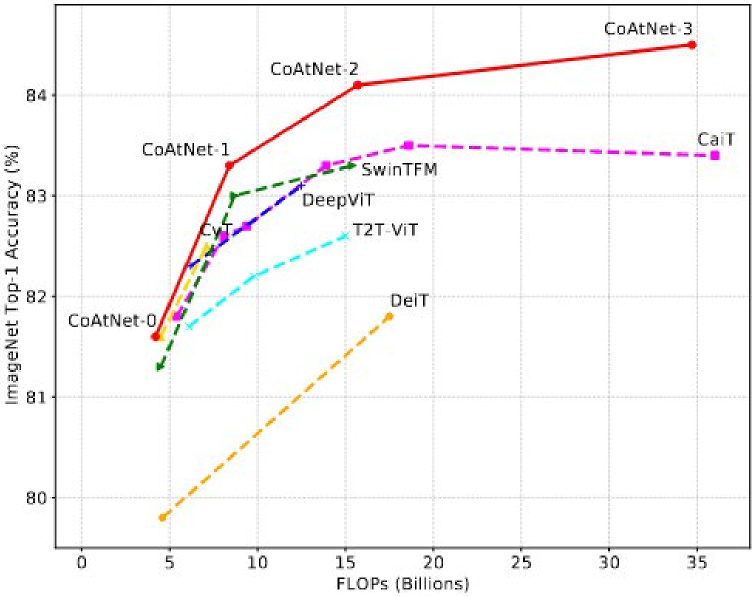


Figure 2: Accuracy-to-FLOPs scaling curve under ImageNet-1K only setting at 224x224.

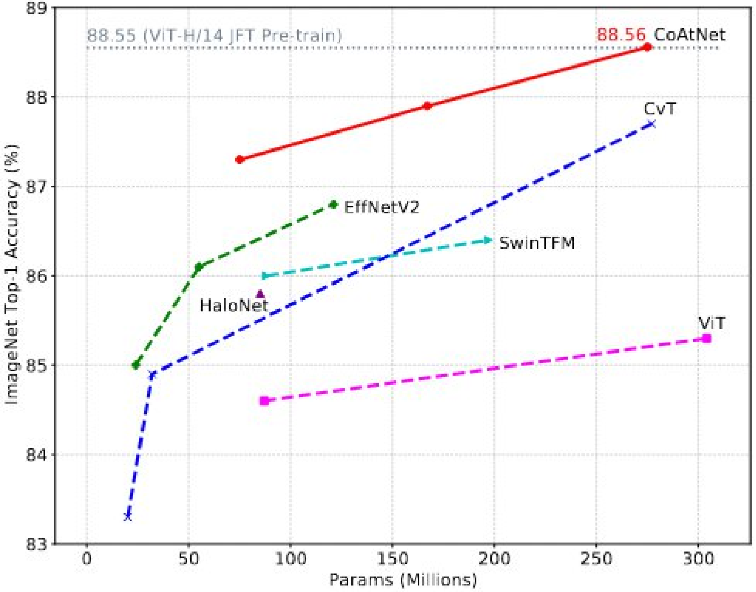


Figure 3: Accuracy-to-Params scaling curve under ImageNet-21K ⇒ ImageNet-1K setting.

[13]

Les Parallélismes de modèle pour les très gros modèles

Pipeline parallelism ◀

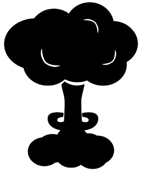
Tensor parallelism ◀

Hybrid parallelism ◀

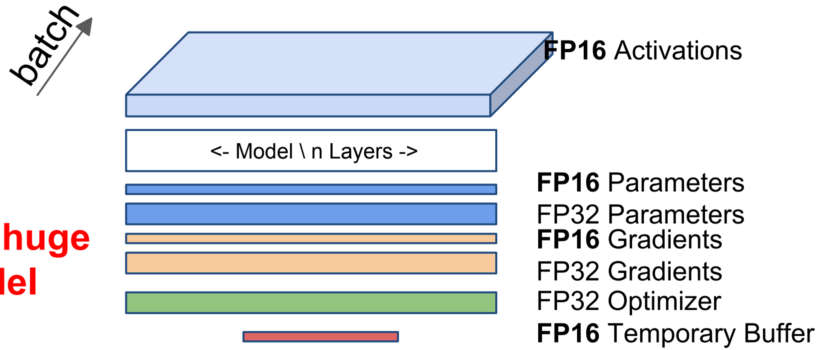
3D parallelism ◀

Énormes modèles > 1 Milliard de paramètres

NLP Transformers



CUDA Out Of Memory



batch

FP16 Activations

<- Model \ n Layers ->

FP16 Parameters

FP32 Parameters

FP16 Gradients

FP32 Gradients

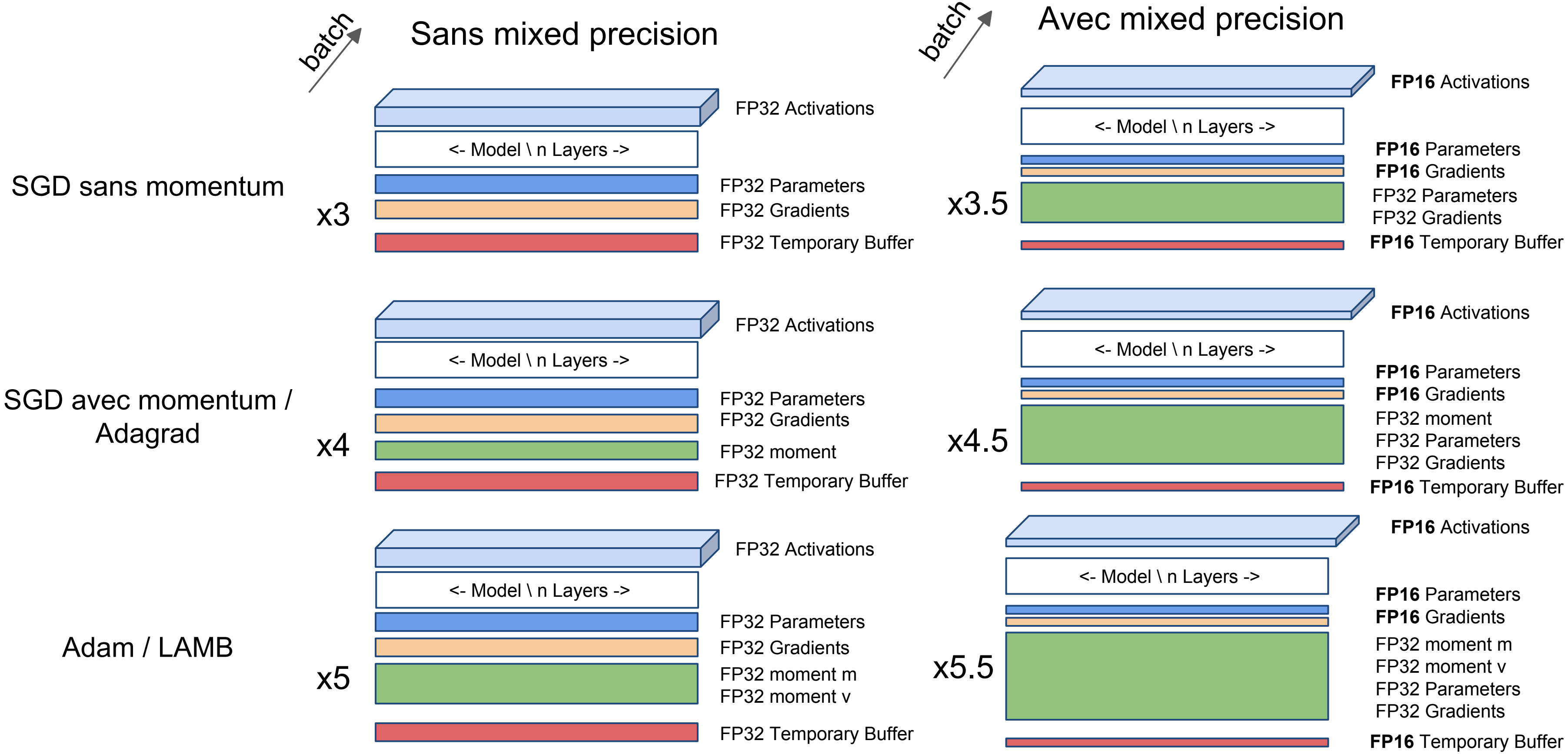
FP32 Optimizer

FP16 Temporary Buffer

~x4 huge model

[1]

