15-779: Lecture 18

Mixture of Experts: Architectures, Kernels, Parallelism

Zhihao Jia

Carnegie Mellon University

Mixture-of-Experts Outline

How do MoE architectures work?

How to optimize MoE kernels?

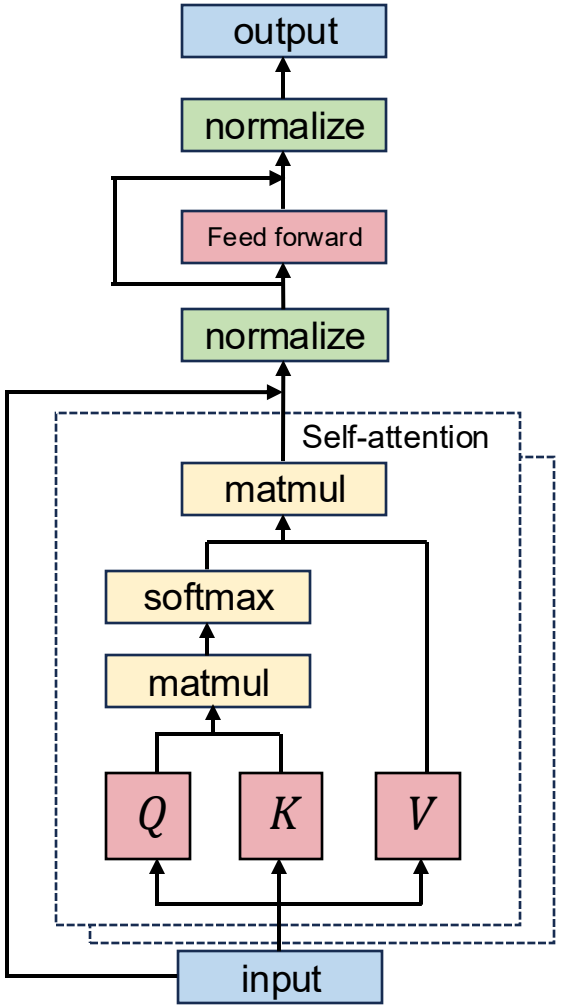
How to parallelize MoE models?

Recap: Transformer Block

A typical transformer block

$$Z = \text{SelfAttention}(XW_K, XW_Q, XW_V)$$
$$Z = \text{LayerNorm}(X + Z)$$
$$H = \text{LayerNorm}(\text{ReLU}(ZW_1)W_2 + Z)$$

(multi-head) self-attention, followed by a linear layer and ReLU and some additional residual connections and normalization



Normal Feed Forward Layer

Feed forward

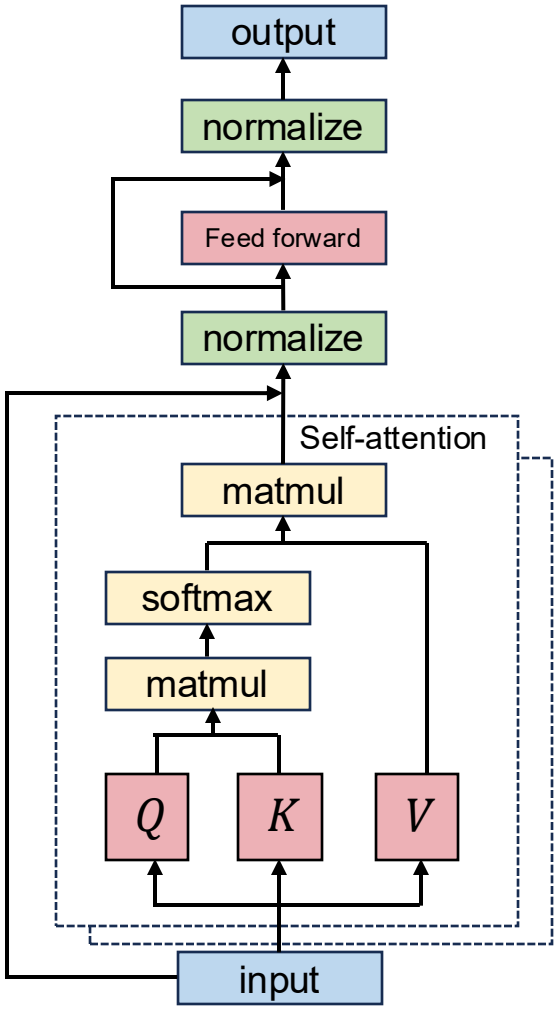
$$H = \text{LayerNorm}(\text{ReLU}(ZW_1)W_2 + Z)$$

