

# Expert Parallelism in LLM Training

Autumn 2025

Lecturer: Yuedong (Steven) Xu

Shenzhen Loop Area Institute

[yuedongxu@slai.edu.cn](mailto:yuedongxu@slai.edu.cn)

Fudan University

[ydxu@fudan.edu.cn](mailto:ydxu@fudan.edu.cn)

# Disclaimer

Machine learning systems is a broad and rapidly evolving field. The course material has been developed using a broad spectrum of resources, including research papers, lecture slides, blogposts, research talks, tutorial videos, and other materials shared by the research community.

# Distributed LLM Training: Outline

- Data Parallelism
  - Parameter-Server, All-Reduce, Memory Optimization
- Model Parallelism
  - Pipeline Parallelism, Tensor Parallelism, Sequence Parallelism
- Mixture of Experts
  - **Principles**
  - Parallel Training
  - DeepEP

# Mixture of Experts

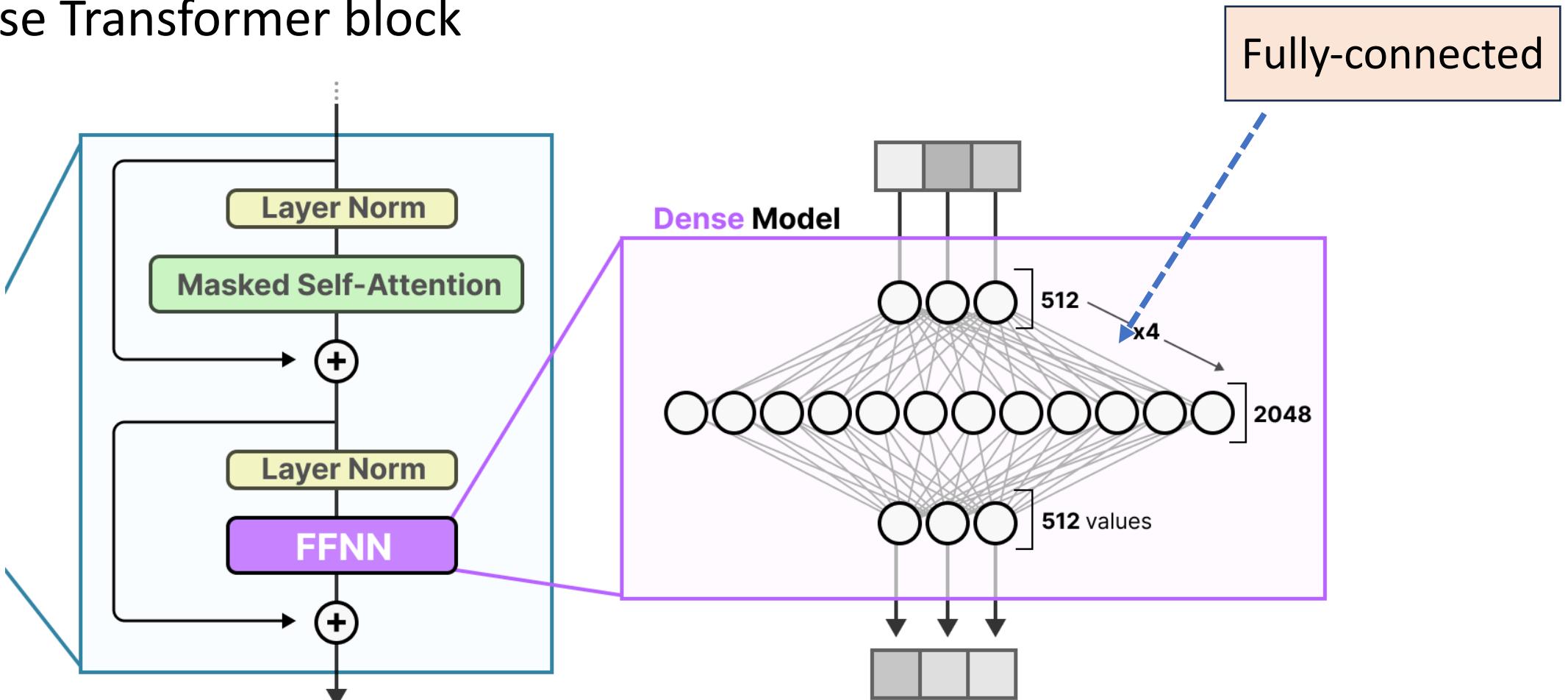
- Mixture of Experts in Early Days
  - Robert Jacobs, Michael Jordan, Steven Nowlan and Geoffrey Hinton (1991)
    - “a new supervised learning procedure for systems composed of many separate networks, each of which learns to handle a subset of the complete set of training cases.”
  - David Eigen, Marc'Aurelio Ranzato, Ilya Sutskever (2013)
    - “extend the Mixture of Experts to a stacked model, the Deep Mixture of Experts, with multiple sets of gating and experts.”
- Mixture of Experts in Transformer
  - GShard by Google

# What shall we learn in this lecture

- We only learn
  - the structure of MoE in today's language models
  - the system challenges faced by MoE model training
  - recent progresses on addressing All-to-All issues
- We encourage students to learn by themselves
  - the instability of MoE model training
  - the design of new MoE framework
  - the fine-tuning techniques of MoE models

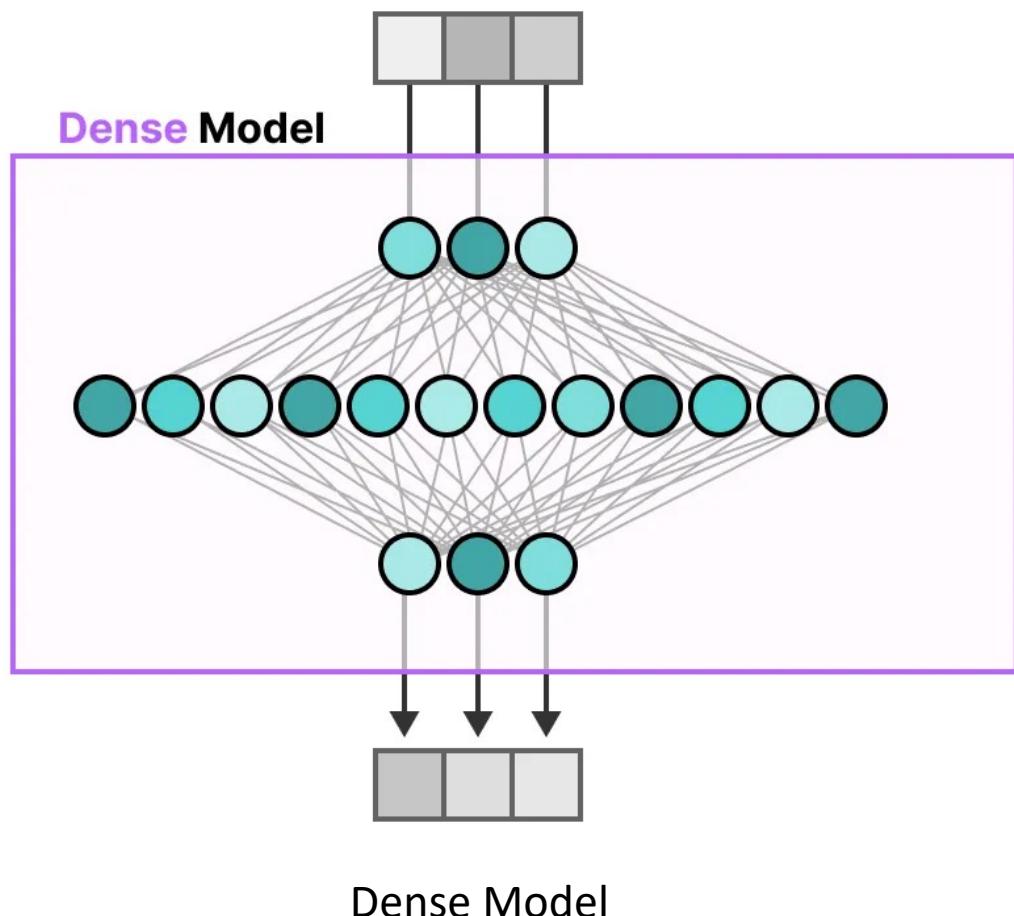
# Mixture of Experts

- Dense Transformer block



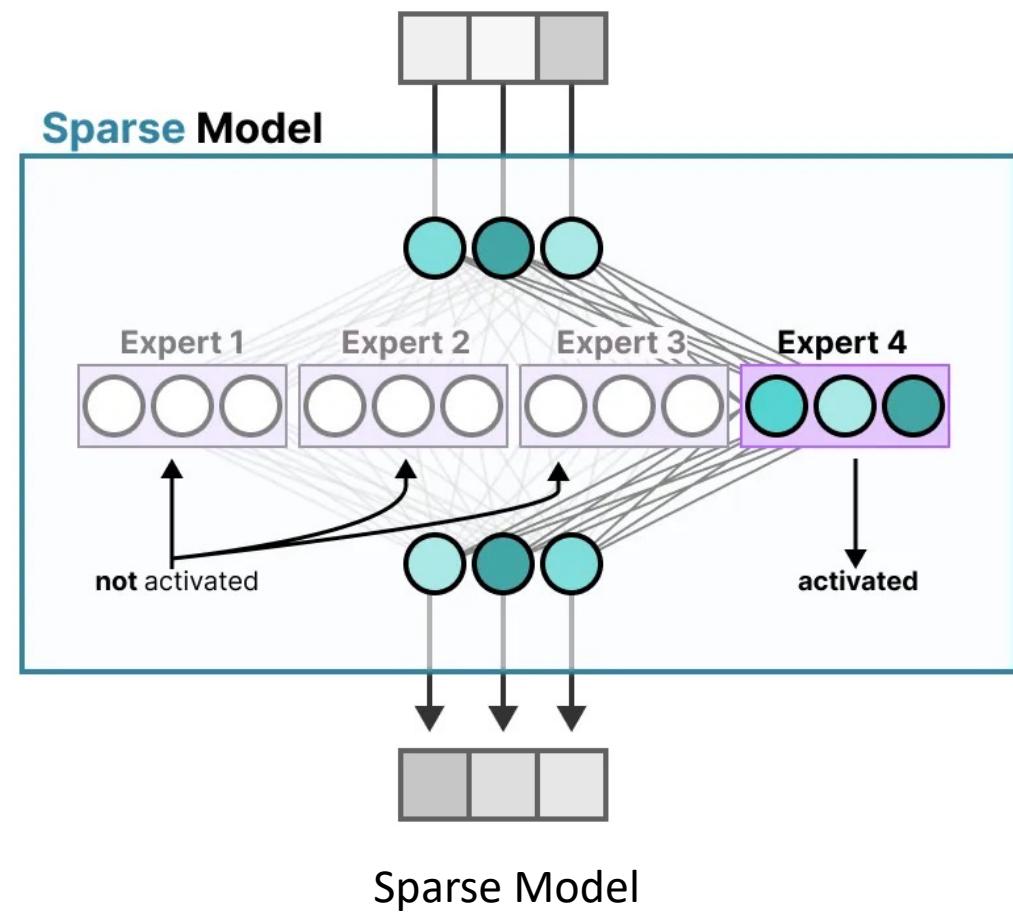
# Mixture of Experts

SLAI is an international institute

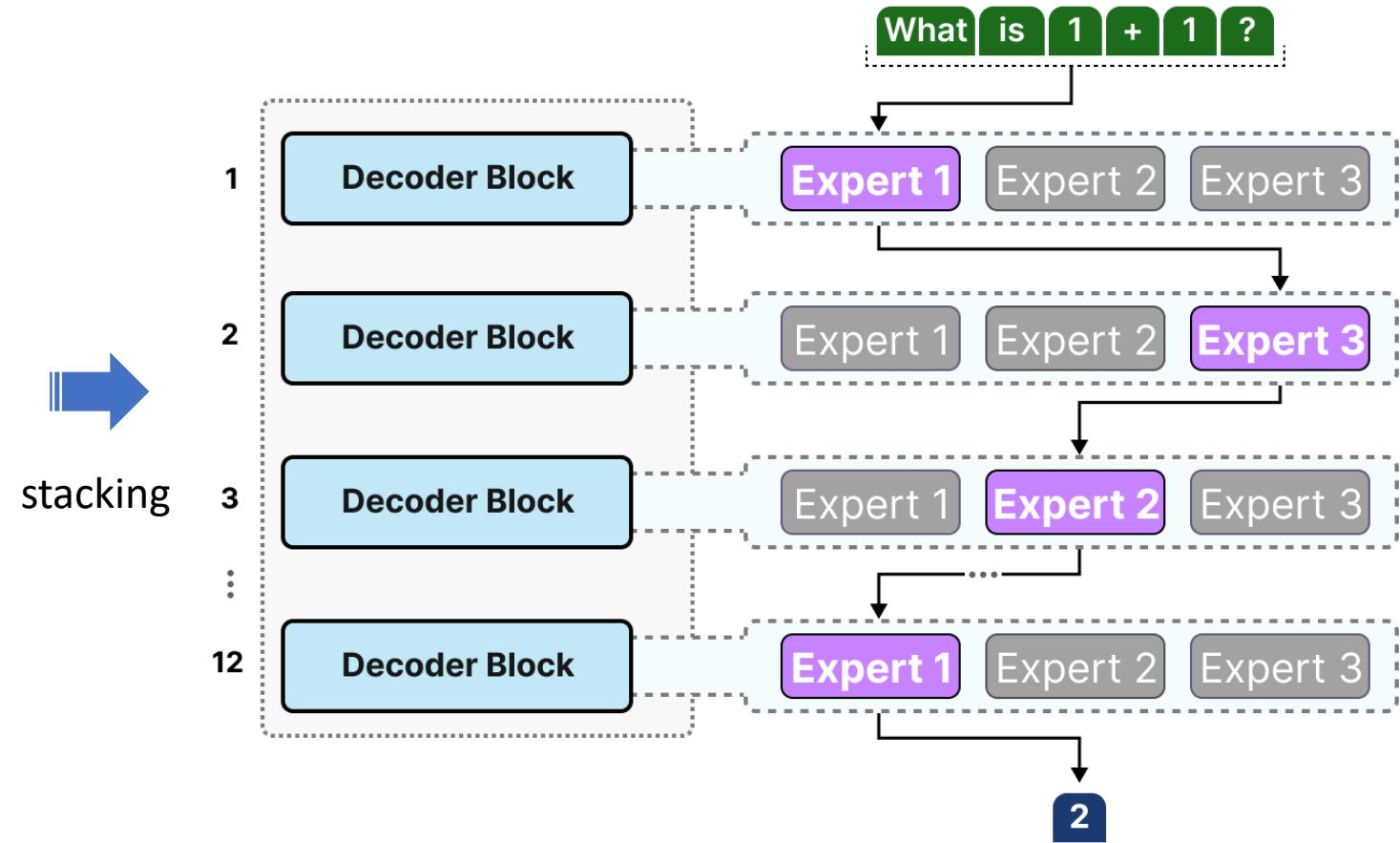
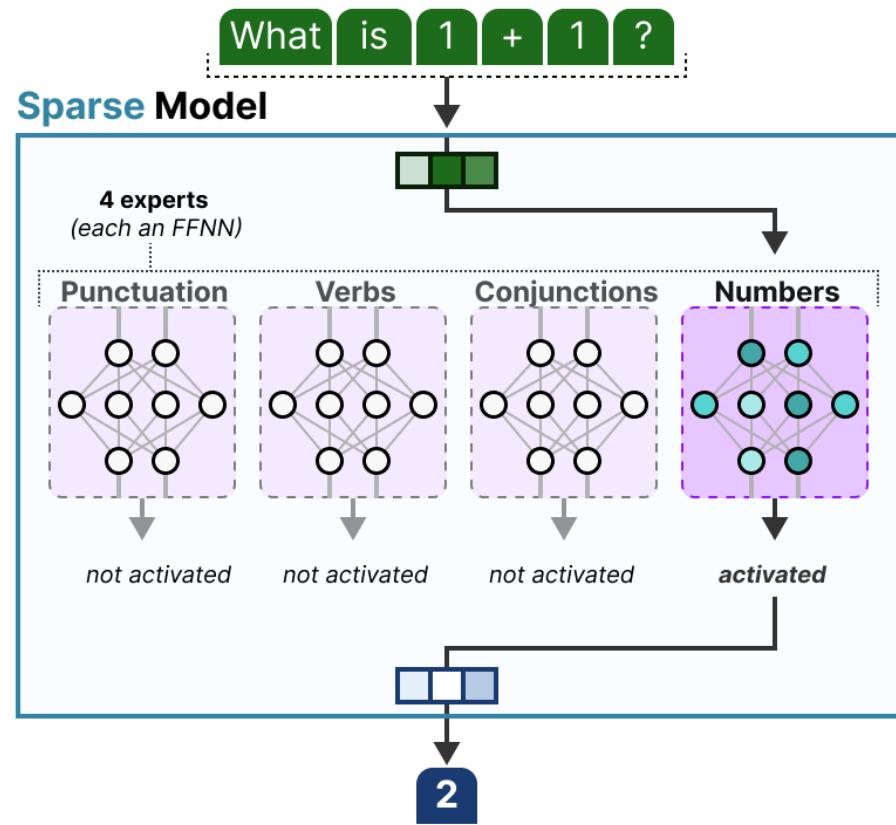


