Deep Learning Optimisé - Jean Zay

Les optimisations des gros modèles

Institut du
Développement et des
Ressources en
Informatique
Scientifique

Les très gros modèles

Transformers ◂

Vision Transformers ◂

CoAtNet ◂

Vision Transformers >> Resnet-152: 60M

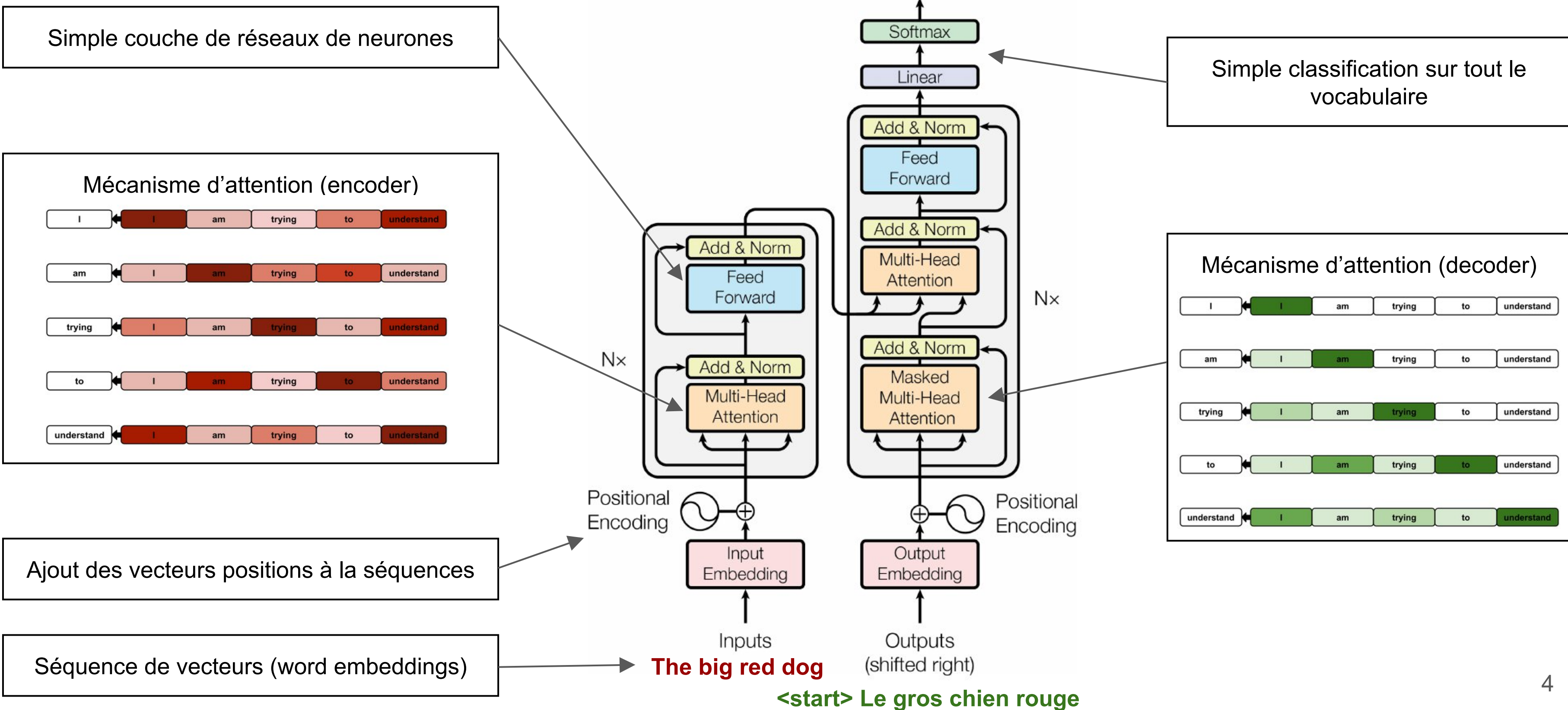
Rank	Model	Top 1 Accuracy	Top 5 Accuracy	Number of params	Extra Training Data	Paper	Code	Result	Year	Tags
1	**CoAtNet-7**	90.88%		2440M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
2	**ViT-G/14**	90.45%		1843M	✓	Scaling Vision Transformers			2021	Transformer 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

<stop>

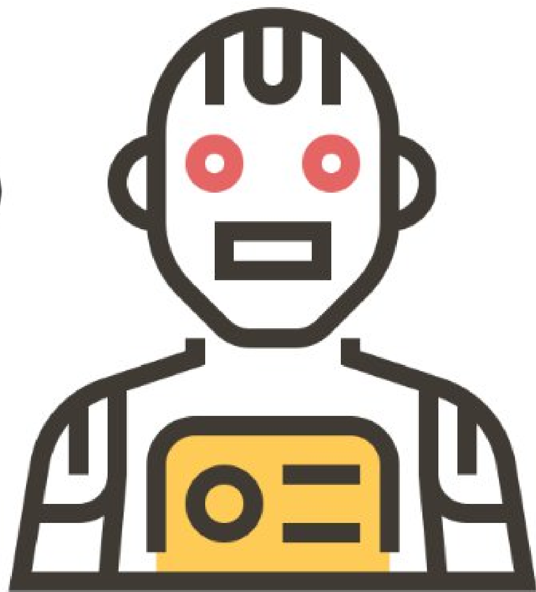
[Attention Is All You Need]

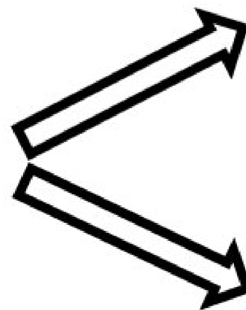


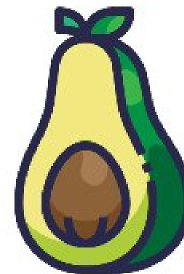
[10]

Mécanisme d’attention (intuition)

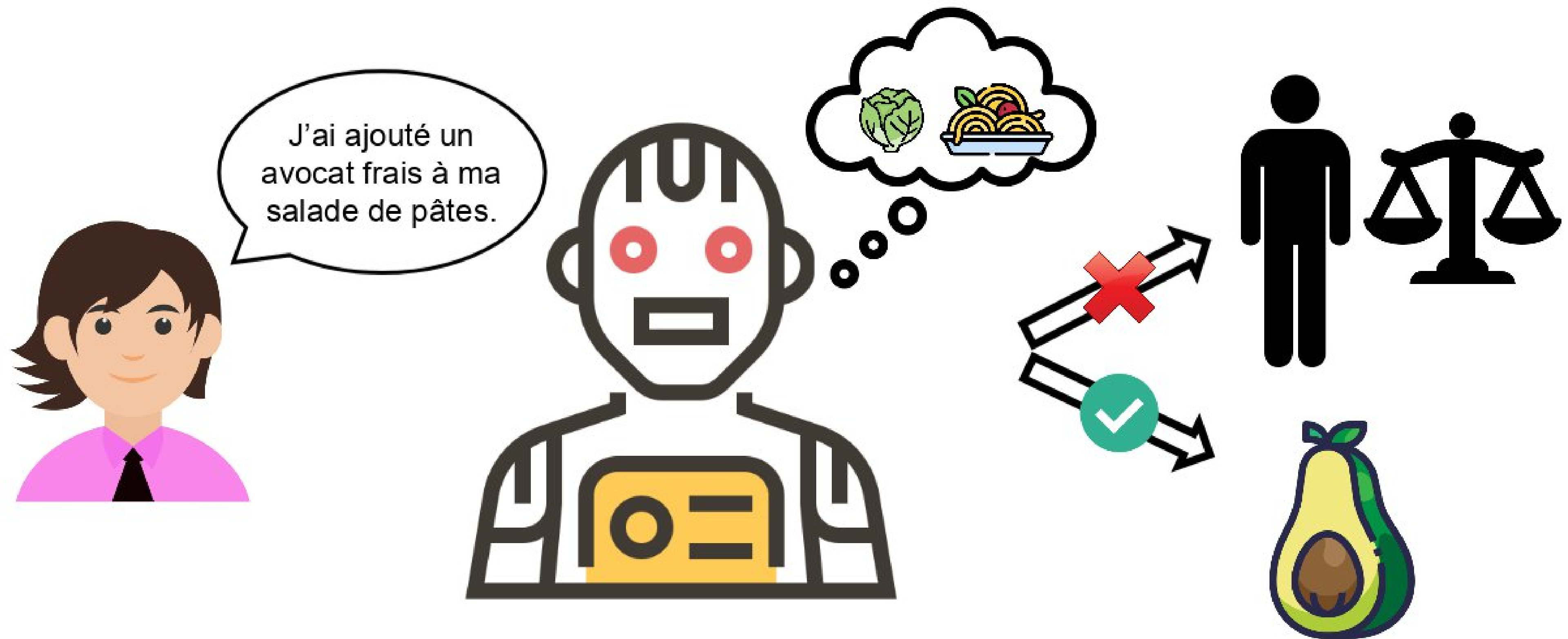
avocat



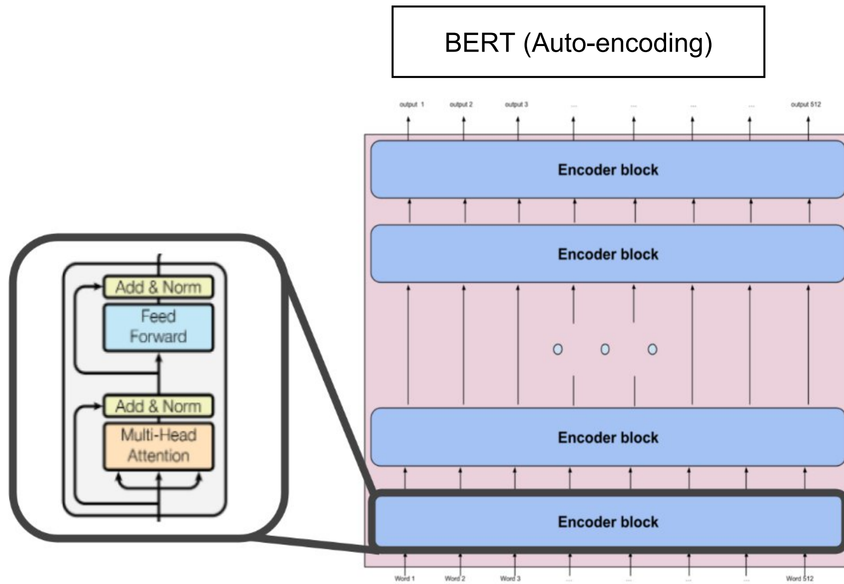


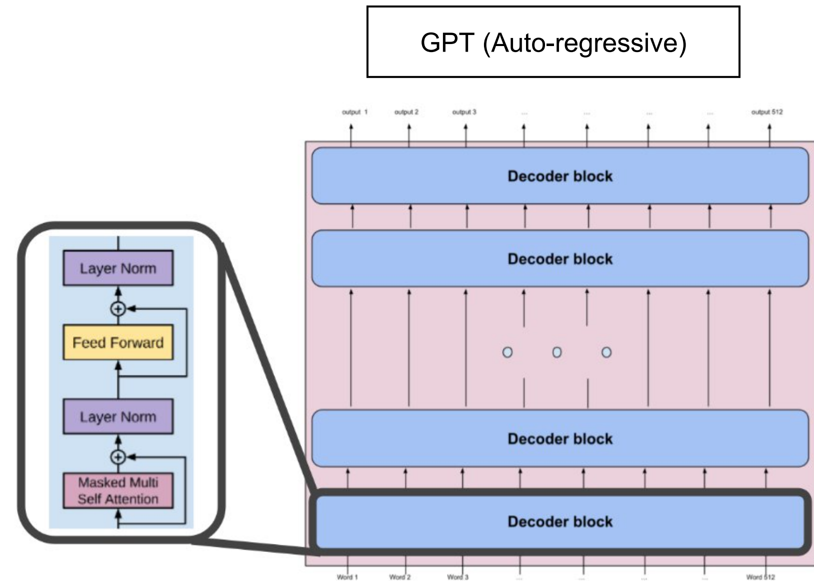


Mécanisme d'attention (intuition)