$W_1 \in R^{n \times m}$

Increasing feature size will increase compute quadratically

Everything is mixed together in the FFN(feed forward network) layer

Most Parameters in Feed-Forward Layers

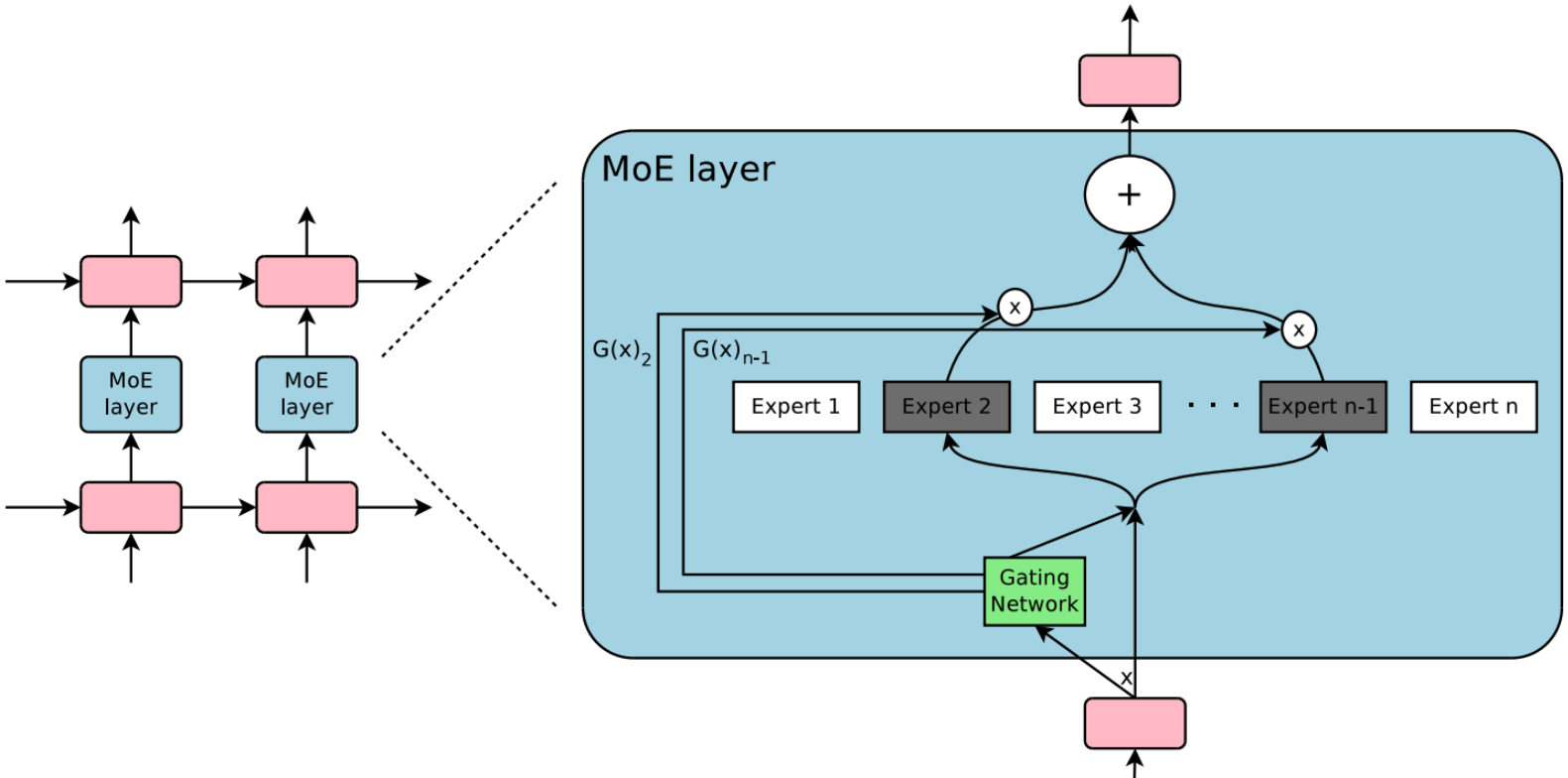


	GPT-3
vocab	50,257
d_model	12288
n_heads	96
n_layers	96
d_ff	49152
LayerNorm (B)	0.00
Embedding (B)	0.62
Attention (B)	57.99
Feed-forward (B)	115.97
Total (B)	**174.57**

Mixture-of-Experts

Key idea: introduce multiple experts, each of which is a FFN layer; make each expert focus on predicting the right answer for a subset of cases

Benefit: sub-linear scaling of FLOPS with model size



A Closer Look at Mixture-of-Experts

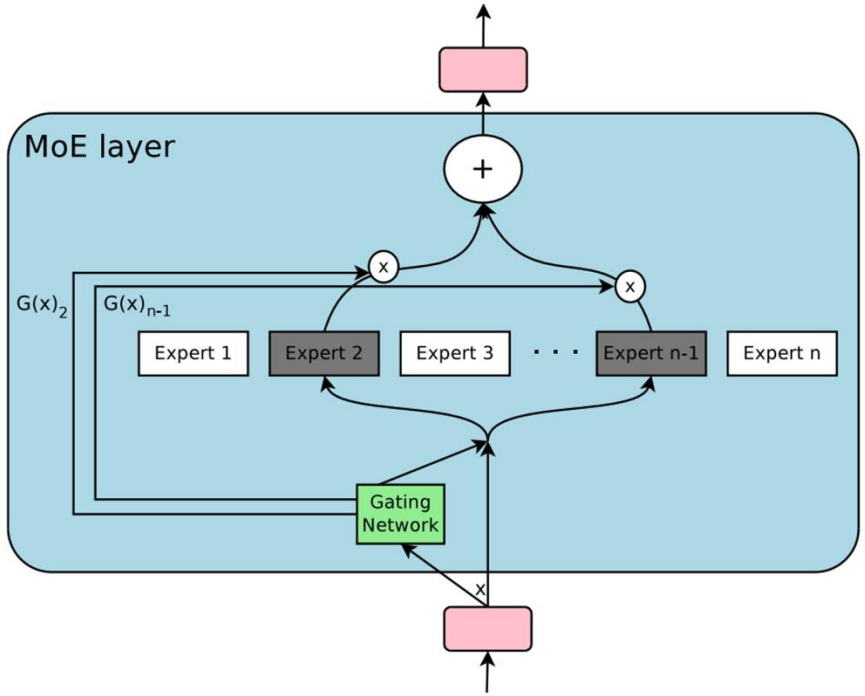
A typical MoE layer (assume single instance and activate two experts)

Gating: $G = \mathrm{Softmax}(W_G X)$

Expert indices: $I = \{i_0, i_1\} = \mathrm{Top}K(G, k = 2)$

Output weight: $s_0 = \frac{G_{i_0}}{(G_{i_0}+G_{i_1})}, s_1 = \frac{G_{i_1}}{(G_{i_0}+G_{i_1})}$