copy

SLAI is an international institute



# Mixture of Experts

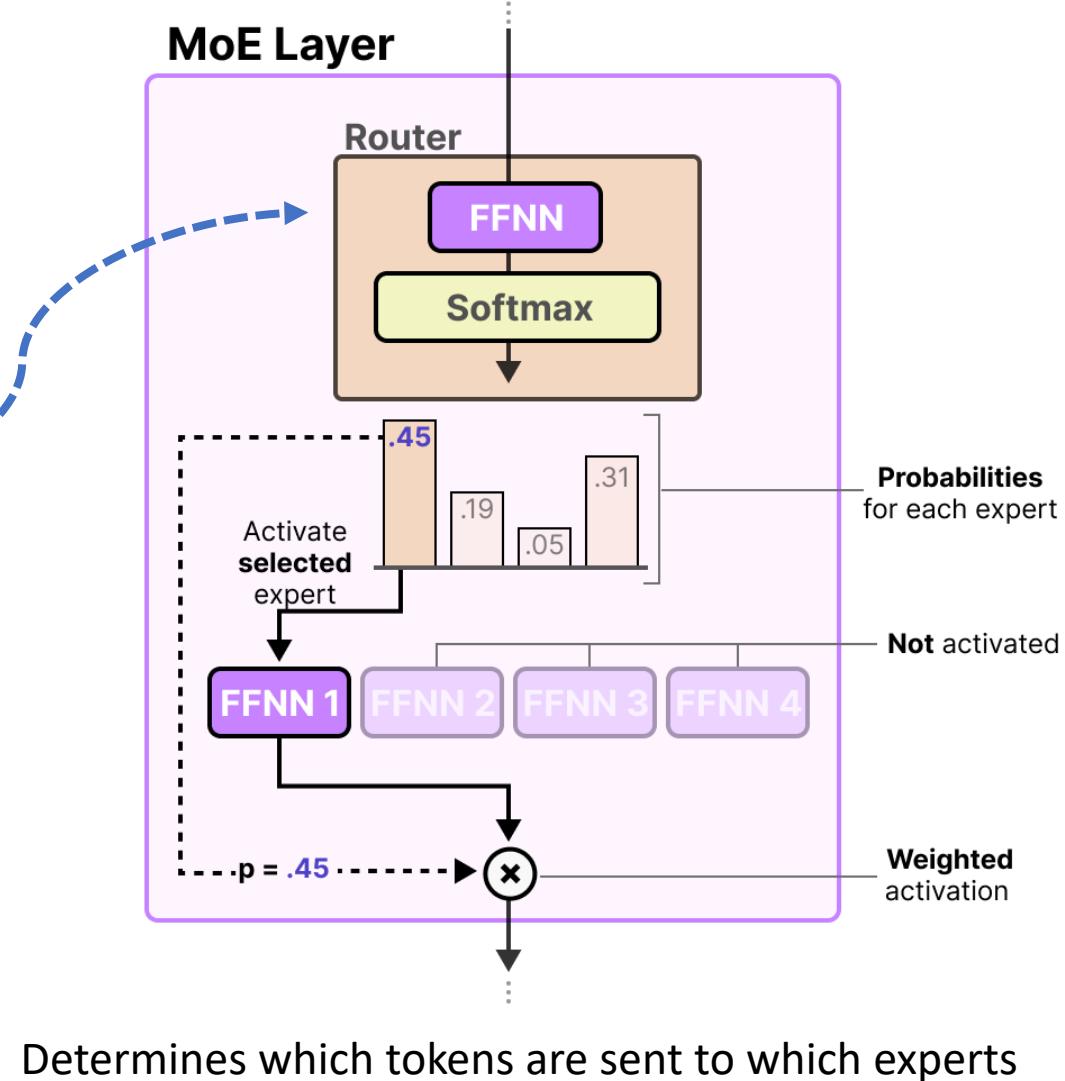
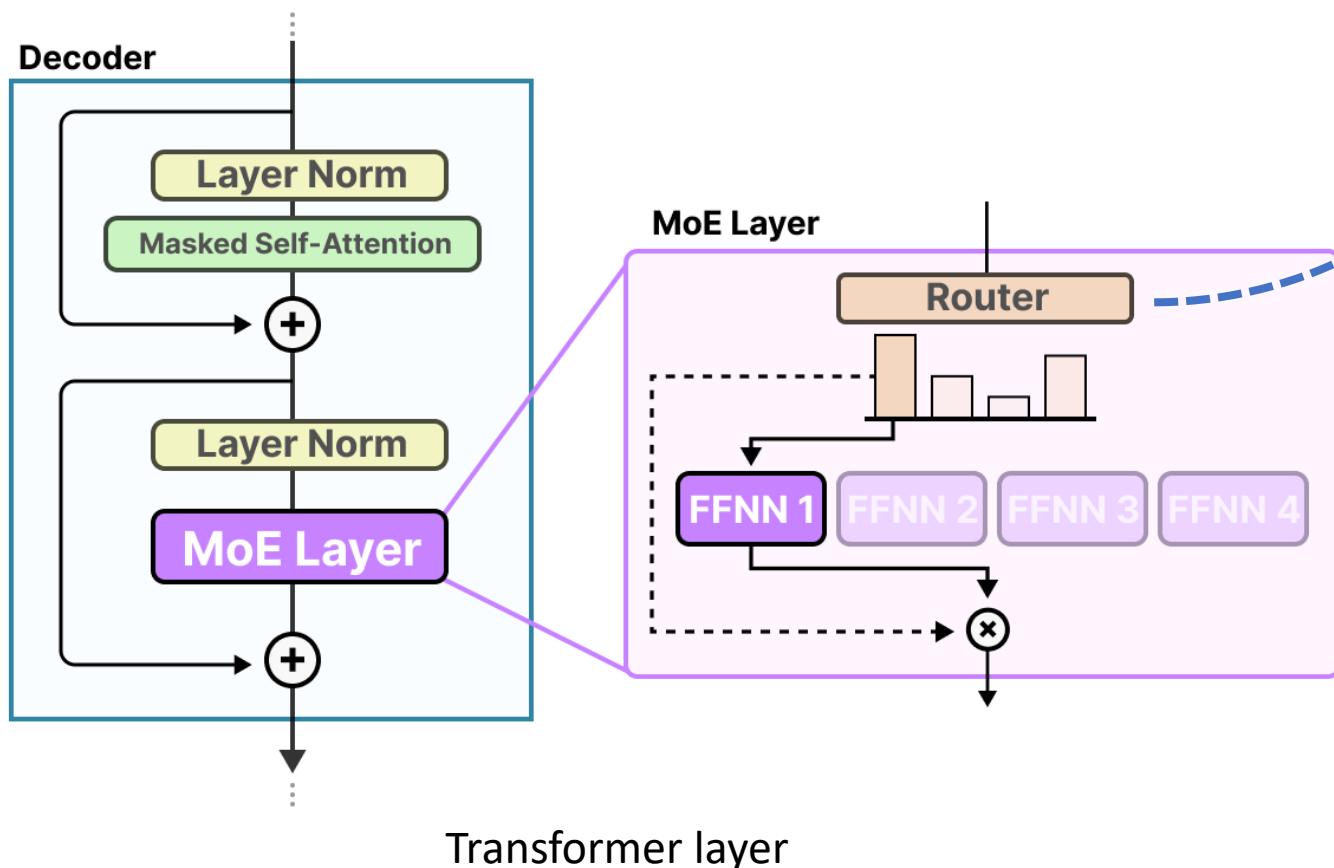


Each expert specializes in different syntactic tokens

More than one decoders → The chosen experts likely differ between tokens, and each token chooses different experts when traversing different encoders → a path exists

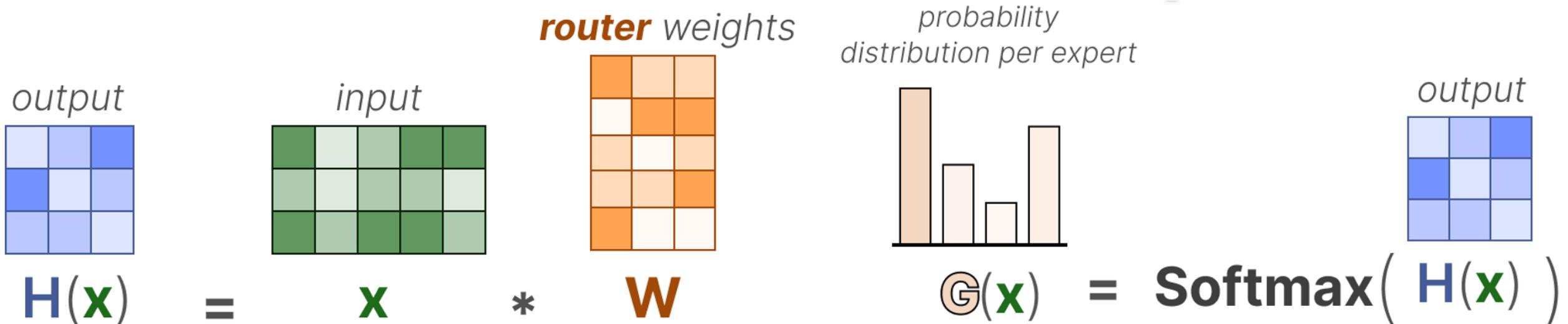
# Mixture of Experts

- How to select an expert?



# Mixture of Experts

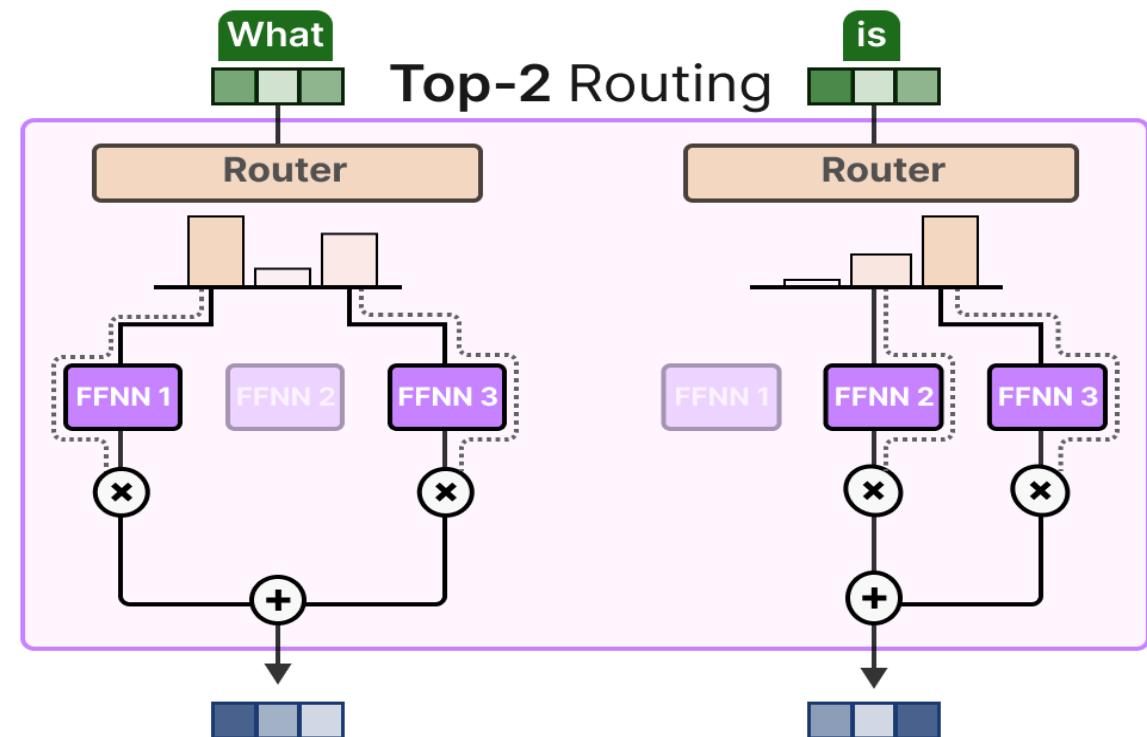
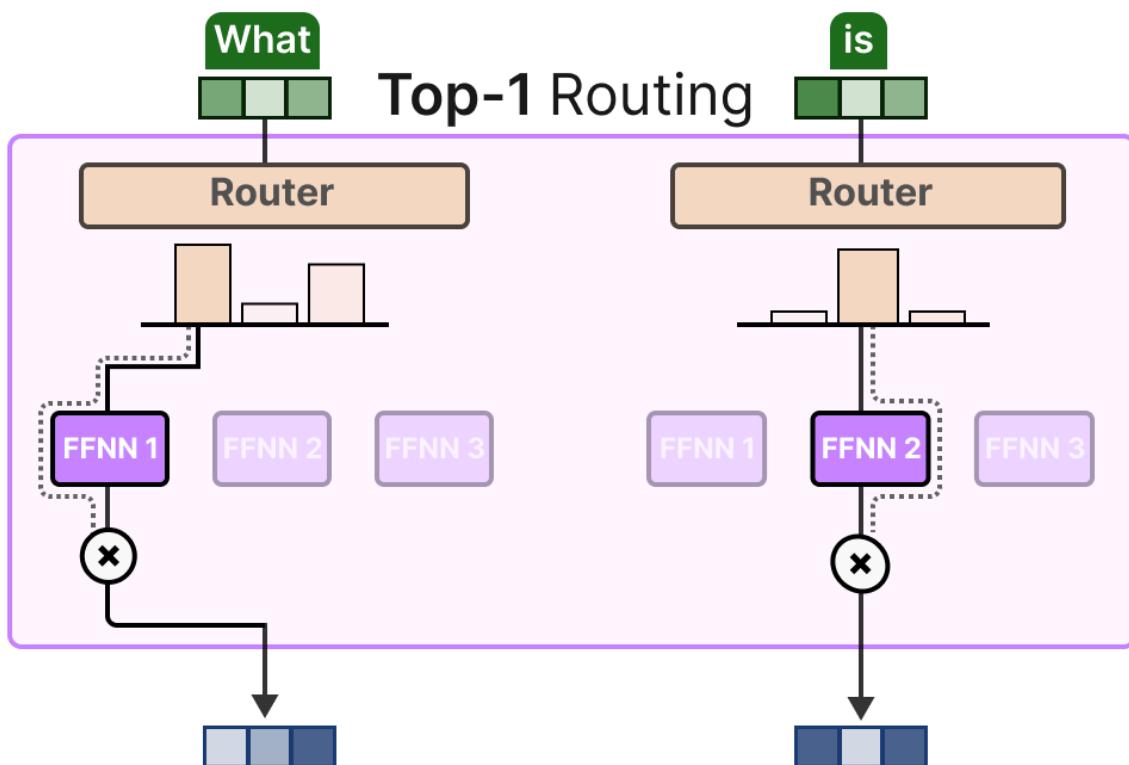
- How to select an expert?



A very basic router

# Mixture of Experts

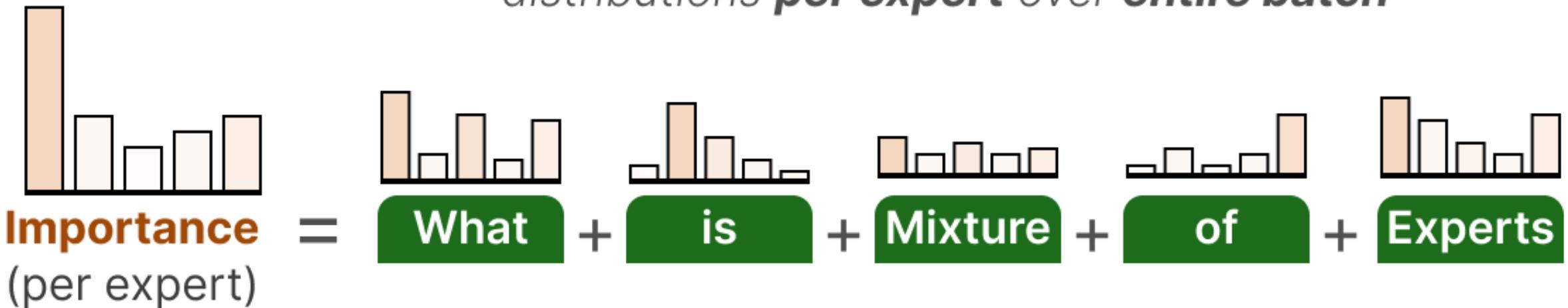
- Top-K routing



# Mixture of Experts

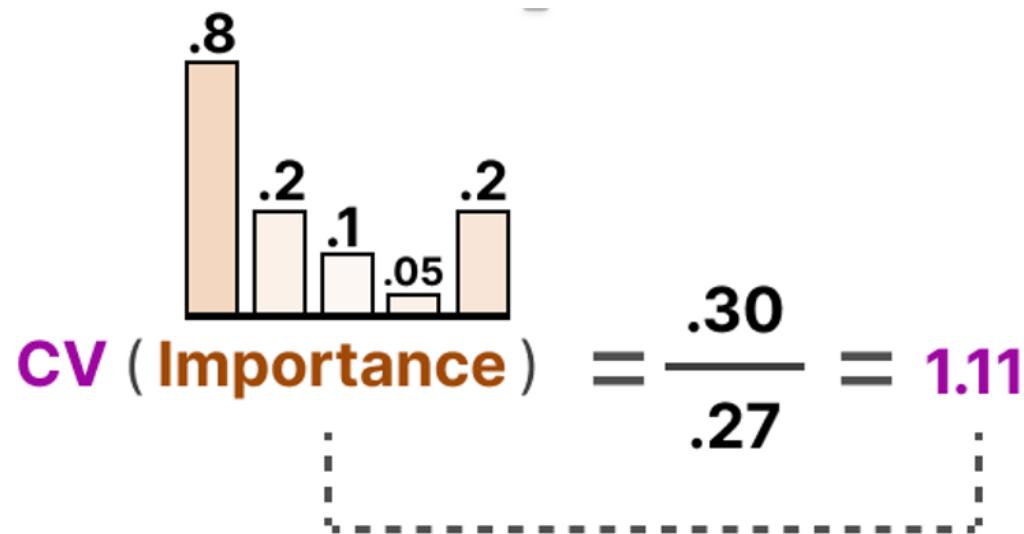
- Load balancing from an algorithmic perspective
  - Quantifying importance of experts

*sum probability  
distributions **per expert** over **entire batch***

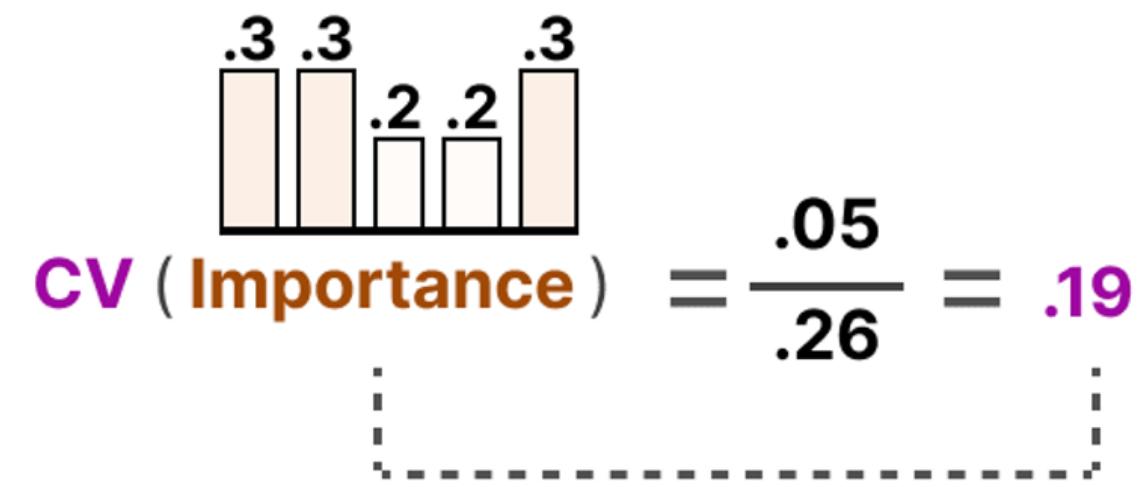


# Mixture of Experts

- Load balancing from an algorithmic perspective
  - Coefficient of variation to capture the diversity across experts



**High** variance in expert  
**importance** result in **high CV**



**Low** variance in expert  
**importance** result in **low CV**

# Mixture of Experts

- Load balancing from an algorithmic perspective
  - Adding “auxiliary loss” to the overall training objective function

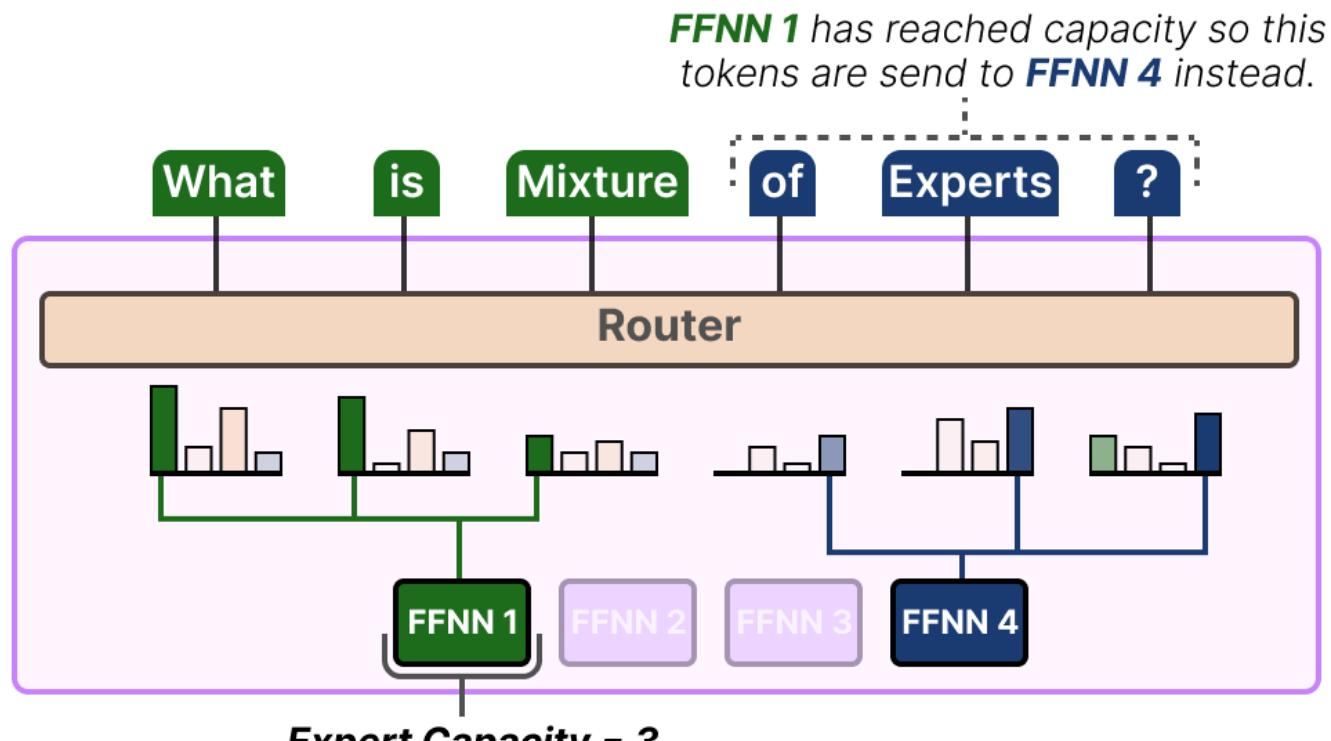
$$\text{Auxiliary Loss} = \frac{\text{(constant)} \cdot \text{scaling factor}}{w_{importance}} * \text{CV} (\text{Importance})^2$$