BERT vs GPT

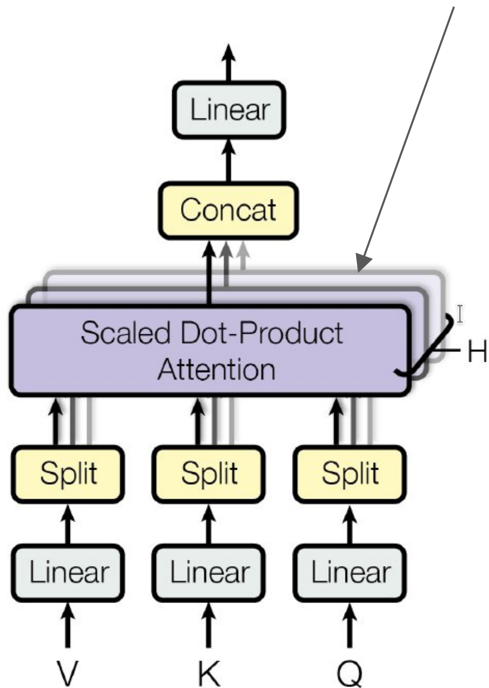


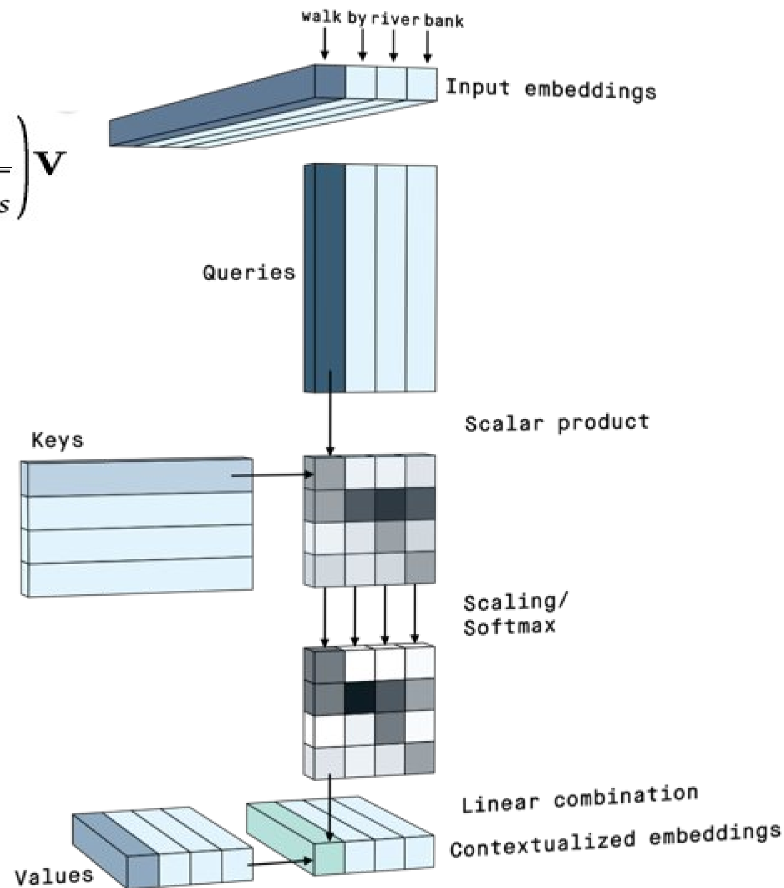


Le mécanisme de Self-Attention

Scaled Dot-Product Attention

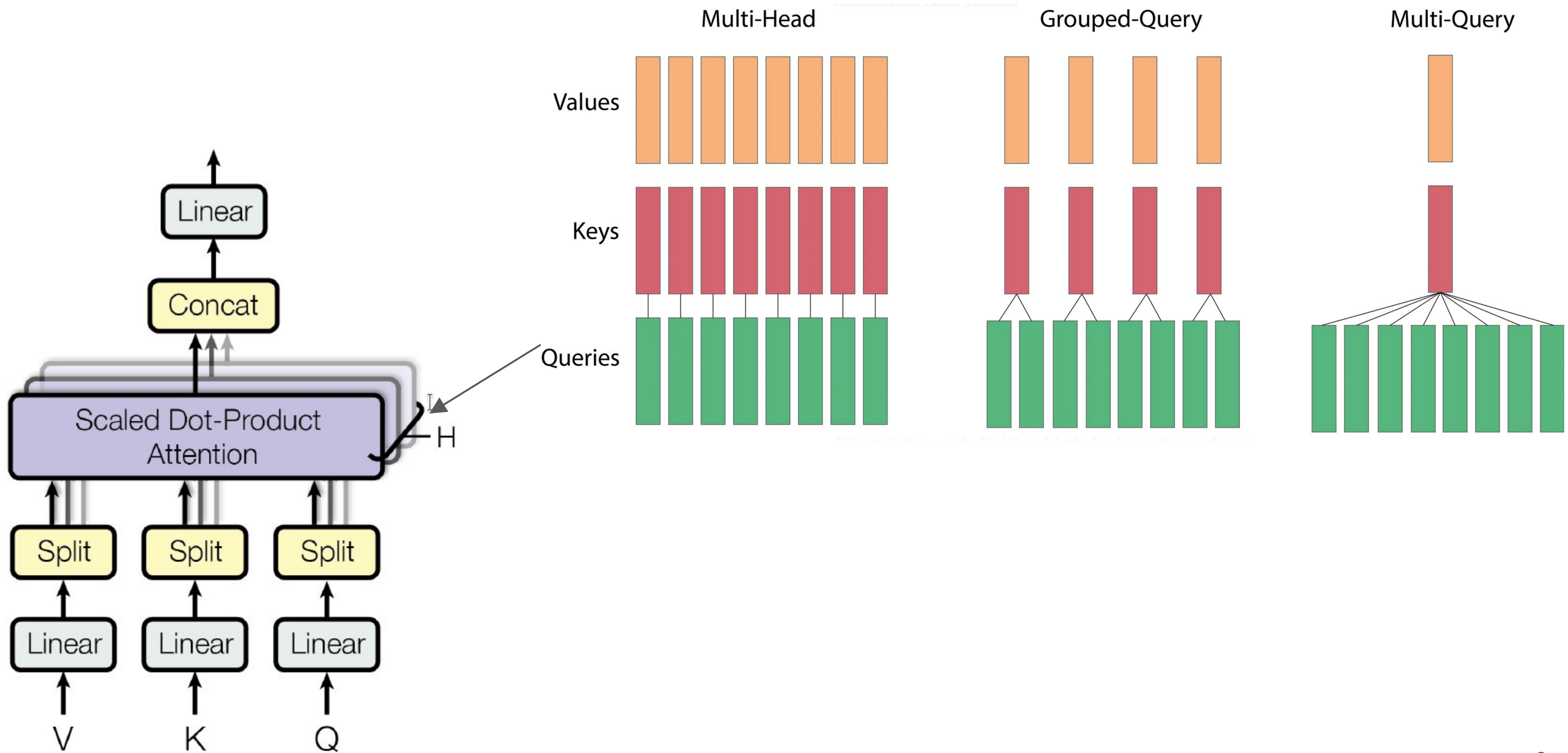
$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_{keys}}}\right)\mathbf{V}$$





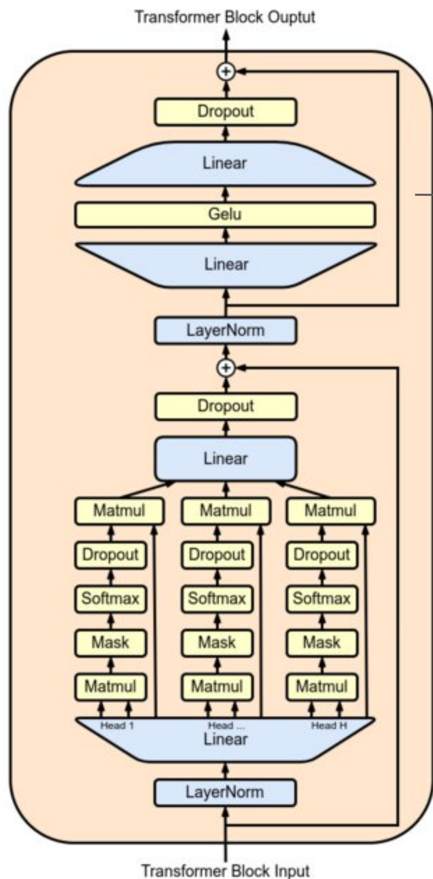
[10, 11]

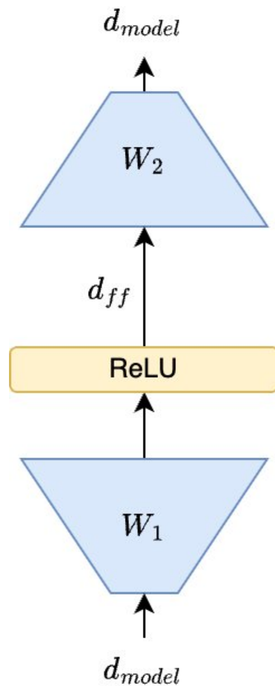
Le mécanisme de Multi-Head Attention



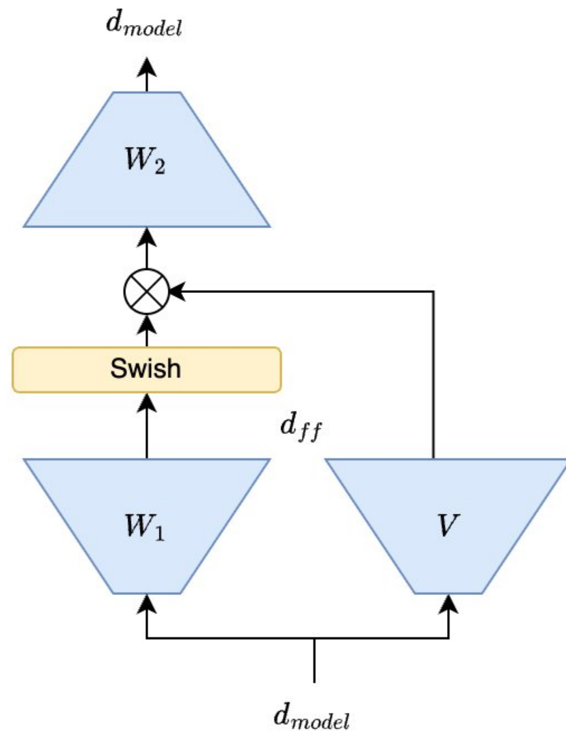
[10, 11]

Le mécanisme de Feed Forward



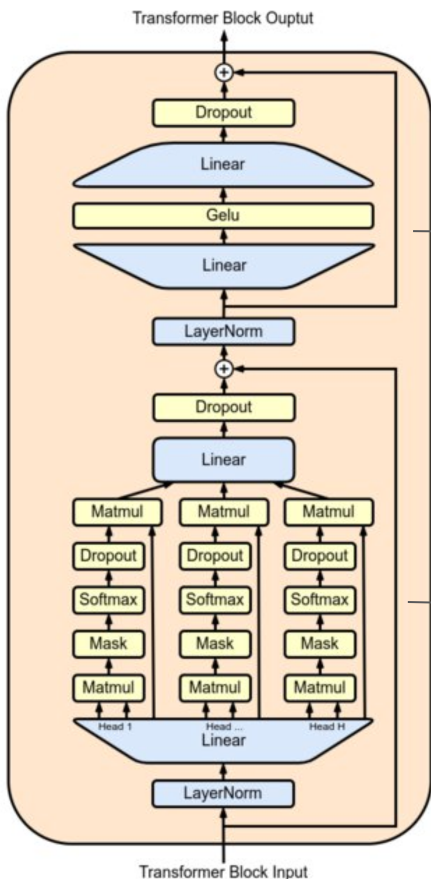


Original Feed Forward Layer



Feed Forward with SwiGLU

Exemple de gros modèles : Llama 3.3-70B



$\text{output_proj} = \texttt{vocab_size} \times \texttt{hidden_size}$
$= 128256 \times 8192 \approx \mathbf{1.05B}$

$\text{proj1: } \texttt{hidden_size} \times \texttt{intermediate_size} = 8192 \times 28672$
$\text{gate1: } \texttt{hidden_size} \times \texttt{intermediate_size} = 8192 \times 28672$
$\text{proj2: } \texttt{intermediate_size} \times \texttt{hidden_size} = 28672 \times 8192$
$3 \times (8192 \times 28672) \approx 704\text{M x } 80 \text{ layers} \approx \mathbf{56B}$

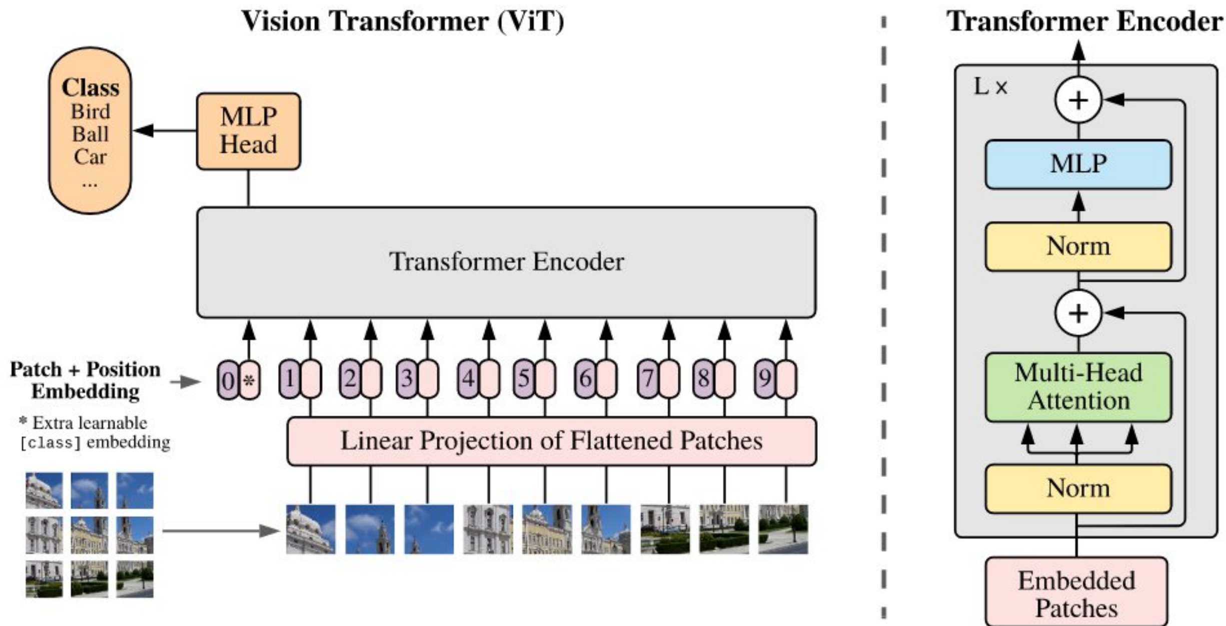
$\text{Q: } \texttt{hidden_size} \times \texttt{hidden_size} = 8192 \times 8192$
$\text{K, V: } \texttt{hidden_size} \times (\texttt{num_key_value_heads} \times \texttt{head_dim})$
$= 8192 \times (8 \times 128) = 8192 \times 1024$
$\text{Output: } \texttt{hidden_size} \times \texttt{hidden_size} = 8192 \times 8192$
$\text{Q:} 8192 \times 8192 = 67.1\text{M K:} 8192 \times 1024 = 8.4\text{M}$
$\text{V:} 8192 \times 1024 = 8.4\text{M Output:} 8192 \times 8192 = 67.1\text{M}$
$\text{TotalAttention: } \approx 151\text{M x } 80 \text{ layers} \approx \mathbf{12B}$

$\text{token_embedding} = \texttt{vocab_size} \times \texttt{hidden_size}$
$= 128256 \times 8192 \approx \mathbf{1.05B}$

meta-llama/Llama-3.3-70B-Instruct


```
{
  "architectures": [
    "LlamaForCausalLM"
  ],
  "attention_bias": false,
  "attention_dropout": 0.0,
  "bos_token_id": 128000,
  "eos_token_id": [
    128001,
    128008,
    128009
  ],
  "head_dim": 128,
  "hidden_act": "silu",
  "hidden_size": 8192,
  "initializer_range": 0.02,
  "intermediate_size": 28672,
  "max_position_embeddings": 131072,
  "mlp_bias": false,
  "model_type": "llama",
  "num_attention_heads": 64,
  "num_hidden_layers": 80,
  "num_key_value_heads": 8,
  "pretraining_tp": 1,
  "rms_norm_eps": 1e-05,
  "rope_scaling": {
    "factor": 8.0,
    "high_freq_factor": 4.0,
    "low_freq_factor": 1.0,
    "original_max_position_embeddings": 8192,
    "rope_type": "llama3"
  },
  "rope_theta": 500000.0,
  "tie_word_embeddings": false,
  "torch_dtype": "bfloat16",
  "transformers_version": "4.47.0.dev0",
  "use_cache": true,
  "vocab_size": 128256
}
```

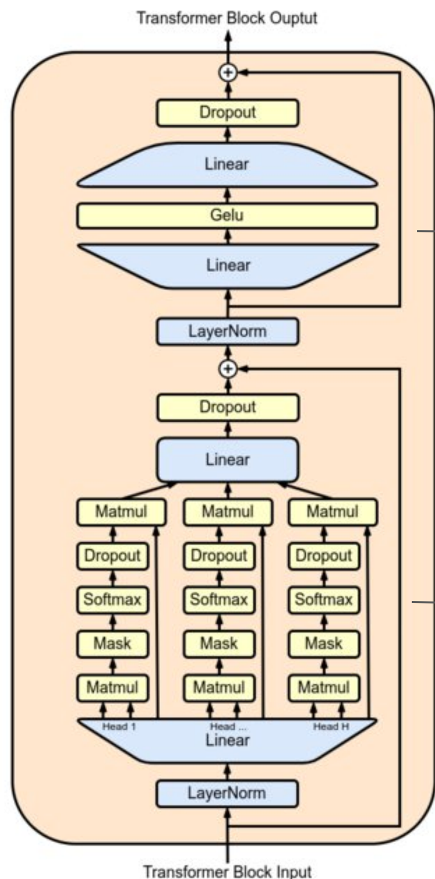
Vision Transformer (ViT)



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

[12]

Exemple de gros modèles : ViT-G-1.84B



proj1: `hidden_size × intermediate_size` = 1664× 8192
proj2: `intermediate_size × hidden_size` = 8192× 1664
2×(1664×8192) ≈ 27M x 48 layers ≈ **1.3B**

Q, K, V: `hidden_size × hidden_size` = 1664×1664
Output: `hidden_size × hidden_size` = 1664×1664
Q, K, V:=2.7M Output:=2.7M
TotalAttention: ≈ 11M x 48 layers ≈ **531M**

patch_embedding = `(patch_size x patch_size) × hidden_size`
= (14x14)×1664 ≈ **326K**

```
"vision_config_dict": {
  "hidden_act": "gelu",
  "hidden_size": 1664,
  "intermediate_size": 8192,
  "num_attention_heads": 16,
  "num_hidden_layers": 48,
  "patch_size": 14,
  "projection_dim": 1280
}
}
```