Empreinte mémoire



Sans mixed precision

SGD sans momentum

x3

FP32 Activations

<- Model \ n Layers ->

FP32 Parameters
FP32 Gradients
FP32 Temporary Buffer

SGD avec momentum / Adagrad

x4

FP32 Activations

<- Model \ n Layers ->

FP32 Parameters
FP32 Gradients
FP32 moment
FP32 Temporary Buffer

Adam / LAMB

x5

FP32 Activations

<- Model \ n Layers ->

FP32 Parameters
FP32 Gradients
FP32 moment m
FP32 moment v
FP32 Temporary Buffer

Avec mixed precision

x3.5

FP16 Activations

<- Model \ n Layers ->

FP16 Parameters
FP16 Gradients
FP32 Parameters
FP32 Gradients
FP16 Temporary Buffer

x4.5

FP16 Activations

<- Model \ n Layers ->

FP16 Parameters
FP16 Gradients
FP32 moment
FP32 Parameters
FP32 Gradients
FP16 Temporary Buffer

x5.5

FP16 Activations

<- Model \ n Layers ->

FP16 Parameters
FP16 Gradients
FP32 moment m
FP32 moment v
FP32 Parameters
FP32 Gradients
FP16 Temporary Buffer

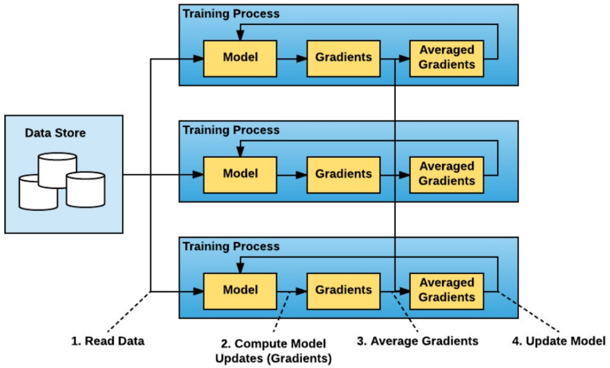
Les différents parallélismes

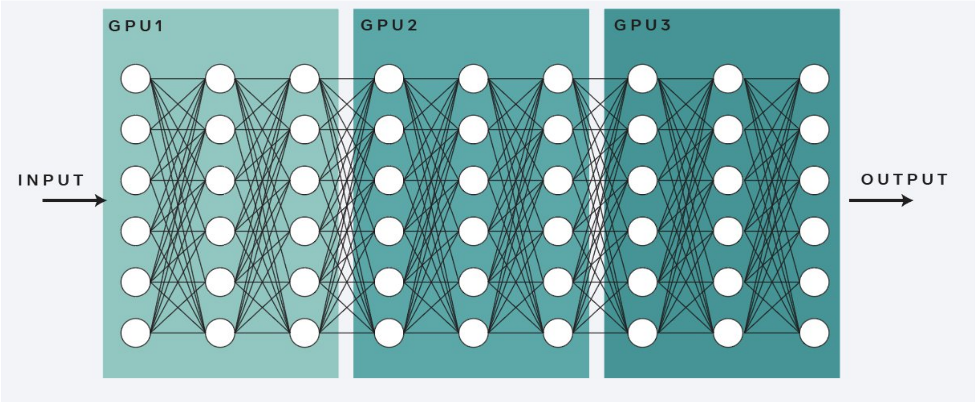
Data Parallelism

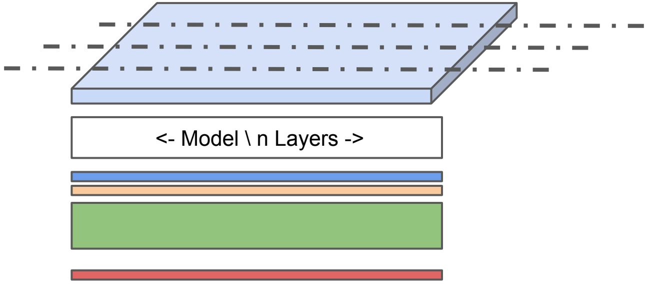
- Meilleur Throughput
- Seule l’empreinte mémoire des activations est distribuée
- Multi Processing

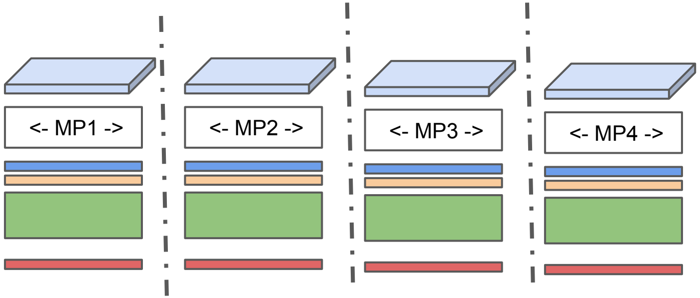
Pipeline Model Parallelism

- Empreinte mémoire distribuée
- Mono ou multi-processing





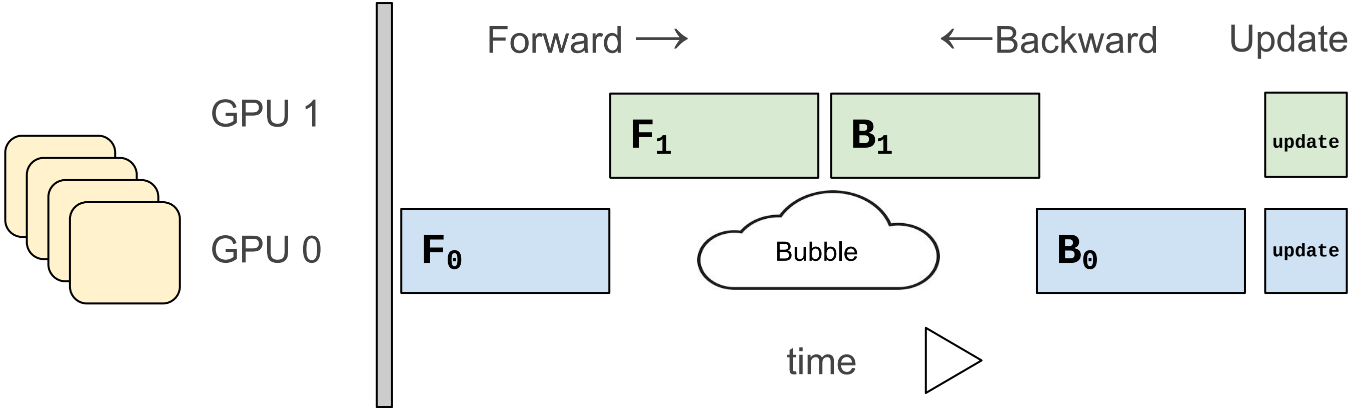


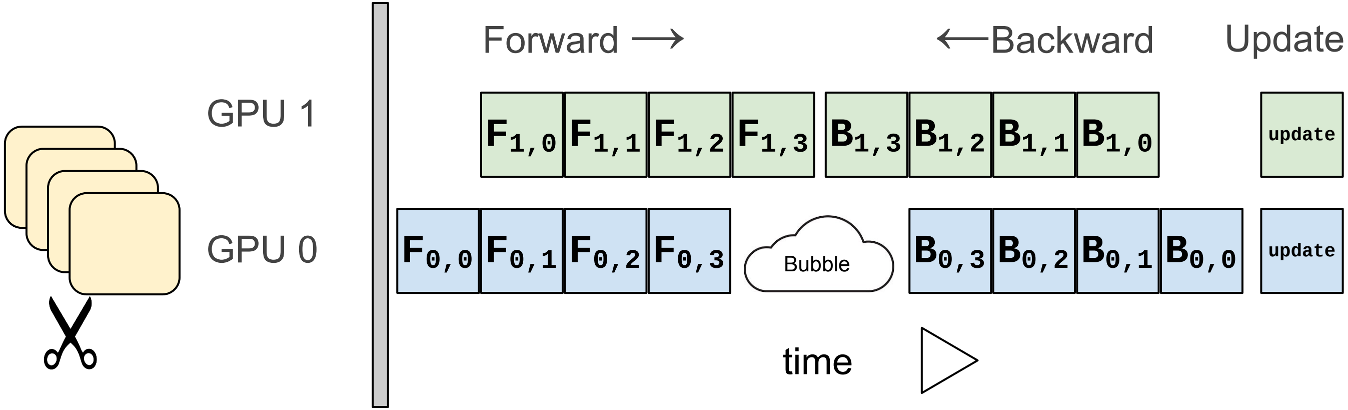


[2, 3]