Output: $Y = \mathrm{s}_0 \mathrm{FFN}_{i_0}(X) + s_1 \mathrm{FFN}_{i_1}(X)$



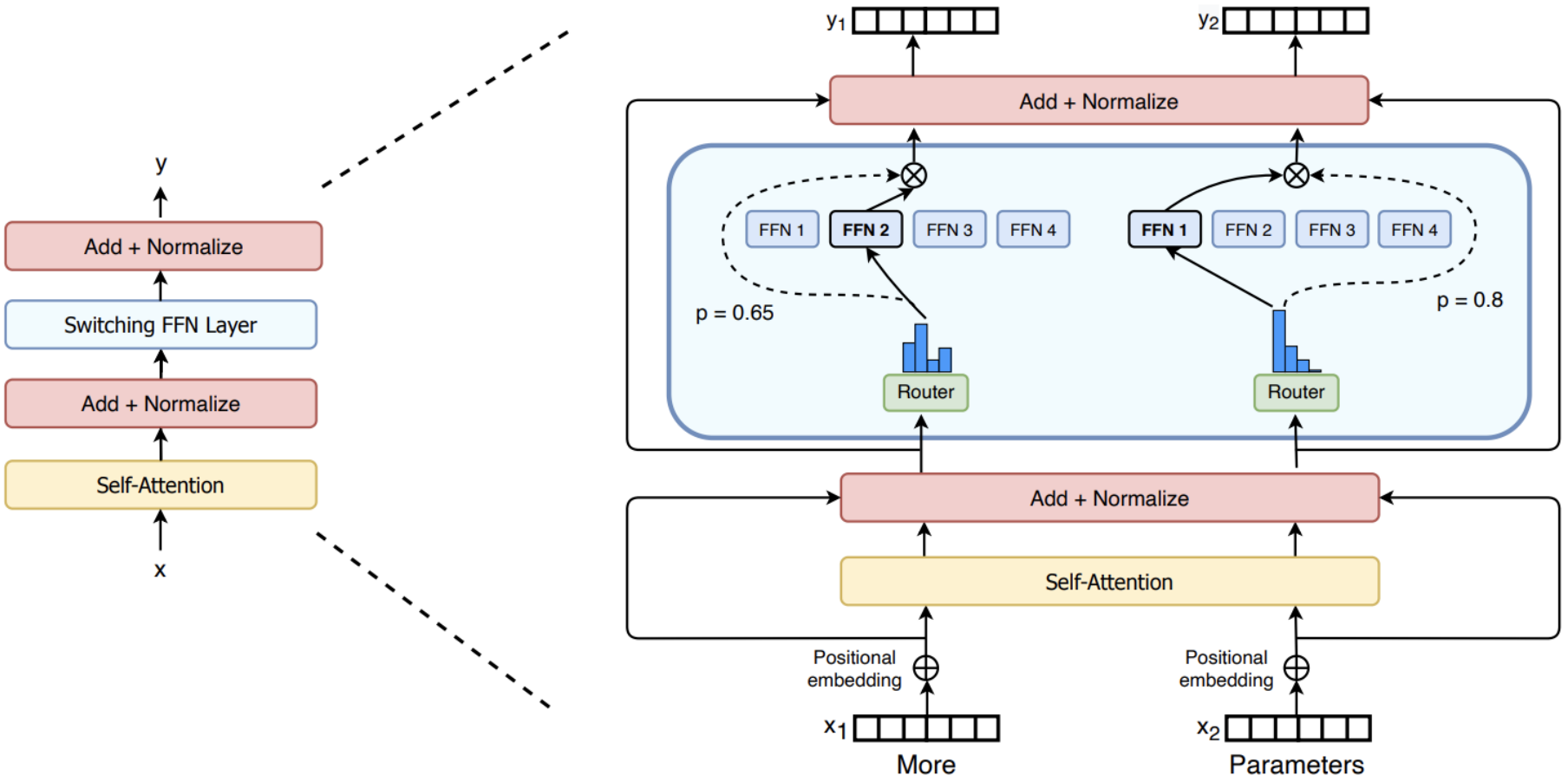
Greedily select top-K experts among N

Example model: Mixtral-8x7B selects 2 experts among eight

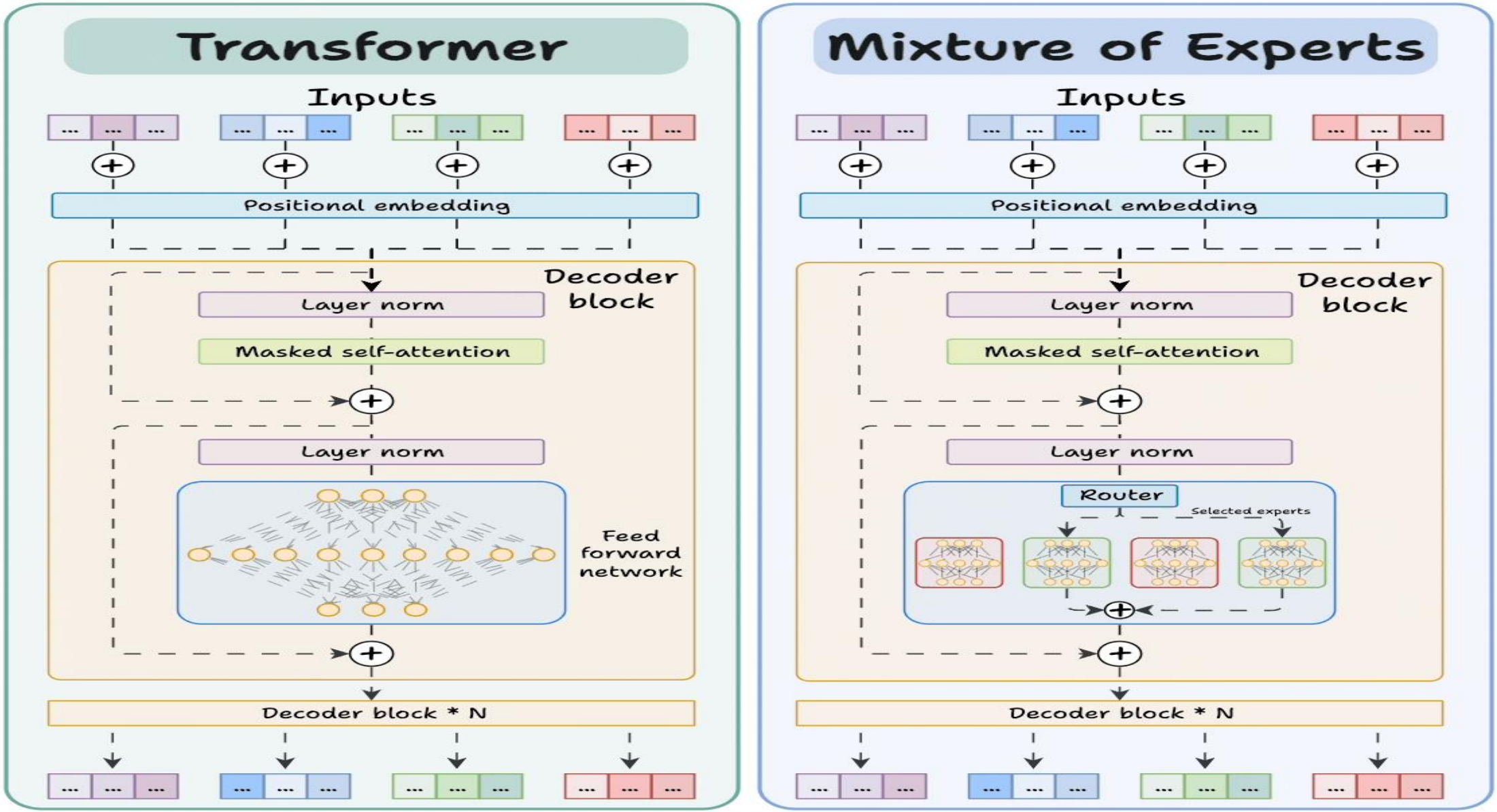
Different models may have different FFN configurations, usually contains multiple linear layers and some non-linear mixing