A high variance in expert importance (CV)  
results in high loss and vice versa

$$l_{aux} = \frac{1}{E} \sum_{e=1}^E \frac{c_e}{S} * m_e$$

# Mixture of Experts

- Expert Capacity
  - limiting the amount of tokens a given expert can handle



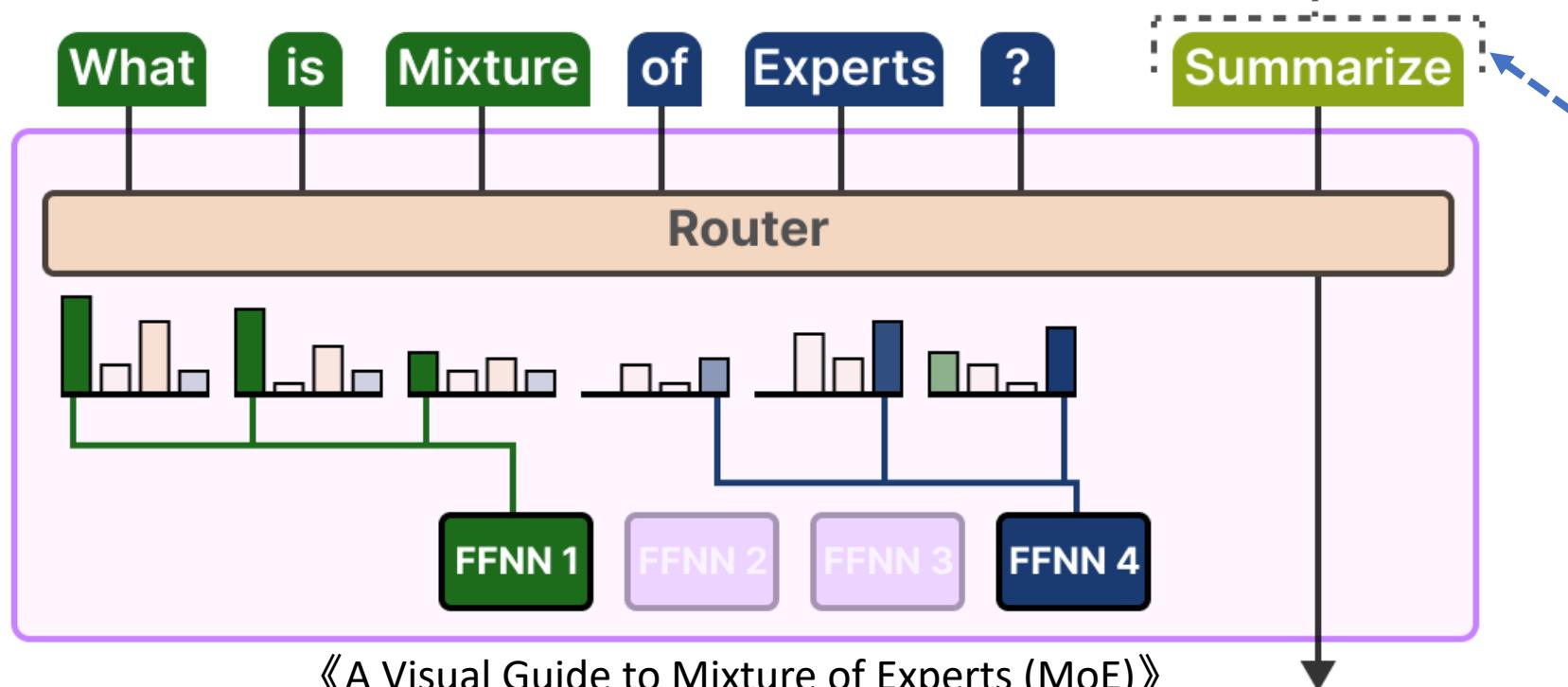
$$capacity = \max\left(\frac{S}{E} * K * capacity\_factor, min\_capacity\right)$$

- Expert Capacity
  - S: # of tokens
  - E: # of experts
  - K: Tok-K parameter

# Mixture of Experts

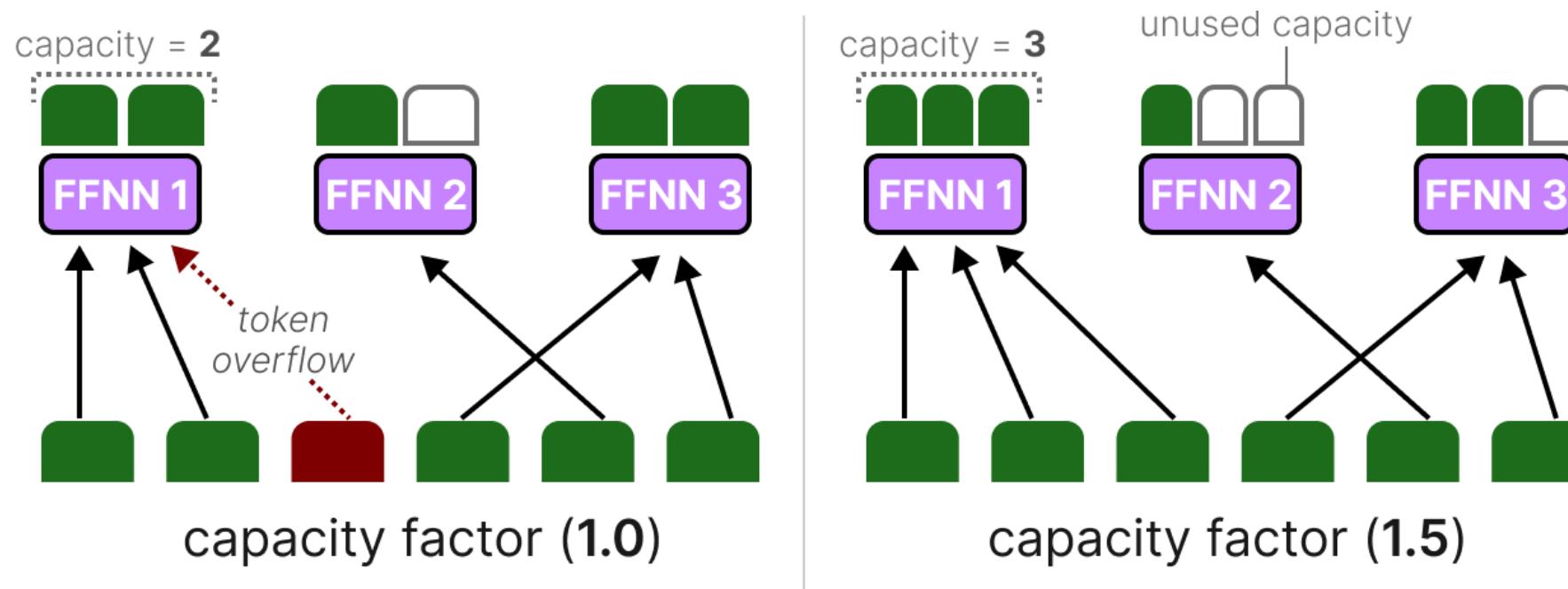
- Expert Capacity
  - limiting the amount of tokens a given expert can handle

**FFNN 1** and **FFNN 4** have **reached capacity**.  
The token is sent to the **next layer** instead.



# Mixture of Experts

- Adjusting Expert Capacity



Increasing the capacity factor increases the quality but increases communication costs and memory of activations.

# Mixture of Experts

- Random Routing/Noisy Routing

$$\text{(somewhat noisy)} \quad \begin{matrix} \text{output} \\ \begin{matrix} \text{H}(\mathbf{x}) \end{matrix} \end{matrix} = \begin{matrix} \text{input} \\ \mathbf{x} \end{matrix} * \begin{matrix} \text{router} \\ \text{weights} \\ \mathbf{W} \end{matrix} + \begin{matrix} \text{(small amount of)} \\ \text{gaussian} \\ \text{noise} \\ \mathbf{n} \end{matrix}$$

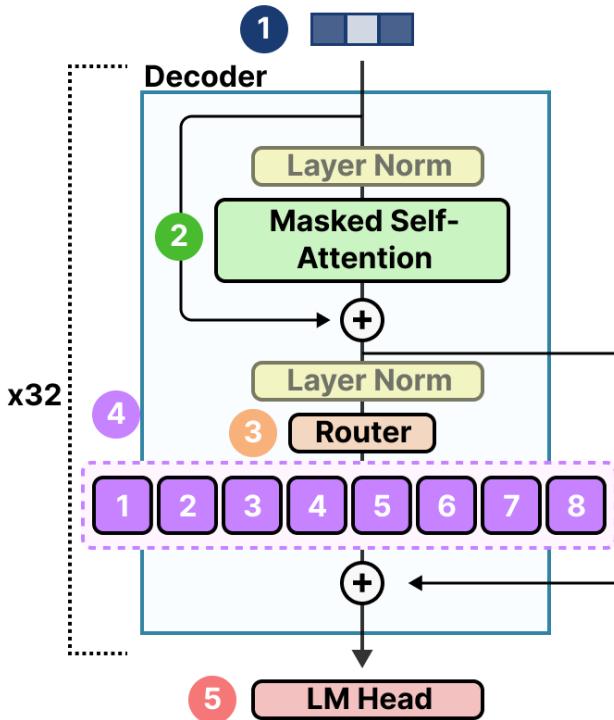
The diagram illustrates the computation of a "somewhat noisy" output. It shows the multiplication of an input vector  $\mathbf{x}$  by a router weights matrix  $\mathbf{W}$ , followed by the addition of Gaussian noise  $\mathbf{n}$ . The input  $\mathbf{x}$  is a 5x1 vector with green blocks. The router weights  $\mathbf{W}$  is a 5x3 matrix with orange blocks. The noise  $\mathbf{n}$  is a 5x3 matrix with red blocks.

preventing the same experts from always being selected

# Mixture of Experts

- Scaling up model parameters

**Mixtral 8x7B**



<b>1</b>	<b>Embeddings</b>	$32000 \times 4096 = 131.072.000$
		embedding size shared parameters
<b>2</b>	<b>Attention</b>	$32 \times 41.943.040 = 1.342.177.280$
		repeated decoder blocks (q, k, v) shared parameters
<b>3</b>	<b>Router</b>	$8 \times 4096 = 32.768$
		# experts shared parameters
<b>4</b>	<b>Experts</b>	$8 \times 5.637.144.576 = 45.097.156.608$
		# experts total parameters
		$2 \times 5.637.144.576 = 11.274.289.152$
		# experts expert size active parameters
<b>5</b>	<b>LM Head</b>	$32000 \times 4096 = 131.072.000$
		shared parameters

Shared parameters		Sparse Parameters (all parameters)		Active Parameters (activated parameters)			
<b>1</b>	<b>Embeddings</b>	<b>131.072.000</b>	<b>46.7B</b>	<b>1</b>	<b>Embeddings</b>	<b>131.072.000</b>	<b>12.8B</b>
<b>2</b>	<b>Attention</b>	<b>1.342.177.280</b>		<b>2</b>	<b>Attention</b>	<b>1.342.177.280</b>	
<b>3</b>	<b>Router</b>	<b>32.768</b>		<b>3</b>	<b>Router</b>	<b>32.768</b>	
<b>4</b>	<b>Experts</b>	<b>45.097.156.608</b>		<b>4</b>	<b>Experts</b>	<b>11.274.289.152</b>	
<b>5</b>	<b>LM Head</b>	<b>131.072.000</b>		<b>5</b>	<b>LM Head</b>	<b>131.072.000</b>	

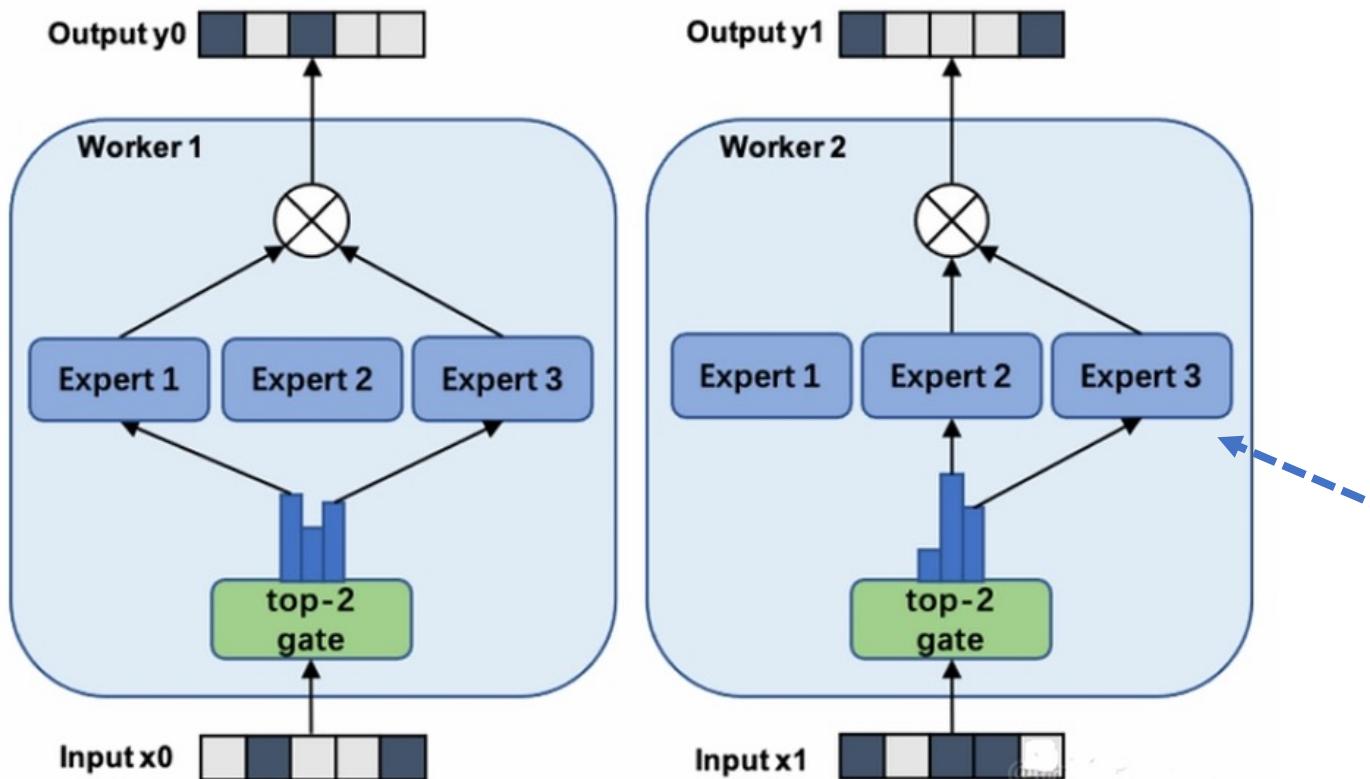
Significant Reduction on Computations

# Distributed LLM Training: Outline

- Data Parallelism
- Model Parallelism
- Mixture of Experts
  - Principles
  - **Parallel Training**
  - Recent Progresses

# Mixture of Experts

- Parallel Training
  - How to distribute experts on different GPUs?
- Data and Expert Parallelism
  - Small # of experts
  - Experts are small in sizes

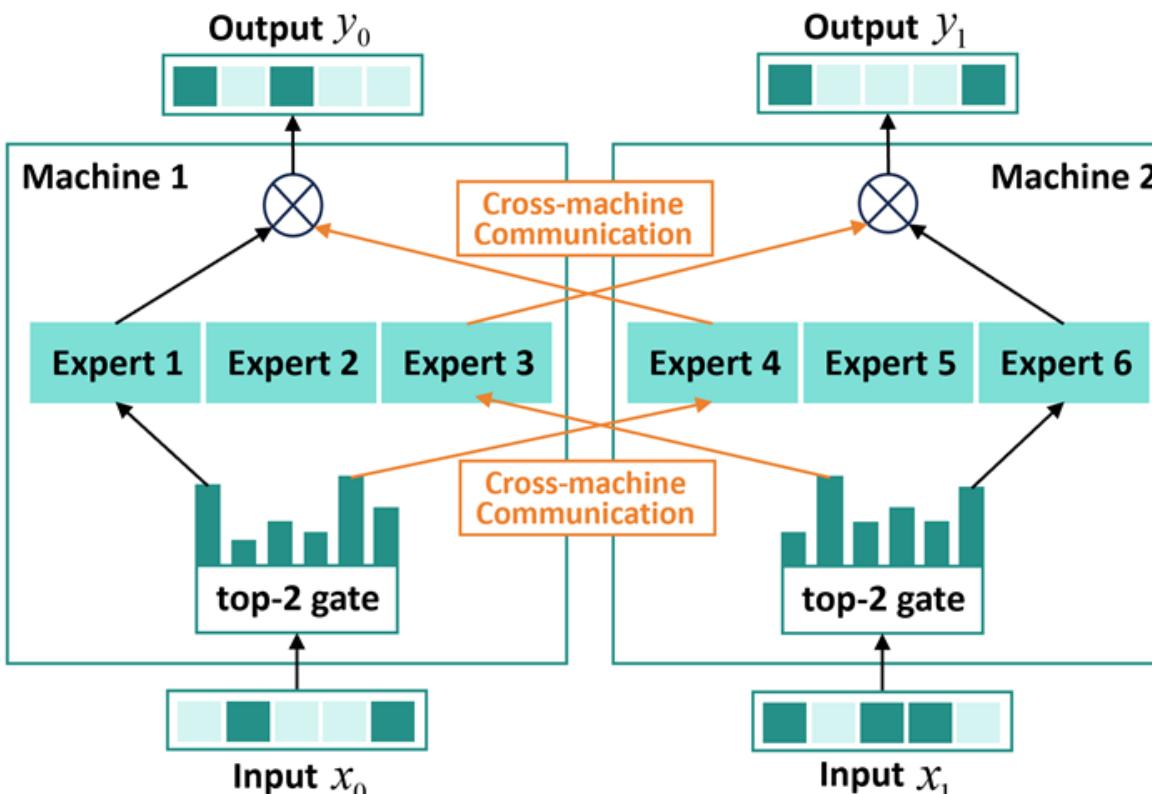


Every GPU/Machine stores the complete replicas of experts

# Mixture of Experts

- Parallel Training

- How to distribute experts on different GPUs?



6 experts, DP = 2, EP = 2 (assuming one machine one GPU)

- Data and Expert Parallelism

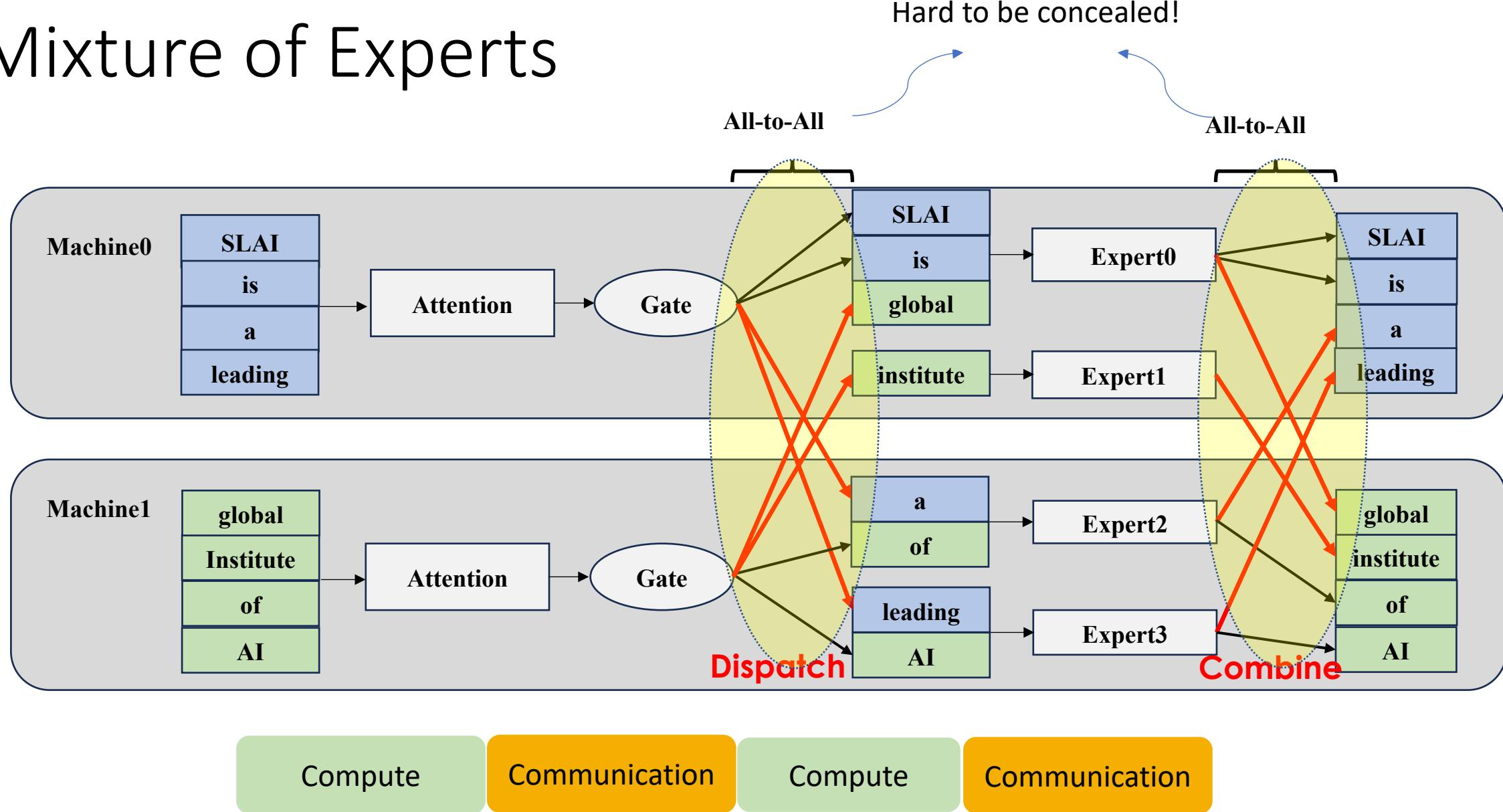
- Scaling up to gigantic experts
  - Experts are placed across DP groups