Pipeline Parallelism

Model parallelism naïf
sur 2 GPU





Model parallelism sur 2
GPU en **pipeline**

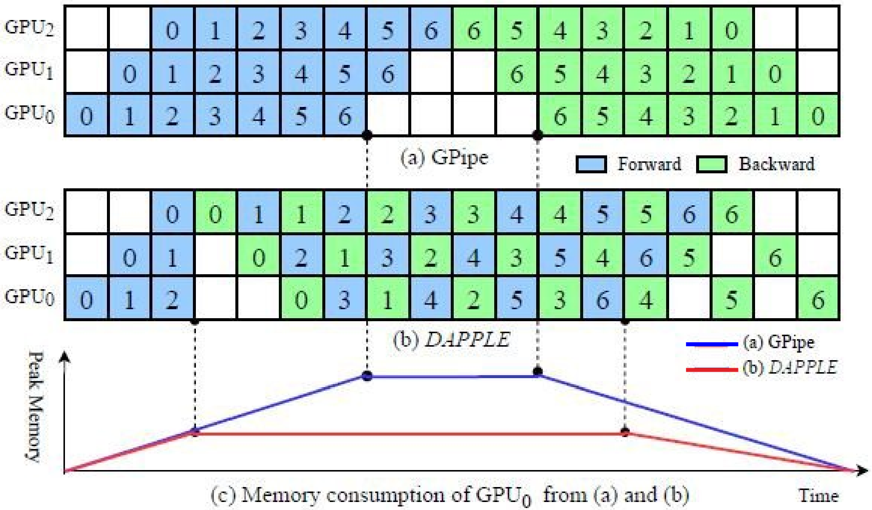
Pipeline Parallelism

Synchronous pipeline :
GPipe, DAPPLE

- \- Throughput
- \+ Memory consumption
- \+ Convergence

Asynchronous pipeline :
PipeDream, PipeMare

- \+ Throughput
- \- Memory consumption
- \- Convergence



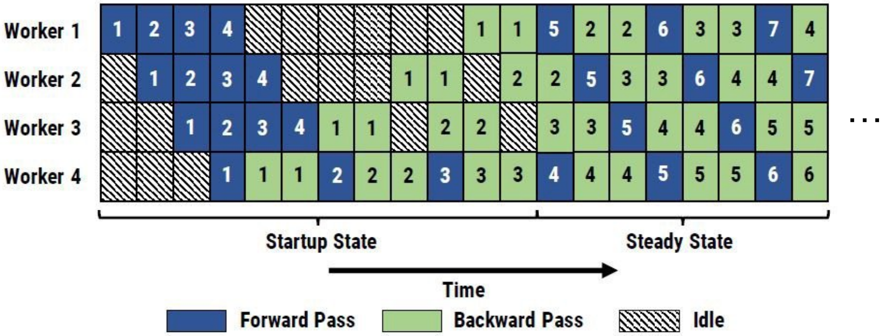
GPU_2			0	1	2	3	4	5	6	6	5	4	3	2	1	0		
GPU_1		0	1	2	3	4	5	6			6	5	4	3	2	1	0	
GPU_0	0	1	2	3	4	5	6					6	5	4	3	2	1	0

(a) GPipe

GPU_2			0	0	1	1	2	2	3	3	4	4	5	5	6	6		
GPU_1		0	1		0	2	1	3	2	4	3	5	4	6	5		6	
GPU_0	0	1	2			0	3	1	4	2	5	3	6	4		5		6

(b) *DAPPLE*

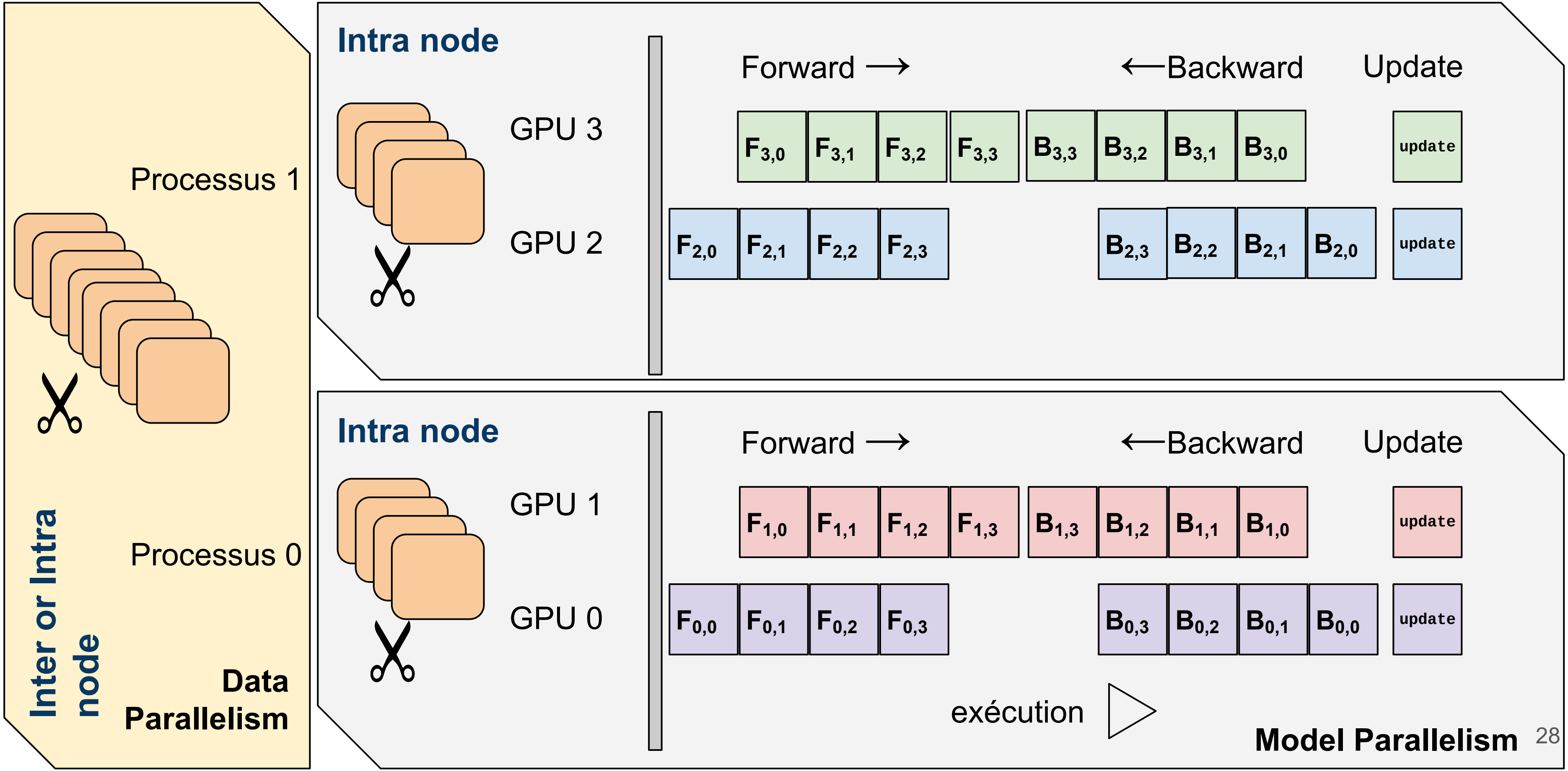
(c) Memory consumption of GPU_0 from (a) and (b)



Worker 1	1	2	3	4	▨	▨	▨	▨	▨	▨	1	1	5	2	2	6	3	3	7	4
Worker 2	▨	1	2	3	4	▨	▨	▨	1	1	▨	2	2	5	3	3	6	4	4	7
Worker 3	▨	▨	1	2	3	4	1	1	▨	2	2	▨	3	3	5	4	4	6	5	5
Worker 4	▨	▨	▨	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6

[4, 5]