Transformers + Mixture-of-Experts

Simply replace the FFN layer in a transformer model with mixture-of-experts



Transformers + Mixture-of-Experts



Potential of MoE in Transformer Models

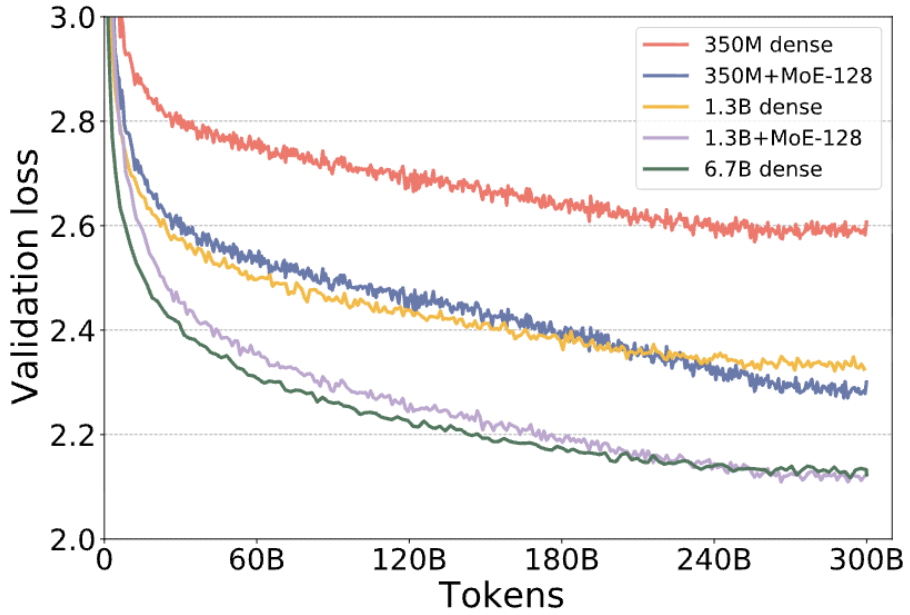
GLaM

- 1.3-trillion-parameter MoE
- 7x more parameters than GPT-3
- 2x less compute than GPT-3

		GPT-3	GLaM	relative
cost	FLOPs / token (G)	350	**180**	**–48.6%**
	Train energy (MWh)	1287	**456**	**–64.6%**
accuracy on average	Zero-shot	56.9	**62.7**	**+10.2%**
	One-shot	61.6	**65.5**	**+6.3%**
	Few-shot	65.2	**68.1**	**+4.4%**

DeepSpeed MoE models

- 6.7B dense LLM = 52B MoE with 1.3B activation
- Training compute reduces 5x



GLaM: Efficient Scaling of Language Models with Mixture-of-Experts
DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale

Case Study: Mixture-of-Experts in DeepSeek-V3

- **Shared experts**: every token passes through (no routing)
- **Routed experts**: selectively activated for each token by a router (gating network)