CoAtNet

“Marrying Convolution and Attention for All Data Sizes”

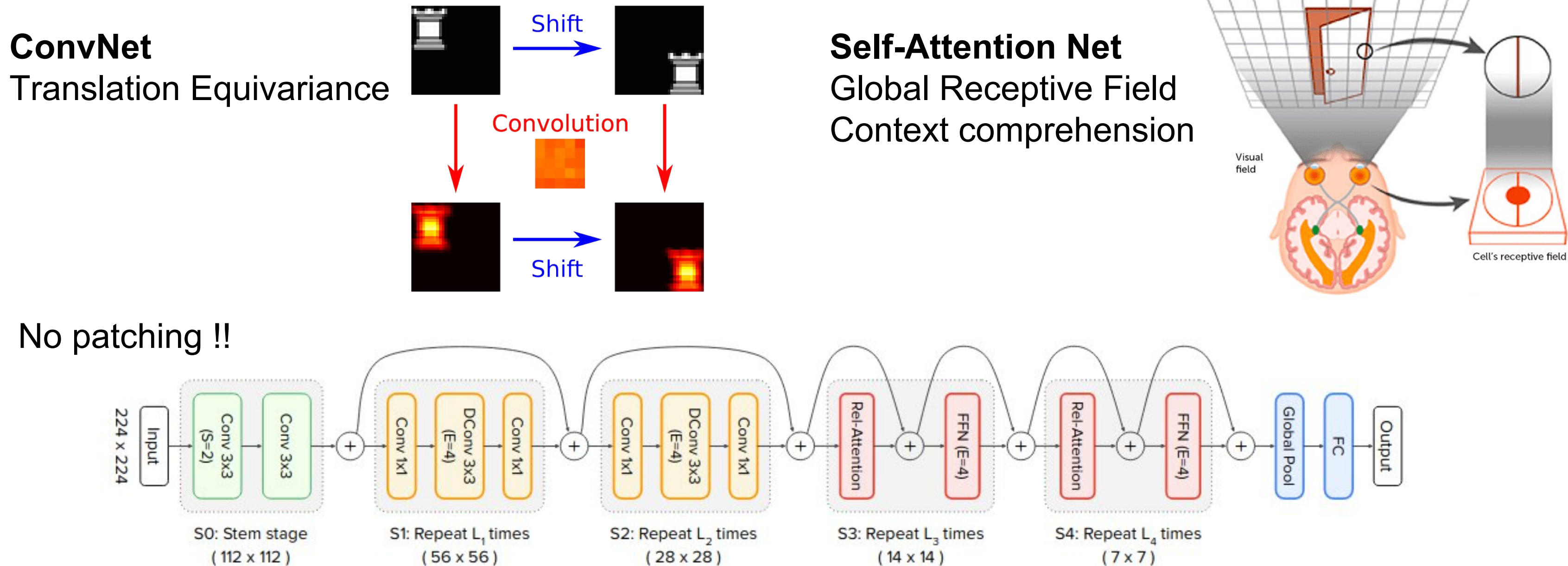


Figure 4: Overview of the proposed CoAtNet.

[13, 14, 15]

CoAtNet - Tailles

Models	Eval Size	#Params	#FLOPs	TPUv3-core-days	Top-1 Accuracy
ResNet + ViT-L/16	384^2	330M	-	-	87.12
ViT-L/16	512^2	307M	364B	0.68K	87.76
ViT-H/14	518^2	632M	1021B	2.5K	88.55
NFNet-F4+	512^2	527M	367B	1.86K	89.2
CoAtNet-3†	384^2	168M	114B	0.58K	88.52
CoAtNet-3†	512^2	168M	214B	0.58K	88.81
CoAtNet-4	512^2	275M	361B	0.95K	89.11
CoAtNet-5	512^2	688M	812B	1.82K	89.77
ViT-G/14	518^2	1.84B	5160B	>30K$^\diamond$	90.45
CoAtNet-6	512^2	1.47B	1521B	6.6K	90.45
CoAtNet-7	512^2	2.44B	2586B	20.1K	**90.88**

[13]

Les Transformers

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

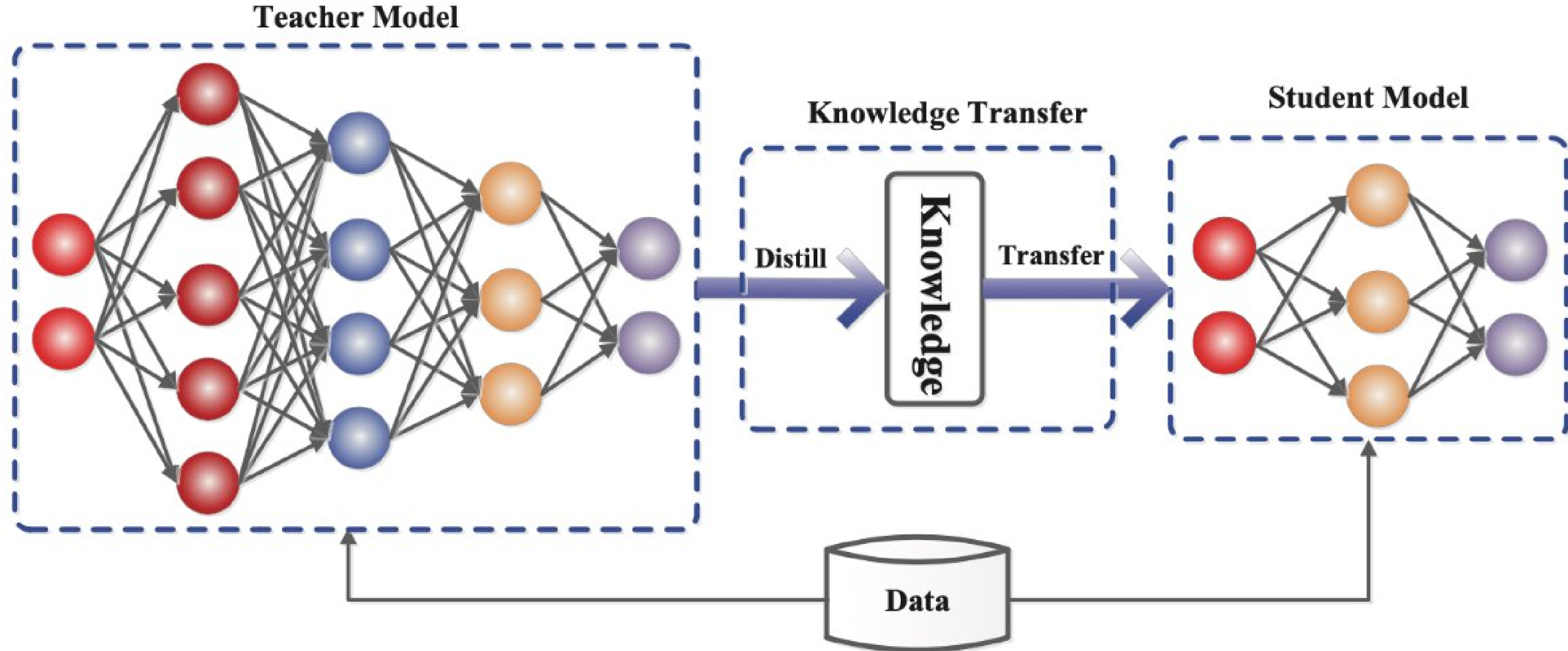
Optimisation de l’inférence des très gros modèles

Distillation ◂

Quantification ◂

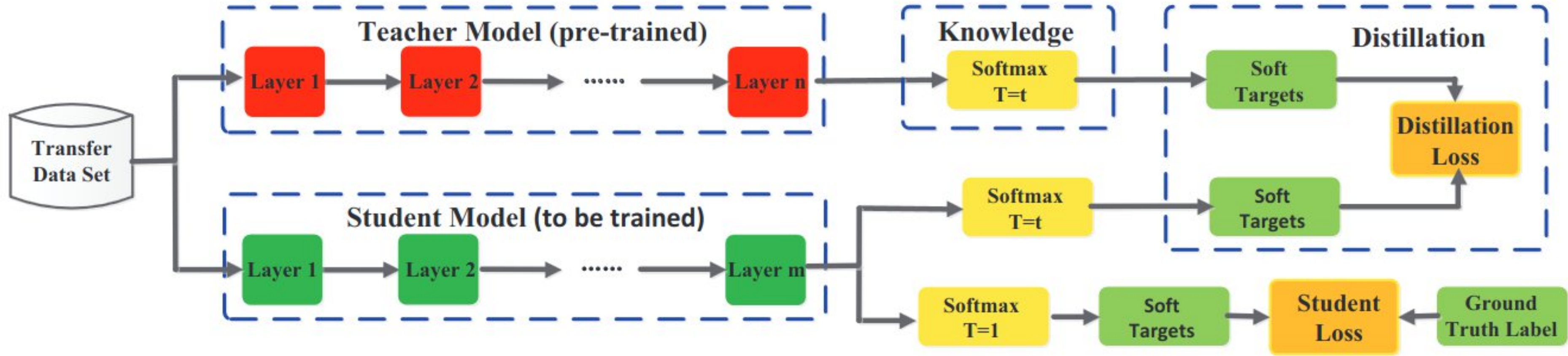
Pruning ◂

Distillation



[18]

Distillation



$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{distil}}\left(y_{\text{teacher}}, y_{\text{student}}\right) + \lambda \mathcal{L}_{\text{CE}}\left(y_{\text{target}}, y_{\text{student}}\right)$$

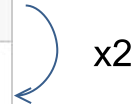
Cross-entropy, Divergence KL, Wasserstein, ...

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*	



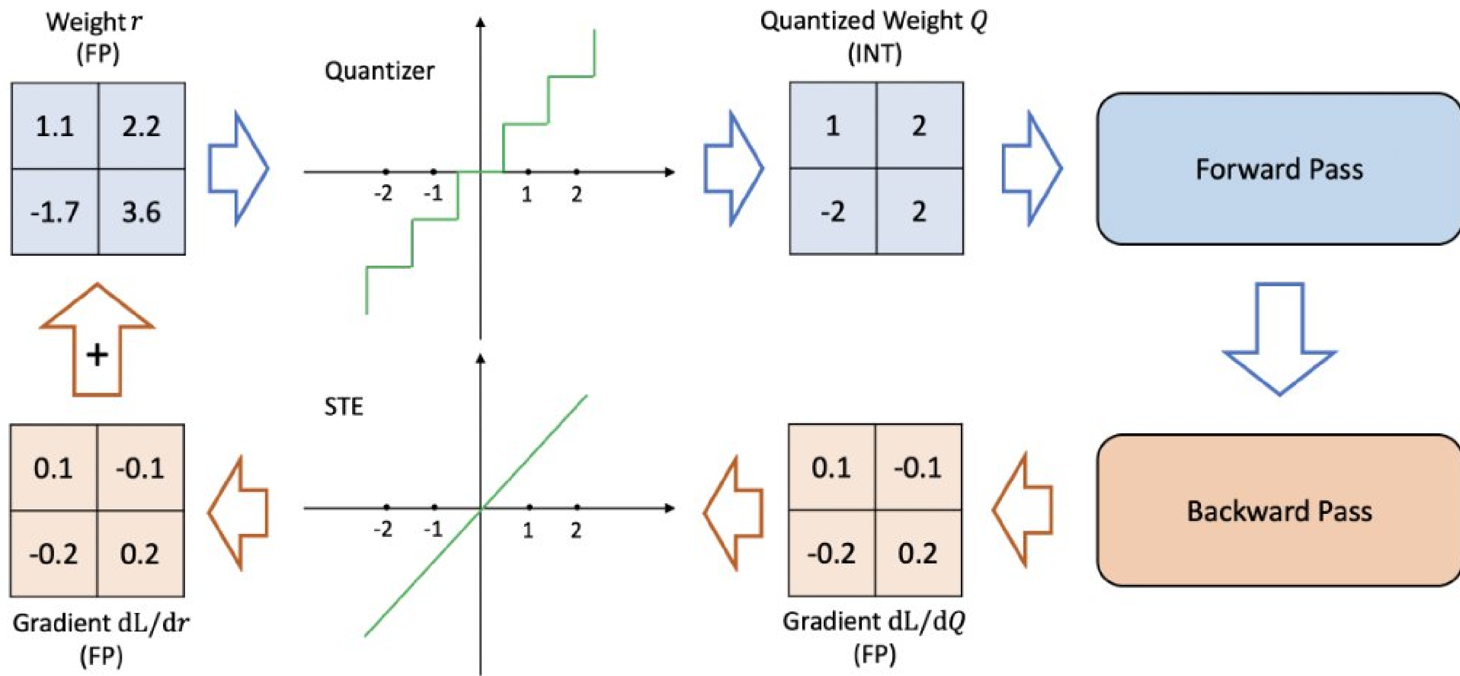
x2

Quantification

cast INT8

Il est possible de rencontrer une perte en performance

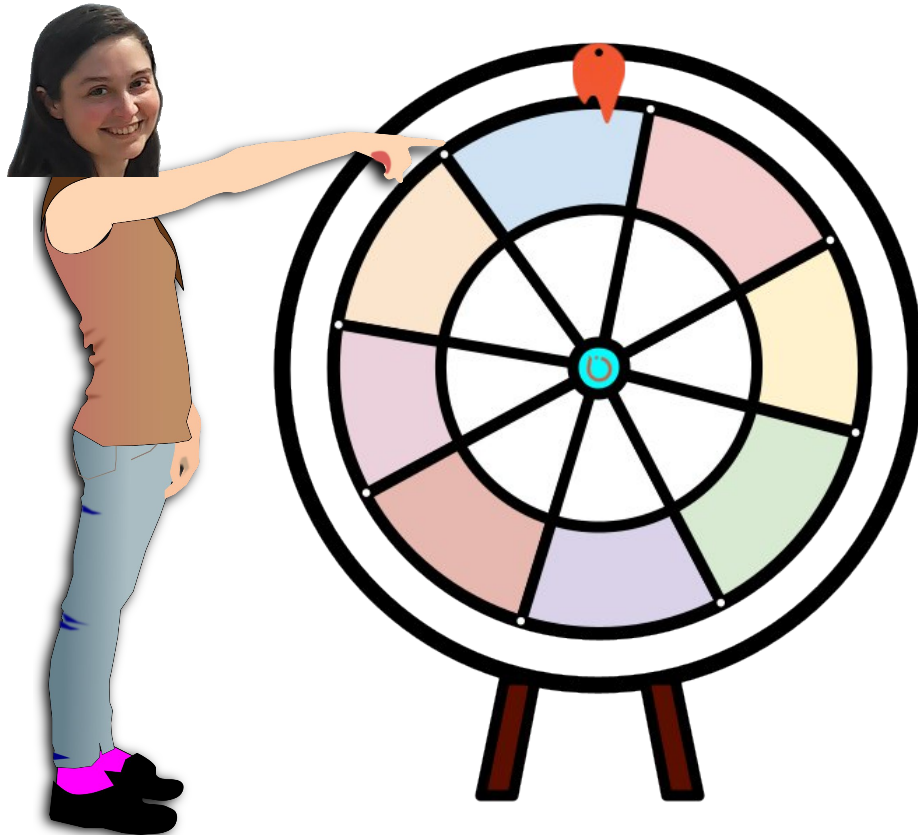
Quantification



Exemple d’entraînement post-quantisation

[19, 23]

