LLM Sys

11868/11968
Large models with Mixture-of-Experts

Lei Li

Carnegie Mellon University
Language Technologies Institute

Outline

- Transformer Mixture-of-Expert Model
  - Switch Transformer architecture
  - Shared-routed Experts
- Training and inference for MoE
  - Expert parallelism (GShard)
- Deepseek MoE (V3 model)
  - code walkthrough

Motivation: Scaling for Dense model is hard

- Background: Compute is the primary challenge of training massive models.

Model	Model Size	Hardware	Days to Train
Megatron-LM GPT-2	8.3B	512 V100 GPU	9.2 days
OPT	175B	992 A100 GPU	56 days
MT-NLG	530B	2200 A100 GPU	60 days
PaLM	540B	6144 TPU v4	57 days

Sparse model is a promising path for improved model quality without increasing training cost, e.g. MOE

Need for Sparse Model

- Dense model is hard to scale, while sparse model scales to larger models

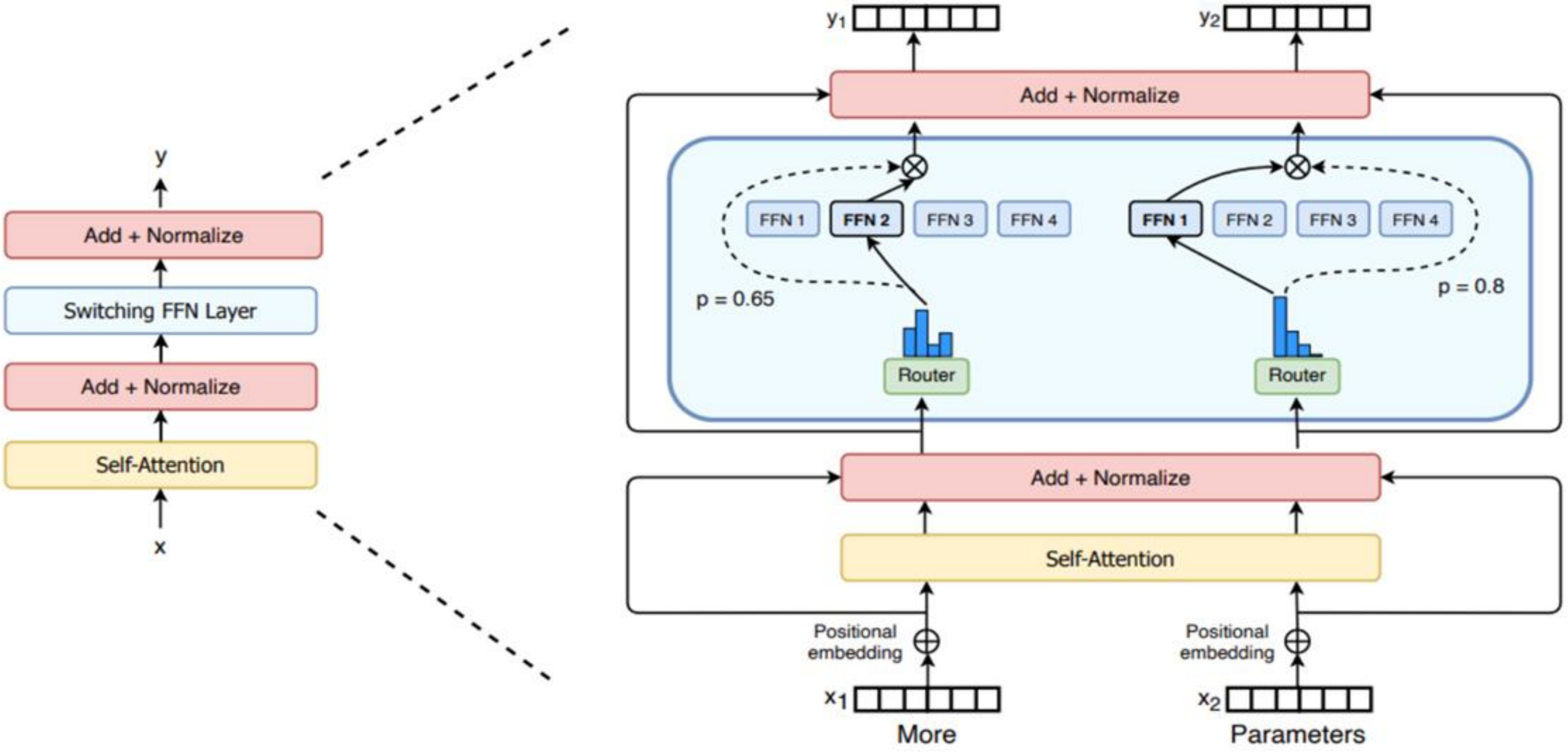
- Mixture-of-Expert is one type of sparse model
  - pretraining is much faster vs. dense models
  - MOE is faster in inference compared to a model with the same number of parameters

Transformer Mixture of Expert Model

- Replacing Transformer’s FFN with
  - multiple small experts, each expert is a neural network (e.g. FFNs)
  - a gating network to choose which expert to activate based on input token
- Not to be confused with Mixture-of-Expert learning, which is a learning algorithm to learn the weighted average of predictor models

Transformer MoE (Switch Transformer)

one token is only passed through one selected FFN



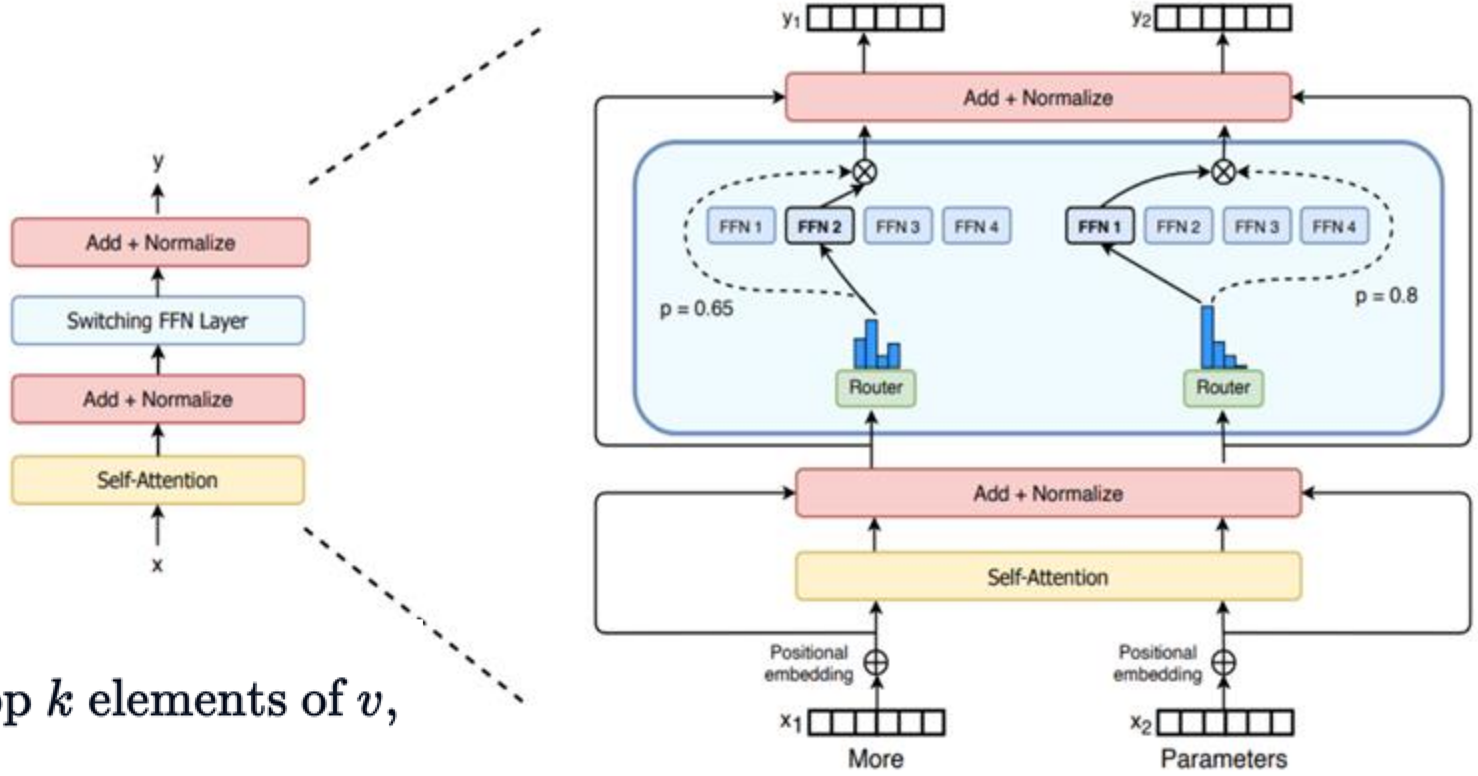
Fedus et al. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. JMLR 2022.