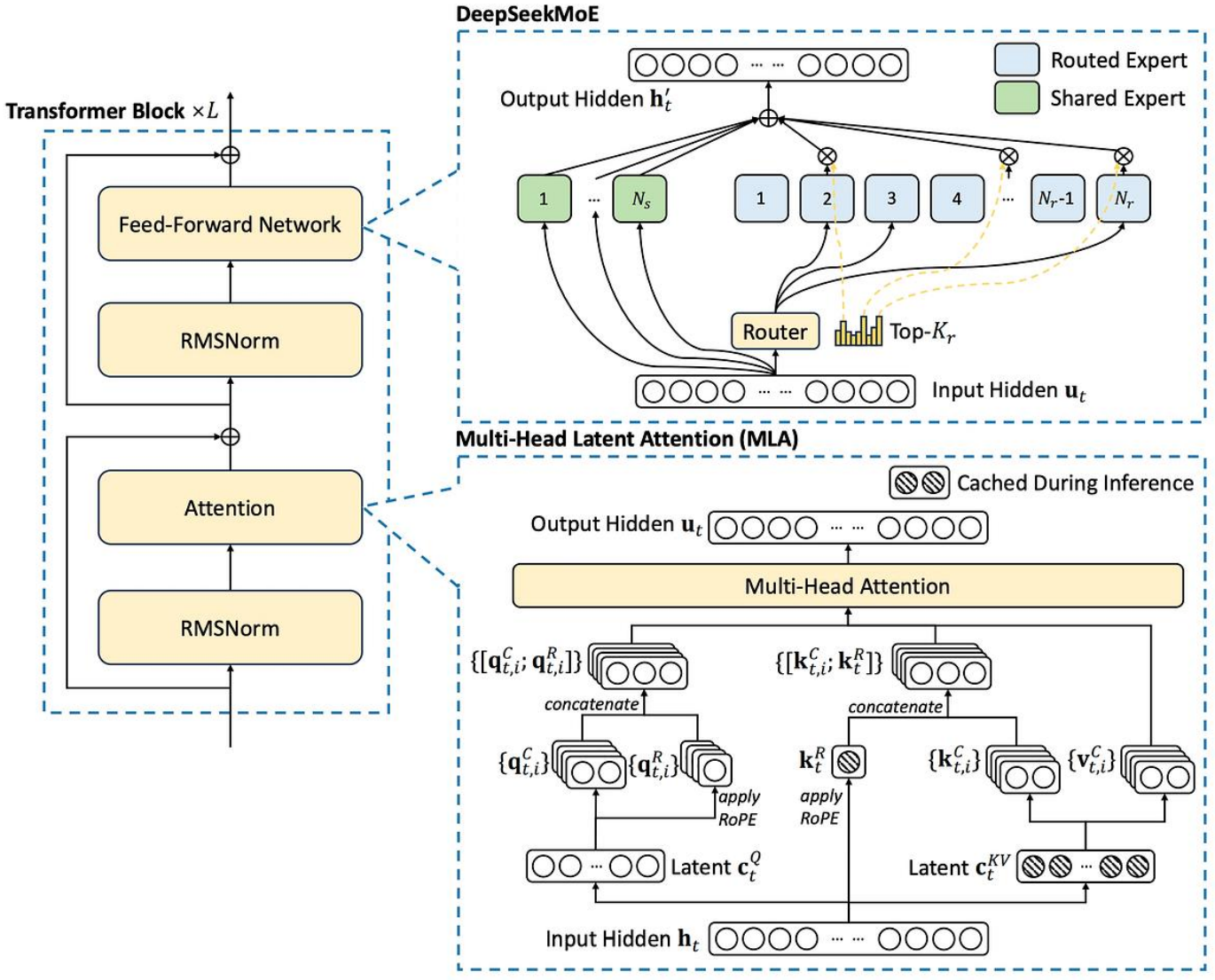
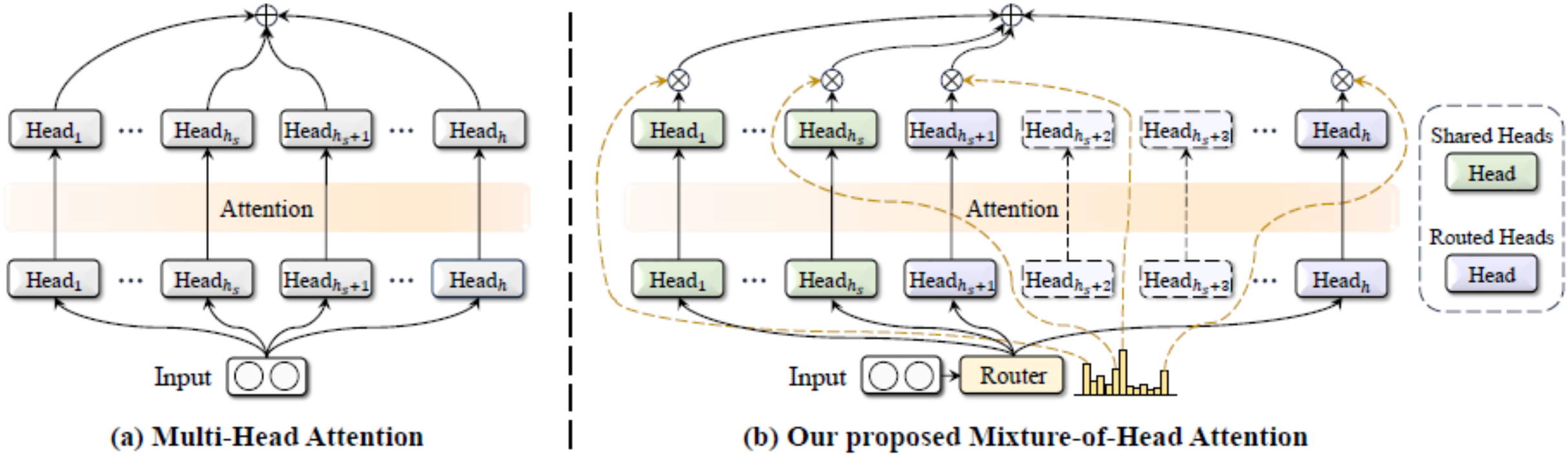


Figure 2 | Illustration of the basic architecture of DeepSeek-V3. Following DeepSeek-V2, we adopt MLA and DeepSeekMoE for efficient inference and economical training.

Case Study: Mixture-of-Head Attention

- **Shared heads**: every token passes through
- **Routed heads**: selectively activated by a router



Discussion: What are the Advantages of MoE?

- More parameters without increasing compute
- Higher model throughput and lower latency (for a given quality target)
- More efficient for distributed training

Mixture-of-Experts Outline

How do MoE architectures work?

How to optimize MoE kernels?

How to parallelize MoE models?

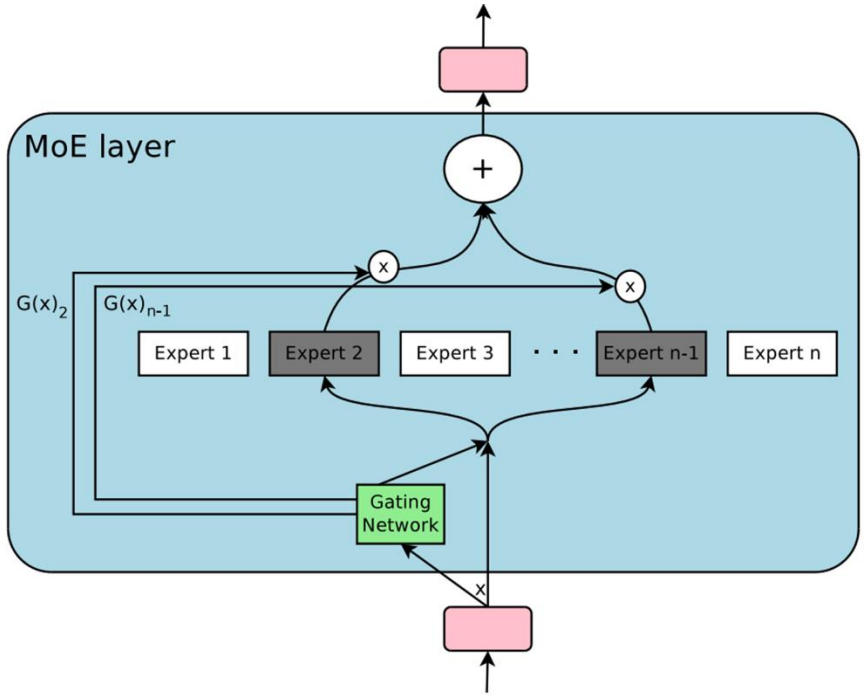
Single Batch Setting

Gating: $G = \mathrm{Softmax}(XW_G)$

Expert indices: $I = \{i_0, i_1\} = \mathrm{Top}K(G, k = 2)$

Output weight: $s_0 = \frac{G_{i_0}}{(G_{i_0} + G_{i_1})}, s_1 = \frac{G_{i_1}}{(G_{i_0} + G_{i_1})}$