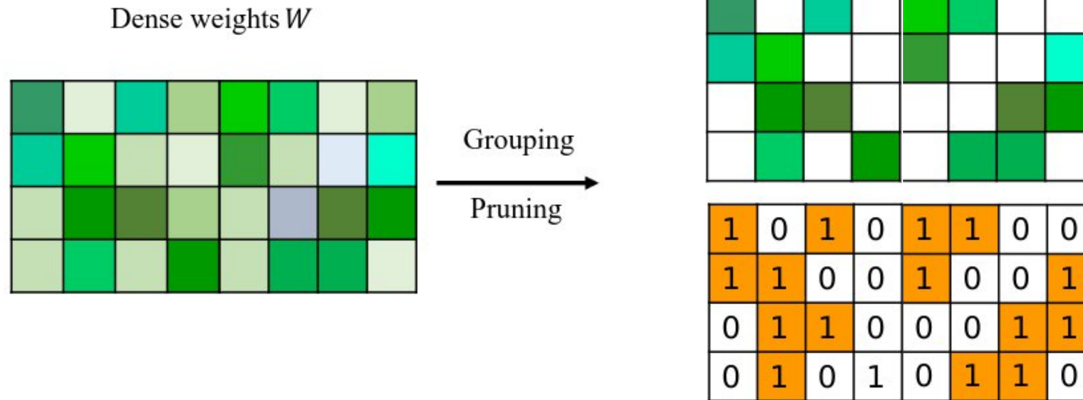
cnrs



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]

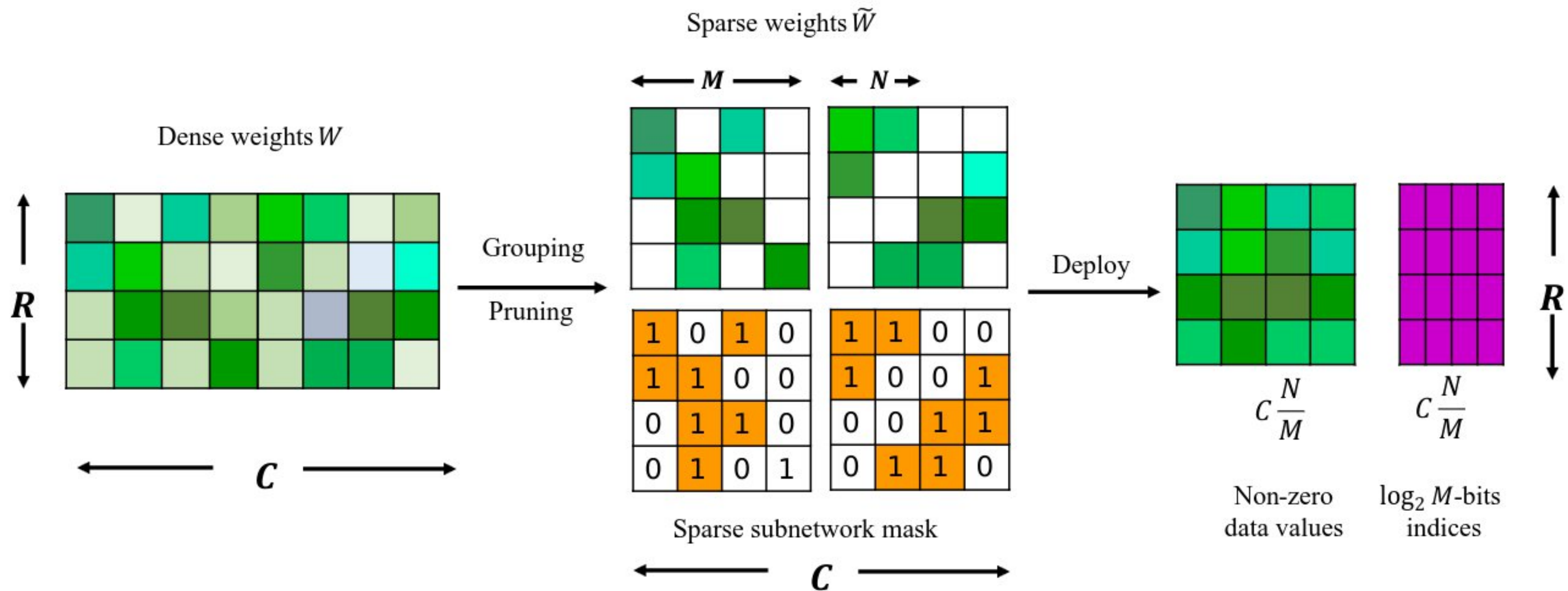
Pruning



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

[20]

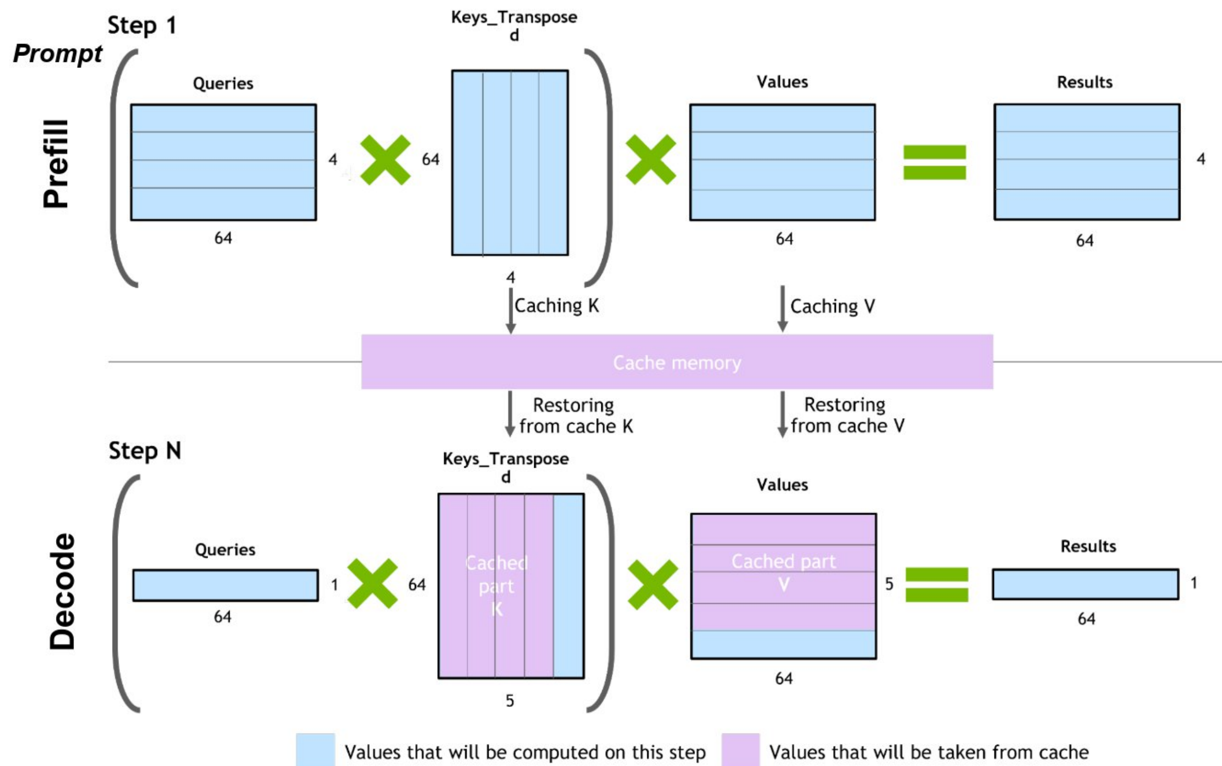
Pruning



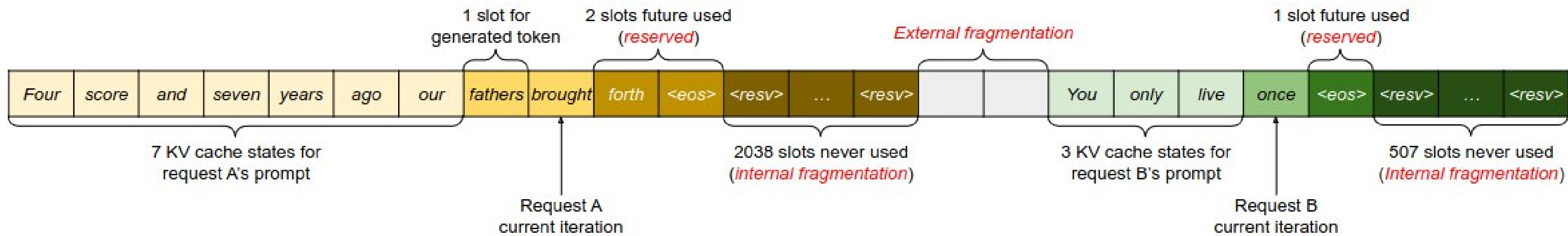
Les Tensor Cores des NVIDIA A100 supportent une dispersion 2:4.

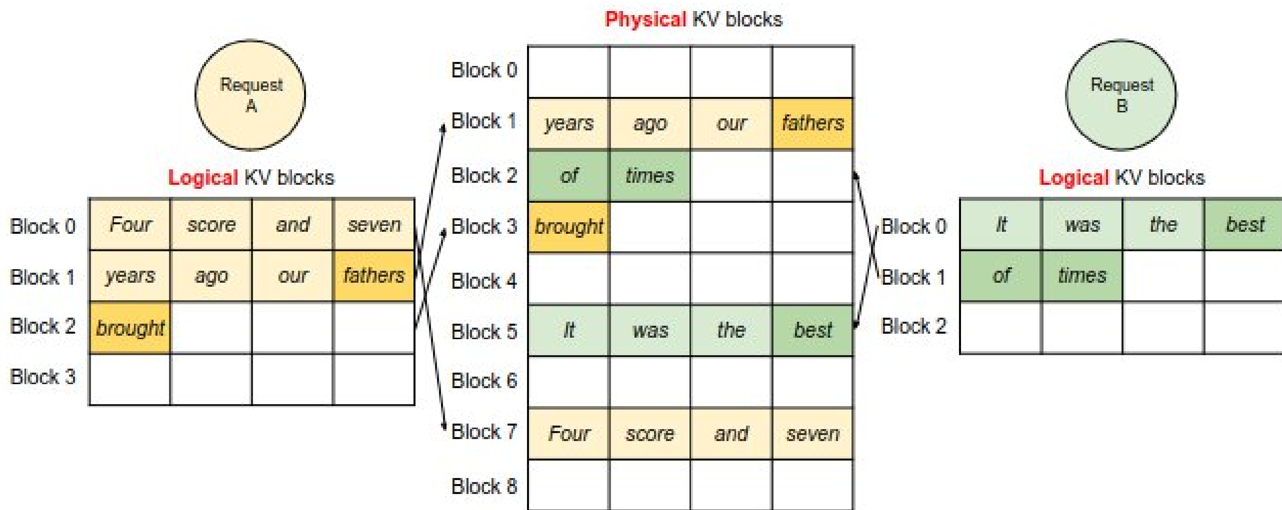
[20, 21]

KV Caching



Paged Attention






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

vLLM

[16]

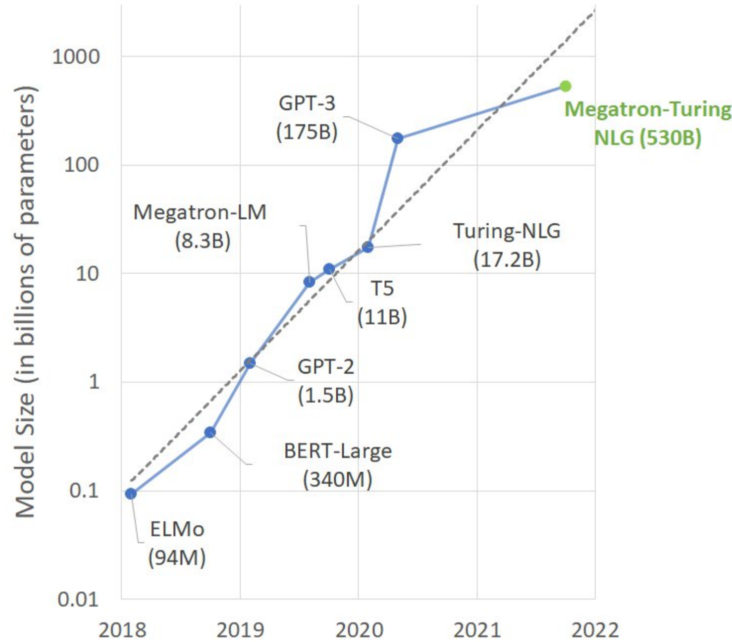
Optimisation du Data Parallelism pour les très gros modèles

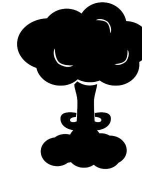
Problématiques des gros modèles ◂

ZeRO & Fully Sharded Data Parallelism ◂

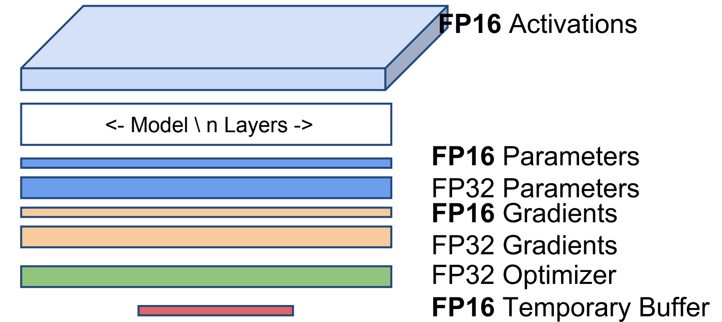
Énormes modèles > 1 Milliard de paramètres