Transformer MoE (Switch Transformer)

- Gating network (G) learns which experts (E) to send a part of the input:

$$G_\sigma(x) = \text{Softmax}(x \cdot W_g)$$

$$y = \sum_{i=1}^{n} G(x)_i E_i(x)$$



Top-k gating:

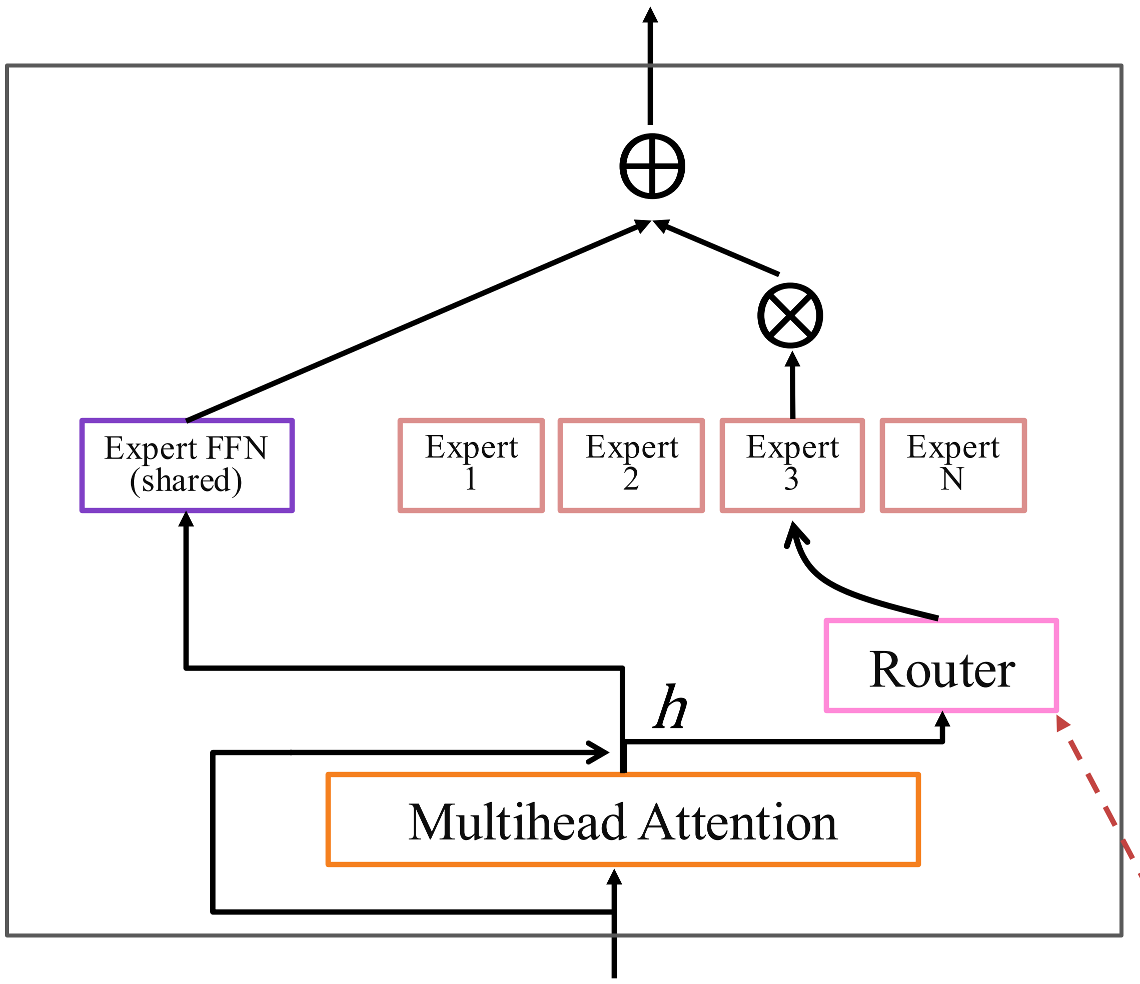
$$\text{KeepTopK}(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v, \\ -\infty & \text{otherwise.} \end{cases}$$

$$G(x) = \text{Softmax}(\text{KeepTopK}(H(x), k))$$

Fedus et al. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. JMLR 2022.

Shared vs. Routed Experts

always pass through one fixed expert FFN



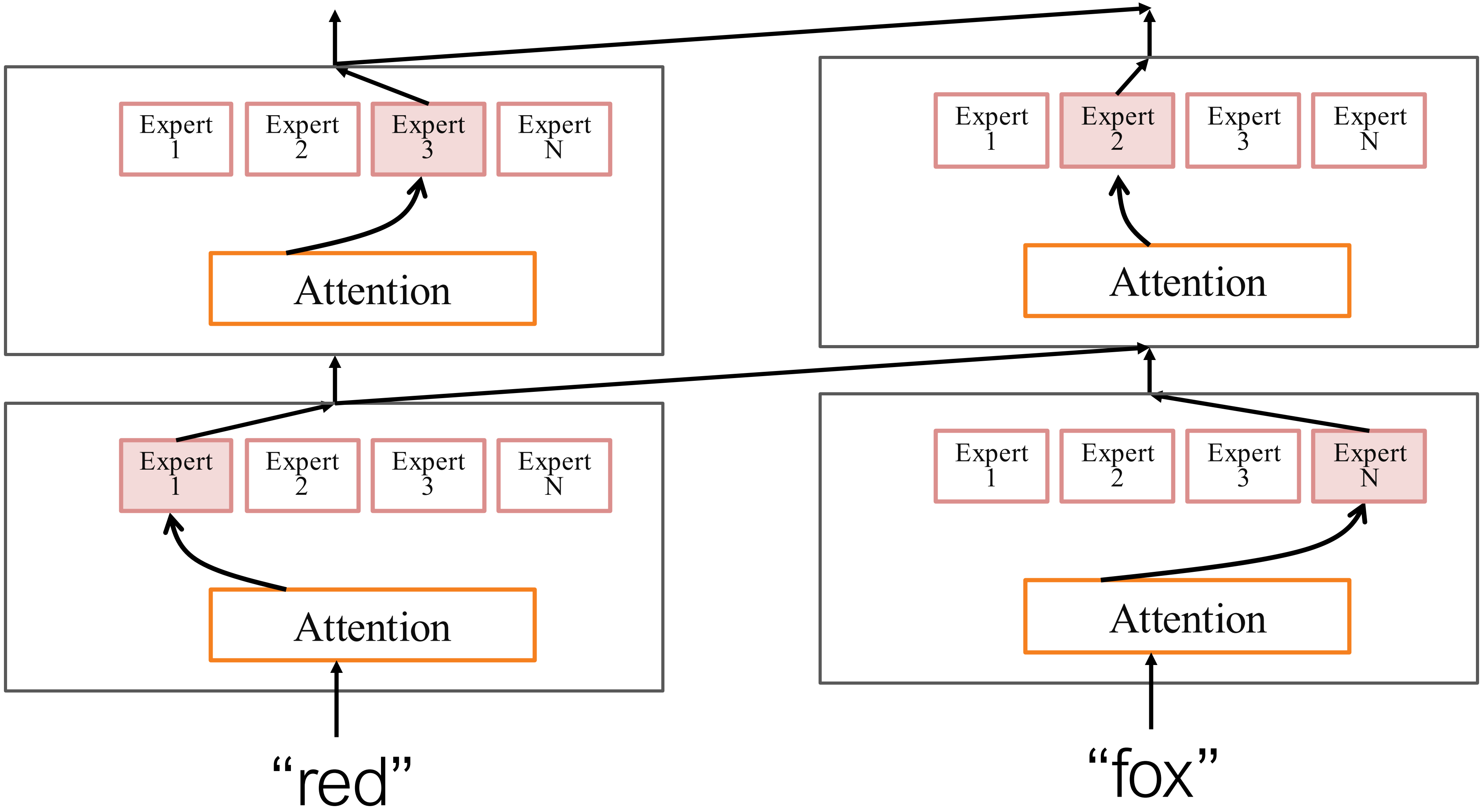
Shared expert: calculating common knowledge