- Size of a “token”

- $\text{hidden\_dim} * 2$  Bytes (fp16)
    - GPT3: 7168 ( $\text{hidden\_dim}$ )
    - GPT4: 16384 ( $\text{hidden\_dim}$ )

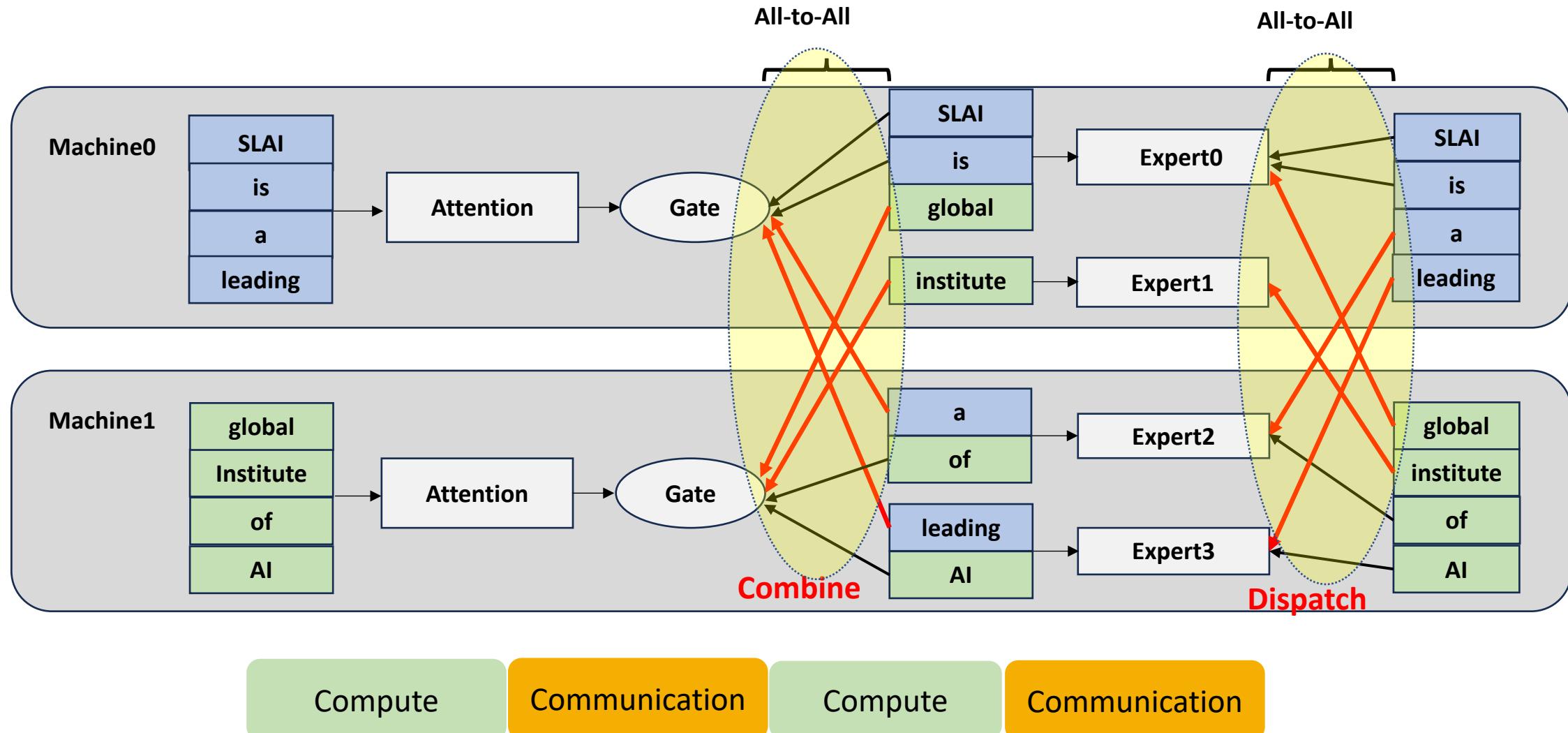
Cross-Machine Communication

# Mixture of Experts

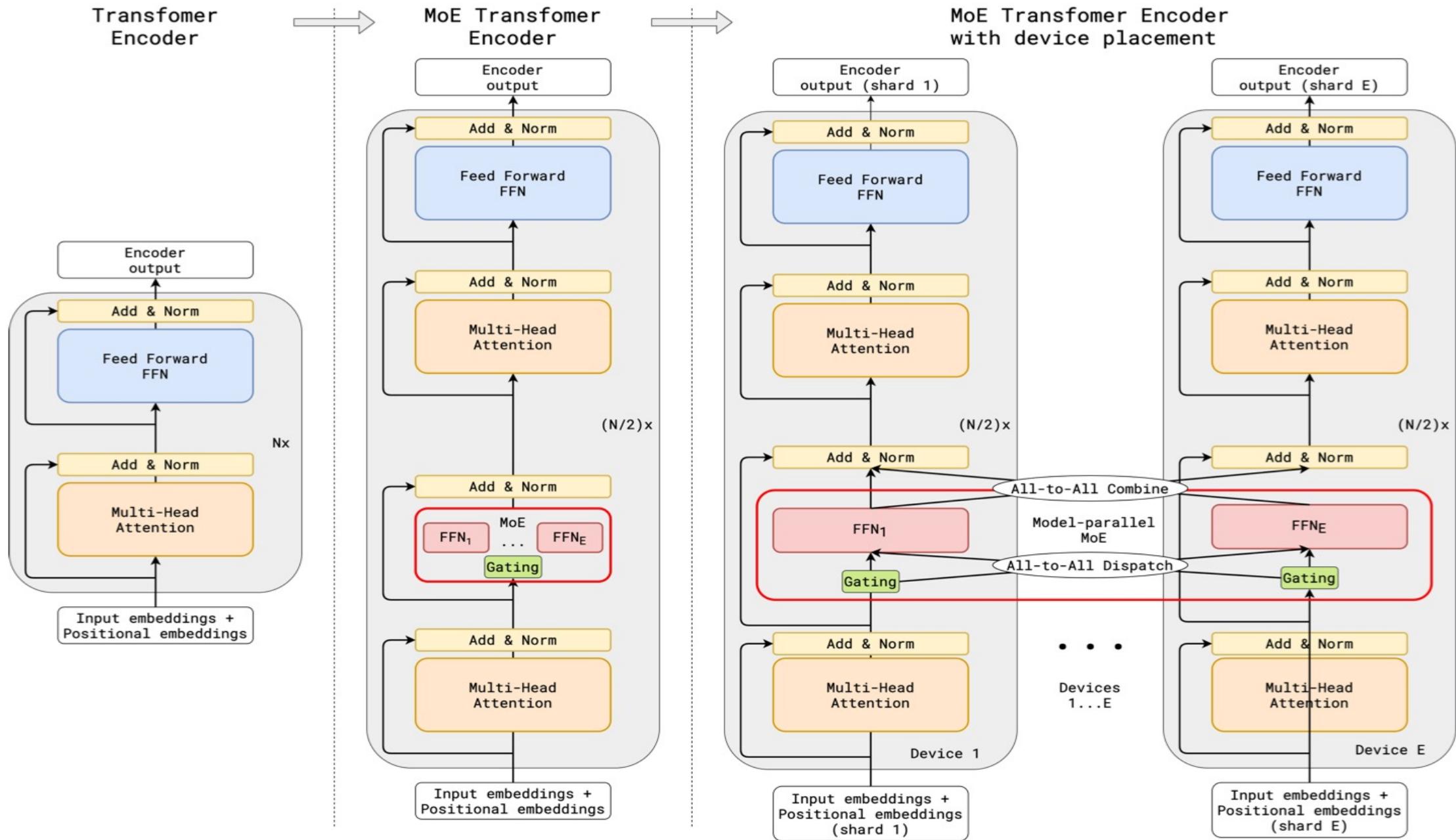


Forward Pass: heavy token communications that hard to be concealed

# Mixture of Experts

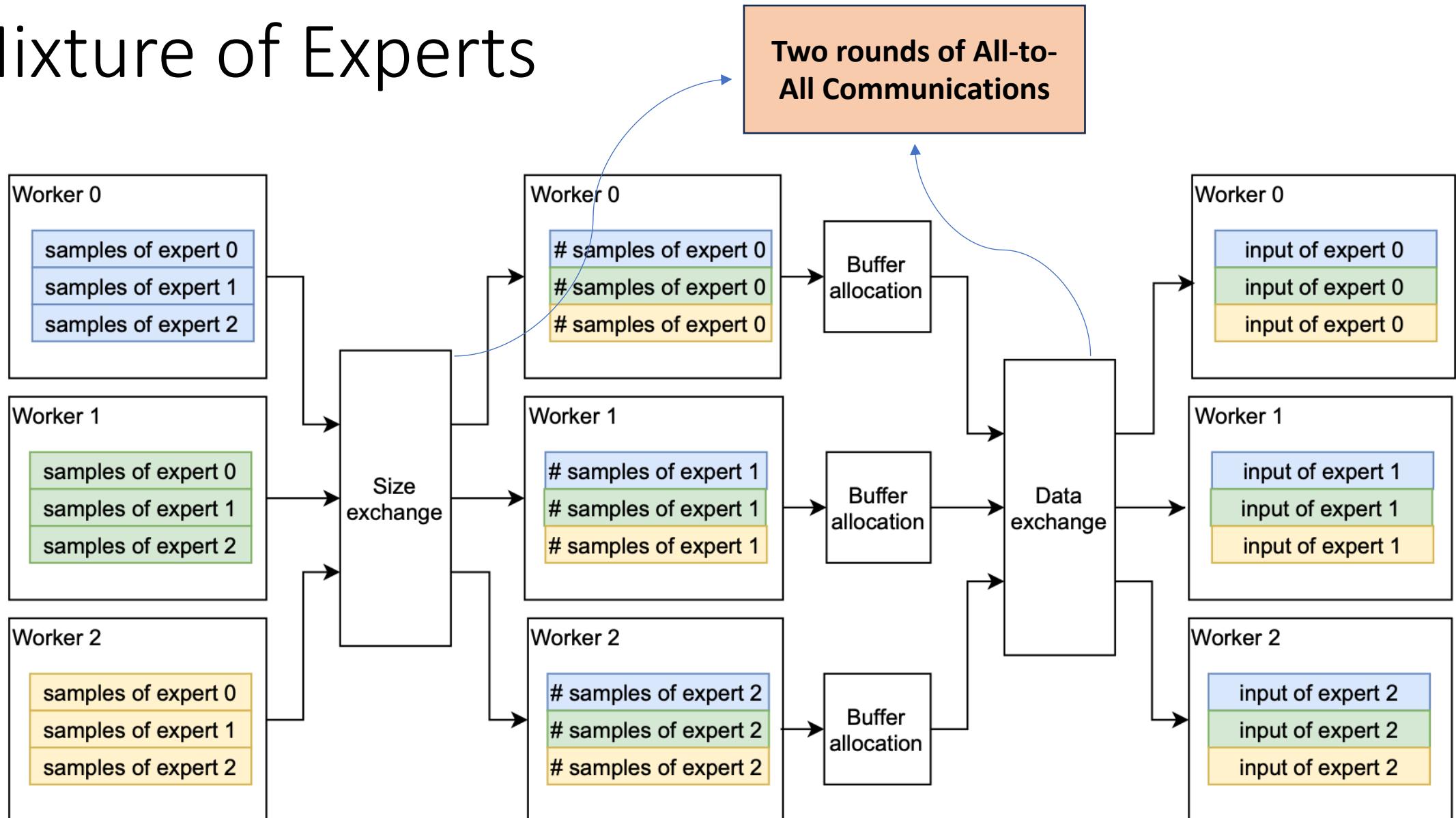


**Backward Pass: heavy token communications that hard to be concealed**



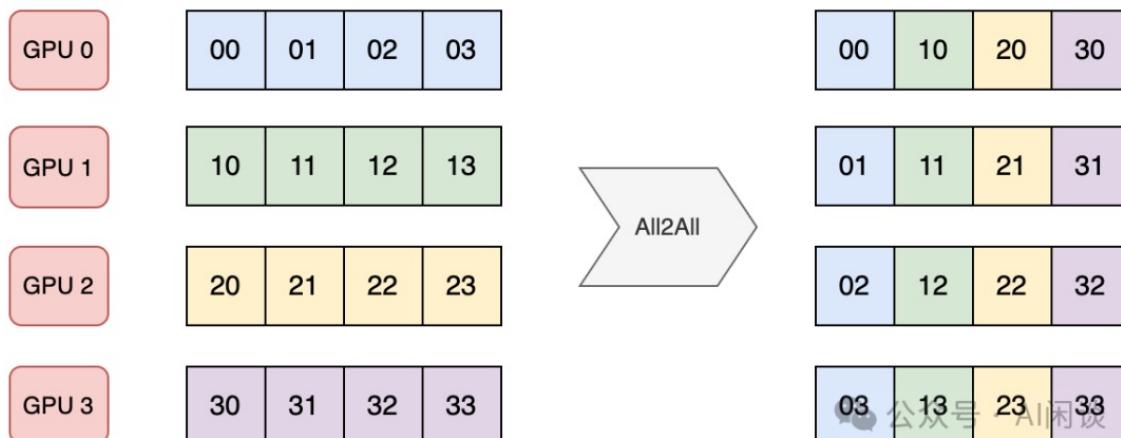
**GShard (2020): imaging the workflow of forward and backward propagations**

# Mixture of Experts



# Mixture of Experts

- All-to-All Communication
  - Standard all-to-all (symmetric)



```
import torch
import torch.distributed as dist

def run_alltoall():
    rank = dist.get_rank()
    size = dist.get_world_size()
    device = torch.device(f'cuda:{rank}')
    input_tensor = torch.ones(size, dtype=torch.int32, device=device) * rank
    print(f"Rank {rank} before alltoall, input_tensor: {input_tensor.tolist()}\n")

    output_tensor = torch.empty(size, dtype=torch.int32, device=device)
    dist.all_to_all_single(output_tensor, input_tensor)
    print(f"Rank {rank} after alltoall, output_tensor: {output_tensor.tolist()}\n\n"

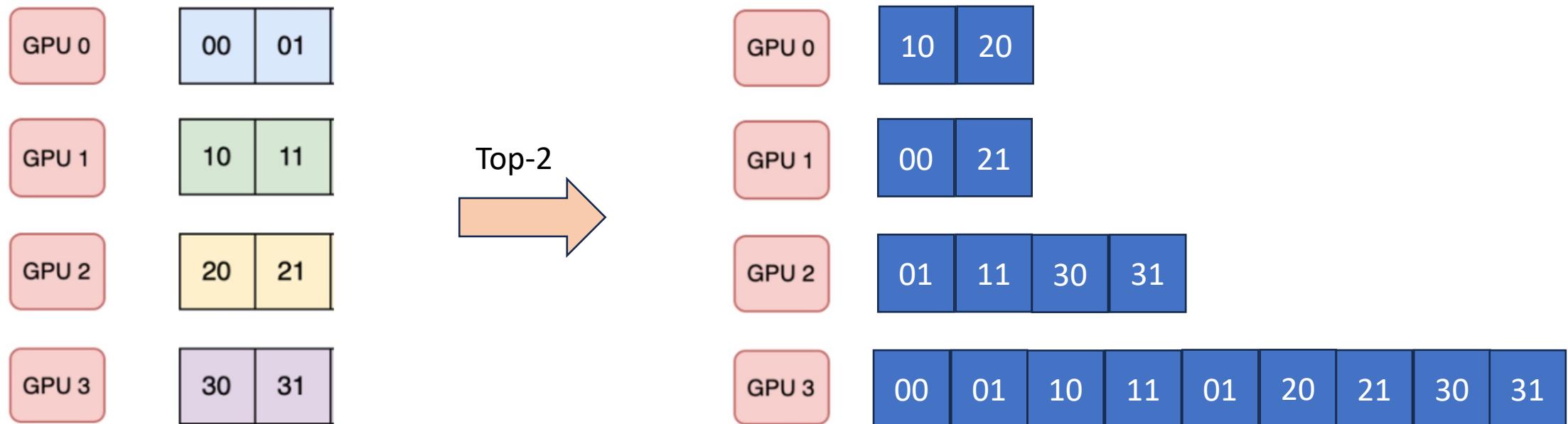
def main():
    dist.init_process_group(backend='nccl')
    run_alltoall()
    dist.destroy_process_group()

if __name__ == "__main__":
    main()
```

公众号 · AI闲谈

# Mixture of Experts

- All-to-All Communication
  - Asymmetric data volume (tokens) between GPUs hosting experts
    - **ZERO padding** to transmit **dummy** tokens under standard All-to-All?



# Mixture of Experts

- All-to-All Communication
  - NCCL (*NVIDIA Collective Commun. Library*) does not have all-to-all implementations
    - ***ncclSend*** and ***ncclRecv*** point-to-point communication instead

```
# 伪代码表示NCCL的all-to-all实现
ncclGroupStart()
    for target_rank in range(world_size):
        if target_rank != my_rank:
            ncclSend(send_buffer[target_rank], target_rank, comm)
            ncclRecv(recv_buffer[target_rank], target_rank, comm)
ncclGroupEnd()
```

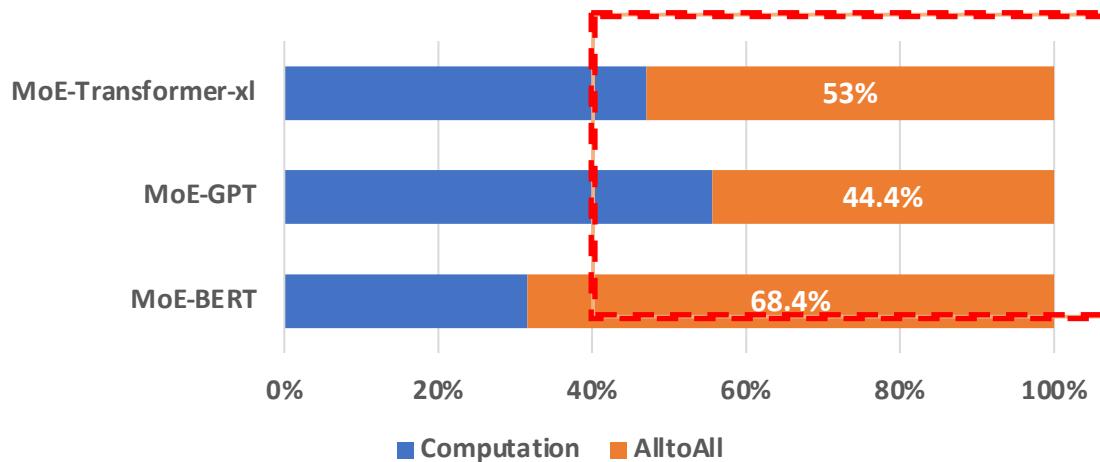
Pseudo codes for “spread-out” implementation

# Mixture of Experts

- MoE All-to-All Communication Practice
  - Dispatch
    - **Gathering # of tokens to send and receive:** one standard All-to-All to exchange the # of tokens to transfer, e.g. via ***torch.distributed.all\_to\_all\_single*** in DeepSpeed-MoE
    - Creating sending and receiving buffers, e.g. by **appending** or **concatenating** vectors
    - Exchanging token vectors: non-standard All-to-All, e.g. via ***torch.distributed.all\_to\_all\_single*** while the # of tokens differ
    - Assigning received tokens to local experts
  - Combine
    - Creating sending and receiving buffers
    - Exchanging the expert outputs and meta data to the corresponding nodes

# Mixture of Experts

- Challenge in MoE All-to-All



All-to-All communication accounts for a significant proportion ( $44.4\% \sim 68.4\%$ ),

iteration time vs time of All-to-All (32 GPUs)

#Parameter	#layer	#AlltoAll per iteration
GPT* + MoE	175 B	96

Upscaling with # of transformer blocks  
(an example)

# Distributed LLM Training: Outline

- Data Parallelism
- Model Parallelism
- Mixture of Experts
  - Principles
  - Parallel Training
  - **Recent Progresses**