Hybrid Parallelism



Processus 1

Processus 0

Inter or Intra node

Data Parallelism

Intra node

GPU 3

GPU 2

Forward ⟶ ⟵Backward Update

$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$ $B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$ update

$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$ $B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$ update

Intra node

GPU 1

GPU 0

Forward ⟶ ⟵Backward Update

$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$ $B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$ update

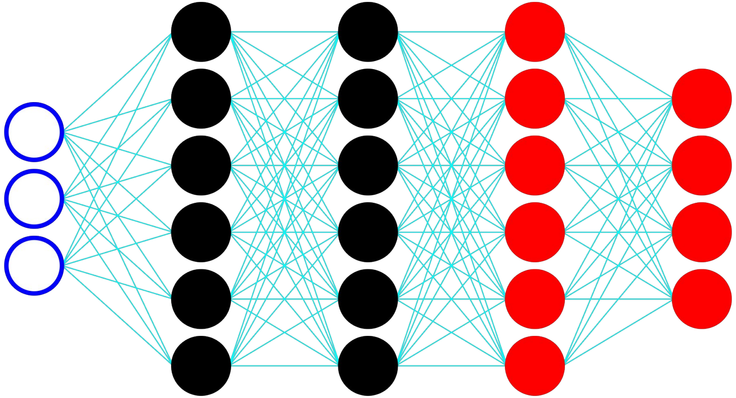
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$ $B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$ update

exécution ▷

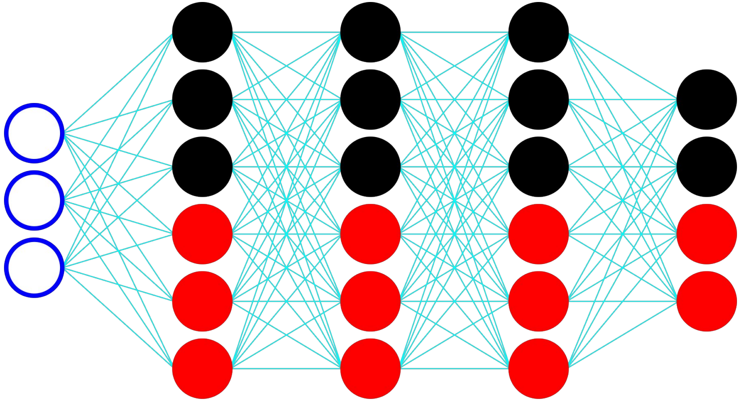
Model Parallelism

Two paradigms for model parallelism

Pipeline Parallelism



Tensor Parallelism



GPU 0

GPU 1

Tensor Parallelism

$$\text{Linear}\left(\mathbf{X}\right) = \mathbf{XW}$$

Découpage par colonne

$$\mathbf{W} = \begin{pmatrix}\mathbf{W_1} & \mathbf{W_2}\end{pmatrix} \qquad \text{Linear}\left(\mathbf{X}\right) = \begin{pmatrix}\mathbf{XW_1} & \mathbf{XW_2}\end{pmatrix}$$

Découpage par ligne

$$\mathbf{W} = \begin{pmatrix}\mathbf{W_1} \\ \mathbf{W_2}\end{pmatrix} \qquad \text{Linear}\left(\begin{pmatrix}\mathbf{X_1} & \mathbf{X_2}\end{pmatrix}\right) = \mathbf{X_1W_1} + \mathbf{X_2W_2}$$

[22]

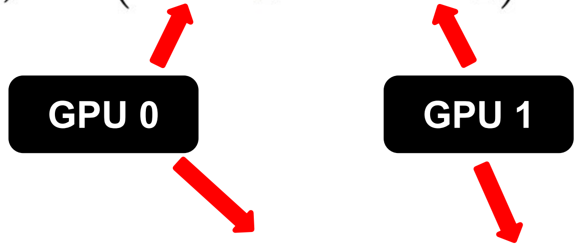
Tensor Parallelism

$$\text{Linear}\left(\mathbf{X}\right) = \mathbf{X}\mathbf{W}$$

Découpage par colonne

$$\mathbf{W} = \begin{pmatrix} \mathbf{W_1} & \mathbf{W_2} \end{pmatrix}$$

$$\text{Linear}\left(\mathbf{X}\right) = \begin{pmatrix} \mathbf{X}\mathbf{W_1} & \mathbf{X}\mathbf{W_2} \end{pmatrix}$$



Découpage par ligne

$$\mathbf{W} = \begin{pmatrix} \mathbf{W_1} \\ \mathbf{W_2} \end{pmatrix}$$

$$\text{Linear}\left(\begin{pmatrix} \mathbf{X_1} & \mathbf{X_2} \end{pmatrix}\right) = \mathbf{X_1}\mathbf{W_1} + \mathbf{X_2}\mathbf{W_2}$$

[22]

Tensor Parallelism

$$\text{Linear}\left(\mathbf{X}\right) = \mathbf{XW}$$

Découpage par colonne

$$\mathbf{W} = \begin{pmatrix}\mathbf{W_1} & \mathbf{W_2}\end{pmatrix} \qquad \text{Linear}\left(\mathbf{X}\right) = \begin{pmatrix}\mathbf{XW_1} & \mathbf{XW_2}\end{pmatrix}$$

AllGather

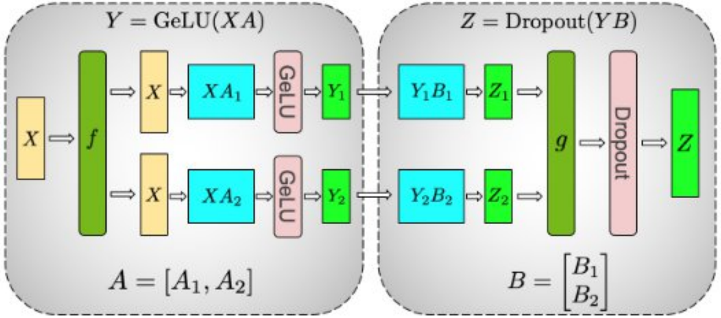
Découpage par ligne

$$\mathbf{W} = \begin{pmatrix}\mathbf{W_1} \\ \mathbf{W_2}\end{pmatrix} \qquad \text{Linear}\left(\begin{pmatrix}\mathbf{X_1} & \mathbf{X_2}\end{pmatrix}\right) = \mathbf{X_1W_1} + \mathbf{X_2W_2}$$

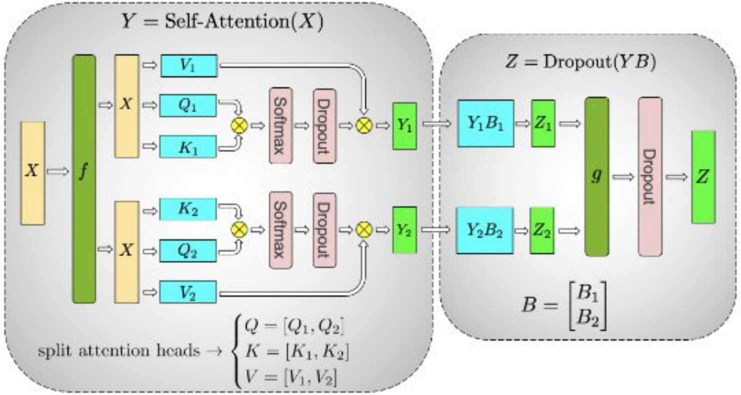
AllReduce

[22]