Output: $Y = s_0 \mathrm{FFN}_{i_0}(X) + s_1 \mathrm{FFN}_{i_1}(X)$



Only two of n experts are used

We only need to load the weights of these experts during computation

Helps to speedup computations

Batched Linear Layer

$Z = X\,W, X \in \mathbb{R}^{b \times n}, W \in \mathbb{R}^{n \times m}$

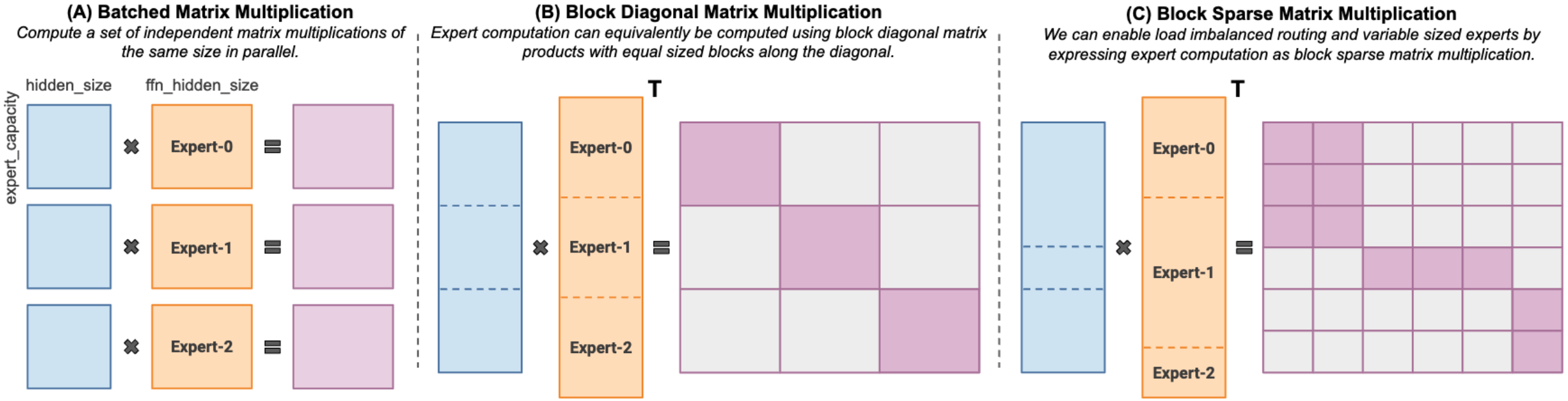
b is the batch size

When b becomes larger, we get better compute efficiency

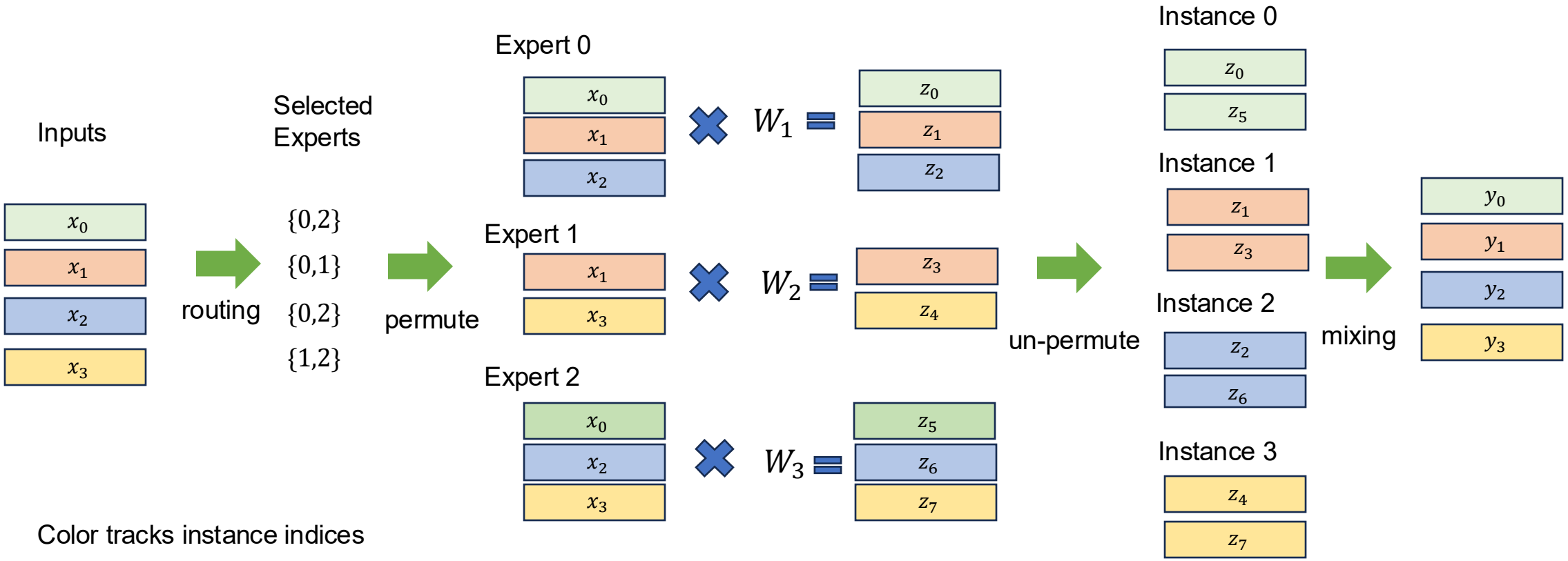
due to memory load reuse in matrix multiply and hardware specialization via TensorCore

Batching MoE = Block-Sparse MatMul

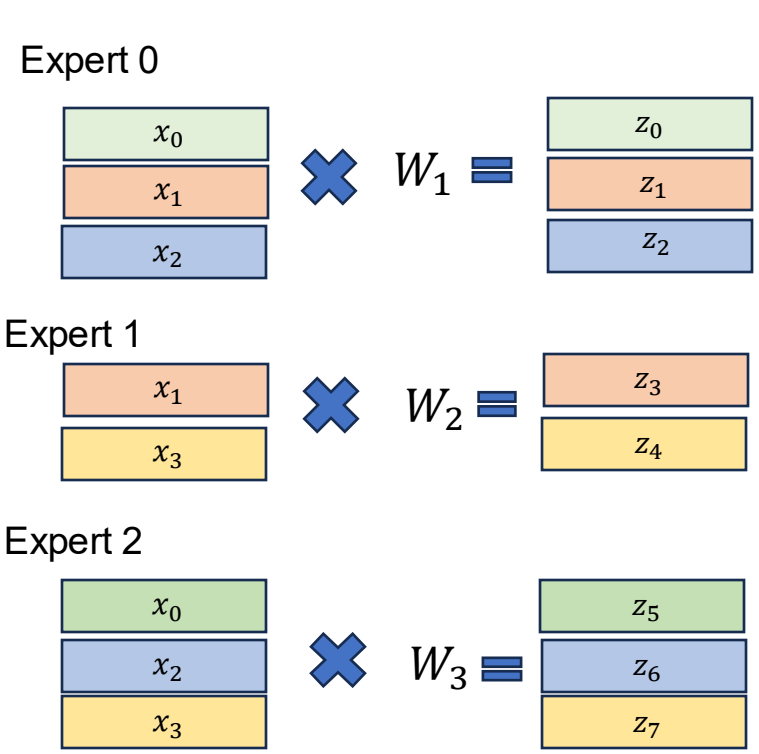
- Fuse all experts' computation into a single kernel to maximize parallelism
- View expert computation as a block-sparse matrix multiplication