Routed experts: calculated token-specific knowledge. First from Deepspeed-MoE. later in deepseek MoE

$\mathrm{Softmax}(\mathrm{TopK}(h_t \cdot W))$

Rajbhandari et al (2022). DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale.

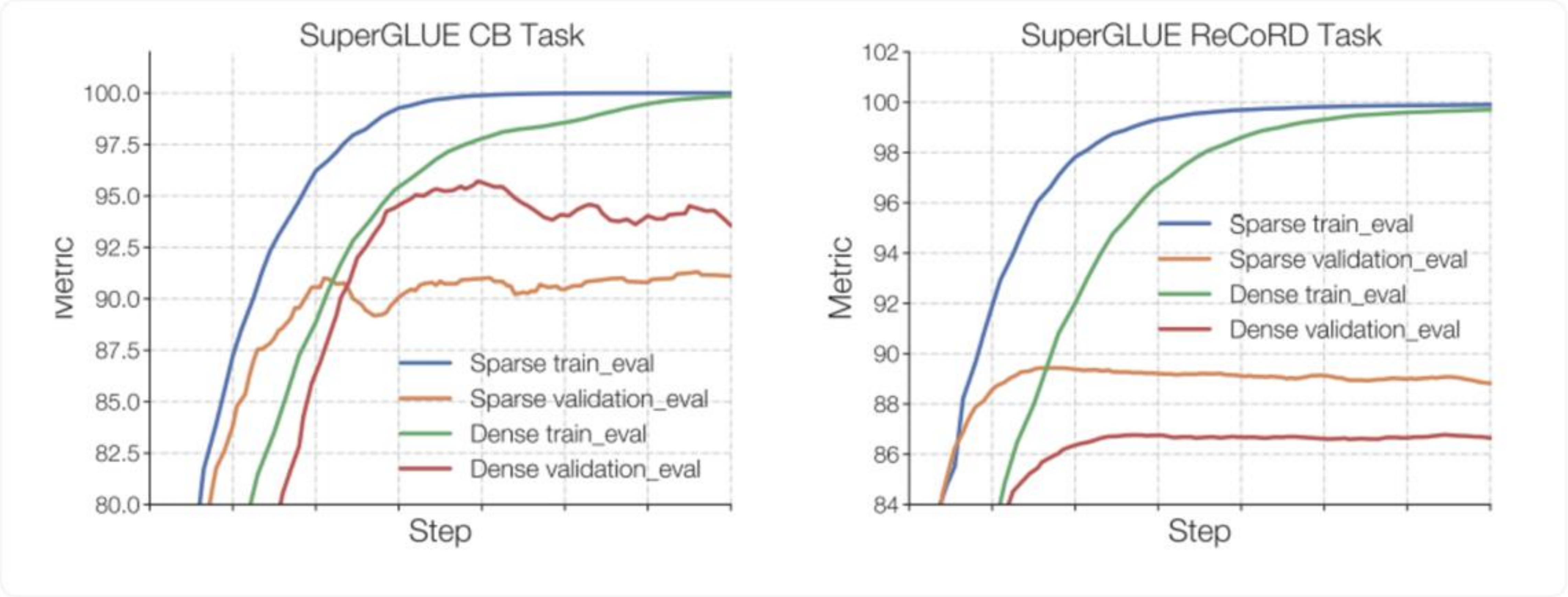
Activated Experts Differ across layers



Parameters of MoE

- How many parameters in Mixtral 8x7B model?
- 56B?
- 47B!
    - since only FFN layers act are experts, the other parameters (attention, embedding) are shared

Danger of MoE over-fitting to small data



In the small task (left), we can see clear overfitting as the sparse model does much worse in the validation set. In the larger task (right), the MoE performs well. This image is from the ST-MoE paper.

What does an Expert network learn?

- Encoder experts tend to specialize in token groups or shallow concepts (e.g., punctuation, proper nouns).
- Decoder experts exhibit less specialization.
- In multilingual setups, experts do not specialize in specific languages due to token routing and load balancing.

Expert specialization	Expert position	Routed tokens
Sentinel tokens	Layer 1	been <extra_id_4><extra_id_7>floral to <extra_id_10><extra_id_12><extra_id_15> <extra_id_17><extra_id_18><extra_id_19>...
	Layer 4	<extra_id_0><extra_id_1><extra_id_2> <extra_id_4><extra_id_6><extra_id_7> <extra_id_12><extra_id_13><extra_id_14>...
	Layer 6	<extra_id_0><extra_id_4><extra_id_5> <extra_id_6><extra_id_7><extra_id_14> <extra_id_16><extra_id_17><extra_id_18>...
Punctuation	Layer 2	, , , , , , , , , - , , , , ,).)
	Layer 6	, , , , , : . : , & , & & ? & - , , ? , , , , <extra_id_27>
Conjunctions and articles	Layer 3	The the the the the the the the the The the the the the the The the the the
	Layer 6	a and and and and and and or and a and . the the if ? a designed does been is not
Verbs	Layer 1	died falling identified fell closed left posted lost felt left said read miss place struggling falling signed died falling designed based disagree submitted develop
Visual descriptions *color, spatial position*	Layer 0	her over her know dark upper dark outer center upper blue inner yellow raw mama bright bright over open your dark blue
Proper names	Layer 1	A Mart Gr Mart Kent Med Cor Tri Ca Mart R Mart Lorraine Colin Ken Sam Ken Gr Angel A Dou Now Ga GT Q Ga C Ko C Ko Ga G
Counting and numbers *written and numerical forms*	Layer 1	after 37 19. 6. 27 I I Seven 25 4, 54 I two dead we Some 2012 who we few lower each

Table from the ST-MoE paper showing which token groups were sent to which expert.