# Mixture of Experts

- Optimizing MoE All-to-All Communication
  - Expert-centric: keeping **experts in-place** and moving tokens via All-to-All
  - Data-centric: keeping **tokens in-place** and moving experts between GPUs

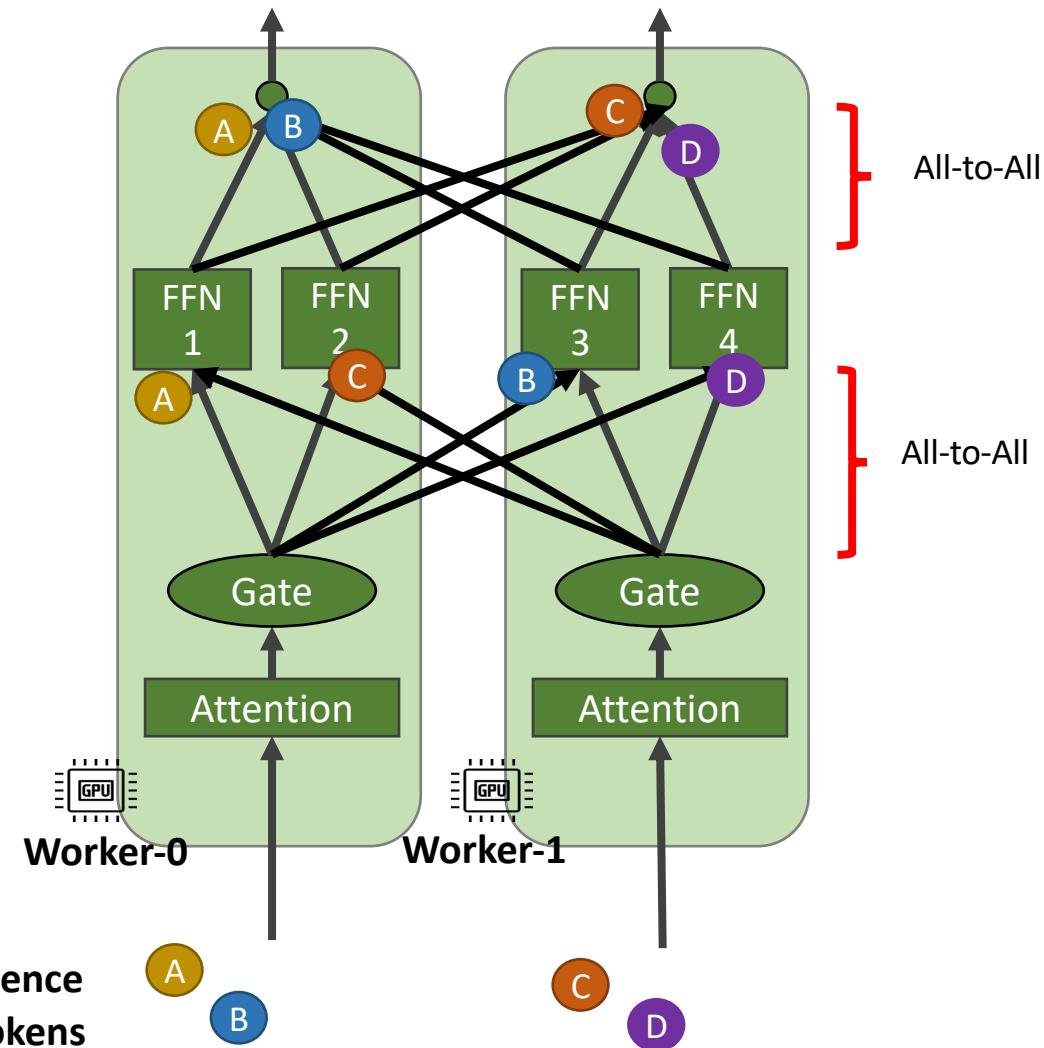
Case Study:

Running on 4 8-A100 machines (32 GPUs)  
Batch=64, Seq=512, Gate's topk=2, hidden size=256, #expert=32

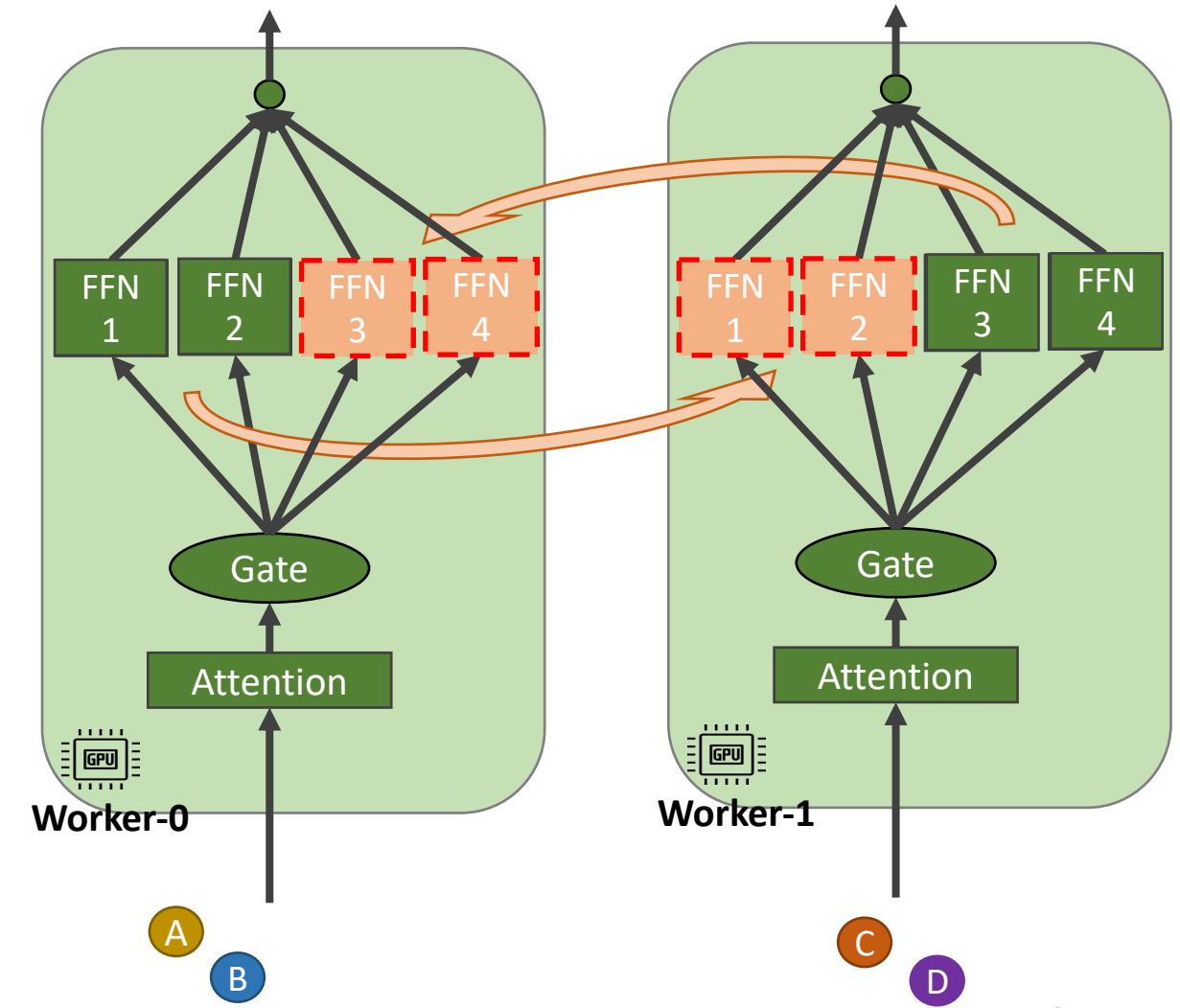
	Operation	Volume of Traffic across a machine
Expert Centric	send out tokens	0.75GB
Data Centric	fetch experts	0.047GB

Data Centric has lower traffic workload !

## Expert Centric

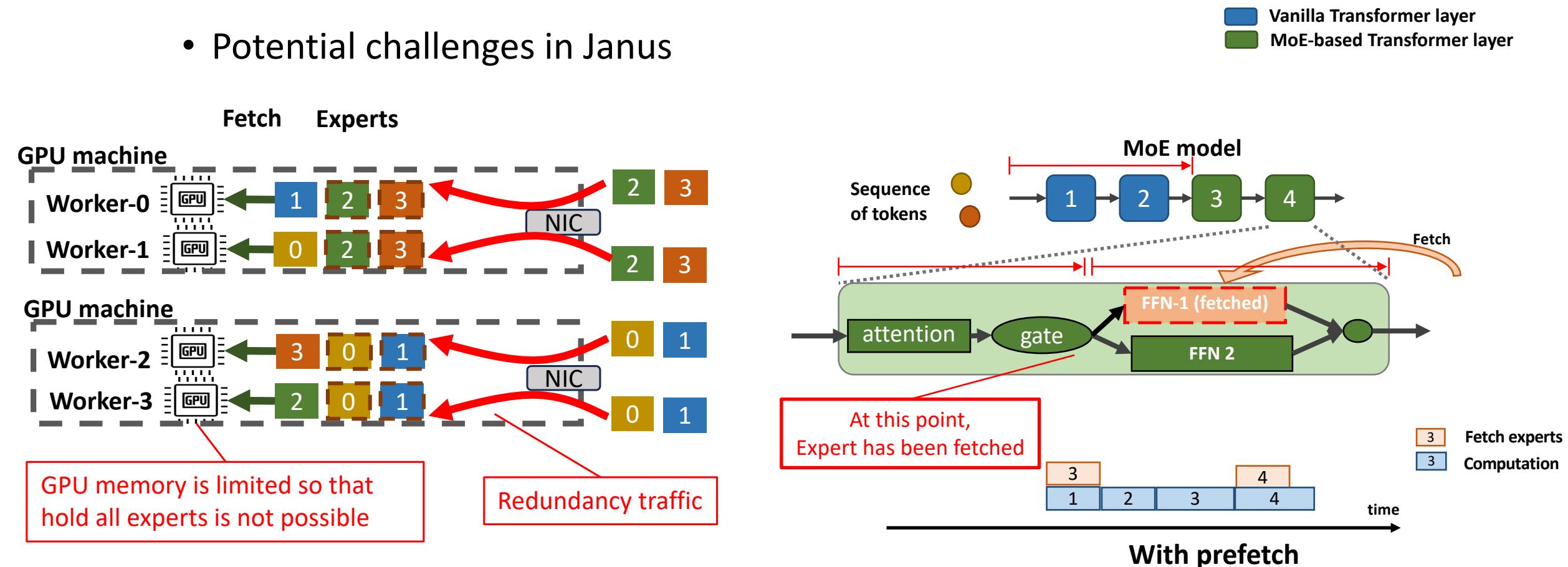


## Our solution: Data Centric



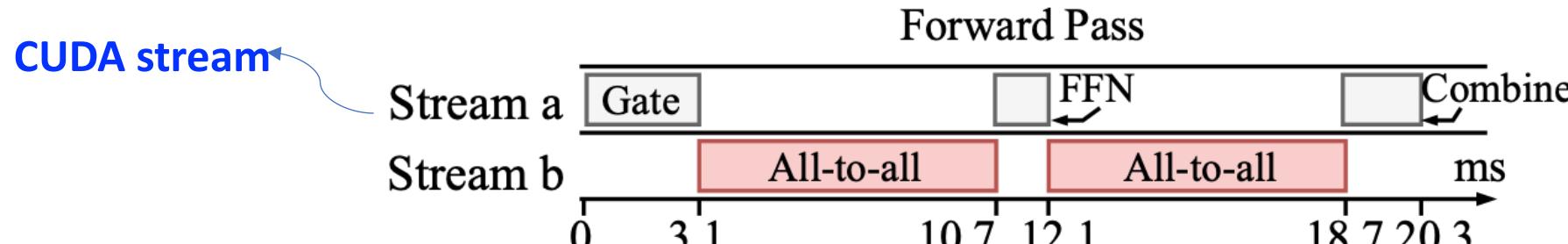
# Mixture of Experts

- Optimizing MoE All-to-All Communication
  - Potential challenges in Janus

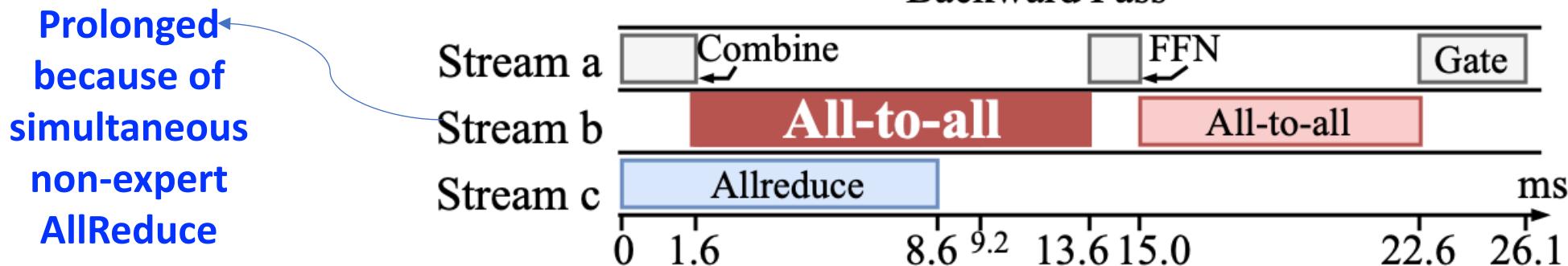


# Mixture of Experts

- Optimizing MoE All-to-All Communication
  - Lina (2023): forward pass is synchronous and blocking

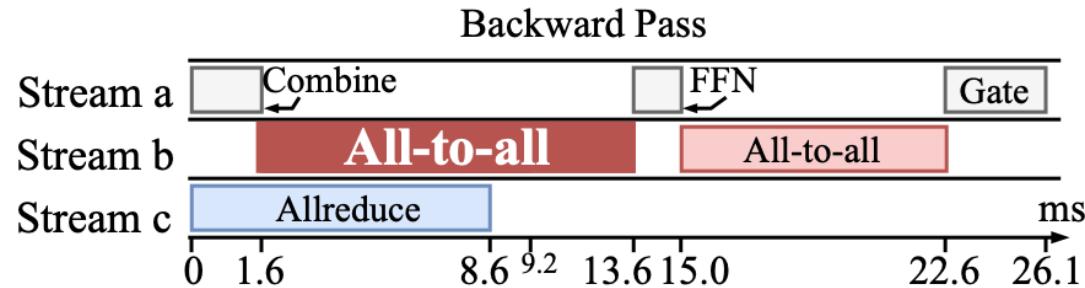


- Lina: backward pass is

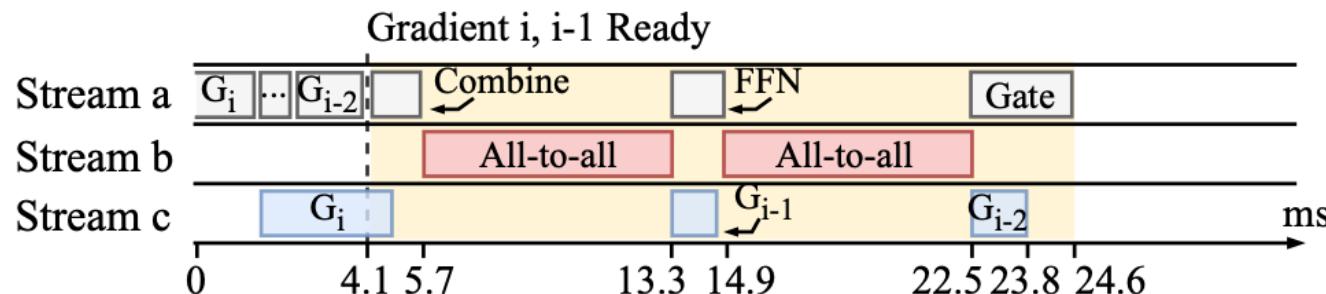


# Mixture of Experts

- Optimizing MoE All-to-All Communication
  - Lina: separate CUDA streams for the expert-parallel and data-parallel groups



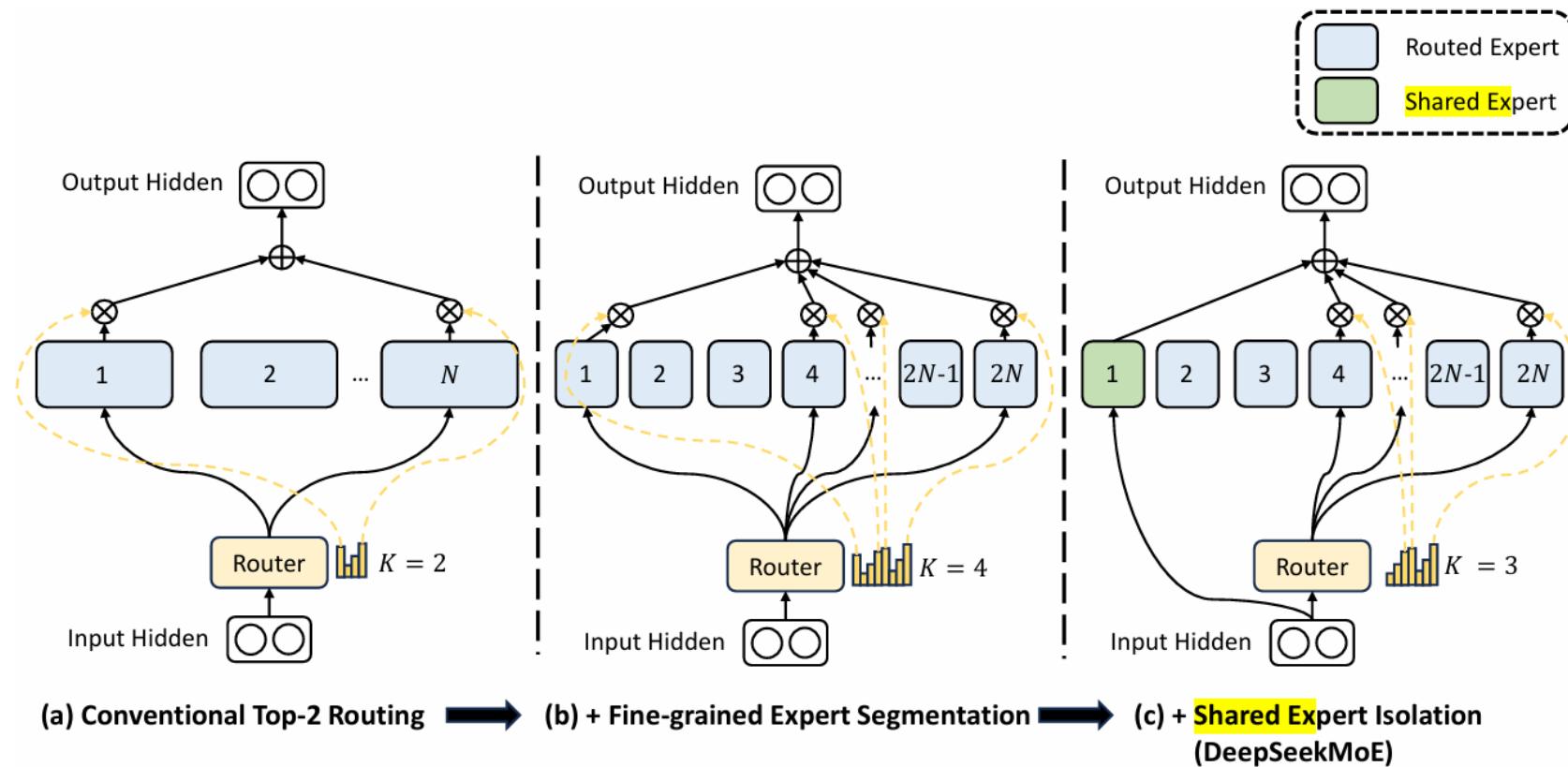
- Lina: always prioritize all-to-all and avoid bandwidth sharing



- More recent approaches: FSMoE (2025), ScheMoE (2024), etc.

# Mixture of Experts

- Overlapping MoE All-to-All in DeepSeek v3



DeepSeek v3: 61 layers, 671B parameters, **256 routing experts**, **Top-8 selection**, 37B parameters activated by one token

# Mixture of Experts

- Overlapping MoE All-to-All in DeepSeek v3

Token-expert affinity:  $s_{i,t} = \text{Sigmoid}(u_t^T e_i)$



Token-expert affinity:  $s_{i,t} = \text{Sigmoid}(u_t^T e_i) + b_i$

Adjusting  $b_i$  dynamically

Token-level load balancing

The Shenzhen Loop Area Institute (SLAI) is a premier global institution dedicated to artificial intelligence, serving as a national pilot for innovative AI talent cultivation under the auspices of China's Ministry of Education.