NLP Transformers





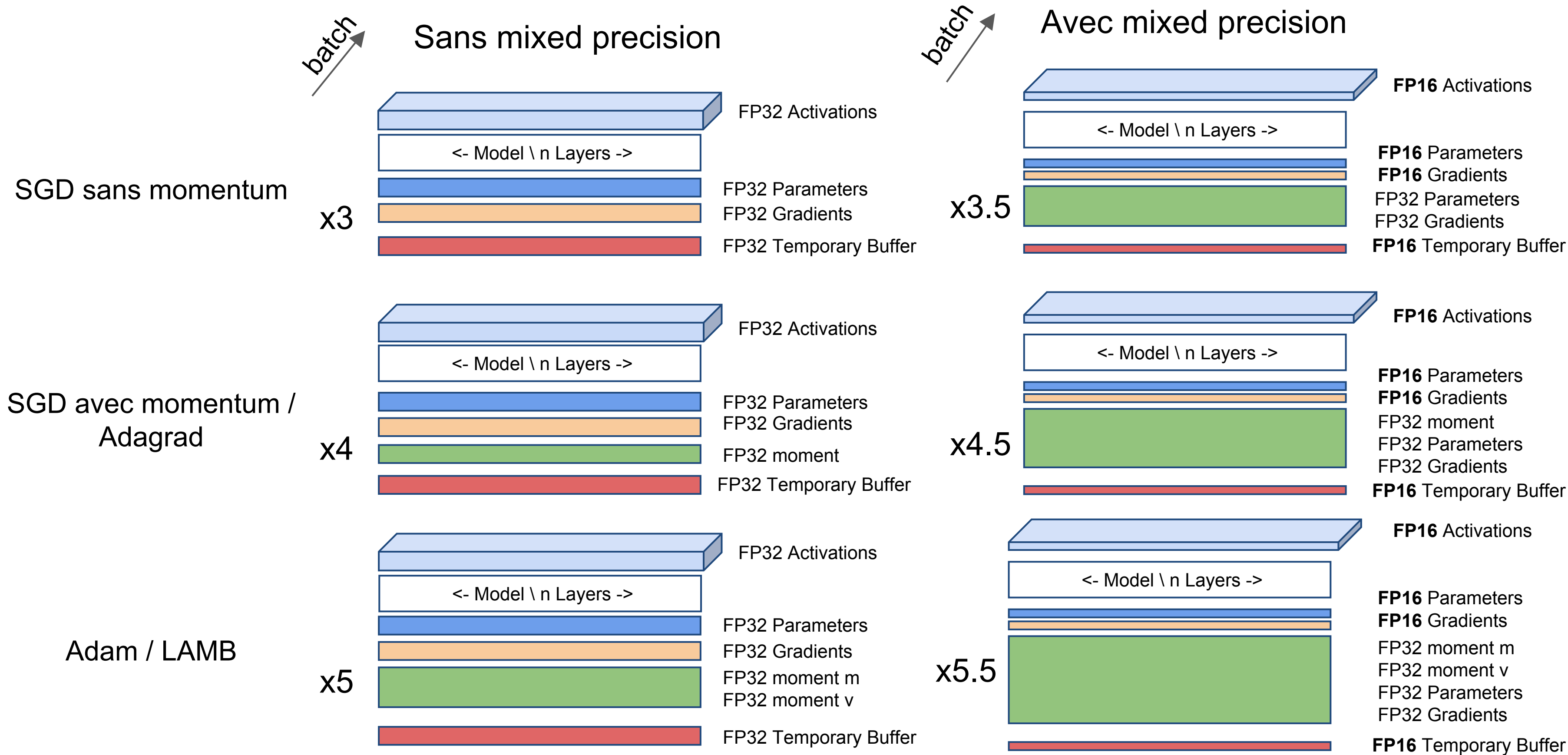
CUDA Out Of Memory



~x4 huge model

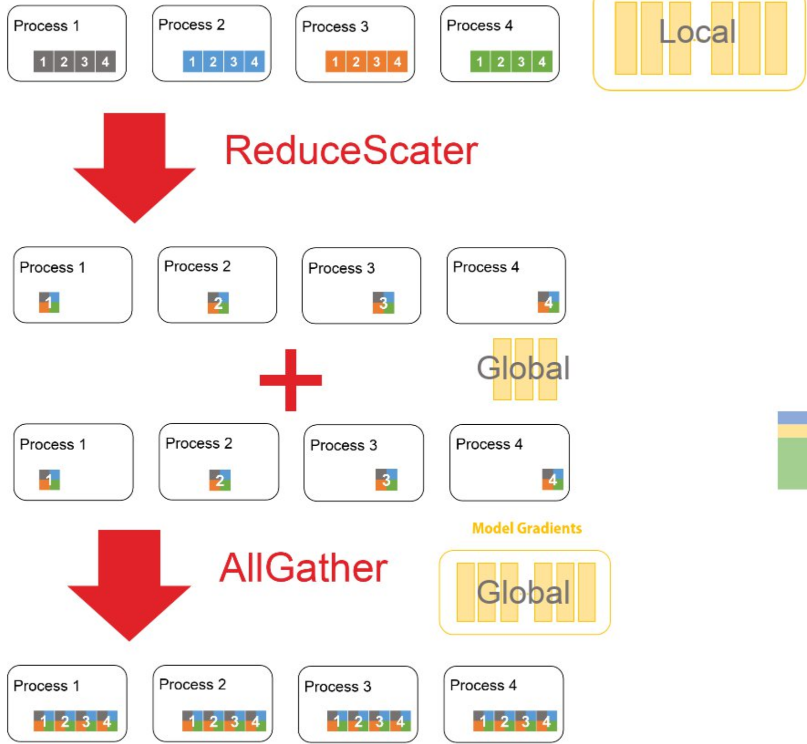
[1]

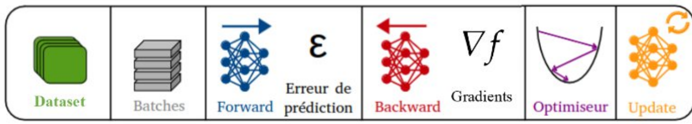
Empreinte mémoire

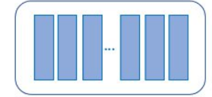


Rappel du Distributed Data Parallelism

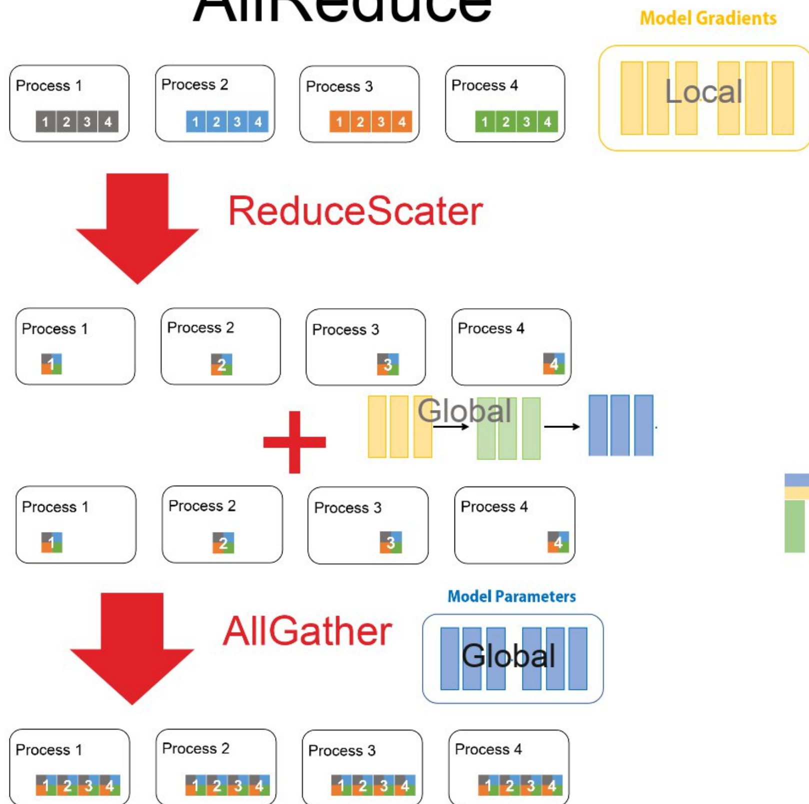
ZeRO-0 / DDP Baseline

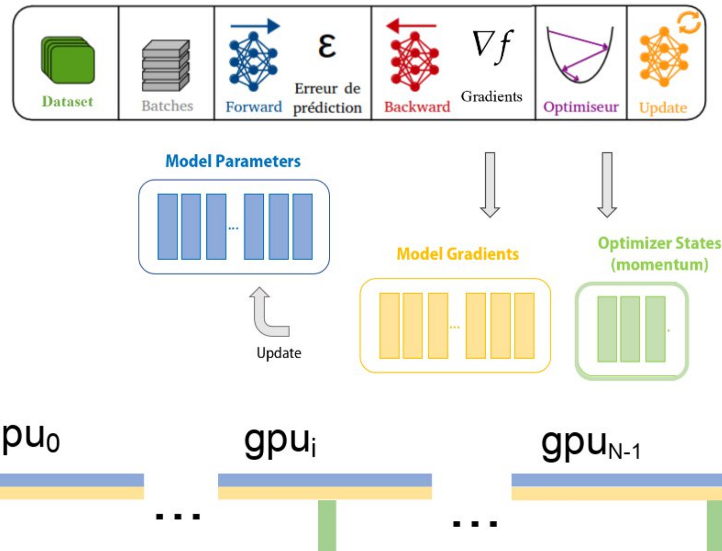




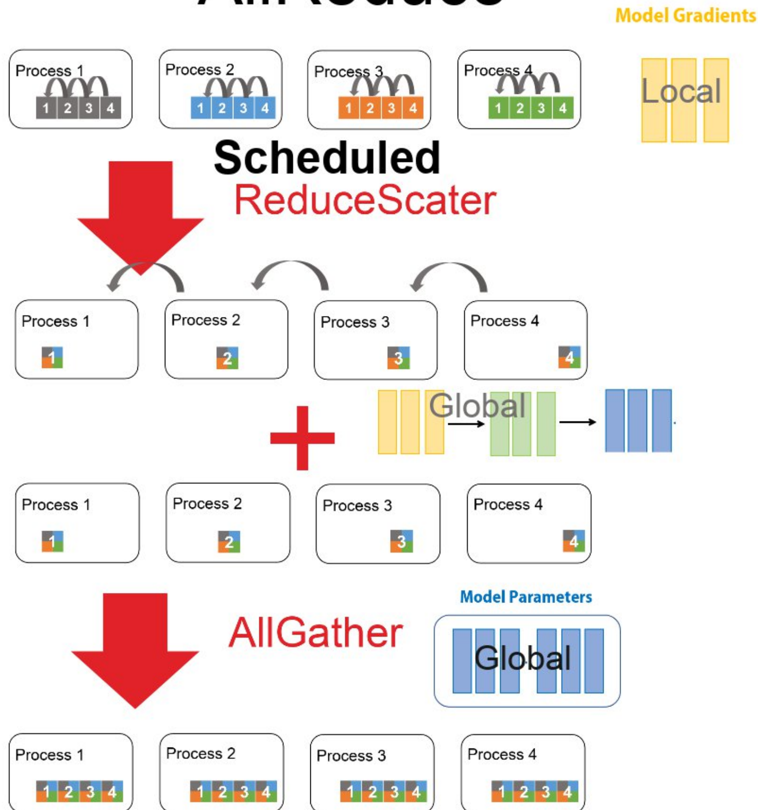


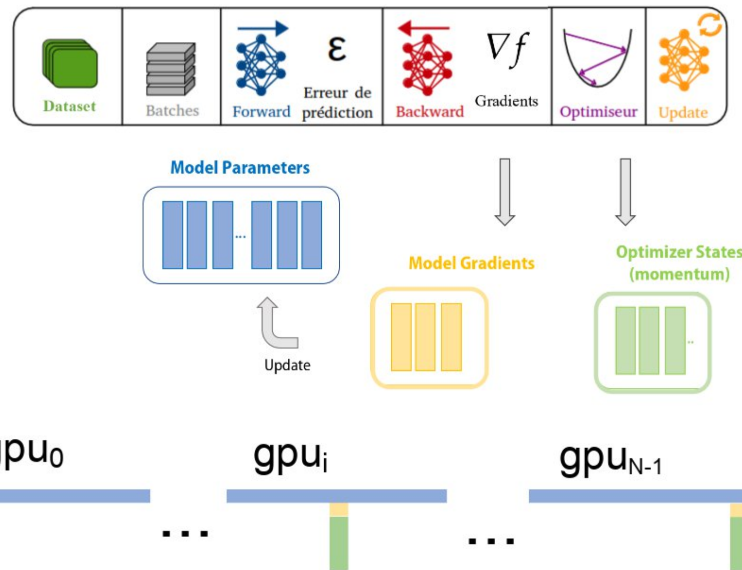
ZeRO-1



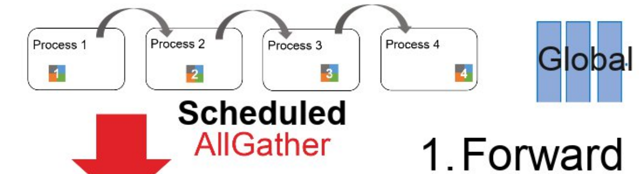


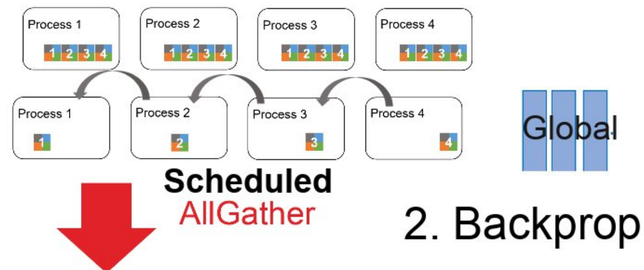
ZeRO-2

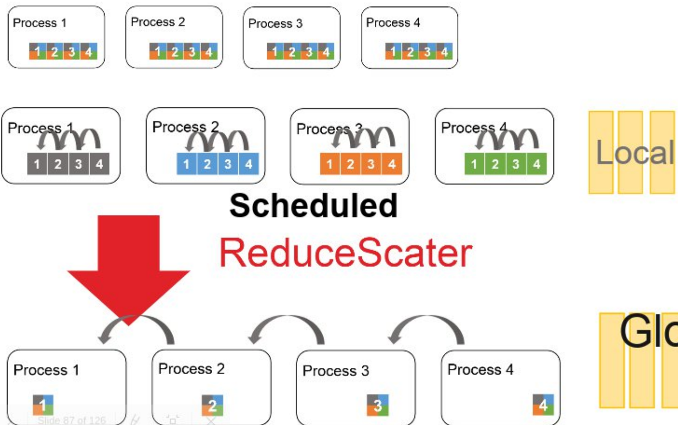


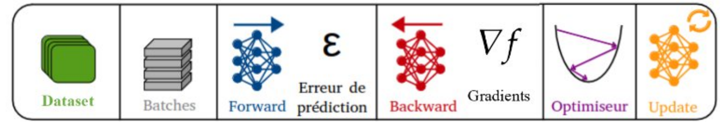


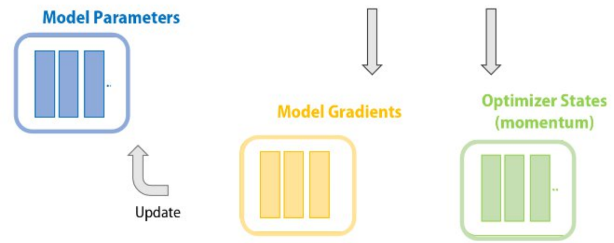
ZeRO-3 / FSDP

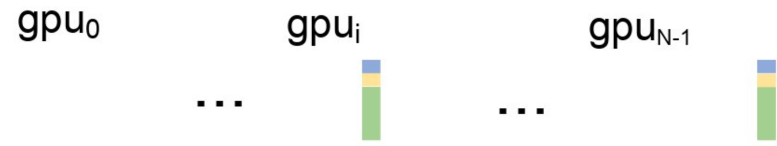




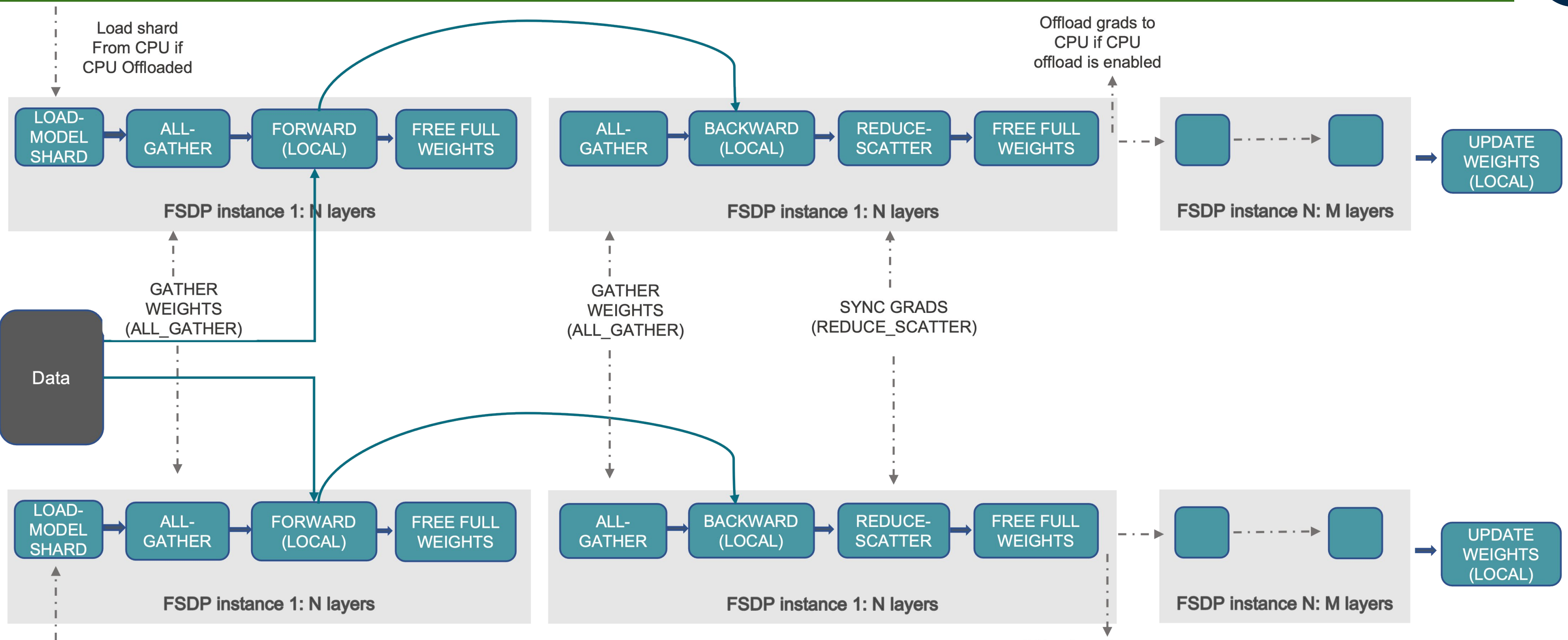








Fully Sharded Data Parallel (Pytorch)




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

[29]

TP : Fully Sharded Data Parallelism

Allez dans le répertoire *tp_fsdp* dans le répertoire *Jour4* du dépôt du cours et suivez le notebook.