Geometric Interpretation of Expert Routing

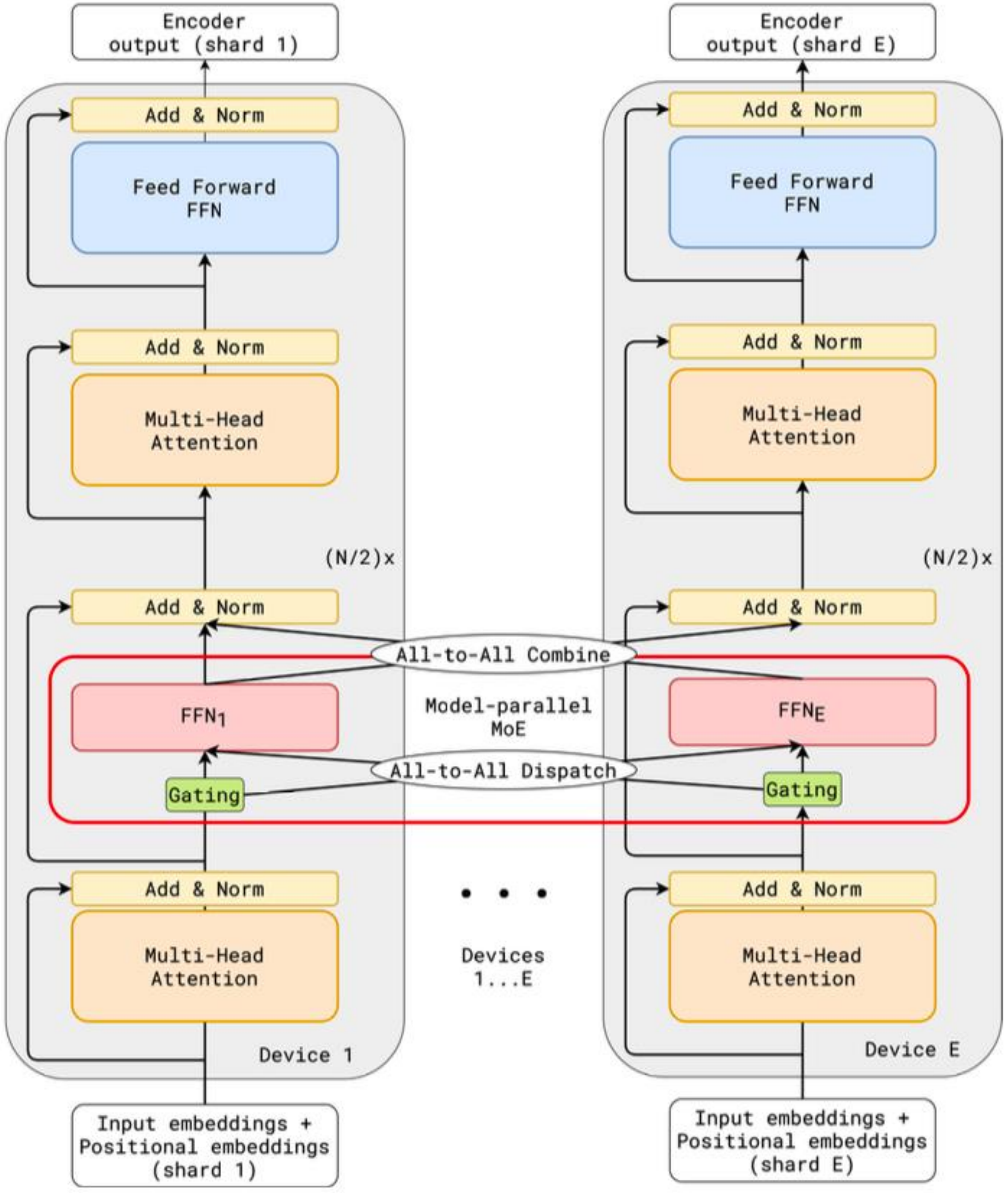
- KeepTop1 with 3 routing experts (finding linear boundaries among expert centroids)

Outline

- Transformer Mixture-of-Expert Model
  - Switch Transformer architecture
  - Shared-routed Experts
- Training and inference for MoE
  - Expert parallelism (GShard)
- Deepseek MoE (V3 model)
  - code walkthrough

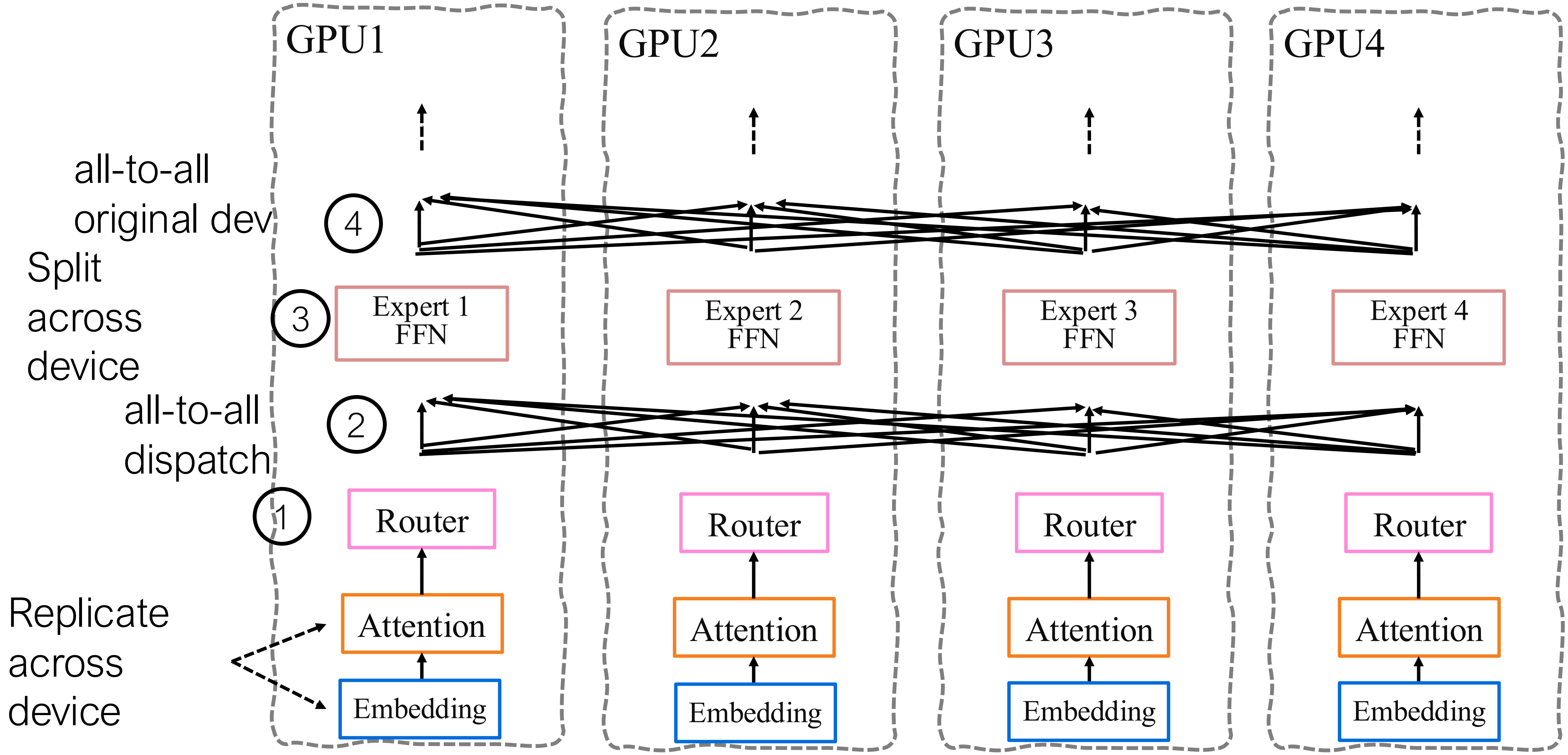
Training of MoE: Expert Parallelism

- Keep one Expert on one worker device
- Replicate all other network components in all devices
- Need fast all-to-all communication