Sequence-level load balancing

Each token can be sent to at most 4 machines

Cross-machine restriction

# Mixture of Experts

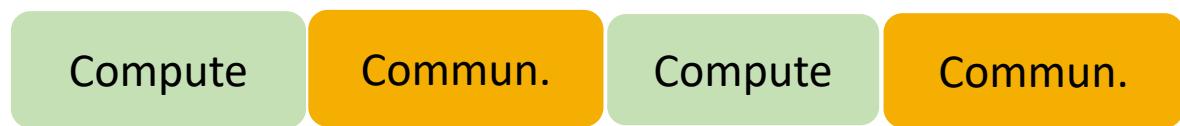
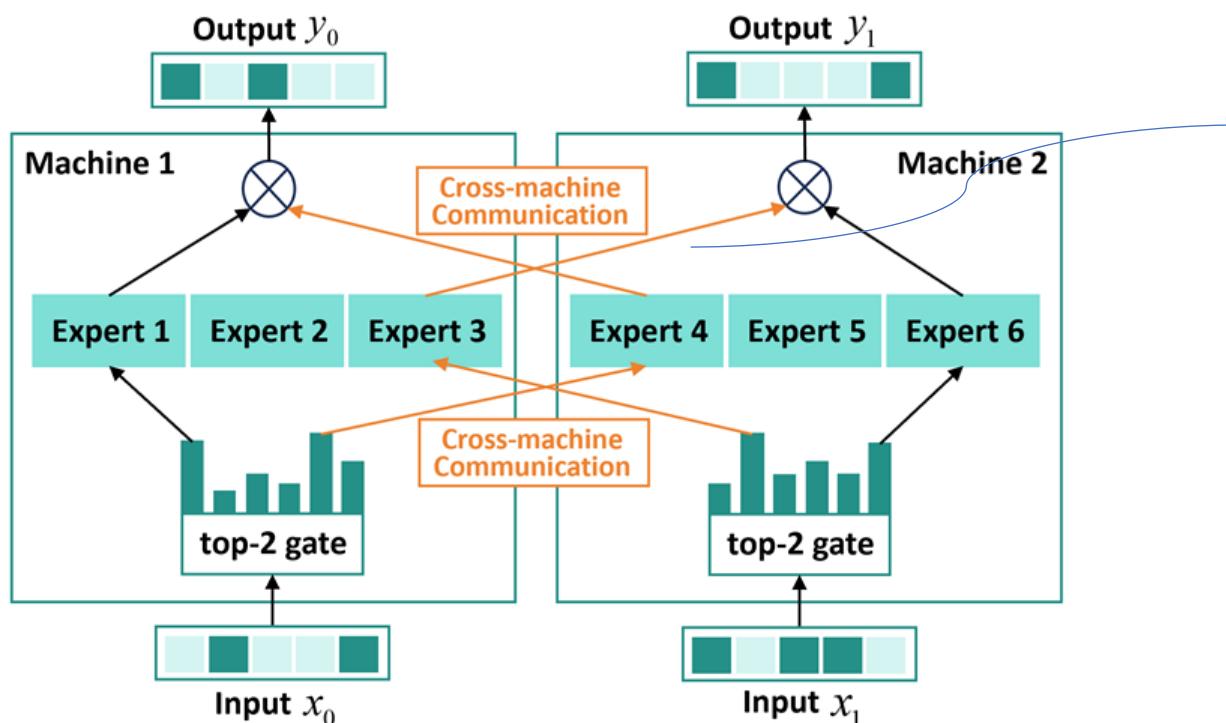
- Overlapping MoE All-to-All in DeepSeek v3
  - 1F1B pipeline: forward pass followed by the immediate backward pass
  - **ZERO bubble**: decomposing backward into backwards for input and parameter
  - **Chimera**: bidirectional pipeline for squeezing bubbles



DualPipe scheduling for 8 PP ranks and 20 micro-batches in two directions. The micro-batches in the reverse direction are symmetric to those in the forward direction

# Mixture of Experts

- Overlapping MoE All-to-All in DeepSeek v3
  - Why All-to-All inter-machine communication can hardly be hidden?



When one model is stored, and what if two models are stored?



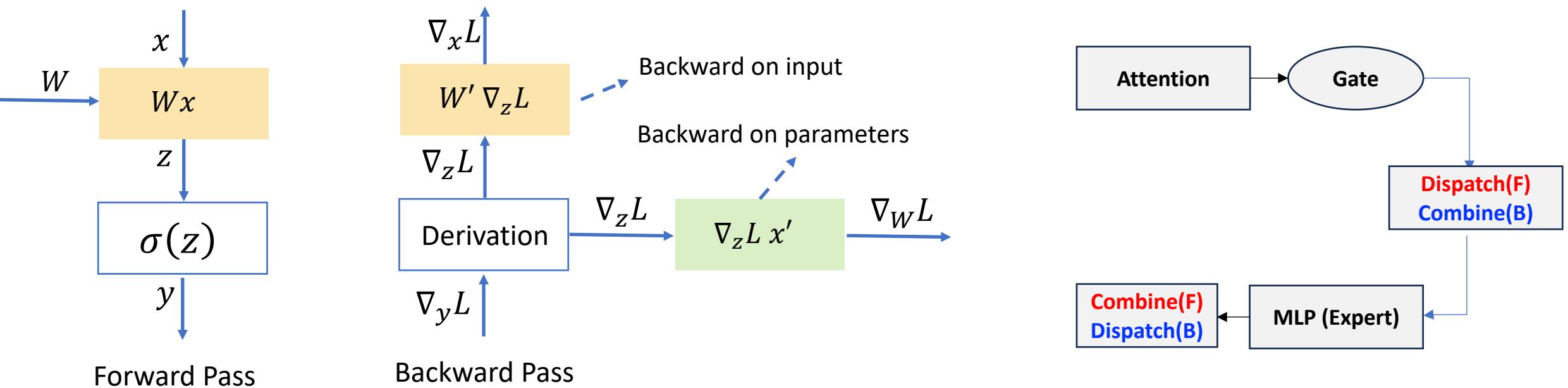
MODEL REPLICA A



MODEL REPLICA B

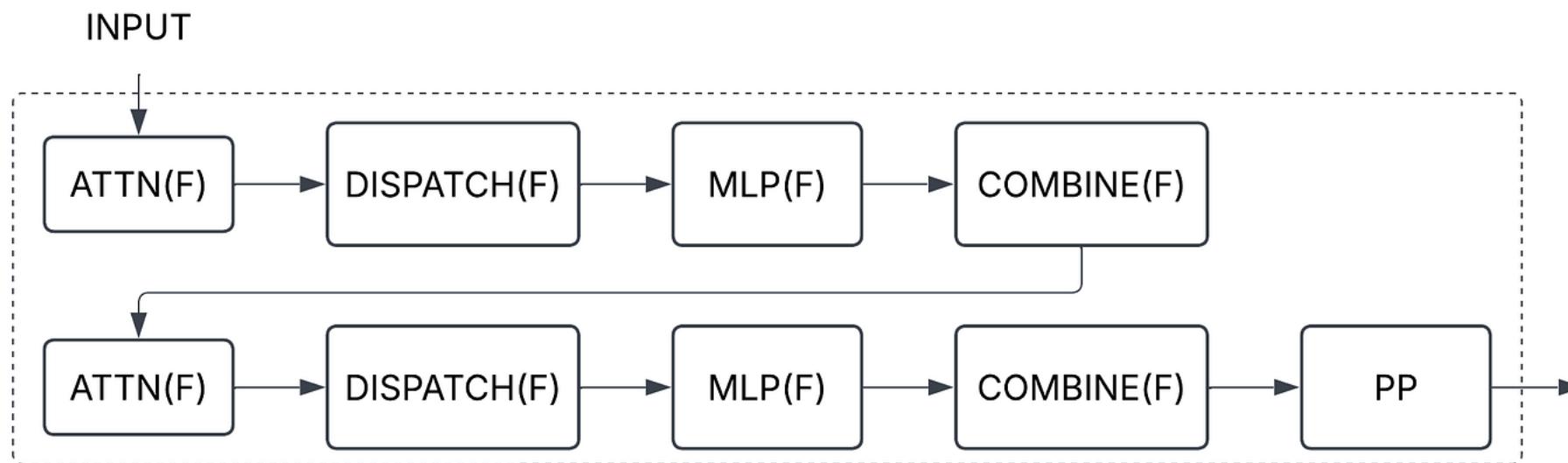
# Mixture of Experts

- Overlapping MoE All-to-All in DeepSeek v3
  - Decomposing computations into  $\text{ATTN}(F)$ ,  $\text{MLP}(F)$ ,  $\text{ATTN}(BI)$ ,  $\text{ATTN}(BW)$ ,  $\text{MLP}(BI)$ ,  $\text{MLP}(BW)$
  - Differentiating  $\text{DISPATCH}(F)$ ,  $\text{DISPATCH}(B)$ ,  $\text{COMBINE}(F)$ ,  $\text{COMBINE}(B)$
  - $\text{DISPATCH}(B)$  after  $\text{MLP}(BI)$ ,  $\text{COMBINE}(B)$  after  $\text{ATTN}(BI)$



# Mixture of Experts

- Overlapping MoE All-to-All in DeepSeek v3
  - Forward pass

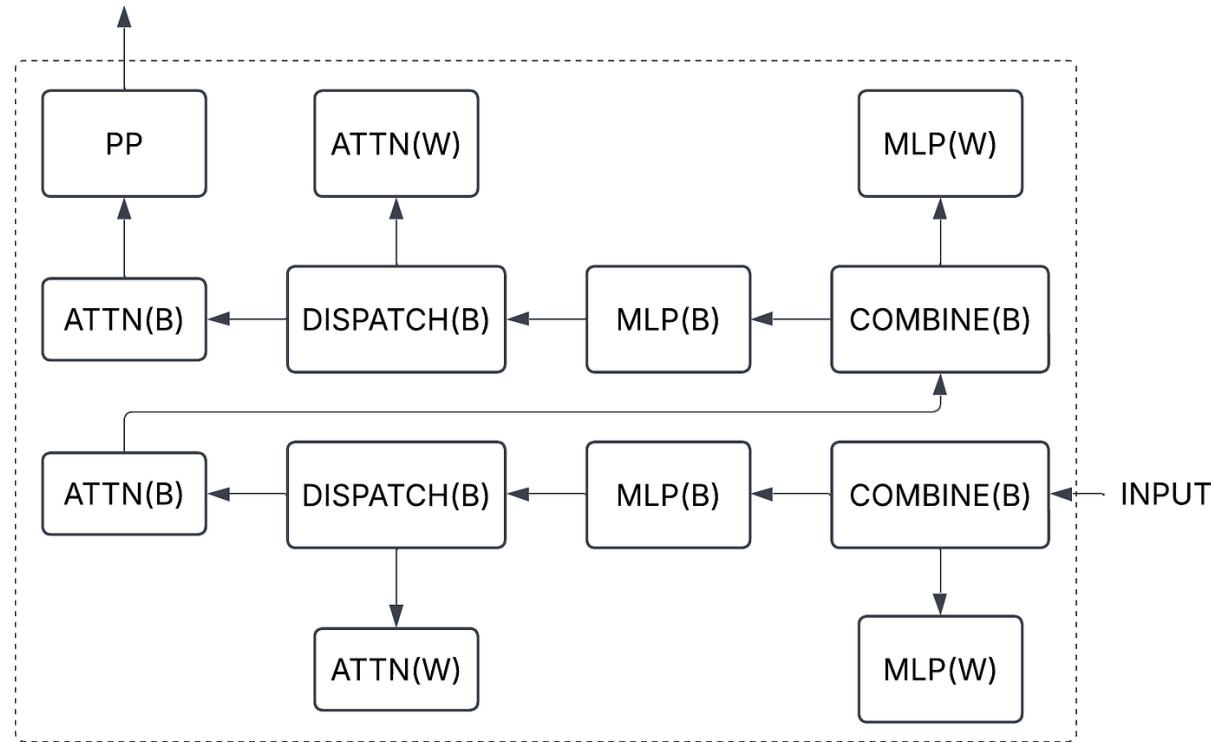


Decouple compute and communication as much as possible

# Mixture of Experts

- Overlapping MoE All-to-All in DeepSeek v3

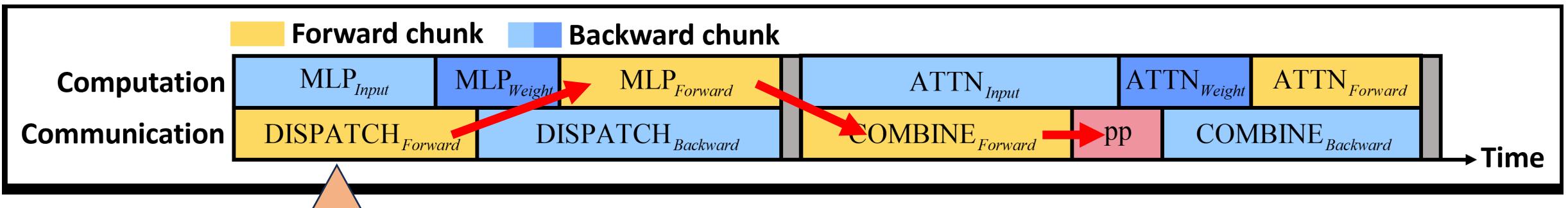
- Backward pass



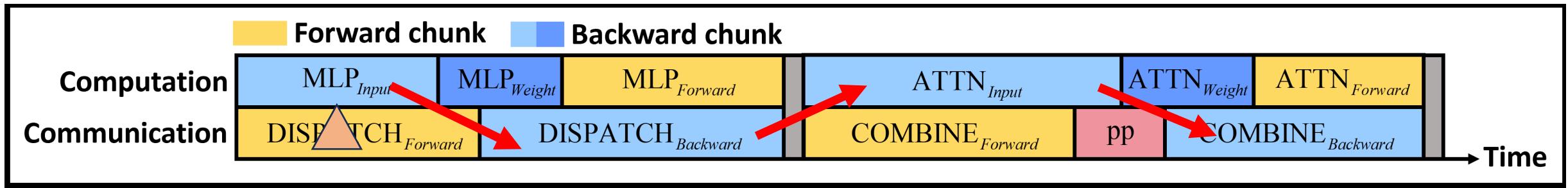
Decouple compute and communication as much as possible

# Mixture of Experts

- Overlapping MoE All-to-All in DeepSeek v3
  - Forward pass

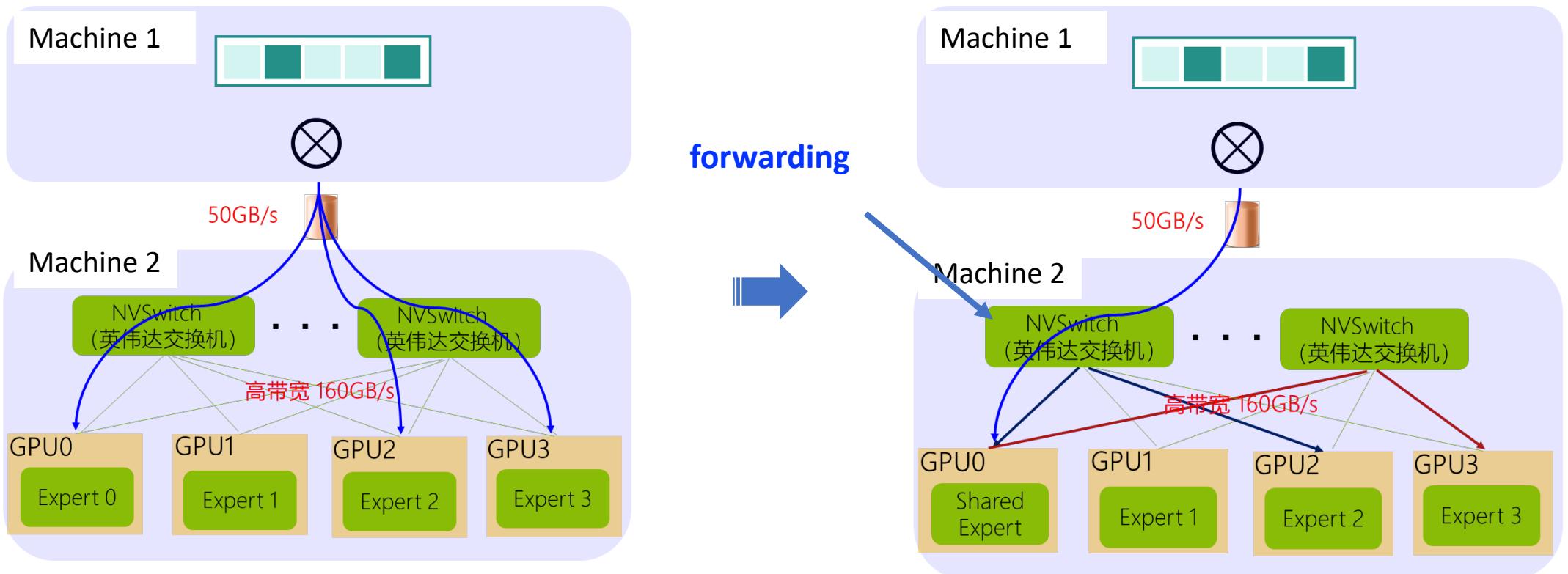


- Backward pass



# Mixture of Experts

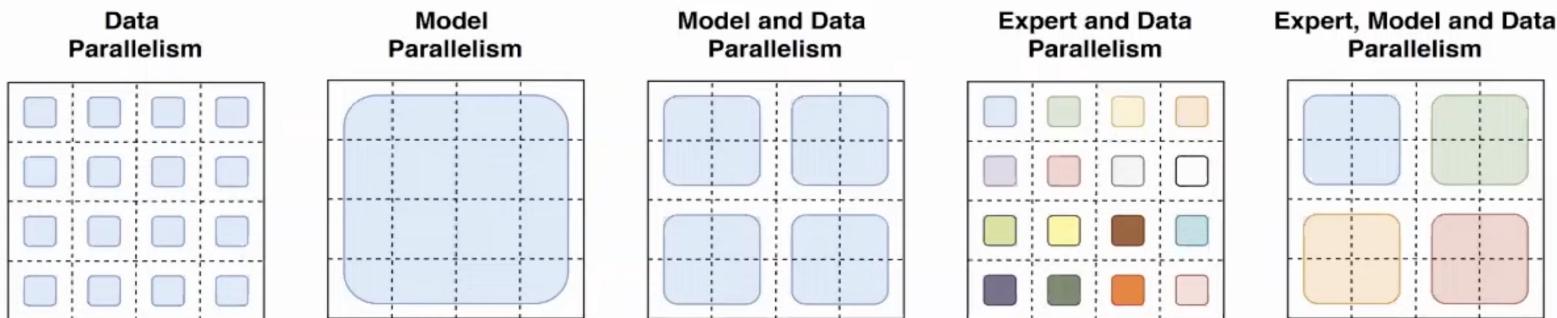
- Overlapping MoE All-to-All in DeepSeek v3
  - Token transfer: forwarding from RDMA to NVLink by use of asymmetric bandwidth