- Limite du *Data Parallelism* avec Llama 3.2 3B
- Implémenter le Fully Sharded Data Parallelism
- Optimisation de la FSDP
- Utilisation de torch.compile sur un module FSDP

Optimisation de l’apprentissage des très gros modèles

Pipeline parallelism ◀

Tensor parallelism ◀

Hybrid parallelism ◀

3D parallelism ◀

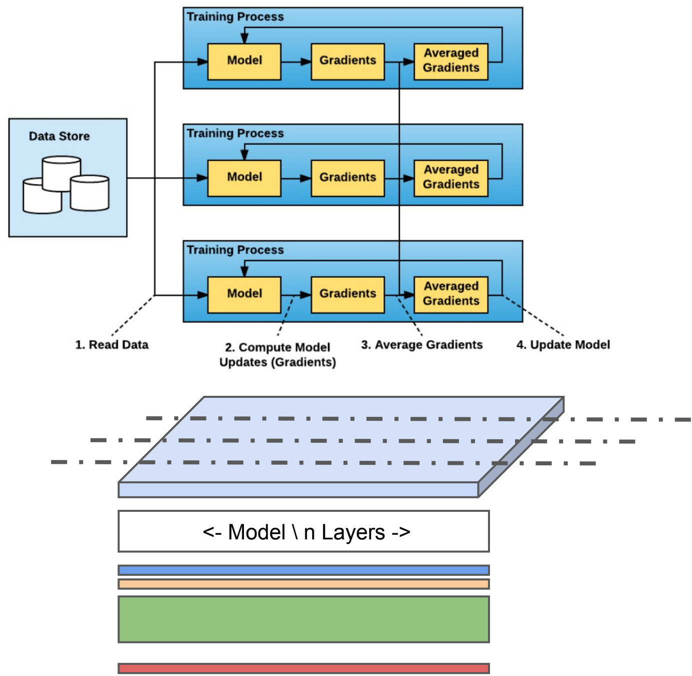
Les différents parallélismes

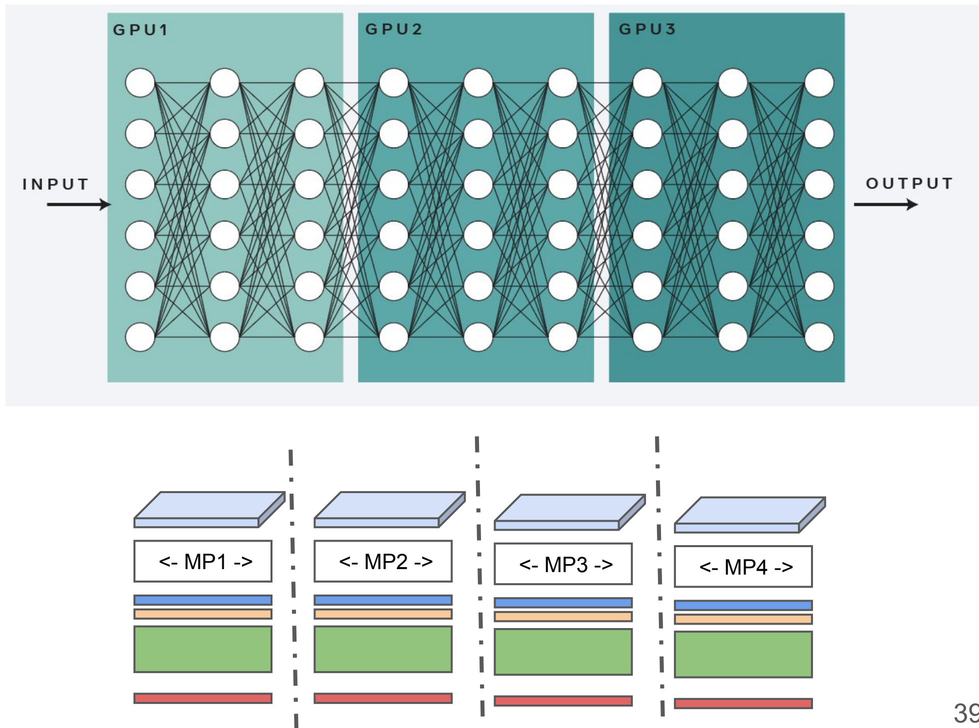
Data Parallelism

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

Pipeline Model Parallelism

- Empreinte mémoire distribuée
- Mono ou multi-processing

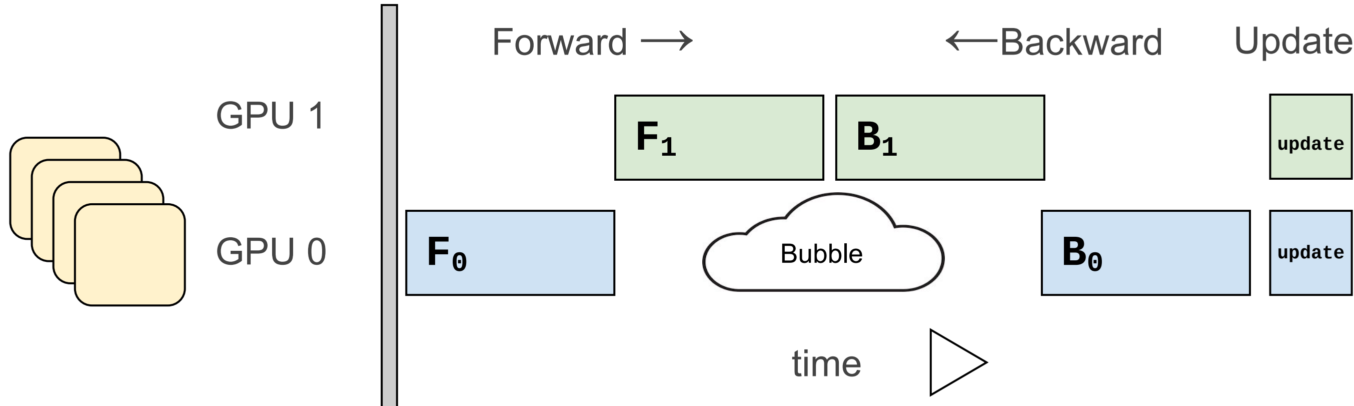


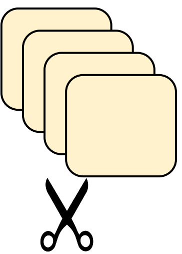


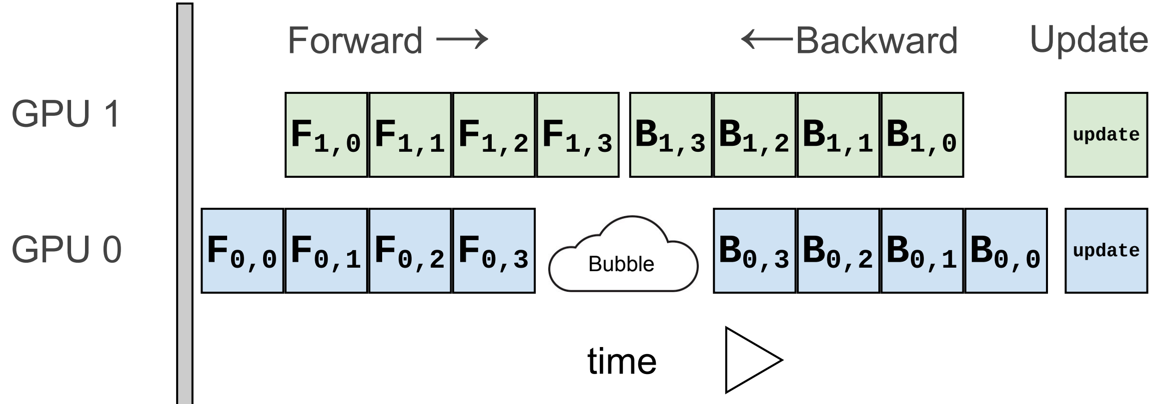
[2, 3]

Pipeline Parallelism

Model parallelism naïf sur 2 GPU





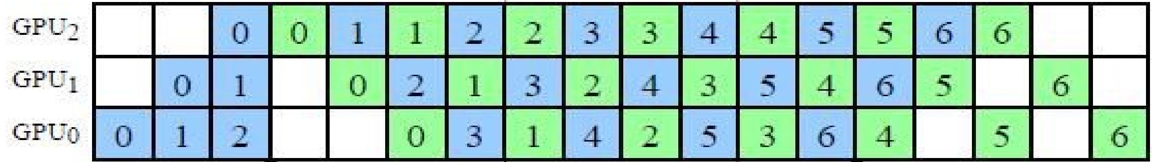


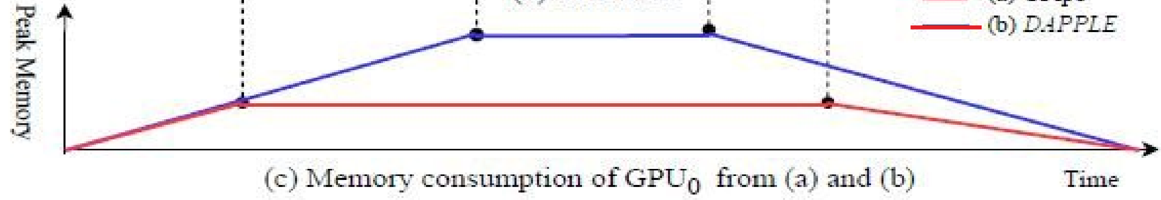
Model parallelism sur 2 GPU en **pipeline**

Pipeline Parallelism

AFAB

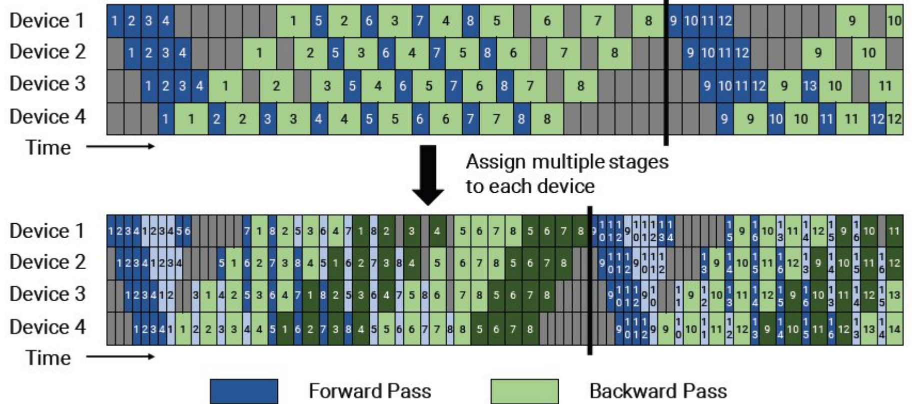
1F1B





[4, 5]

Pipeline Parallelism



Zero Bubble

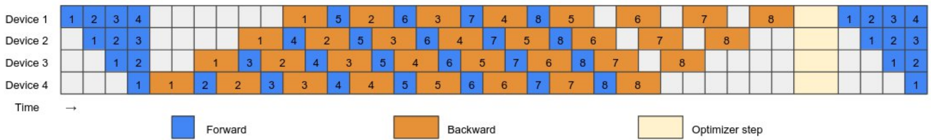


Figure 2: 1F1B pipeline schedule.