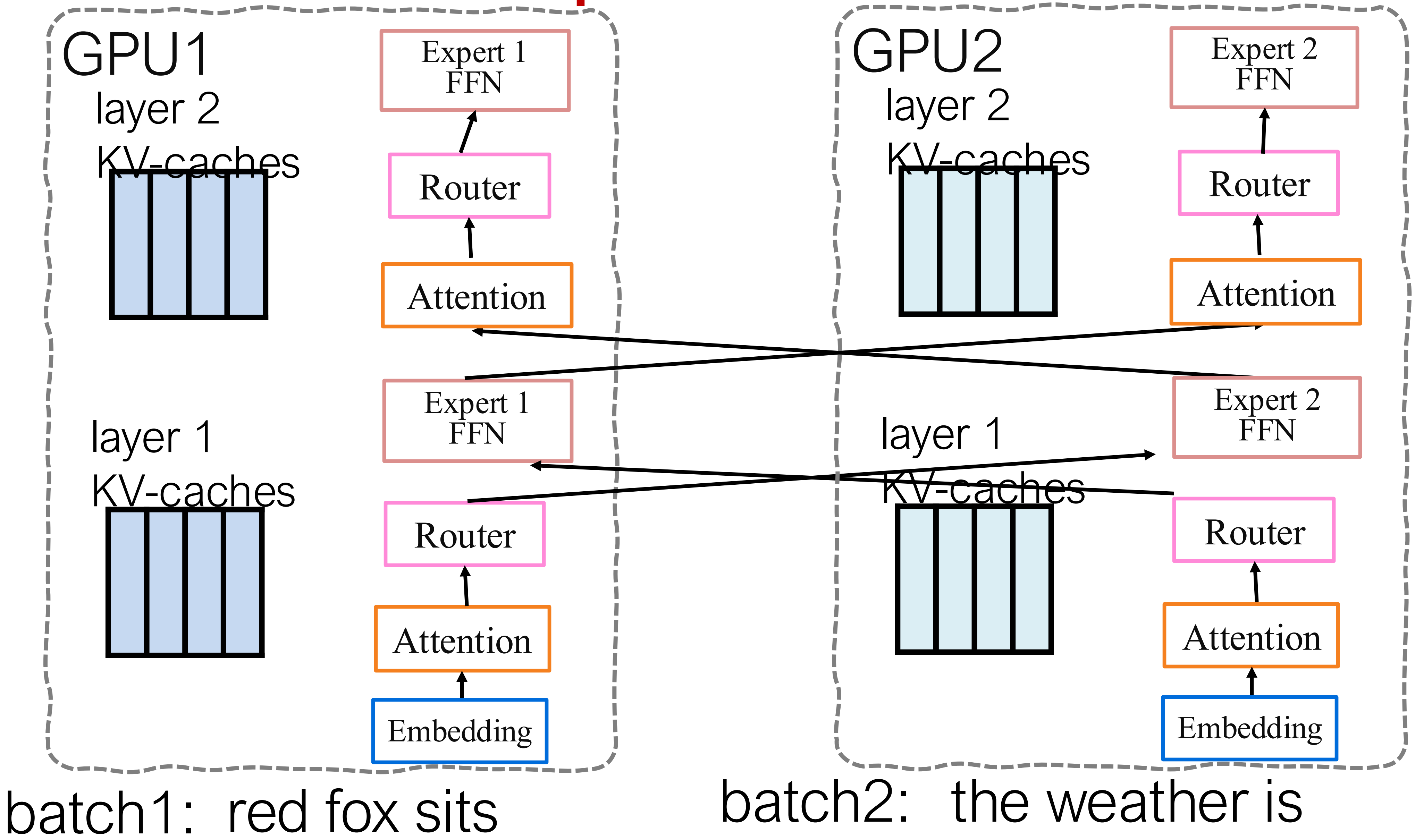


Lepikhin et al. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. ICLR 2021.

Training MoE: Expert Parallelism



Token Computation Path in MoE



Gshard's Interleaving Expert

- For every other layer, use MoE

Load Balancing in MoE Training

- Expert-Level Balance Loss (to avoid routing collapse to experts)

$$L_{ExpBal} = \alpha_1 M \sum_{i=1}^{\#experts} f_i P_i$$

$$f_i = \frac{\#\text{tokens to expert } i}{\#\text{tokens}}$$

M: num of experts

$$P_i = \frac{1}{\#\text{tokens}} \sum_{t=1}^{\#\text{tokens}} s_{i,t}$$

routing weight

MOE Inference

- MoE inference performance depends on:
  - overall model size
  - how many activated experts
  - overall memory bandwidth

- Default implementation:
  - Keep all experts in GPU memory (need large mem)

Optimizing MoE inference

- System Design Goal: minimize the critical data path per device, maximize the achievable aggregate memory bandwidth
- group and route all tokens with the same critical data path together to reduce data access per device and achieve maximum aggregate bandwidth;
- Optimize communication scheduling with parallelism coordination
- Optimize transformer and MoE related kernels to improve per-device performance

Expert Parallelism and Tensor-Parallelism

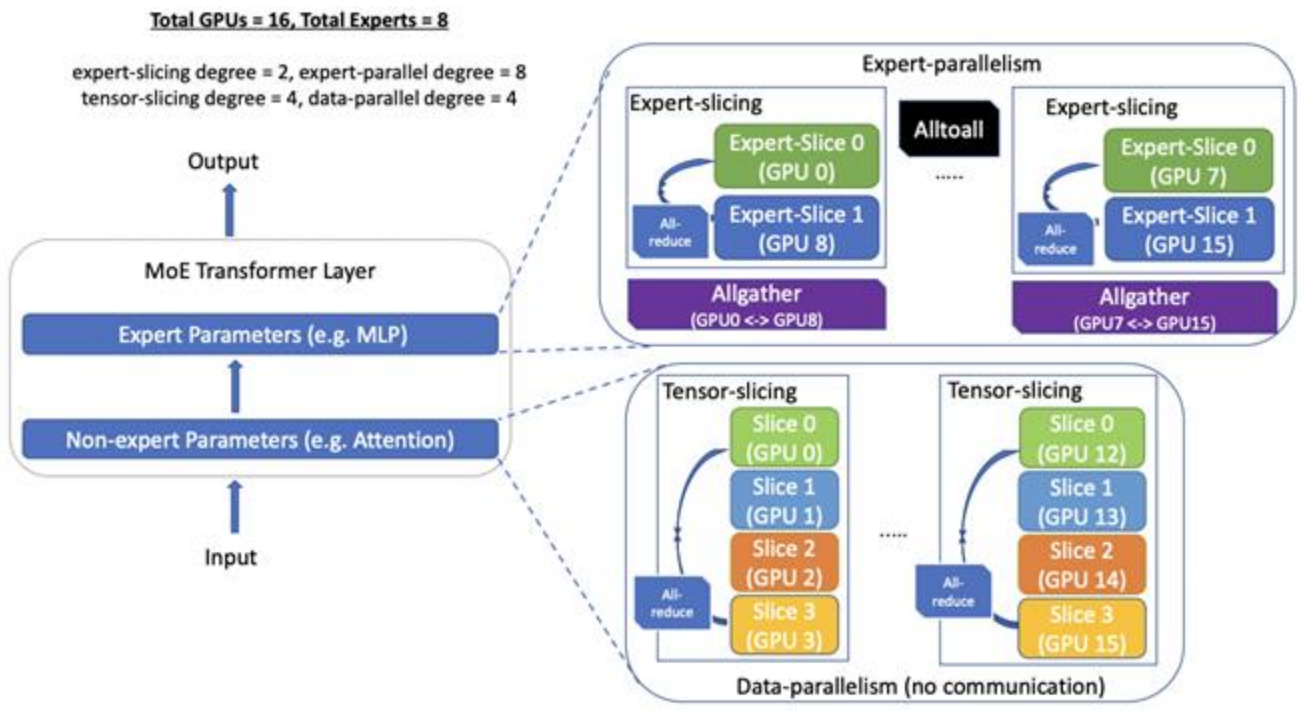


Figure 7: DS-MoE design that embraces the complexity of multi-dimensional parallelism for different partitions (expert and non-expert) of the model.