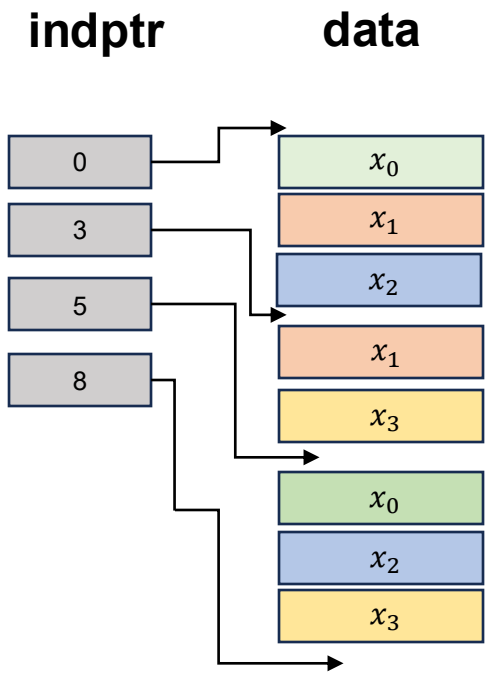


Batching MoE Computation



Batched Expert Compute





weights

W_1

W_2

W_3

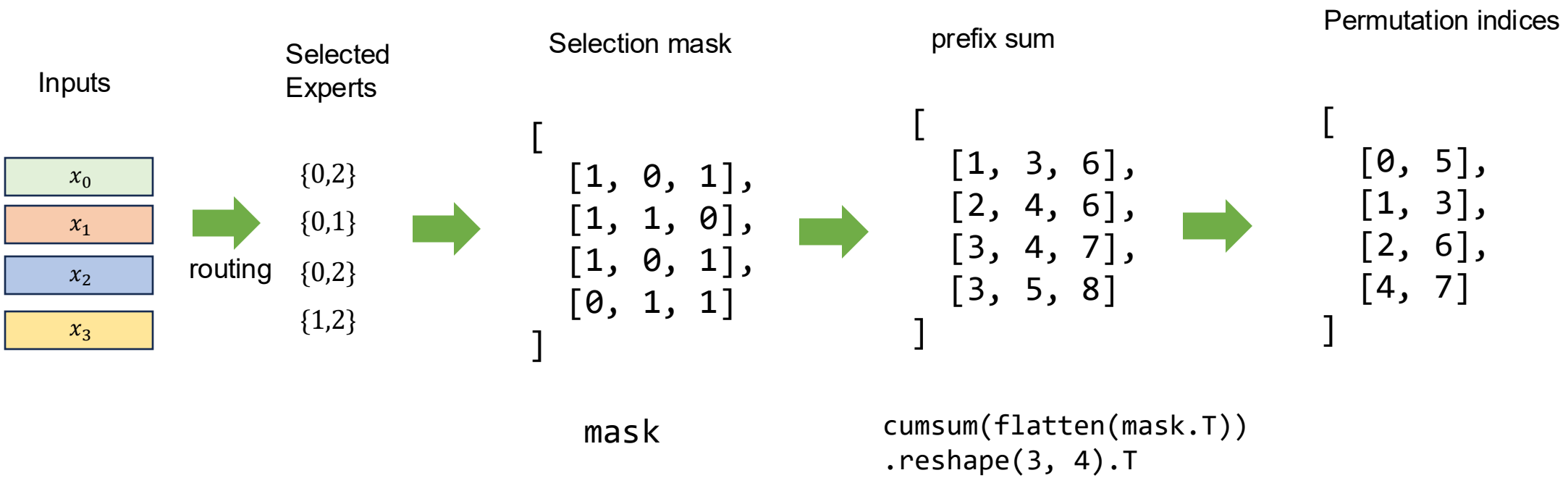
$$Z = \text{GroupGemm}(\text{data}, \text{indptr}, \text{weights})$$

In practice can be computed in compressed sparse row (CSR) format

A Closer Look at Permutation

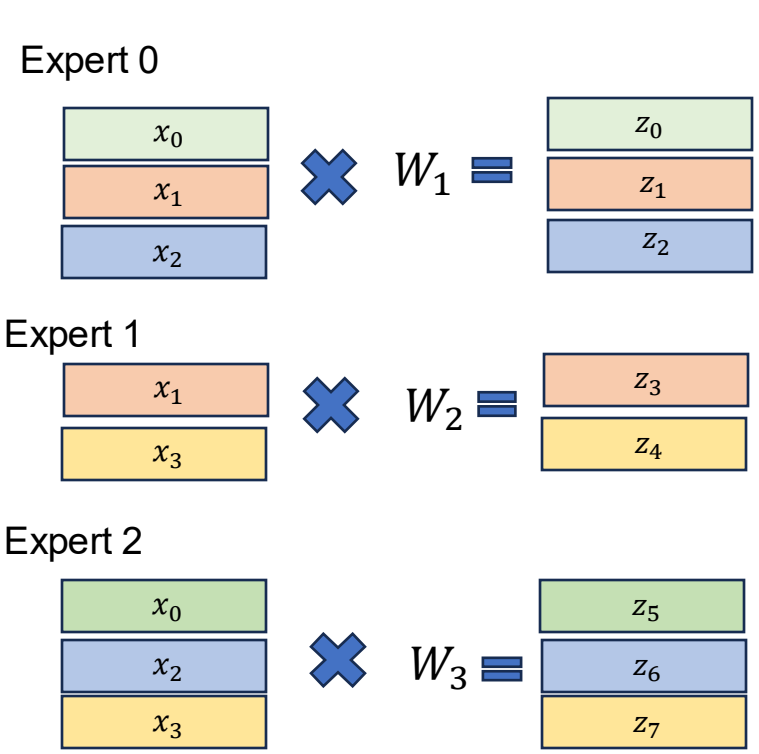
How to get the permutation indices efficiently in GPU?