Tensor Parallelism



(a) MLP



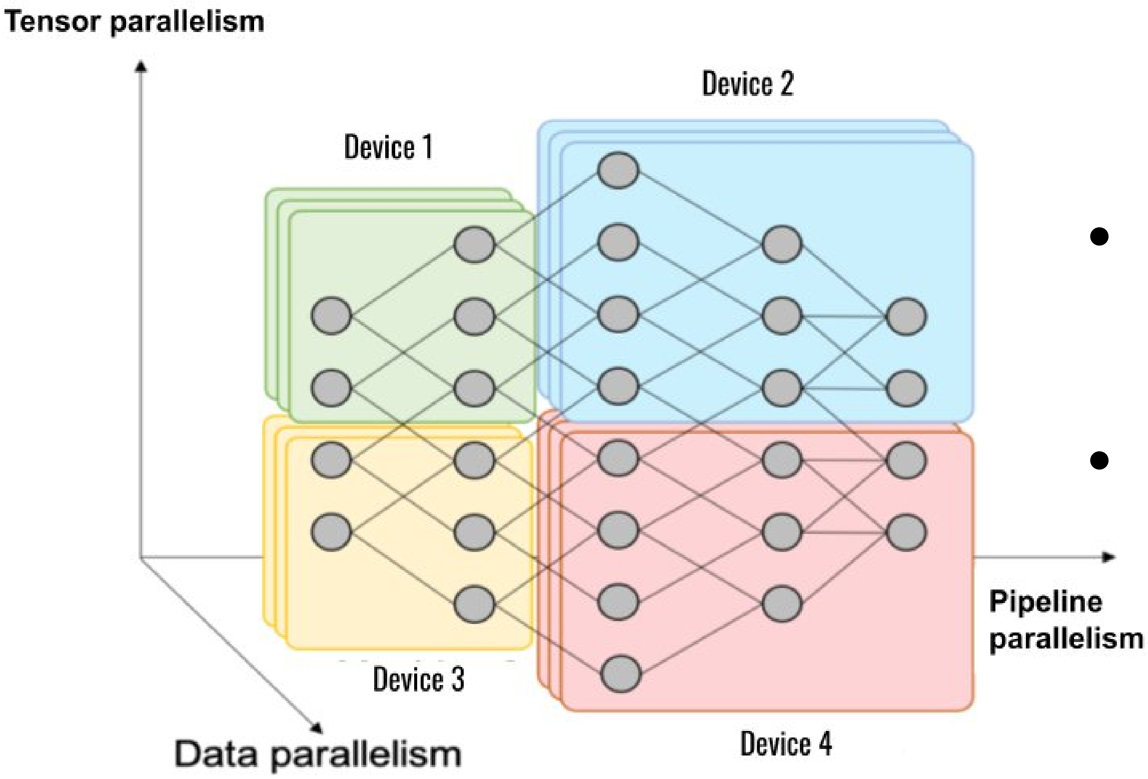
(b) Self-Attention

Par défaut, le tensor parallelism exige des synchronisations à **chaque** couche.

En alternant coupure en lignes et coupure en colonnes, on peut se permettre de ne communiquer qu’une fois toutes les **deux** couches denses.

[22]

3D Parallelism



- **Data Parallelism**
  - Simple à implémenter
  - Meilleure performance
  - Augmente la taille du batch (problème de convergence)

- **Pipeline Parallelism**
  - Effort d'implémentation.
  - Équilibre entre mémoire, performance et convergence.

- **Tensor Parallelism**
  - Effort important d'implémentation
  - Bonne accélération des calculs
  - **Bande passante très sollicitée** (implémentation Intra-nœud)

API pour les gros modèles

- Deepspeed ◂
- Fully Sharded Data Parallel ◂
- Megatron-LM ◂
- Accelerate, Fabric & vLLM ◂

Deepspeed

Model Scale

Support 200B

Toward 100 Trillion

Speed

Up to 10x faster

Scalability

Superlinear speedup

Usability

Few lines of code changes

```
# Include DeepSpeed configuration arguments
parser = deepspeed.add_config_arguments(parser)
```

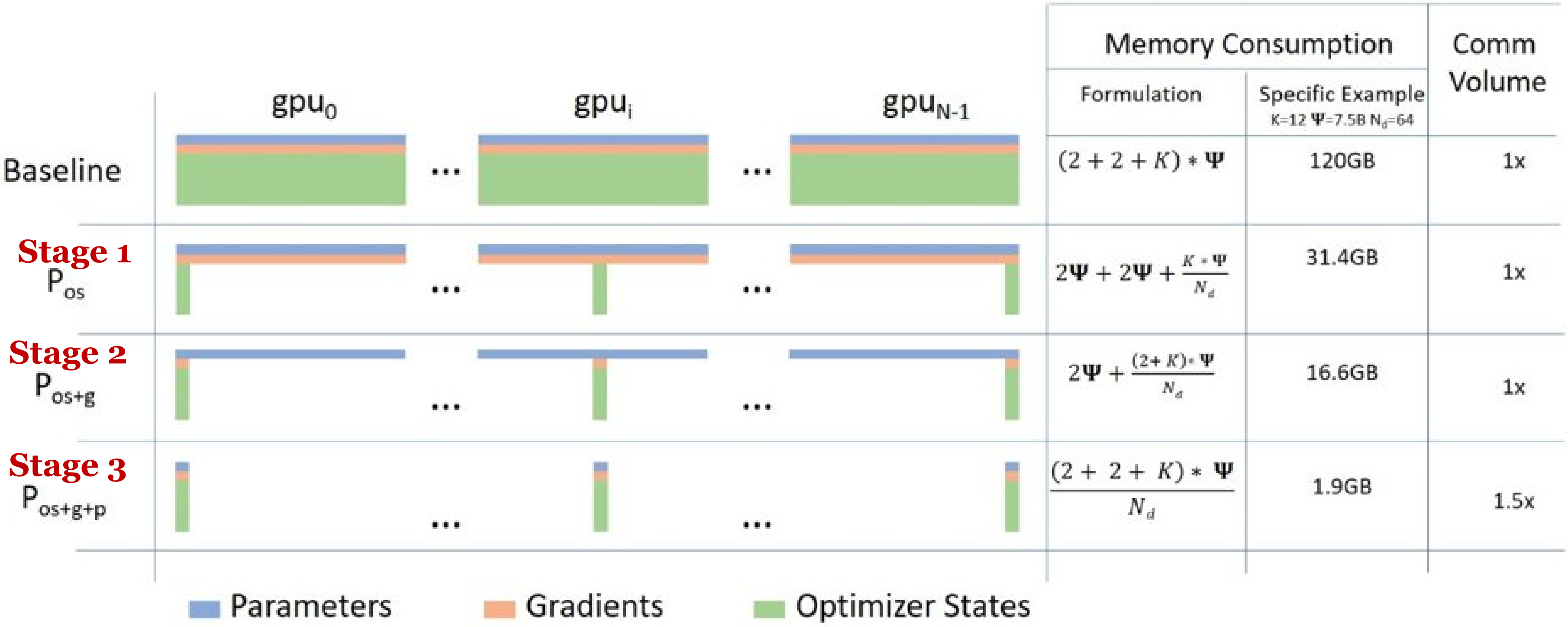
```
# Initialize DeepSpeed to use the following features
# 1) Distributed model
# 2) DeepSpeed optimizer
model_engine, optimizer, _, _ = deepspeed.initialize(
                                 args=args, model=model,
                                 model_parameters=parameters,
                                 optimizer=optimizer)
```

```
for step, batch in enumerate(data_loader):
    #forward() method
    loss = model_engine(batch)

    #runs backpropagation
    model_engine.backward(loss)

    #weight update
    model_engine.step()
```

```
{
    "zero_optimization": {
        "stage":2,
        "contiguous_gradients": true,
        "overlap_comm": true,
        "reduce_scatter": true,
        "reduce_bucket_size": 5e8,
        "allgather_bucket_size": 5e8
    }
}
```

```
# SLURM Job submission
srun train.py -b 28 -s 200 --image-size 288
 —deepspeed --deepspeed_config
ds_config_zero2.json
```

ZeRO — Optimisation du data parallelism



	Memory Consumption		Comm Volume
	Formulation	Specific Example $K=12$ $\Psi=7.5B$ $N_d=64$	
Baseline	$(2+2+K)*\Psi$	120GB	1x
Stage 1 P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
Stage 2 P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
Stage 3 P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

🟦 Parameters 🟧 Gradients 🟩 Optimizer States

[6]

Fused optimizers

Implémentations présentent dans *APEX*

Fusionne des ***kernels* GPU** pour économiser les opérations de lecture / écriture de mémoire