Expert Parallelism / Expert slicing
Group all input tokens assigned to the same experts under the same critical data path, and parallelize processing of the token groups with different critical paths among different devices using expert parallelism.

Tensor Parallelism / Tensor slicing:
Partition the non-expert parameters (Attention) across devices (usually within a node)

Further with Data parallelism

Rajbhandari et al (2022). DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale.

Optimizing MoE Kernels

- MOE Specific Optimizations:
  - fuse the gating function into a single kernel
  - dense token-to-expert mapping table
- Result: over 6x reduction in MoE Kernel related latency

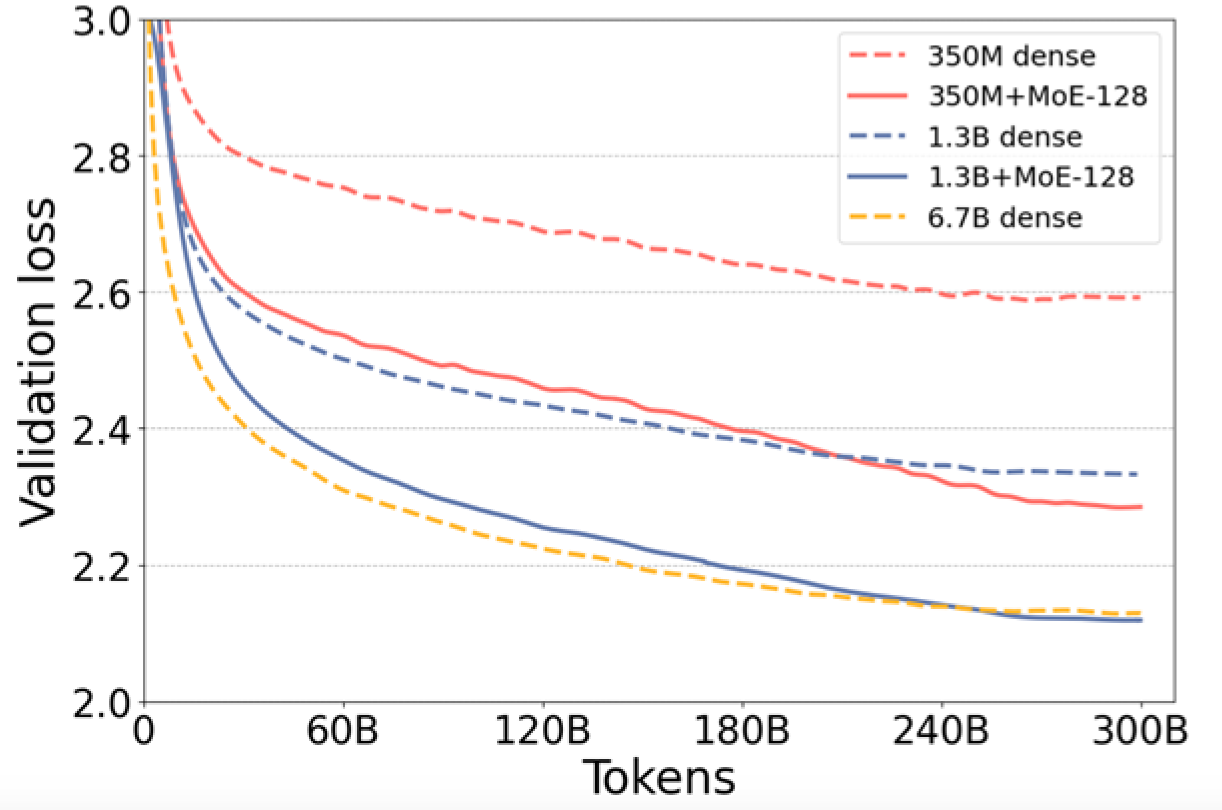
Opportunity for Optimized All-to-All Communication

- Expert parallelism requires all-to-all communication between all expert parallel devices; the latency increases linearly with the increase in devices
- Optimization:
  - hierarchical all-to-all communication pattern: reduces the communication hops
  - parallelism-coordinated communication optimization: schedules communications based on the model's parallelism strategy to minimize overhead.

Rajbhandari et al (2022). DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale.

MOE Training Loss and Throughput

Token-wise validation loss curves for dense and MoE LLMs

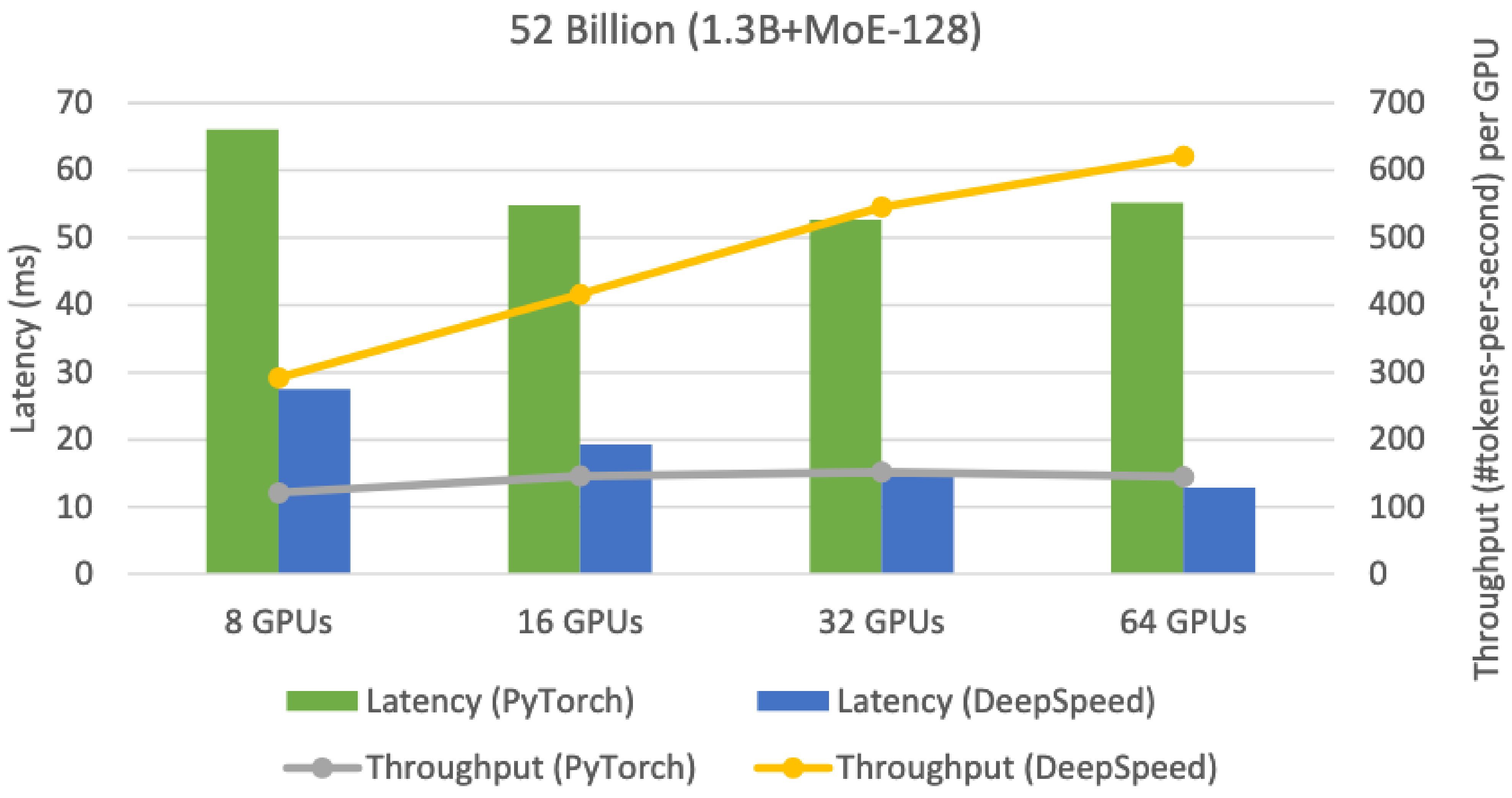


	Training samples per sec	Throughput gain/ Cost Reduction
6.7B dense	70	1x
1.3B+MoE-128	372	5x

Training throughput (on 128 A100 GPUs) comparing MoE based model vs dense model that can both achieve the same model quality.