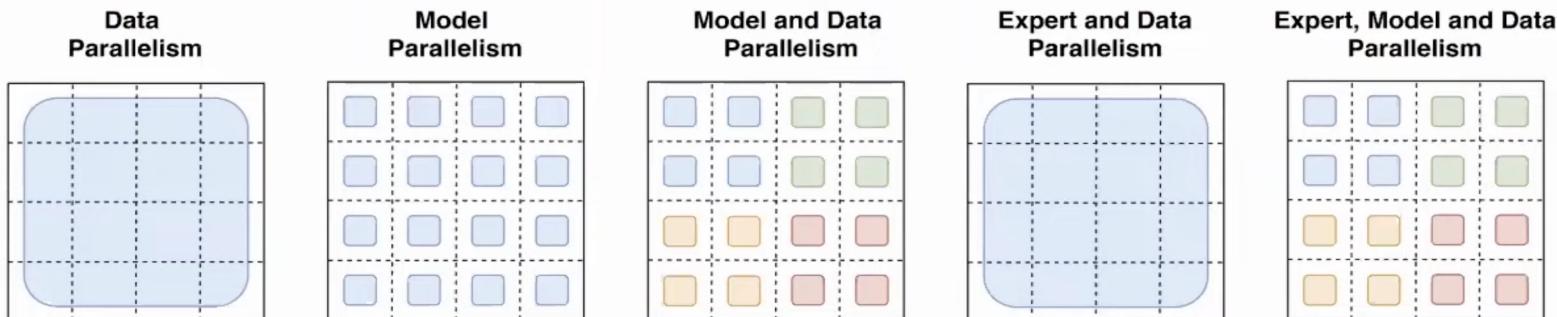
# Hybrid Parallelism

- Segmenting data, model and activation

How the *model weights* are split over cores



How the *data* is split over cores



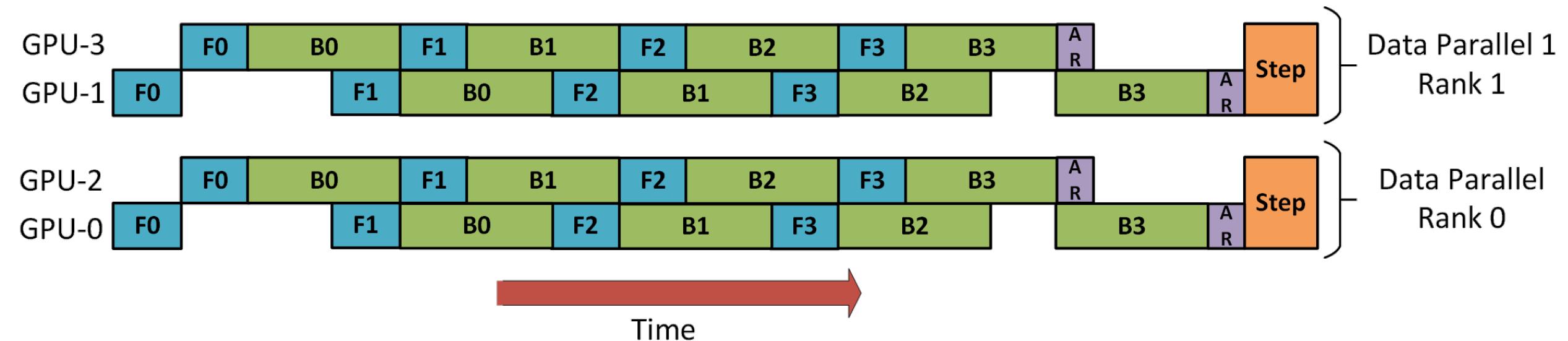
# A Brief Summary

- Hybrid parallelism (classical models)

Models	TP	DP	PP	Micro batchsize	global batchsize	sequence length	ZeRO	GPUs	GPU types
Bloom-176B	4	8	12	2	2048	2048	ZeRO-1	384	A100-80GB
Megatron-175B	8	\	16	1	1536	2048	\	\	\
OPT-175B	8	124	\	\	\	2048	\	996	A100-80GB
GPT3-175B	\	\	\	\	\	2048	\	\	\
Megatron-NLG 530B	8	16	35	\	1960	2048	\	4480	A100-80GB
GLM-130B	4	24	8	\	\	4096	ZeRO-1	768	A100-80GB

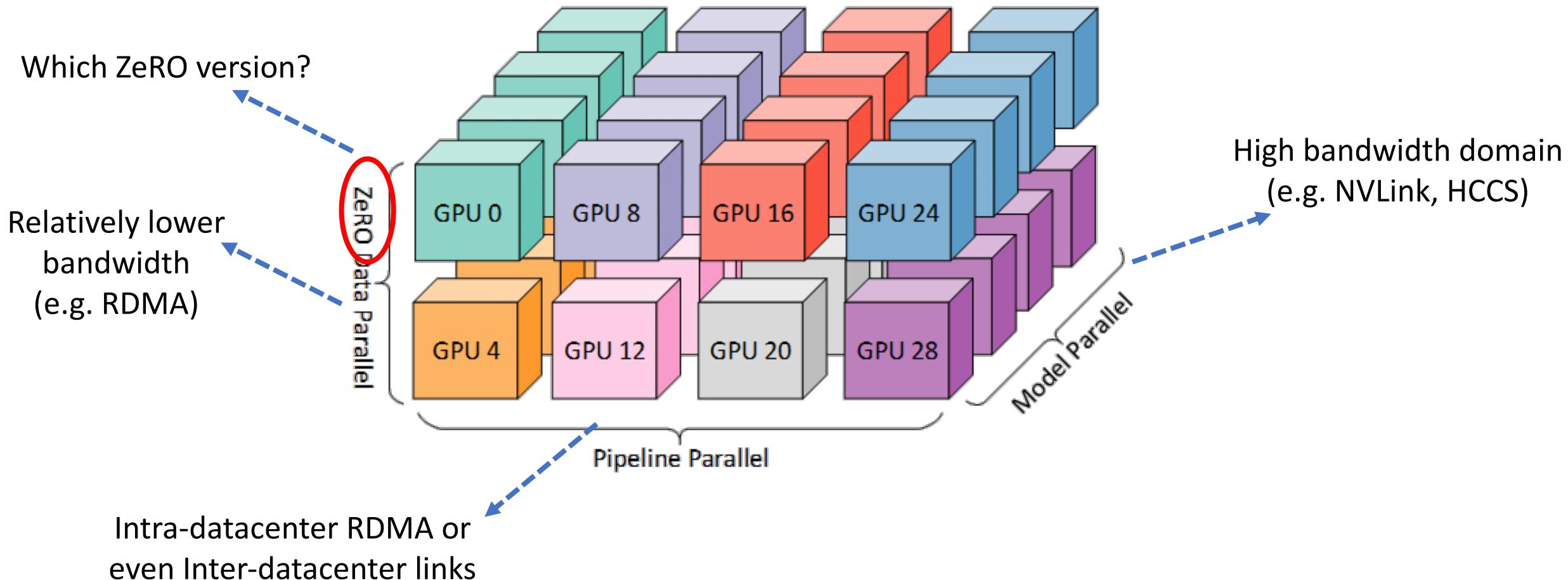
# A Brief Summary

- Data Parallelism + Pipeline Parallelism



# A Brief Summary

- Data Parallelism + Pipeline Parallelism + Tensor Parallelism + ZeRO



Thanks!



## **Backup slides to explain why LLM training OPS is important**

---

**Be Brief**

# Reliability

## Hardware Failure



GPU



Router



SmartNIC

## Software Failure



Out-of-Memory

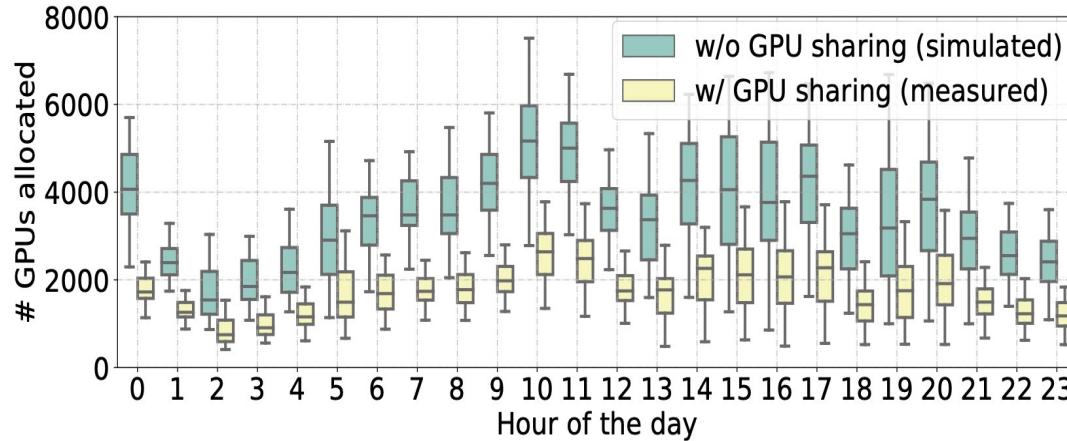


Framework Runtime Error

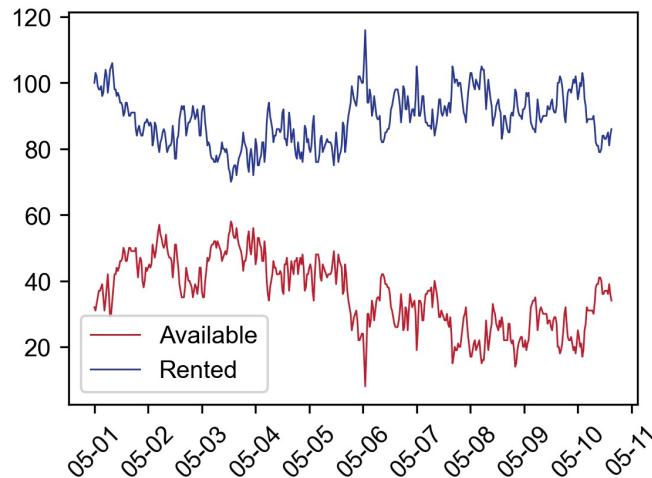
- Meta AI encounters 110 failures when training OPT-175B using 992 A100 GPUs

- Our home-made GPUs and network devices may encounter more failures than NVIDIA systems

# Reliability



Model FLOPS Utilization in a Mega-Scale Cluster

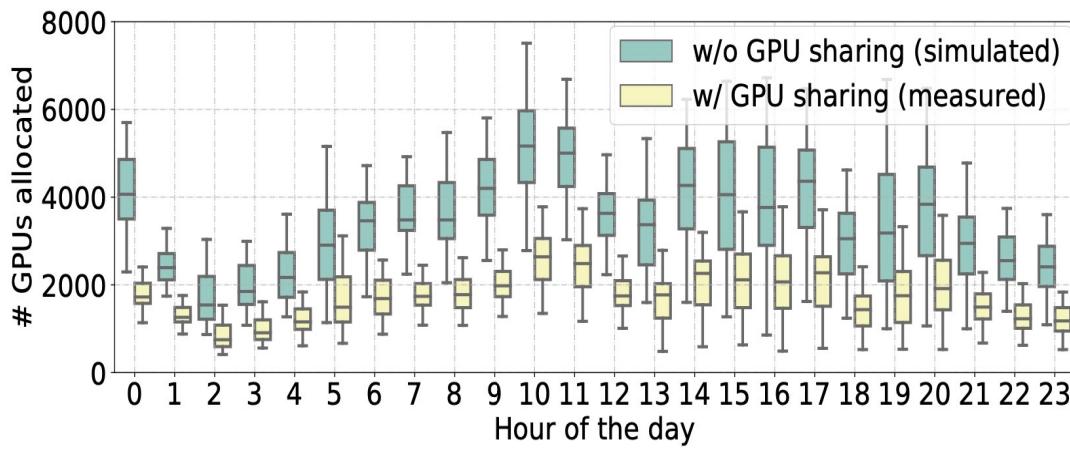


Available A100  
GPUs in VAST.ai

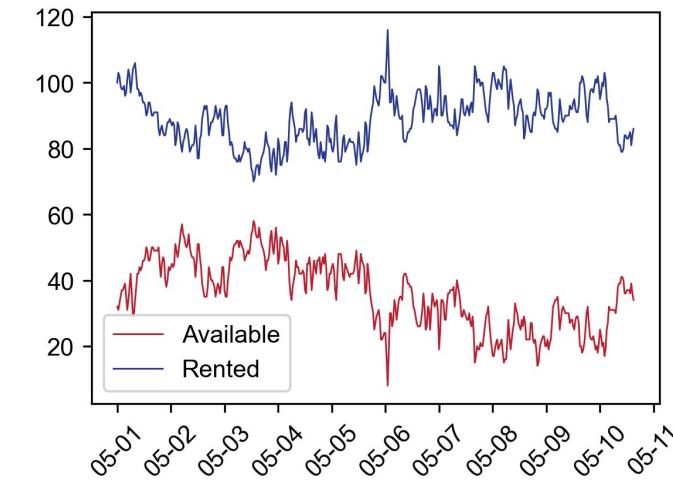
Low-priority GPU users will pause and resume training from time to time

- Dynamic GPU availability
- Variable prices of GPU spot instances

# Reliability



Model FLOPS Utilization in a Mega-Scale Cluster



Available A100 GPUs in VAST.ai

Low-priority GPU users will pause and resume training from time to time

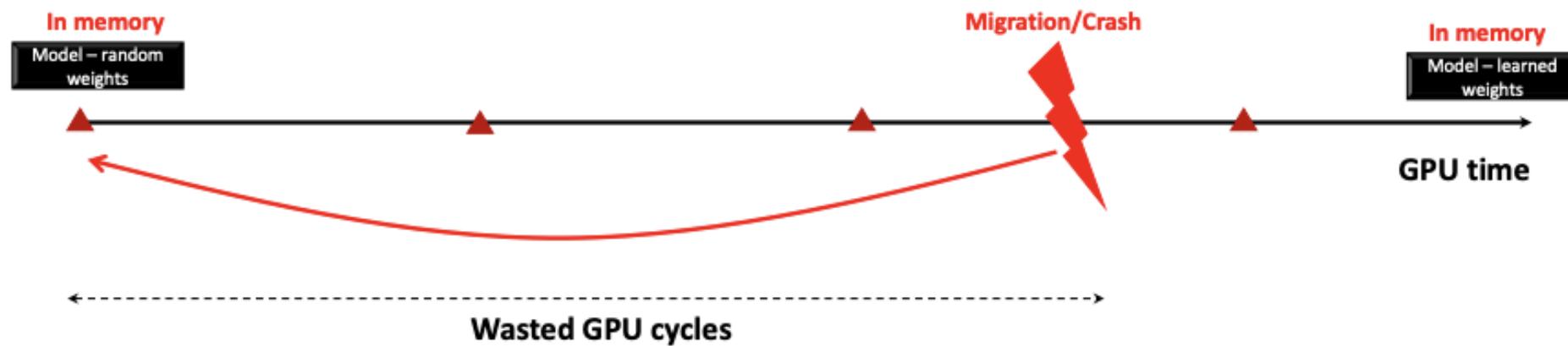
# Reliability

Component	Category	Interrupt Count	% of Interruptions
Faulty GPU	GPU	148	30.1%
GPU HBM3 Memory	GPU	72	17.2%
Software Bug	Dependency	54	12.9%
Network Switch/Cable	Network	35	8.4%
Host Maintenance	Unplanned Maintenance	32	7.6%
GPU SRAM Memory	GPU	19	4.5%
GPU System Processor	GPU	17	4.1%
NIC	Host	7	1.7%
NCCL Watchdog Timeouts	Unknown	7	1.7%
Silent Data Corruption	GPU	6	1.4%
GPU Thermal Interface + Sensor	GPU	6	1.4%
SSD	Host	3	0.7%
Power Supply	Host	3	0.7%
Server Chassis	Host	2	0.5%
IO Expansion Board	Host	2	0.5%
Dependency	Dependency	2	0.5%
CPU	Host	2	0.5%
System Memory	Host	2	0.5%

**Table 5 Root-cause categorization of unexpected interruptions during a 54-day period of Llama 3 405B pre-training.** About 78% of unexpected interruptions were attributed to confirmed or suspected hardware issues.

# Reliability

- Checkpointing



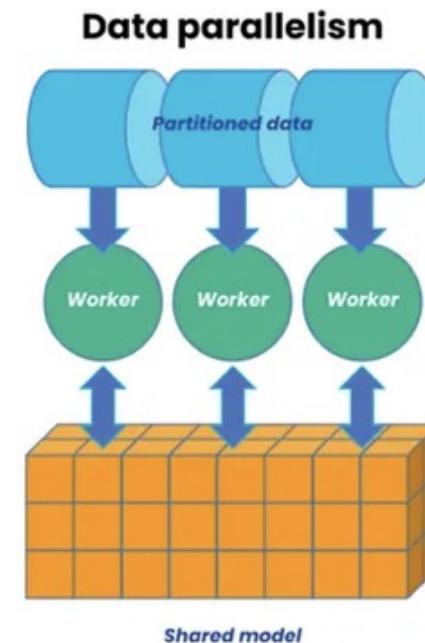
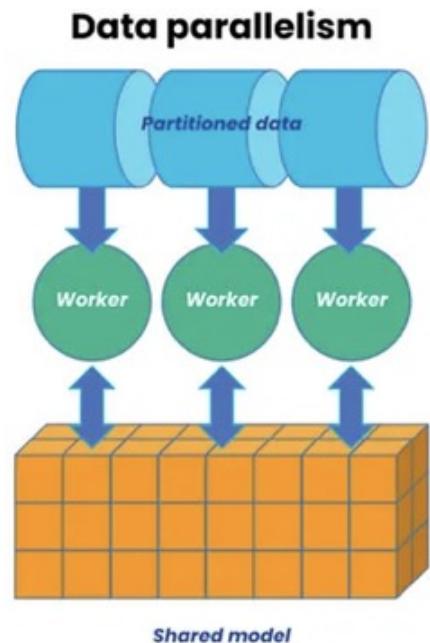
- Frequent (e.g. synchronous) checkpointing causes training stalls
  - How frequent to checkpoint?
  - How to avoid contention with training traffic?
  - How to resume correctly and quickly from a checkpoint?

# Reliability

- Checkpointing: some confusions
  - A checkpoint includes
    - Model parameters
    - Momentum and variance in optimizer states
    - Hyperparameters such as learning rate and random seeds, etc.
  - Is checkpointing really necessary?
    - Data parallelism naturally duplicates model parameters and optimizer states!

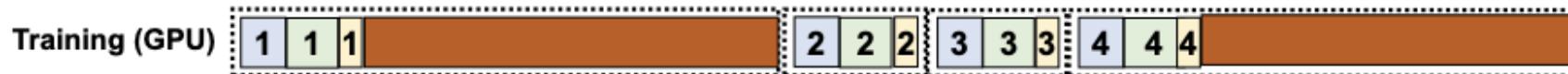
# Reliability

- Checkpointing: some confusions
  - Is checkpointing really necessary?
    - Data parallelism naturally duplicates model parameters and optimizer states!



# Reliability

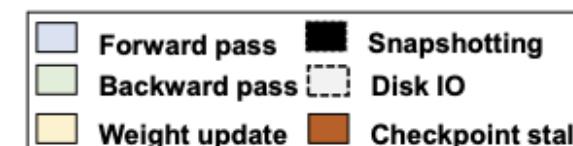
- Checkpointing
  - Early trials: CheckFreq



(a) Baseline : Synchronous checkpointing



(b) Only persist() pipelining



(c) Snapshot() and persist() pipelining

- Snapshot: Serialize and copy into a user-space buffer
- Persist: Write out the serialized contents to disk

# Reliability

- Checkpointing
  - Early trials: CheckFreq
    - To further reduce the checkpoint cost, CheckFreq snapshots on the GPU, and asynchronously writes it to CPU memory if it profiles spare memory on the GPU

Model	Parameters	Checkpoint size	Checkpoint time (20Gbps)
Gopher [56]	280B	3.4 TB	23 min
MT-NLG [62]	530B	6.4 TB	43 min
PaLM [23]	540B	6.5 TB	44 min

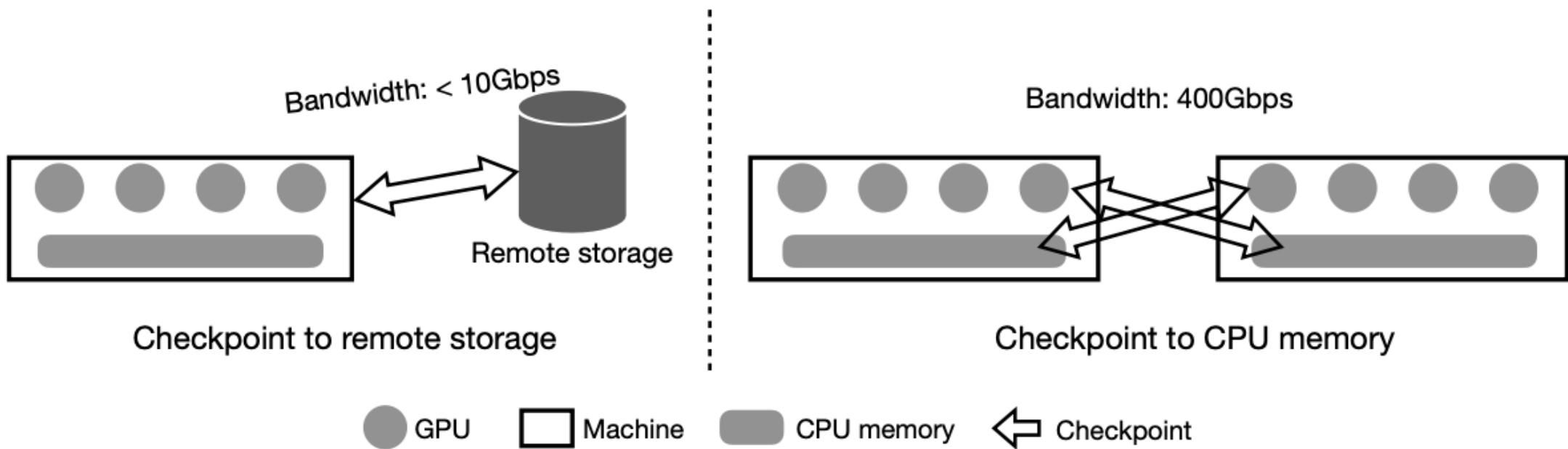
# Reliability

- Checkpointing
  - Early trials: GEMINI
    - As model grows large, checkpoint frequency is throttled by checkpointing time
    - CPU memory is much larger than GPU memory