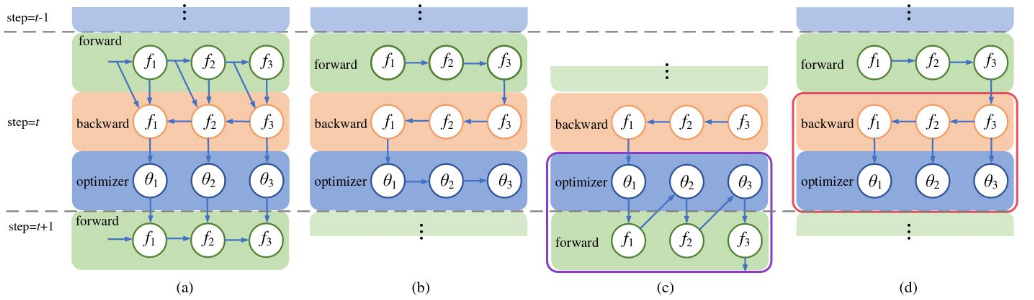
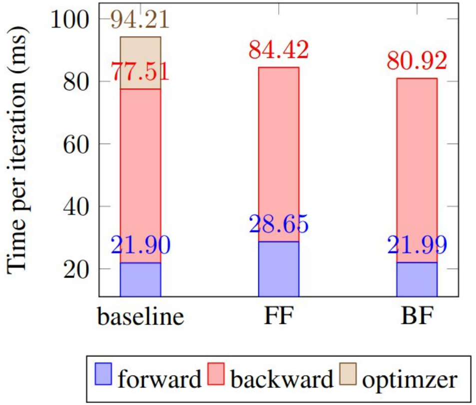


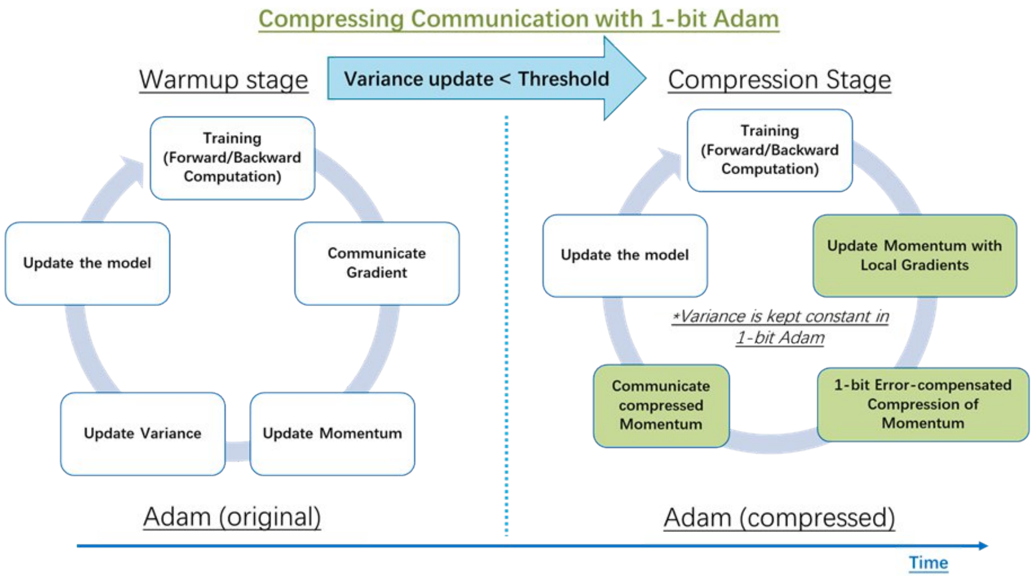
Figure 1: (a) Data dependency graph. (b) Baseline method. (c) `Forward-fusion`. (d) `Backward-fusion`. θ_i represents the trainable parameters in the layer f_i.



But : accélère l’étape des optimiseurs sur GPU.

[7]

1-bit optimizers



Compressing Communication with 1-bit Adam

Warmup stage

Variance update < Threshold

Compression Stage

Training (Forward/Backward Computation)

Communicate Gradient

Update Momentum

Update Variance

Update the model

Adam (original)

Training (Forward/Backward Computation)

Update Momentum with Local Gradients

1-bit Error-compensated Compression of Momentum

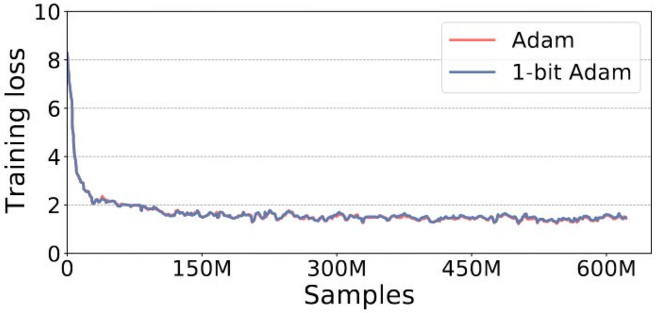
Communicate compressed Momentum

Update the model

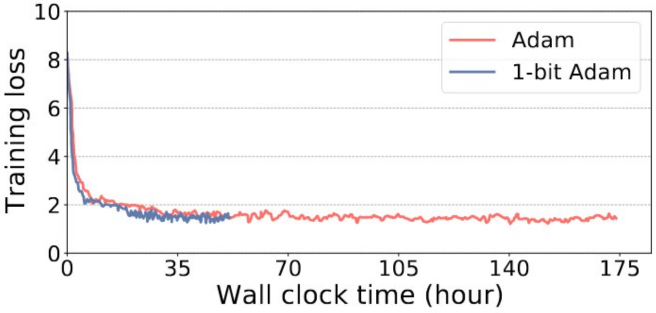
Variance is kept constant in 1-bit Adam

Adam (compressed)

Time



(a) Sample-wise



(b) Time-wise

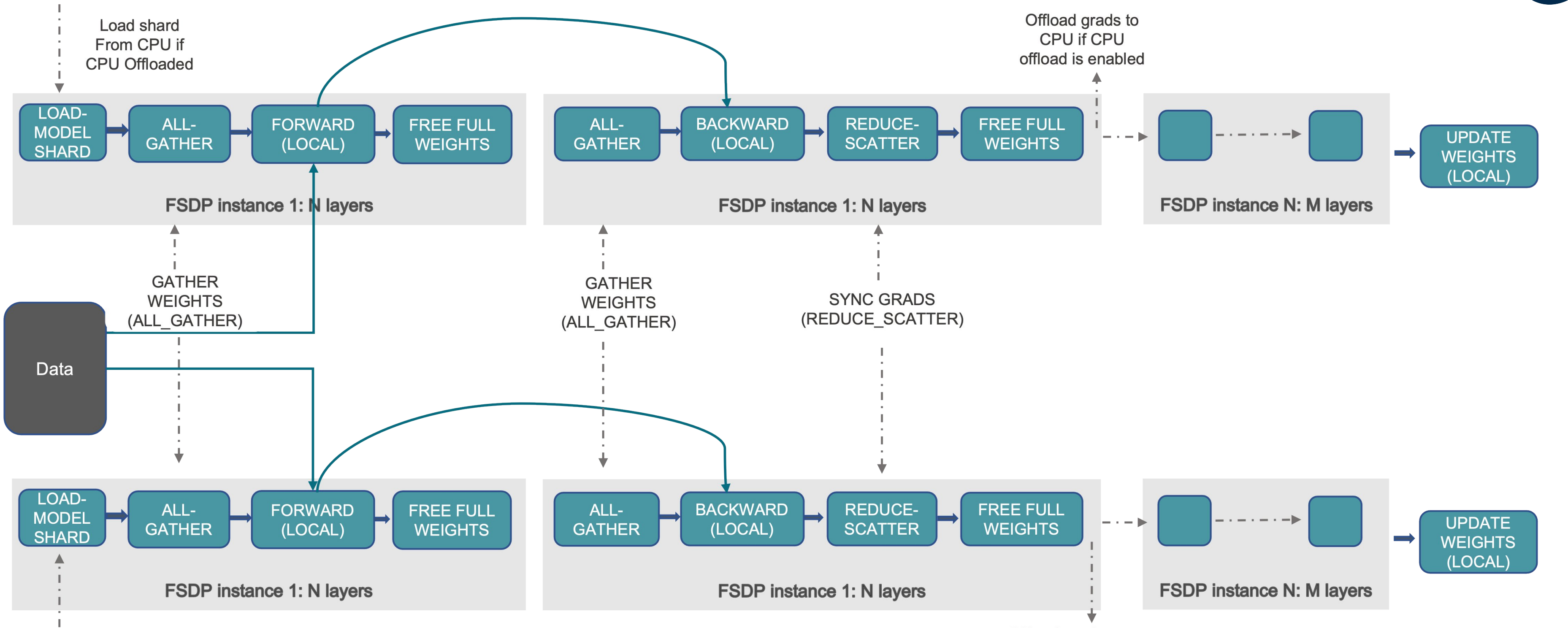
But : diminuent les communications nécessaires et donc accélère l’étape des optimiseurs pour un modèle distribué.