Rajbhandari et al (2022). DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale.

DeepSpeed MOE Inference Performance



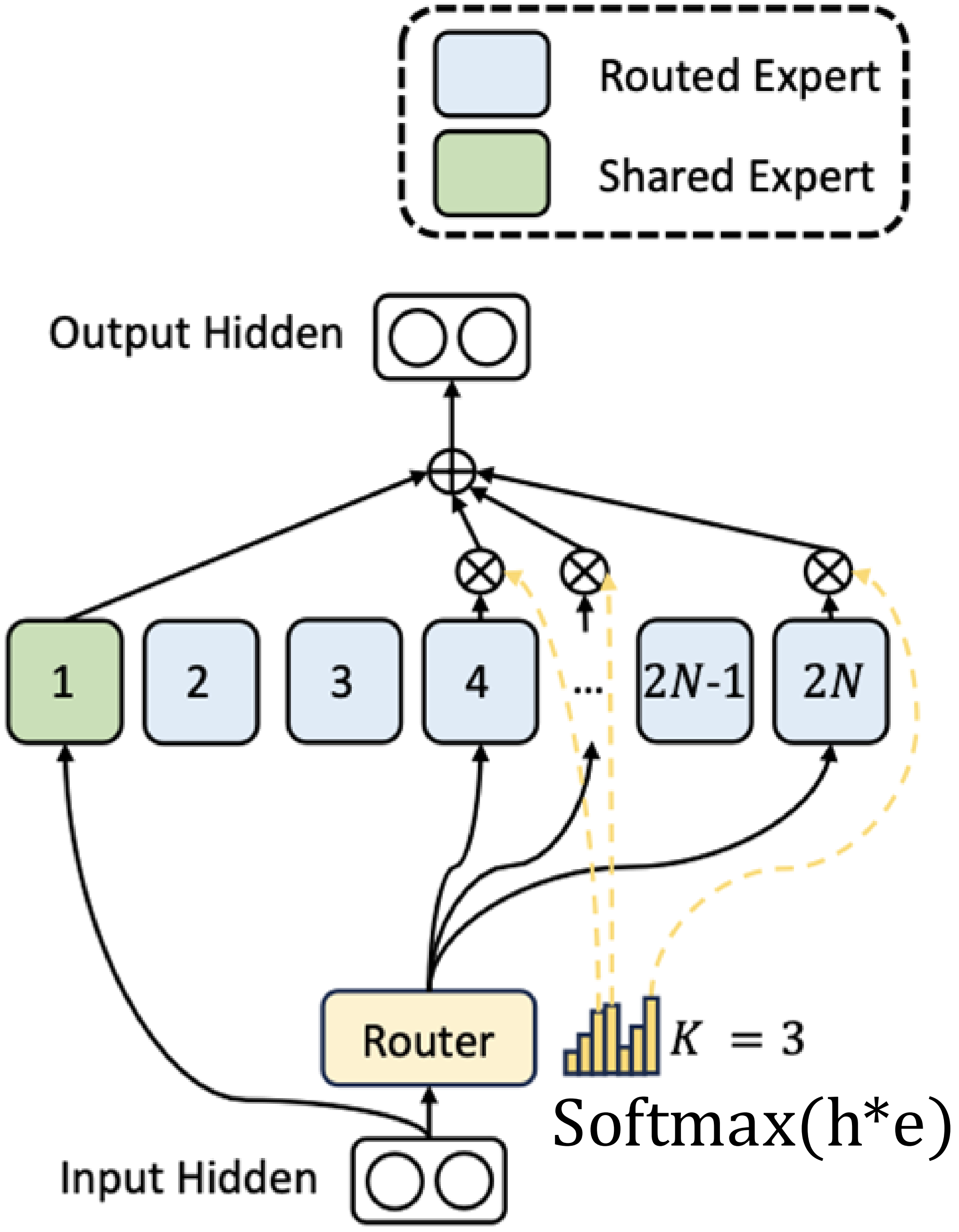
Rajbhandari et al (2022). DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale.

Outline

- Transformer Mixture-of-Expert Model
  - Switch Transformer architecture
  - Shared-routed Experts
- Training and inference for MoE
  - Expert parallelism (GShard)
- Deepseek MoE (V3 model)
  - code walkthrough

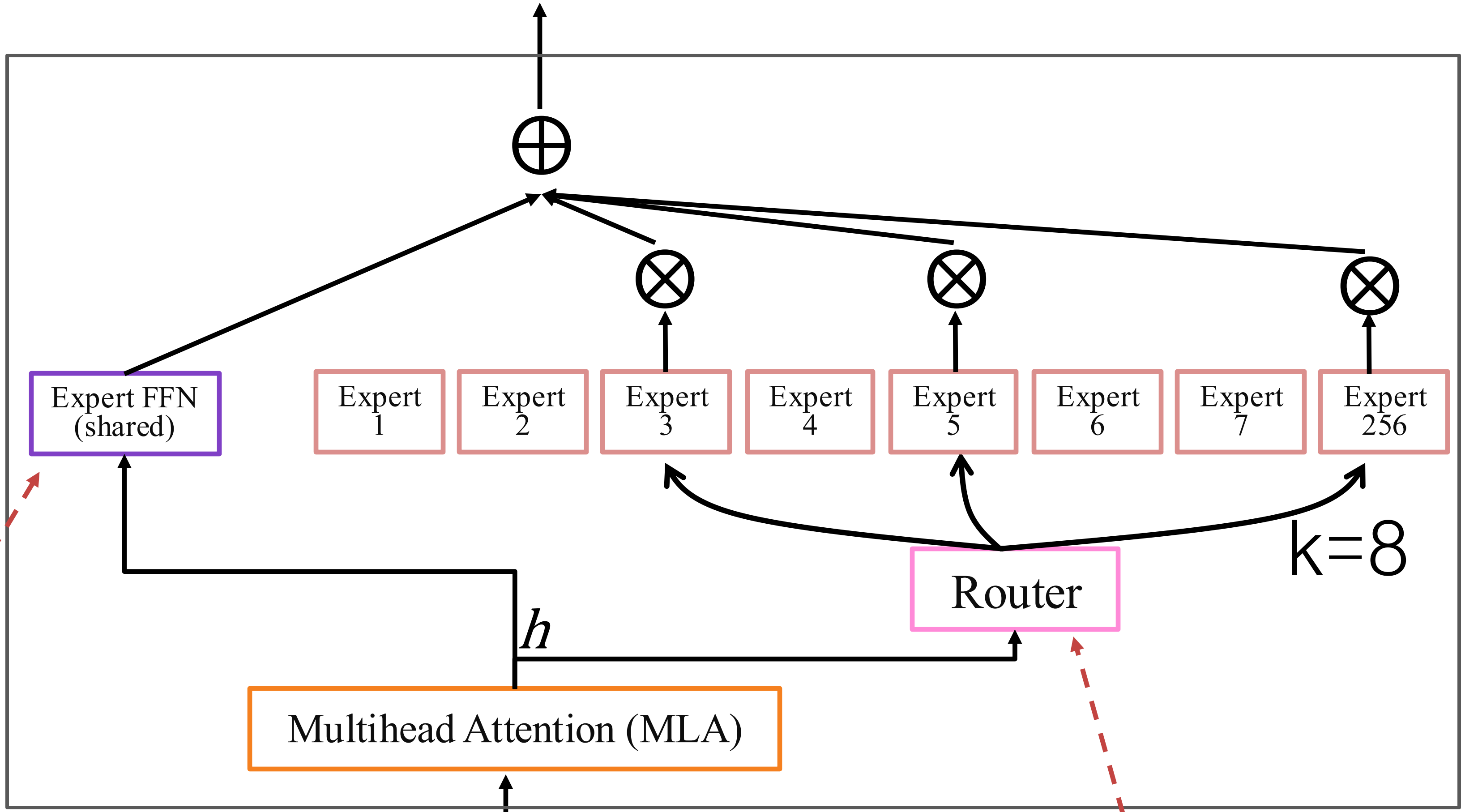
DeepSeek MoE

- Fine-grained experts: each FFN is split to k smaller experts, total kN (N=original experts)
- shared experts + routing experts
- topk weighted average of routing experts (activating kM)