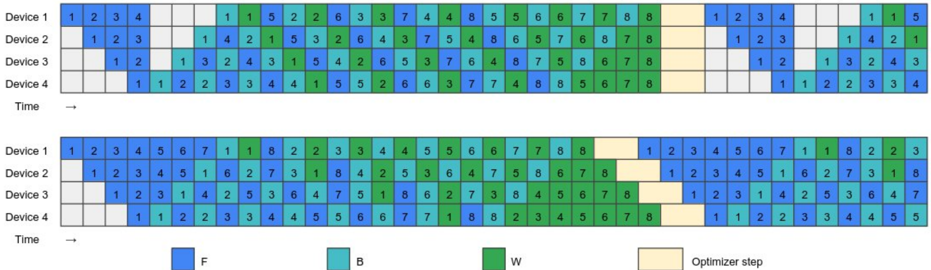
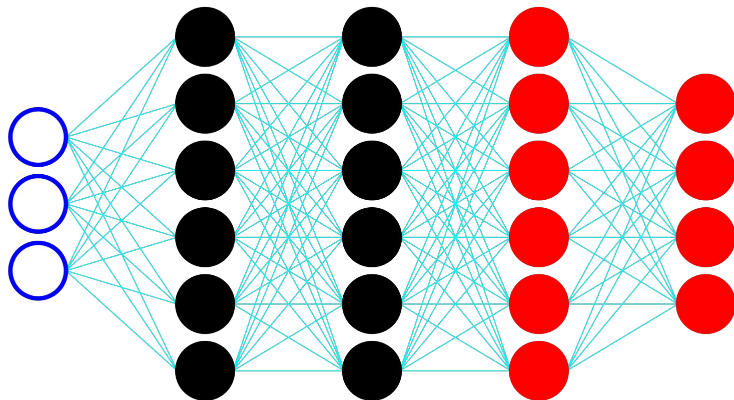


Figure 3: Handcrafted pipeline schedules, top: ZB-H1; bottom: ZB-H2

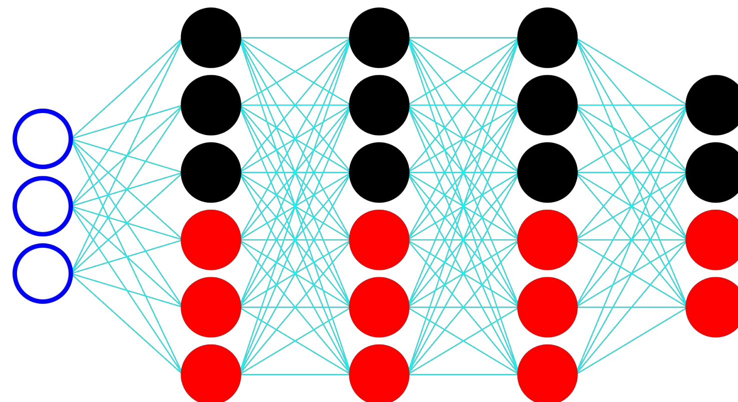
[4, 5]

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

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

[22]

Tensor Parallelism

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

Découpage par colonne

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

GPU 0

GPU 1

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

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

Découpage par colonne

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

AllGather

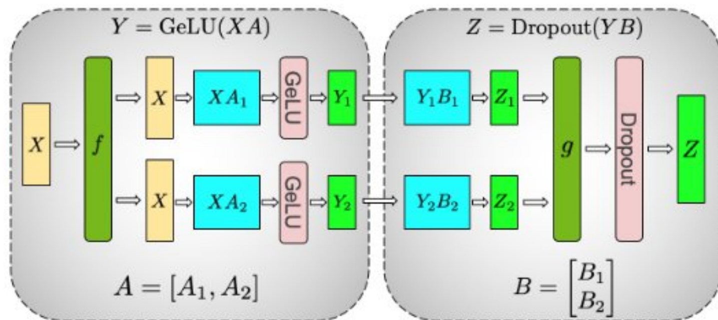
Découpage par ligne

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

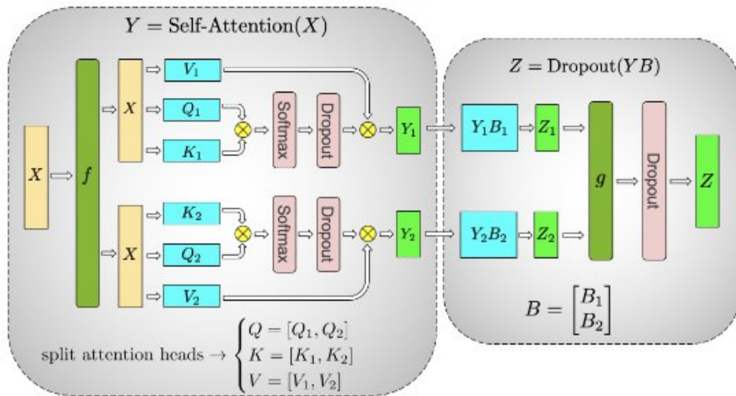
AllReduce

[22]

Tensor Parallelism



(a) MLP



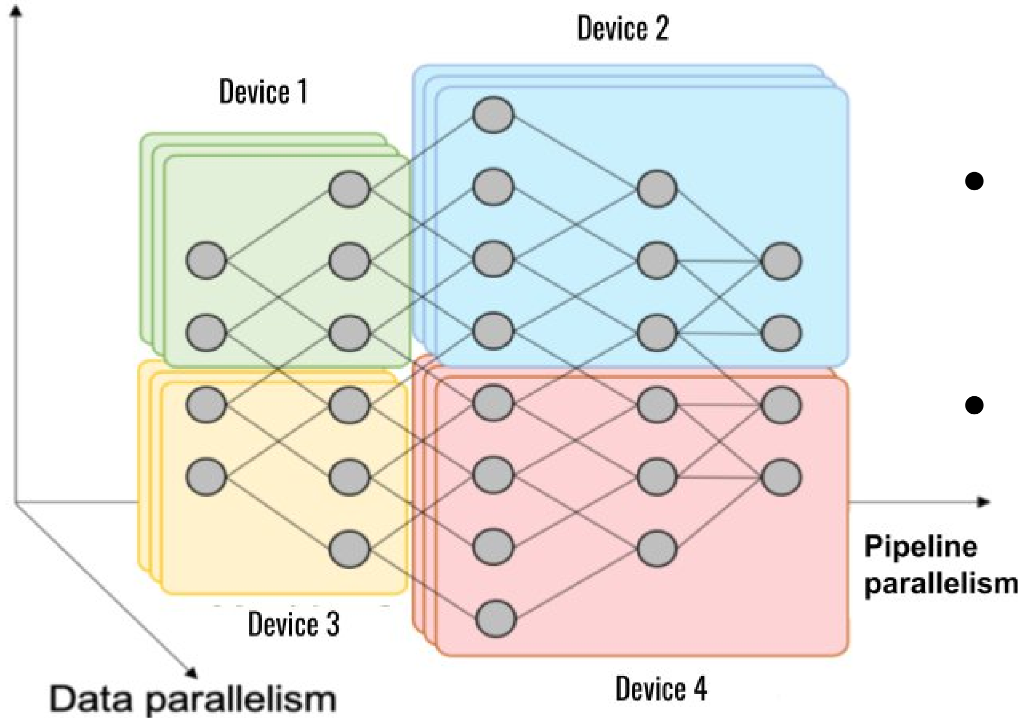
(b) Self-Attention

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 important d'implémentation.
  - Équilibre entre mémoire, performance et convergence.
- **Tensor Parallelism**
  - Effort d'implémentation
  - Bonne accélération des calculs
  - **Bande passante très sollicitée** (implémentation Intra-nœud)

Context Parallelism

Seulement pour les transformers et modèles séquentiels
Semblable au Data Parallelism dans la dimension de la séquence

GPU 0
Lorem ipsum dolor sit

GPU 1
amet, consectetur adipiscing elit

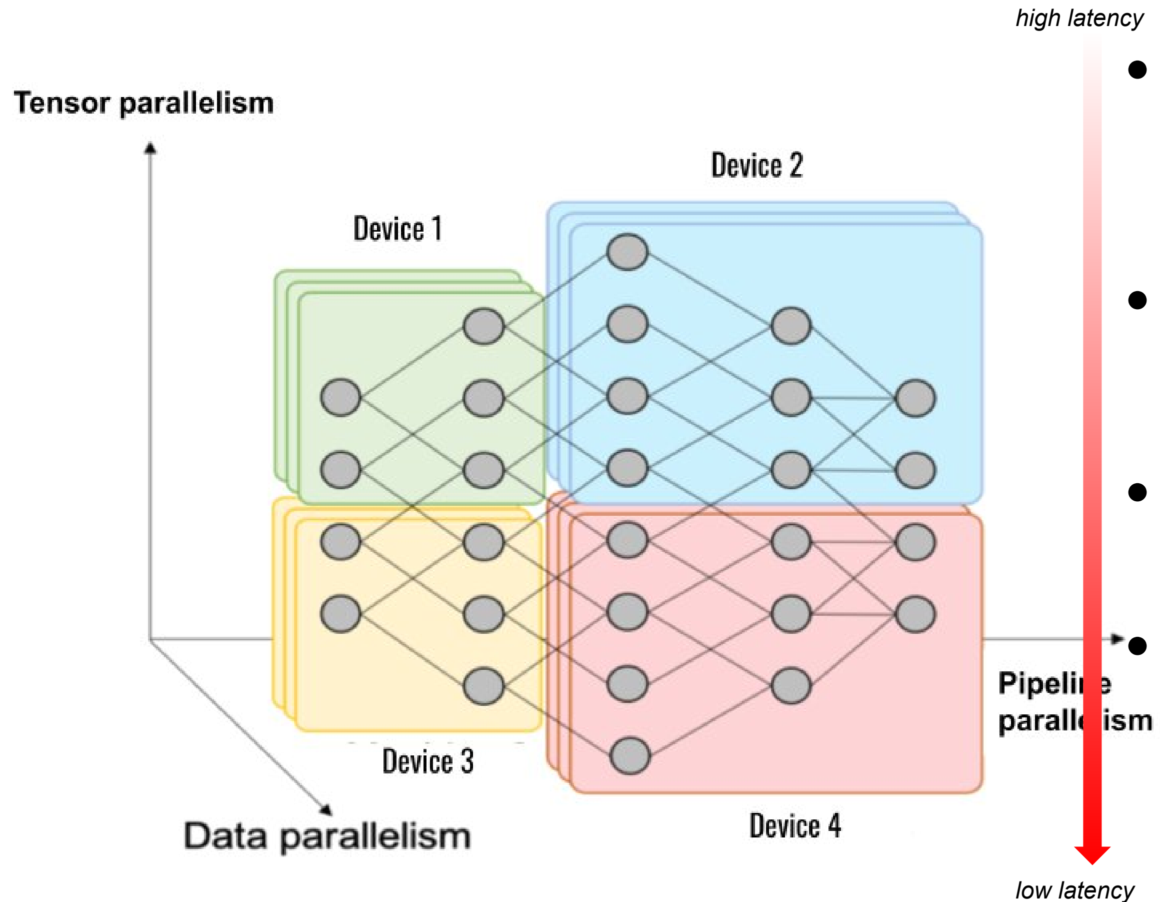
GPU 2
A transformer is a

GPU 3
deep learning architecture which

Data Parallelism

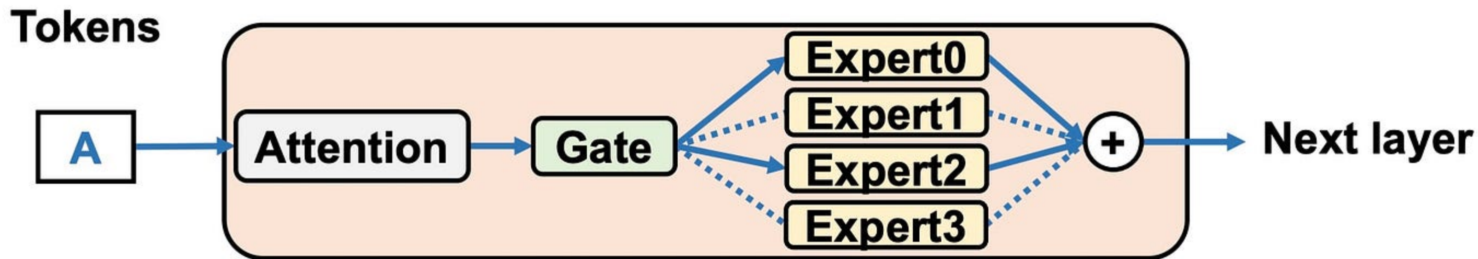
Context Parallelism

4D Parallelism

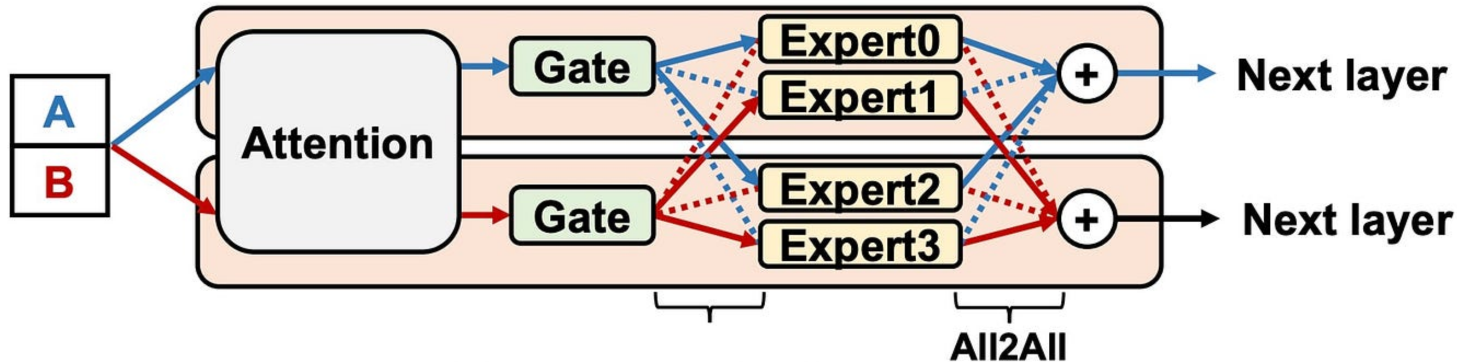


- **Data Parallelism**
  - Simple à implémenter
  - Meilleure performance
  - Augmente la taille du batch (problème de convergence)
- **Pipeline Parallelism**
  - Effort important d'implémentation.
  - Équilibre entre mémoire, performance et convergence.
- **Context parallelism**
  - Uniquement pour transformers
  - Seulement pour **très** longues séquences
- **Tensor Parallelism**
  - Effort d'implémentation
  - Bonne accélération des calculs
  - **Bande passante très sollicitée** (implémentation Intra-nœud)

Expert Parallelism



(a) MoE layer.



(b) Expert parallelism.

API pour les gros modèles

Deepspeed ◂
Megatron-LM ◂
Accelerate, Fabric & vLLM ◂
Native Pytorch◂

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)
```

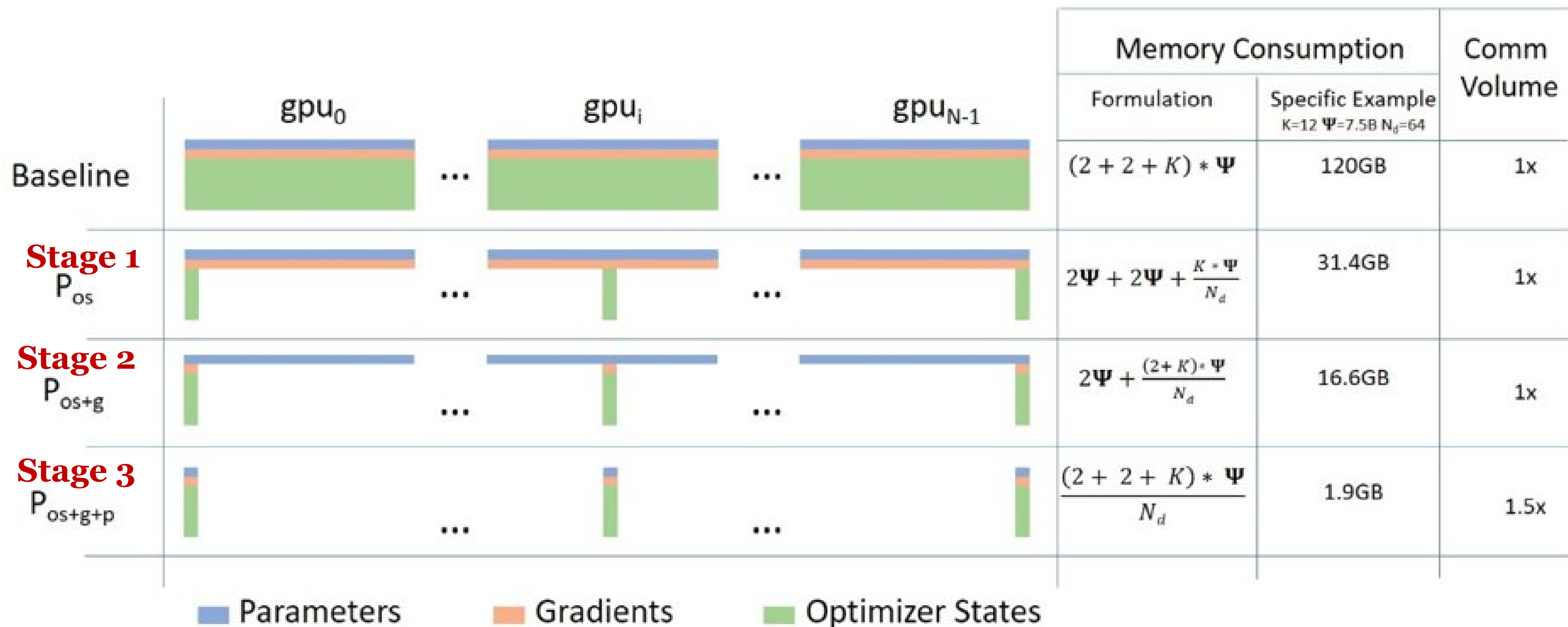
```
# Initialize DeepSpeed to use the following features
# 1) Distributed model
# 2) DeepSpeed optimizer
model_engine, optimizer, _, _ = deepspeed.initialize(
                                args=args, model=model,
                                model_parameters=parameters,
                        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