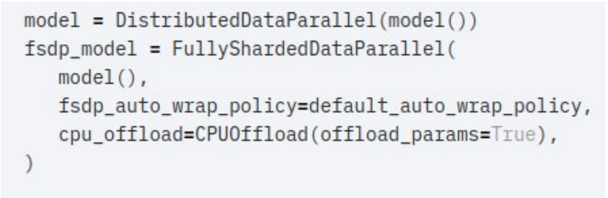
[8, 9]

Toutes les applications de Deepspeed

- **Distributed Training with Mixed Precision**
  - 16-bit mixed precision
  - Single-GPU/Multi-GPU/Multi-Node
- **Model Parallelism**
  - Support for Custom Model Parallelism
  - **Integration with Megatron-LM**
- **Pipeline Parallelism**
  - 3D Parallelism
- **The Zero Redundancy Optimizer (ZeRO)**
  - Optimizer State and Gradient Partitioning
  - Activation Partitioning
  - Constant Buffer Optimization
  - Contiguous Memory Optimization
- ZeRO-Offload
  - Leverage both CPU/GPU memory for model training
  - Support 10B model training on a single GPU
- Ultra-fast dense transformer kernels
- Sparse attention
  - Memory- and compute-efficient sparse kernels
  - Support 10x longer sequences than dense
  - Flexible support to different sparse structures
- **1-bit Adam and 1-bit LAMB**
  - Custom communication collective
  - Up to 5x communication volume saving
- Additional Memory and Bandwidth Optimizations
  - Smart Gradient Accumulation
  - Communication/Computation Overlap
- Training Features
  - Simplified training API
  - Gradient Clipping
  - Automatic loss scaling with mixed precision
- Training Optimizers
  - Fused Adam optimizer and arbitrary torch.optim.Optimizer
  - Memory bandwidth optimized FP16 Optimizer
  - Large Batch Training with LAMB Optimizer
  - Memory efficient Training with ZeRO Optimizer
  - CPU-Adam
- Training Agnostic Checkpointing
- Advanced Parameter Search
  - Learning Rate Range Test
  - 1Cycle Learning Rate Schedule
- Simplified Data Loader
- Performance Analysis and Debugging

Fully Sharded Data Parallel



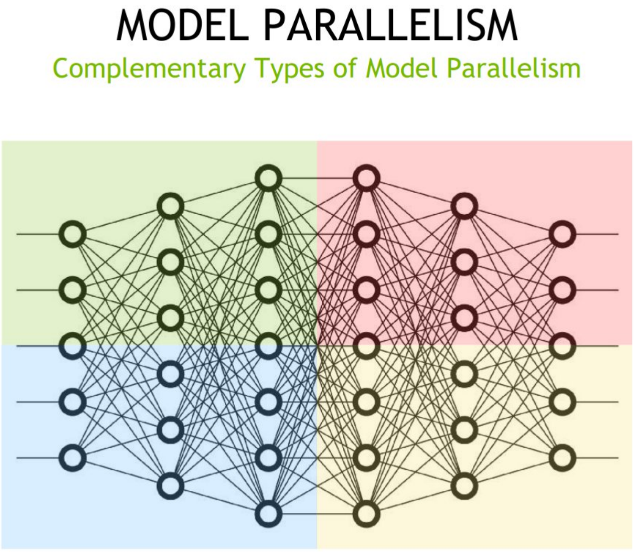


```
model = DistributedDataParallel(model())
fsdp_model = FullyShardedDataParallel(
   model(),
   fsdp_auto_wrap_policy=default_auto_wrap_policy,
   cpu_offload=CPUOffload(offload_params=True),
)
```

[29]

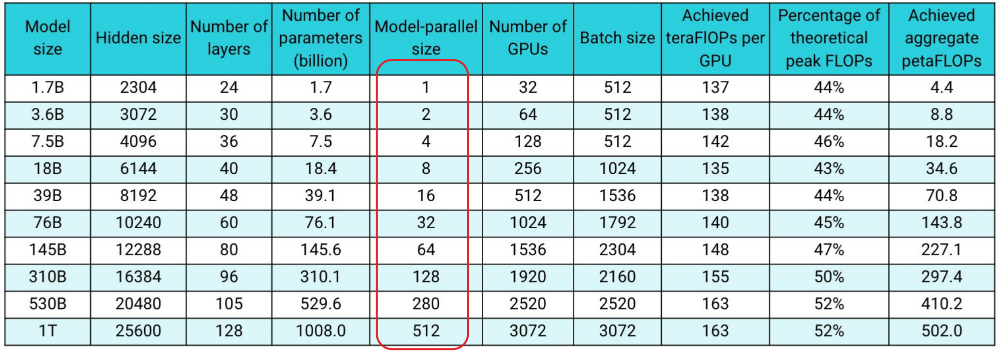
Megatron-LM

Model Parallelism de GPU NVIDIA (tensor and pipeline) efficace en multi-nœud pour le *pre-training* de *Transformer* comme [GPT](), [BERT](), et [T5]() utilisant la *mixed precision*.


MODEL PARALLELISM

Complementary Types of Model Parallelism

Inter + Intra Parallelism

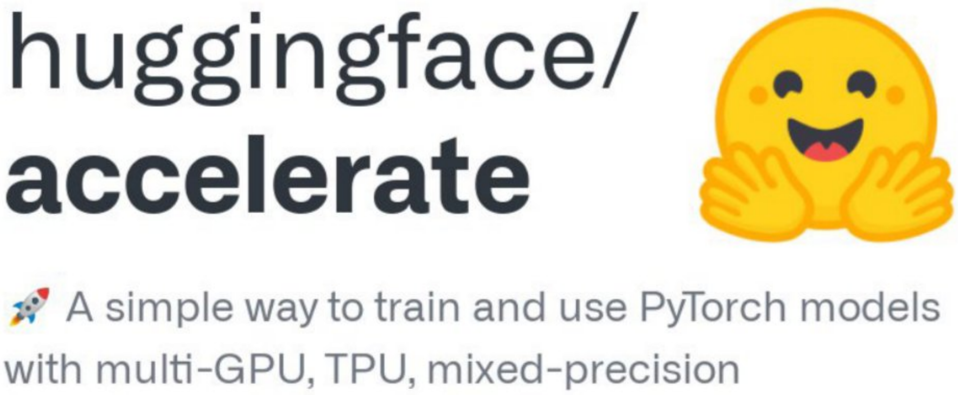


Model size	Hidden size	Number of layers	Number of parameters (billion)	Model-parallel size	Number of GPUs	Batch size	Achieved teraFlOPs per GPU	Percentage of theoretical peak FLOPs	Achieved aggregate petaFLOPs
1.7B	2304	24	1.7	1	32	512	137	44%	4.4
3.6B	3072	30	3.6	2	64	512	138	44%	8.8
7.5B	4096	36	7.5	4	128	512	142	46%	18.2
18B	6144	40	18.4	8	256	1024	135	43%	34.6
39B	8192	48	39.1	16	512	1536	138	44%	70.8
76B	10240	60	76.1	32	1024	1792	140	45%	143.8
145B	12288	80	145.6	64	1536	2304	148	47%	227.1
310B	16384	96	310.1	128	1920	2160	155	50%	297.4
530B	20480	105	529.6	280	2520	2520	163	52%	410.2
1T	25600	128	1008.0	512	3072	3072	163	52%	502.0

La colonne *Model-parallel size* décrit un degré de *Tensor Parallelism* et de *Pipeline Parallelism* combinés

Pour les nombres supérieurs à 8, un *Tensor Parallelism* de taille 8 est typiquement utilisé. Ainsi, par exemple, le modèle de *145B* indique une taille de *Model Parallelism* totale de 64, ce qui signifie que cette configuration a utilisé TP=8 et PP=8.