DeepSeek V3 MoE (670B)

Vocab: 129,280
dimension=7168
num layer=61
num dense layer=3 (lowest)
num head = 128
dim ffn (inter dim)=18432
moe dim = 2048
num shared experts = 1
num routed experts = 256
num activated experts = 8
num expert group=8
num limited group=4



$\mathrm{FFN}_{SwiGLU}(x) = \big(Swish(x \cdot W_1) \odot (x \cdot W_2)\big) \cdot W_3$

$\mathrm{TopK}(\mathrm{Softmax}(h_t \cdot W), 8)$

https://github.com/deepseek-ai/DeepSeek-V3/blob/main/inference/model.py

Load Balancing in Deepseek MoE

- Expert-Level Balance Loss (to avoid routing collapse to experts)

$$L_{ExpBal} = \alpha_1 \sum_{i=1}^{\#\text{experts}} f_i P_i$$

$$f_i = \frac{\#\text{experts}}{\#\text{activated\_experts}} \cdot \frac{\#\text{tokens to expert } i}{\#\text{tokens}} \qquad P_i = \frac{1}{\#\text{tokens}} \sum_{t=1}^{\#\text{tokens}} s_{i,t}$$

routing weight

- Device-level balance loss (balance computation across dev)

$$L_{DevBal} = \alpha_2 \sum_{j=1}^{\#\text{groups}} f_j P_j$$

$$f_j = avg\ f\ in\ group\ j) \qquad P_j = sum\ of\ P\ in\ group\ j$$

Deepseek Libraries to Accelerate MOE

- DeepEP is a communication library tailored for Mixture-of-Experts (MoE) and expert parallelism (EP).
  - https://github.com/deepseek-ai/DeepEP

- Expert Parallelism Load Balancer (EPLB)
  - https://github.com/deepseek-ai/EPLB

Deepseek V3 MoE Code Walkthrough

- https://github.com/deepseek-ai/DeepSeek-V3/blob/main/inference/model.py

Deepspeed MoE Code Example

```python
import torch
import deepspeed
import deepspeed.utils.groups as groups
from deepspeed.moe.layer import MoE

WORLD_SIZE = 4
EP_WORLD_SIZE = 2
EXPERTS = 8

fc3 = torch.nn.Linear(84, 84)
fc3 = MoE(hidden_size=84, expert=self.fc3, num_experts=EXPERTS, ep_size=EP_WORLD_SIZE, k=1)
fc4 = torch.nn.Linear(84, 10)
```

https://www.deepspeed.ai/tutorials/mixture-of-experts/

Deepspeed MoE Code Example

```python
class MoE(nn.Module):
    """Initialize an MoE layer.

    Arguments:
        hidden_size (int): the hidden dimension of the model, importantly this is also the input and output dimension.
        expert (nn.Module): the torch module that defines the expert (e.g., MLP, torch.linear).
        num_experts (int, optional): default=1, the total number of experts per layer.
        ep_size (int, optional): default=1, number of ranks in the expert parallel world or group.
        k (int, optional): default=1, top-k gating value, only supports k=1 or k=2.
        capacity_factor (float, optional): default=1.0, the capacity of the expert at training time.
        eval_capacity_factor (float, optional): default=1.0, the capacity of the expert at eval time.
        min_capacity (int, optional): default=4, the minimum capacity per expert regardless of the capacity_factor.
        use_residual (bool, optional): default=False, make this MoE layer a Residual MoE (https://arxiv.org/abs/2201.05596) layer.
        noisy_gate_policy (str, optional): default=None, noisy gate policy, valid options are 'Jitter', 'RSample' or 'None'.
        drop_tokens (bool, optional): default=True, whether to drop tokens - (setting to False is equivalent to infinite capacity).
        use_rts (bool, optional): default=True, whether to use Random Token Selection.
        use_tutel (bool, optional): default=False, whether to use Tutel optimizations (if installed).
        enable_expert_tensor_parallelism (bool, optional): default=False, whether to use tensor parallelism for experts
        top2_2nd_expert_sampling (bool, optional): default=True, whether to perform sampling for 2nd expert
    """
```

https://github.com/microsoft/DeepSpeed/blob/master/deepspeed/moe/layer.py

Deepspeed MoE Code Example

```
experts = Experts(expert, self.num_local_experts, self.expert_group_name)
self.deepspeed_moe = MOELayer(TopKGate(hidden_size, num_experts, k, capacity_factor, eval_capacity_factor,
                                       min_capacity, noisy_gate_policy, drop_tokens, use_rts, None,
                                       top2_2nd_expert_sampling),
                              experts,
                              self.expert_group_name,
                              self.ep_size,
                              self.num_local_experts,
                              use_tutel=use_tutel)
if self.use_residual:
    self.mlp = expert
    # coefficient is used for weighted sum of the output of expert and mlp
    self.coefficient = nn.Linear(hidden_size, 2)
```

https://github.com/microsoft/DeepSpeed/blob/master/deepspeed/moe/layer.py

Deepspeed MoE Code Example

```python
class Experts(nn.Module):

    def __init__(self, expert: nn.Module, num_local_experts: int = 1, expert_group_name: Optional[str] = None) -> None:
        super(Experts, self).__init__()

        self.deepspeed_experts = nn.ModuleList([copy.deepcopy(expert) for _ in range(num_local_experts)])
        self.num_local_experts = num_local_experts

        # TODO: revisit allreduce for moe.gate...
        for expert in self.deepspeed_experts:
            # TODO: Create param groups to handle expert + data case (e.g. param.group = moe_group)
            for param in expert.parameters():
                param.allreduce = False
                param.group_name = expert_group_name

    def forward(self, inputs: torch.Tensor) -> torch.Tensor:
        chunks = inputs.chunk(self.num_local_experts, dim=1)
        expert_outputs: List[torch.Tensor] = []

        for chunk, expert in zip(chunks, self.deepspeed_experts):
            out = expert(chunk)
            if isinstance(out, tuple):
                out = out[0]  # Ignore the bias term for now
            expert_outputs += [out]

        return torch.cat(expert_outputs, dim=1)
```

https://github.com/microsoft/DeepSpeed/blob/master/deepspeed/moe/experts.py

Summary

- LLM Mixture-of-Expert Model
  - Instead of a single dense FFN, using multiple FFNs (experts)
  - Routing network to select one/multiple experts
  - Shared-routed experts (deepspeed-MOE, deepseek MOE)
  - a few dense layers, then MOE (deepseek MOE)

- Scalable training/inference
  - expert parallelism: split experts and replicate non-expert (GShard)
  - all-to-all communication for expert output
  - load balancing: grouping and avoid collapse (deepseek)
  - optimized kernel for MoE

Reference

- Rajbhandari et al (2022). DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale.
- Dai, D. et al. (2024). DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models.