Instance type	Cloud	GPU	GPU memory	CPU memory
p3dn.24xlarge [14]	AWS	8 V100	8 × 32 GB	768 GB
p4d.24xlarge [15]	AWS	8 A100	8 × 40 GB	1152 GB
ND40rs_v2 [10]	Azure	8 V100	8 × 32 GB	672 GB
ND96asr_v4 [11]	Azure	8 A100	8 × 40 GB	900 GB
n1-8-v100 [9]	GCP	8 V100	8 × 32 GB	624 GB
a2-highgpu-8g [9]	GCP	8 A100	8 × 40 GB	640 GB
DGX A100 [12]	NVIDIA	8 A100	8 × 80 GB	2 TB

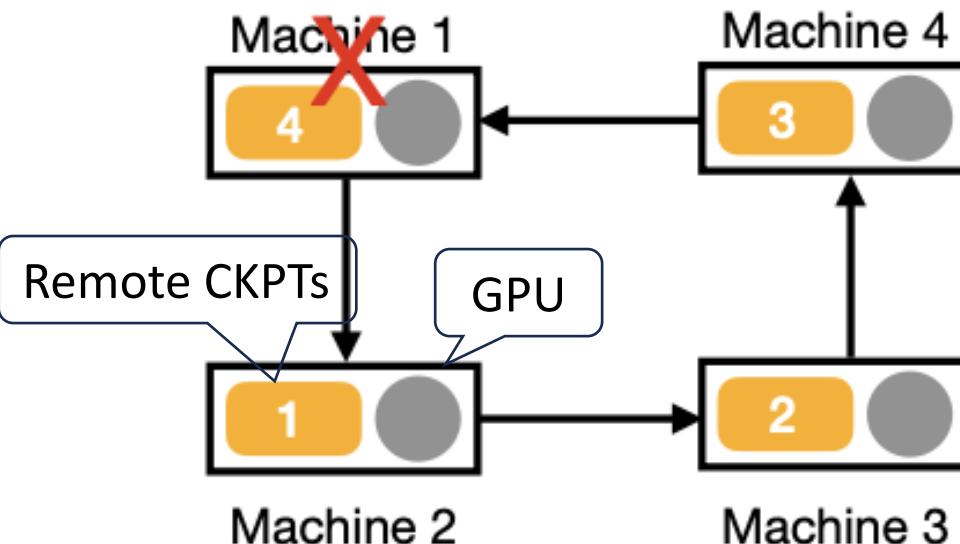
# Reliability

- Checkpointing
  - Early trials: GEMINI

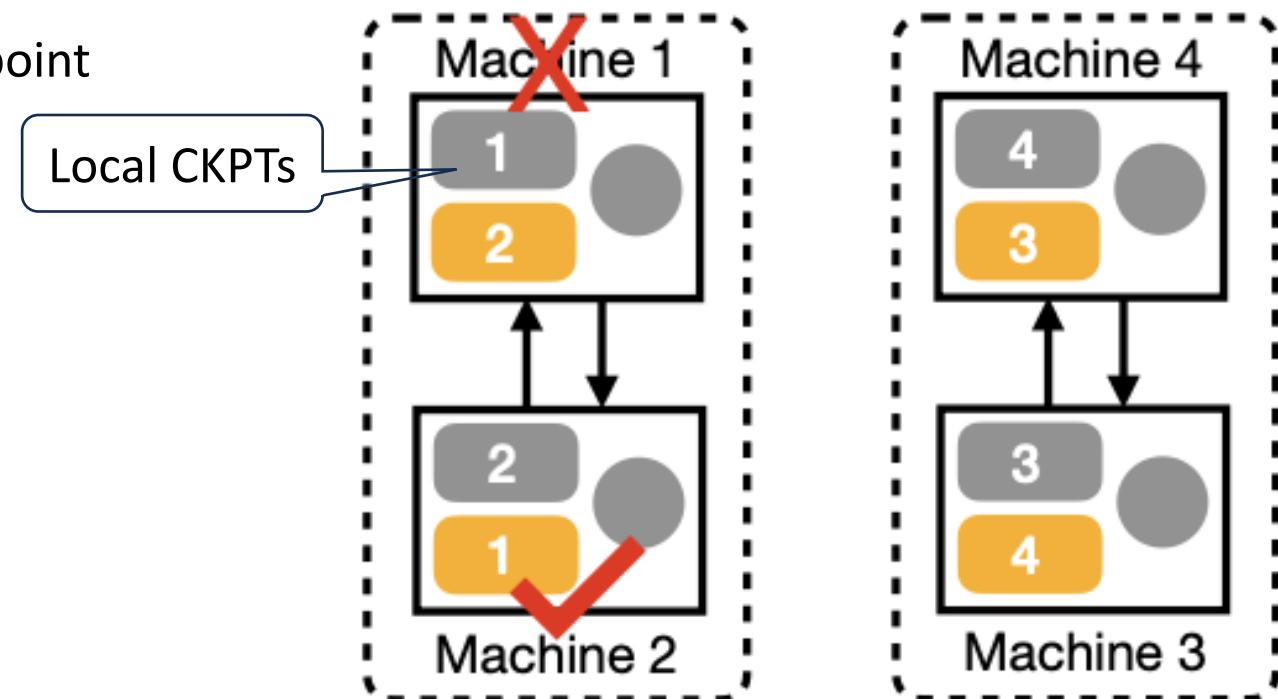


# Reliability

- Checkpointing
  - Early trials: GEMINI
    - How many shards to checkpoint



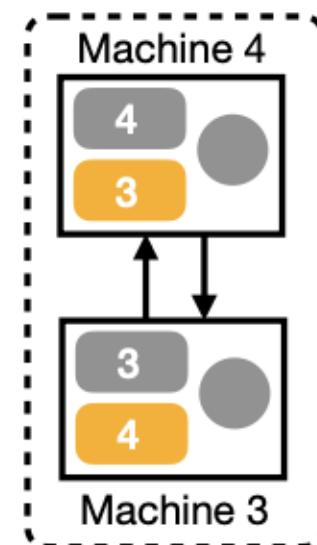
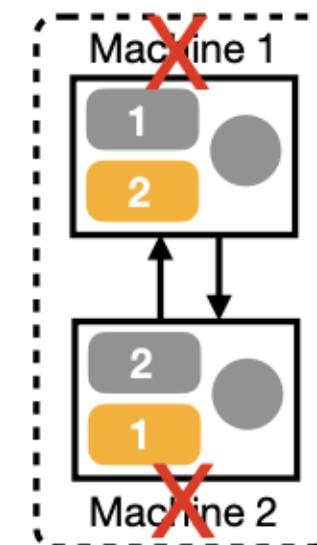
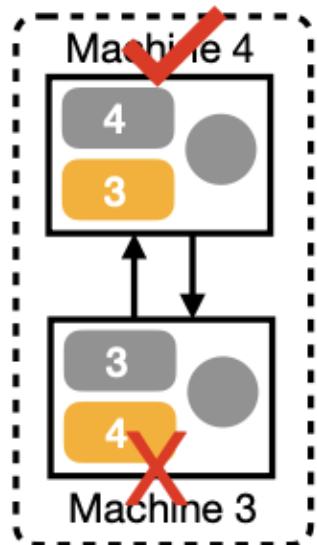
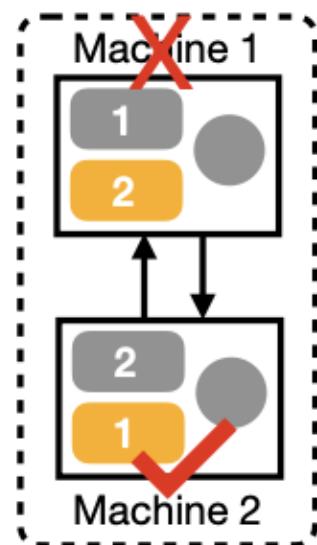
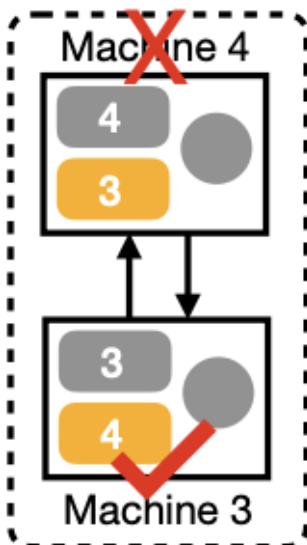
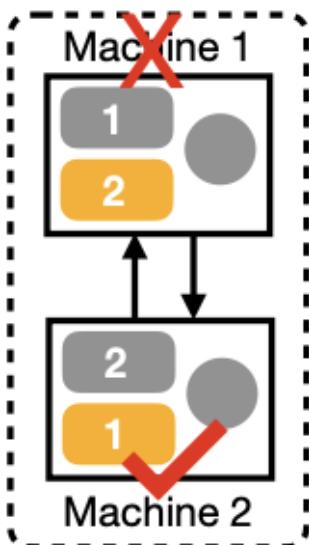
Checkpointing: one remote replica with one machine failure



Checkpointing: one remote replica and one local replica

# Reliability

- Checkpointing
  - Early trials: GEMINI
    - How to distribute different shards in the presence of multiple simultaneous failures



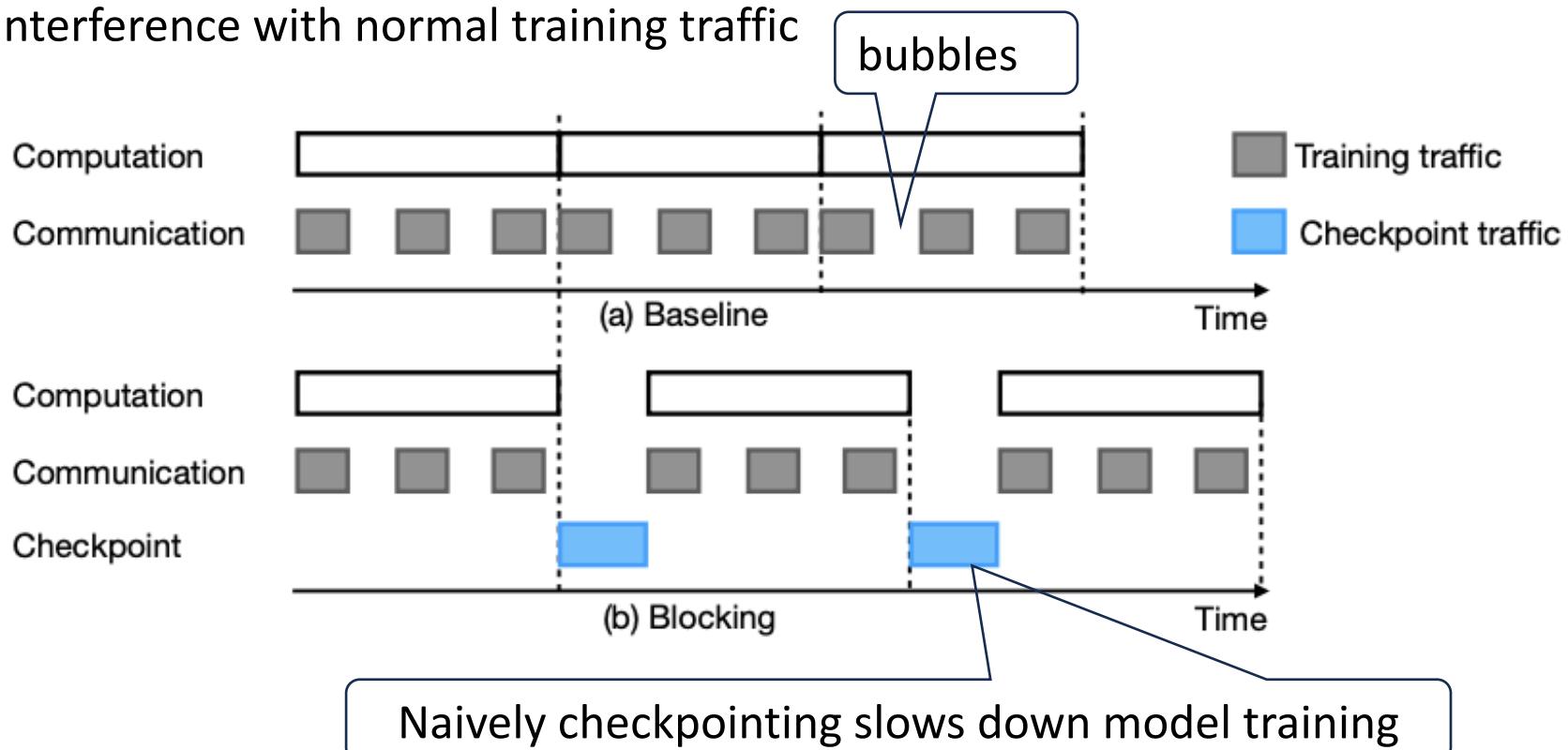
Successful recovery

Successful recovery

Incomplete CKPTs: when all machines  
in a group fail simultaneously

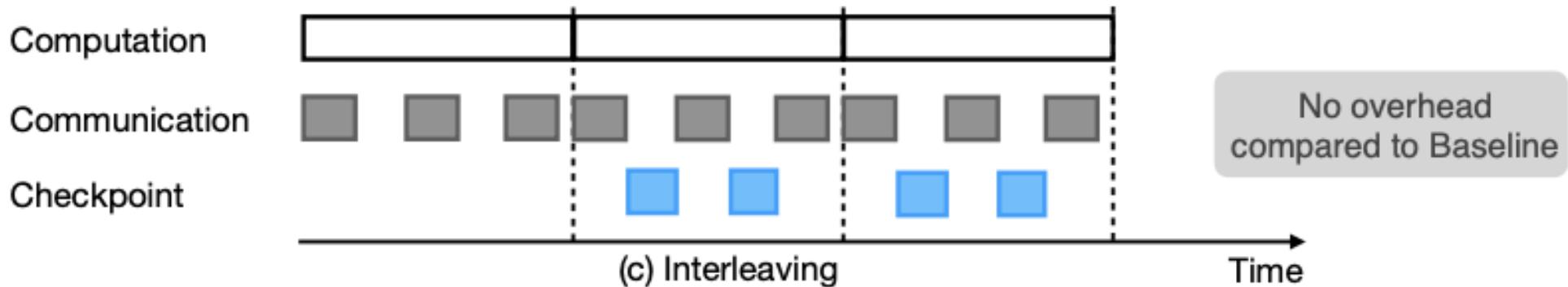
# Reliability

- Checkpointing
  - Early trials: GEMINI
    - Interference with normal training traffic



# Reliability

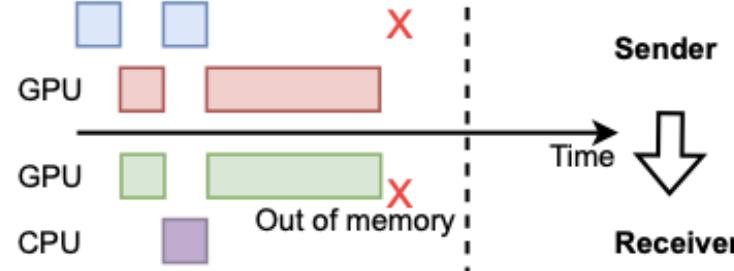
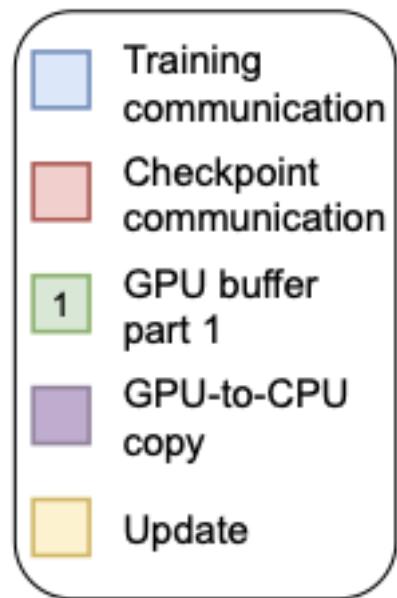
- Checkpointing
  - Early trials: GEMINI
    - Idea checkpointing: inserting checkpoint transmission in the bubbles



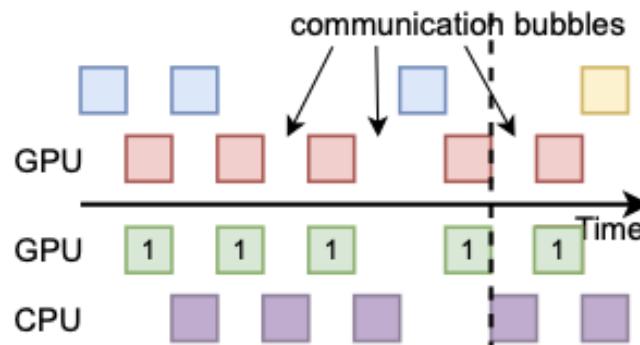
- Methodology
  - GPU-to-GPU communication (GDR: GPU Direct over RDMA)
  - Application-layer scheduling of chunked checkpoint transmission
  - Proper management of GPU memory for avoiding out-of-memory (OOM)

# Reliability

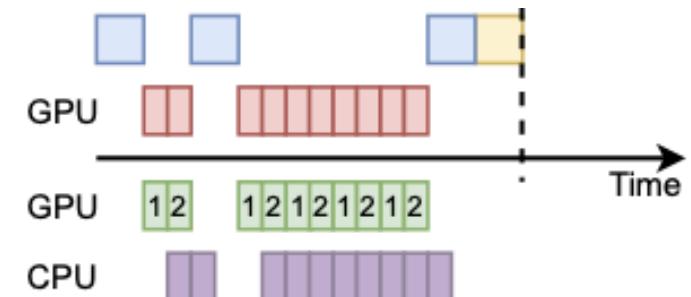
- Checkpointing
  - Early trials: GEMINI
    - Idea checkpointing: inserting checkpoint transmission in the bubbles



Even if communication is not blocked, storing a replica may cause OOM



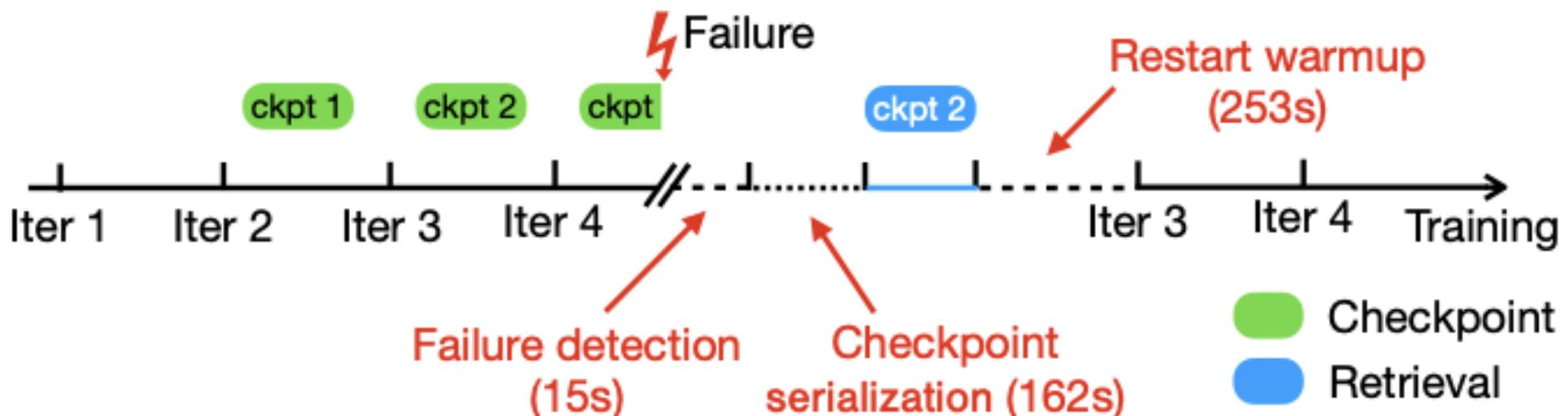
Even if checkpoint is partitioned, the slow GPU-to-CPU copies block checkpoint communication



Do everything in a pipeline: receiving incoming checkpoint chunks and copying them to CPU memory at the same time

# Reliability

- Checkpointing
  - Early trials: GEMINI
    - Fast recovery (training GPT2 100B parameters)



# Reliability

- Checkpointing
  - Recent advances
    - Further reducing checkpointing cost/improving checkpointing frequency
  - Just-in-Time checkpointing
    - Intercept API calls to the synchronization primitives such as `cudaStreamWaitEvent`, `cudaEventRecord` and a few NCCL APIs
    - Failure recovery
      - Errors during forward and backward passes
      - Errors during optimizer step
      - Errors due to GPU hardware