Fabric & Accelerate



huggingface/
accelerate

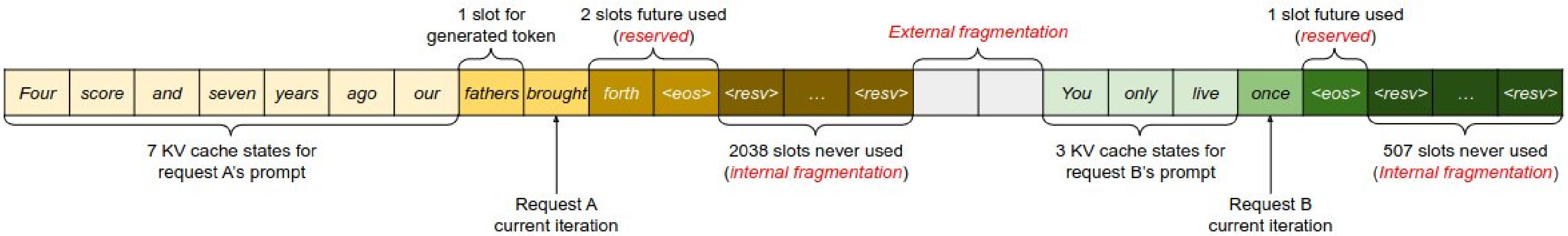
🚀 A simple way to train and use PyTorch models with multi-GPU, TPU, mixed-precision

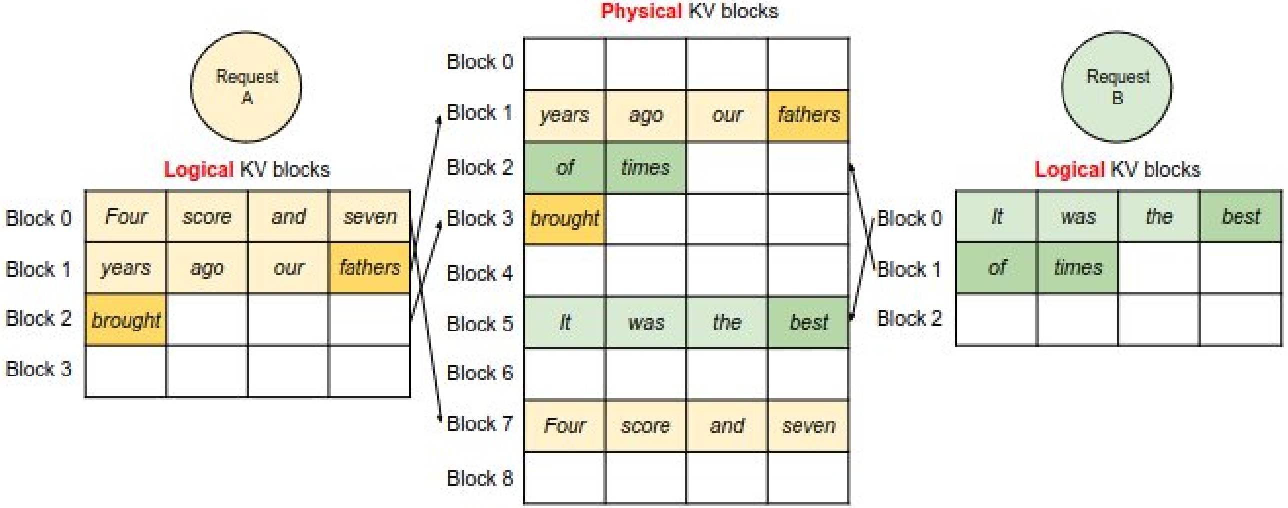
```
srun idr_accelerate --config_file myconfig.json --zero_stage 3 train.py --lr 0.5
```

Lightning Fabric

[17, 28]

vLLM (Inférence des transformers)





Logical KV blocks (Request A)

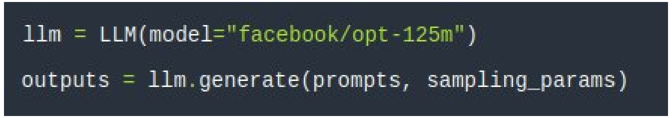
Block 0	*Four*	*score*	*and*	*seven*
Block 1	*years*	*ago*	*our*	*fathers*
Block 2	*brought*			
Block 3				

Physical KV blocks

Block 0				
Block 1	*years*	*ago*	*our*	*fathers*
Block 2	*of*	*times*		
Block 3	*brought*			
Block 4				
Block 5	*It*	*was*	*the*	*best*
Block 6				
Block 7	*Four*	*score*	*and*	*seven*
Block 8				

Logical KV blocks (Request B)

Block 0	*It*	*was*	*the*	*best*
Block 1	*of*	*times*		
Block 2				

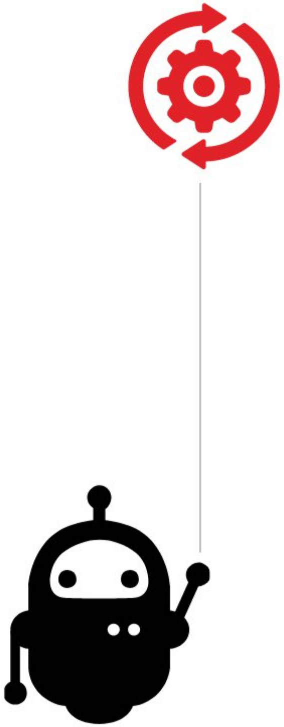


```
llm = LLM(model="facebook/opt-125m")
outputs = llm.generate(prompts, sampling_params)
```

vLLM

[16]

TP : CoAtNet-7



- Limite du *Data Parallelism* avec CoAtNet

- Implémenter ZeRO
- Implementer le Pipeline Parallelism
- Recherche du meilleur compromis

Références des images utilisées et articles

1. HuggingFace 2021, https://huggingface.co/blog/large-language-models
2. Nicolae, Bogdan, et al. "Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models." *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020.
3. FairScale authors. (2021). FairScale: A general purpose modular PyTorch library for high performance and large scale training. https://fairscale.readthedocs.io/en/latest/deep_dive/pipeline_parallelism.html
4. Fan, Shiqing, et al. "DAPPLE: A pipelined data parallel approach for training large models." *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2021.
5. Narayanan, Deepak, et al. "PipeDream: Generalized pipeline parallelism for DNN training." *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 2019.
6. Rajbhandari, Samyam, et al. "Zero: Memory optimizations toward training trillion parameter models." *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020.
7. Jiang, Zixuan, et al. "Optimizer Fusion: Efficient Training with Better Locality and Parallelism." *arXiv preprint arXiv:2104.00237* (2021).
8. Deepspeed 2020, https://www.deepspeed.ai/2020/09/08/onebit-adam-blog-post.html
9. Tang, Hanlin, et al. "1-bit adam: Communication efficient large-scale training with adam's convergence speed." *International Conference on Machine Learning*. PMLR, 2021.
10. Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
11. Peltarion, https://peltarion.com/blog/data-science/self-attention-video
12. Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).
13. Dai, Zihang, et al. "Coatnet: Marrying convolution and attention for all data sizes." *Advances in Neural Information Processing Systems* 34 (2021): 3965-3977.
14. Medium, https://medium.com/@oskyhn_77789/current-convolutional-neural-networks-are-not-translation-equivariant-2f04bb9062e3
15. AI Summer, https://theaisummer.com/receptive-field/
16. https://vllm.ai/
17. https://huggingface.co/docs/accelerate/index
18. Gou, Jianping, et al. "Knowledge distillation: A survey." *International Journal of Computer Vision* 129 (2021): 1789-1819.
19. Gholami, Amir, et al. "A survey of quantization methods for efficient neural network inference." *arXiv preprint arXiv:2103.13630* (2021).
20. Zhou, Aojun, et al. "Learning N: M fine-grained structured sparse neural networks from scratch." *arXiv preprint arXiv:2102.04010* (2021).
21. https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/
22. Shoeybi, Mohammad, et al. "Megatron-lm: Training multi-billion parameter language models using model parallelism." *arXiv preprint arXiv:1909.08053* (2019).
23. Bengio, Yoshua, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." *arXiv preprint arXiv:1308.3432* (2013).
24. Frankle, Jonathan, and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks." *arXiv preprint arXiv:1803.03635* (2018).
25. Gale, Trevor, Erich Elsen, and Sara Hooker. "The state of sparsity in deep neural networks." *arXiv preprint arXiv:1902.09574* (2019).
26. Zhu, Michael, and Suyog Gupta. "To prune, or not to prune: exploring the efficacy of pruning for model compression." *arXiv preprint arXiv:1710.01878* (2017).
27. Sanh, Victor, et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." *arXiv preprint arXiv:1910.01108* (2019).
28. https://lightning.ai/docs/fabric/stable/
29. https://pytorch.org/blog/introducing-pytorch-fully-sharded-data-parallel-api/