[6]

Fused optimizers

Implémentations présentent dans *APEX*

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

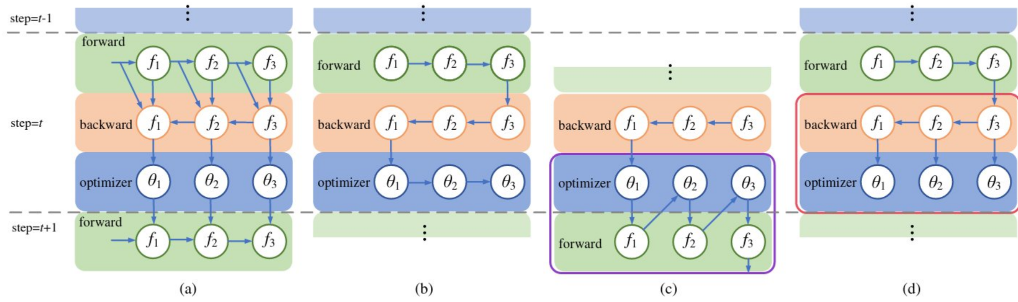
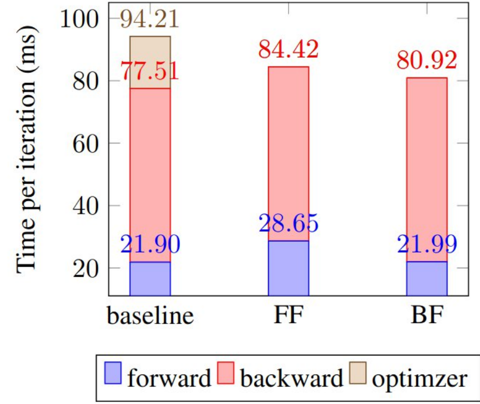


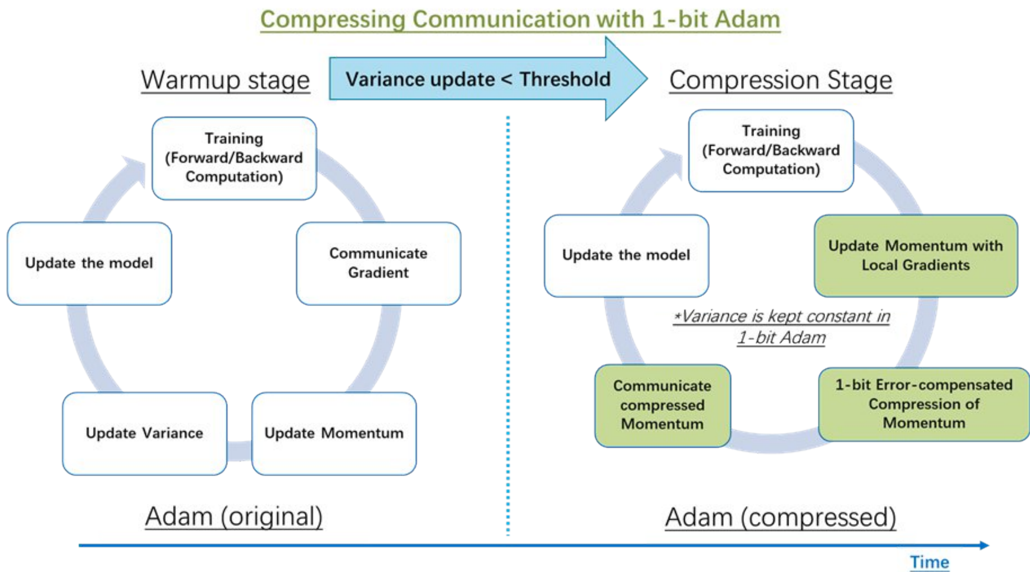
Figure 1: (a) Data dependency graph. (b) Baseline method. (c) `Forward-fusion`. (d) `Backward-fusion`. θ_i represents the trainable parameters in the layer f_i.

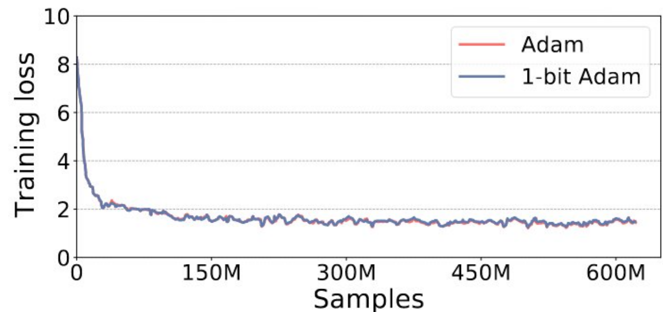


But : accélère l’étape des optimiseurs sur GPU.

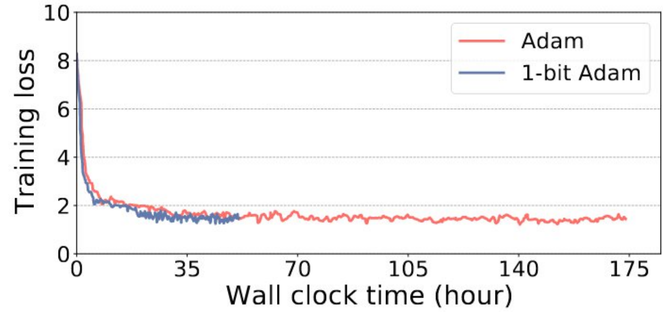
[7]

1-bit optimizers





(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

Megatron-LM

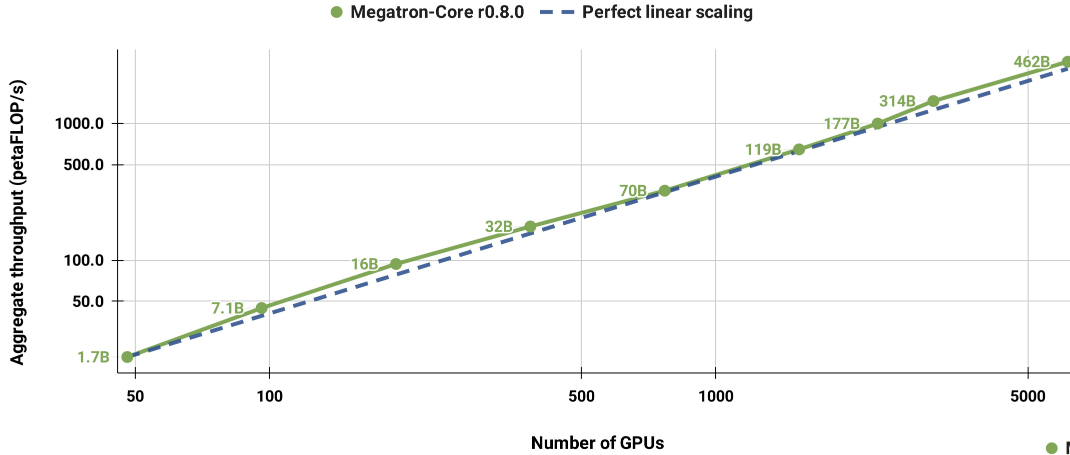
NVIDIA

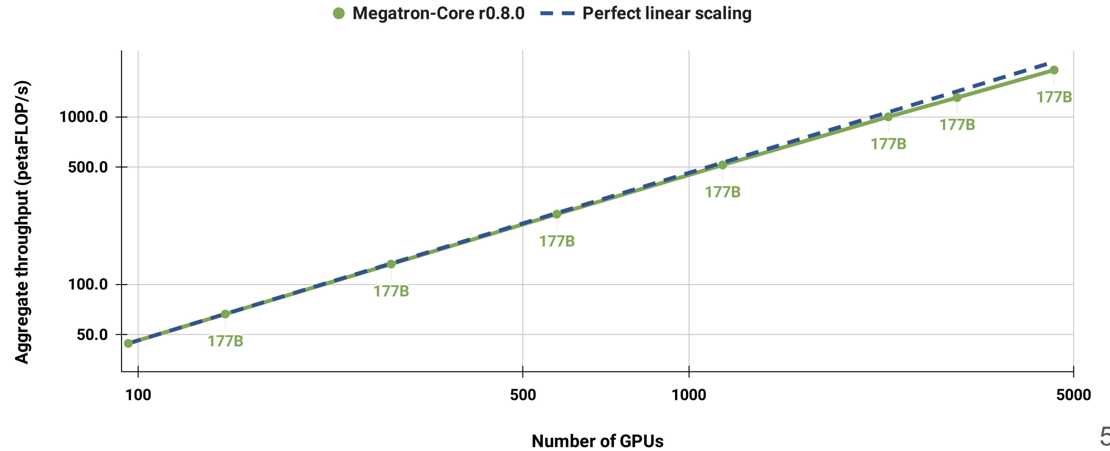
cnrs

Model size	Attention heads	Hidden size	Number of layers	Tensor MP size	Pipeline MP size	Data-parallel size	Number of GPUs	Batch size	Per-GPU teraFLOP/s	MFU	Aggregate petaFLOP/s
1.7B	16	2048	24	1	1	48	48	192	408.8	41%	19.6
7.1B	32	4096	30	2	1	48	96	192	465.9	47%	44.7
16B	48	6144	32	4	1	48	192	192	489.1	49%	93.9
32B	56	7168	48	8	1	48	384	192	459.6	46%	176.5
70B	64	8192	84	8	2	48	768	384	419.7	42%	322.3
119B	80	10240	92	8	4	48	1536	768	420.5	43%	645.9
177B	96	12288	96	8	6	48	2304	1152	432.8	44%	997.2
314B	128	16384	96	8	8	48	3072	1536	474.4	48%	1457.4
462B	144	18432	112	8	16	48	6144	3072	459.9	47%	2825.6

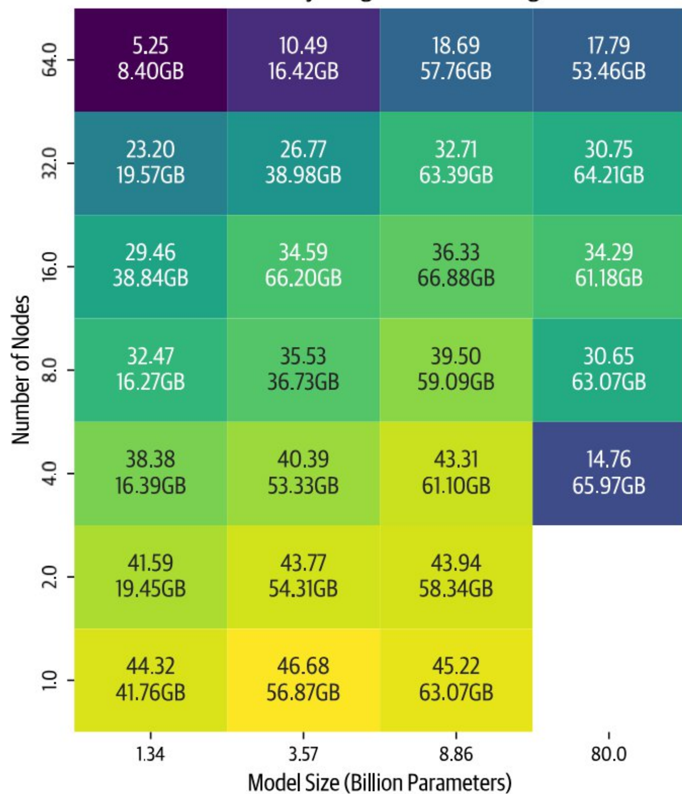
$$MFU = \frac{\text{Model FLOPs per token} \times \text{Observed tokens per second}}{\text{Theoretical peak FLOPs of the hardware}}$$

Megatron-LM





Nanotron



Best Configuration by Model Size and Number of Nodes

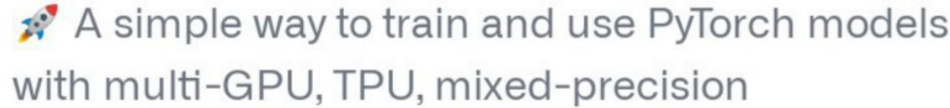
Number of Nodes	1.34	3.57	8.86	80.0
64.0	DP: 128 TP: 2 PP: 2 GAS: 1 MBS: 2 ZeRO-1	DP: 128 TP: 4 PP: 1 GAS: 1 MBS: 2 ZeRO-1	DP: 128 TP: 4 PP: 1 GAS: 1 MBS: 2 ZeRO-0	DP: 32 TP: 16 PP: 1 GAS: 4 MBS: 2 ZeRO-1
32.0	DP: 64 TP: 4 PP: 1 GAS: 1 MBS: 4 ZeRO-0	DP: 64 TP: 4 PP: 1 GAS: 1 MBS: 4 ZeRO-0	DP: 32 TP: 8 PP: 1 GAS: 1 MBS: 8 ZeRO-0	DP: 4 TP: 4 PP: 16 GAS: 64 MBS: 1 ZeRO-1
16.0	DP: 64 TP: 2 PP: 1 GAS: 1 MBS: 4 ZeRO-0	DP: 32 TP: 4 PP: 1 GAS: 1 MBS: 8 ZeRO-0	DP: 16 TP: 8 PP: 1 GAS: 2 MBS: 8 ZeRO-0	DP: 1 TP: 4 PP: 32 GAS: 256 MBS: 1 ZeRO-0
8.0	DP: 32 TP: 1 PP: 2 GAS: 8 MBS: 1 ZeRO-1	DP: 32 TP: 1 PP: 2 GAS: 8 MBS: 1 ZeRO-1	DP: 32 TP: 1 PP: 2 GAS: 8 MBS: 1 ZeRO-1	DP: 1 TP: 4 PP: 16 GAS: 256 MBS: 1 ZeRO-0
4.0	DP: 16 TP: 1 PP: 2 GAS: 16 MBS: 1 ZeRO-1	DP: 16 TP: 1 PP: 2 GAS: 8 MBS: 2 ZeRO-1	DP: 16 TP: 1 PP: 2 GAS: 16 MBS: 1 ZeRO-1	DP: 1 TP: 8 PP: 4 GAS: 256 MBS: 1 ZeRO-0
2.0	DP: 8 TP: 1 PP: 2 GAS: 32 MBS: 1 ZeRO-1	DP: 8 TP: 1 PP: 2 GAS: 16 MBS: 2 ZeRO-1	DP: 4 TP: 4 PP: 1 GAS: 16 MBS: 4 ZeRO-1	
1.0	DP: 4 TP: 1 PP: 2 GAS: 16 MBS: 4 ZeRO-1	DP: 4 TP: 1 PP: 2 GAS: 32 MBS: 2 ZeRO-1	DP: 2 TP: 1 PP: 4 GAS: 128 MBS: 1 ZeRO-1	

Model Size (Billion Parameters)

MFU scale: 45, 40, 35, 30, 25, 20, 15, 10

Fabric & Accelerate

huggingface/
accelerate




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

Lightning Fabric

[17, 28]

Pytorch ++

deepspeed

PyTorch Foundation Welcomes DeepSpeed

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/