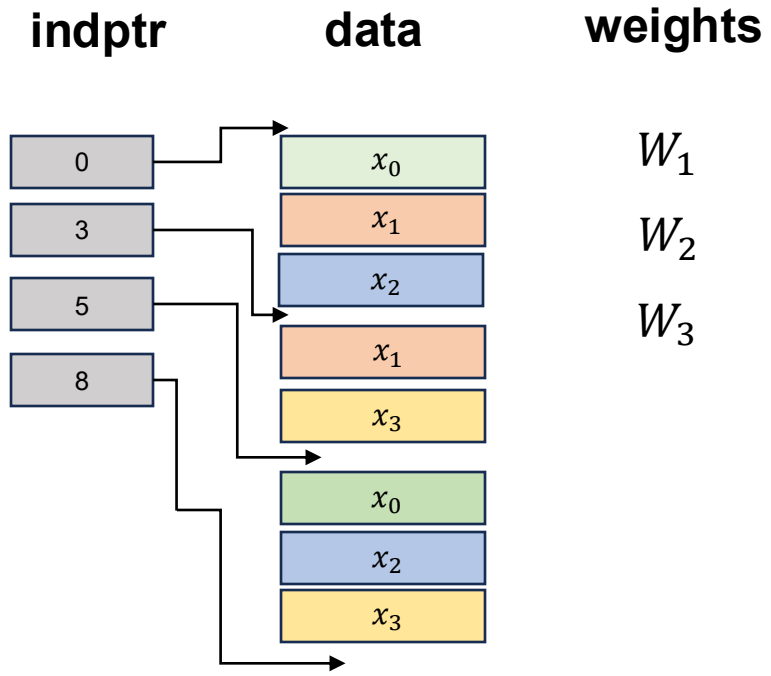
Getting Permutation Indices with Prefix Sum



Prefix sum(scan) can be efficiently parallelized in GPU

Revisit the Batched Compute





Discussion: How to get indptr from the existing data?

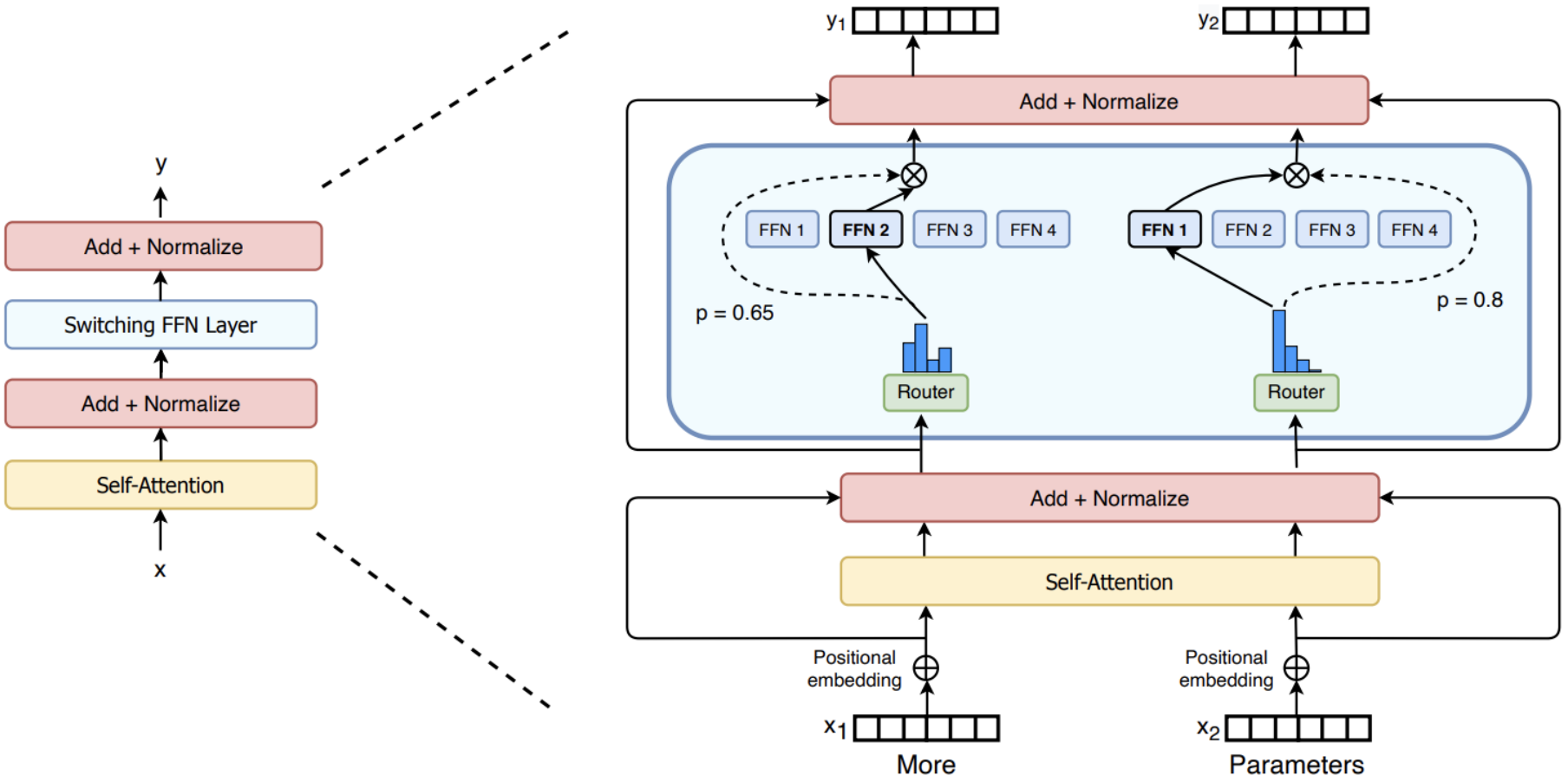
Mixture-of-Experts Outline

How do MoE architectures work?

How to optimize MoE kernels?

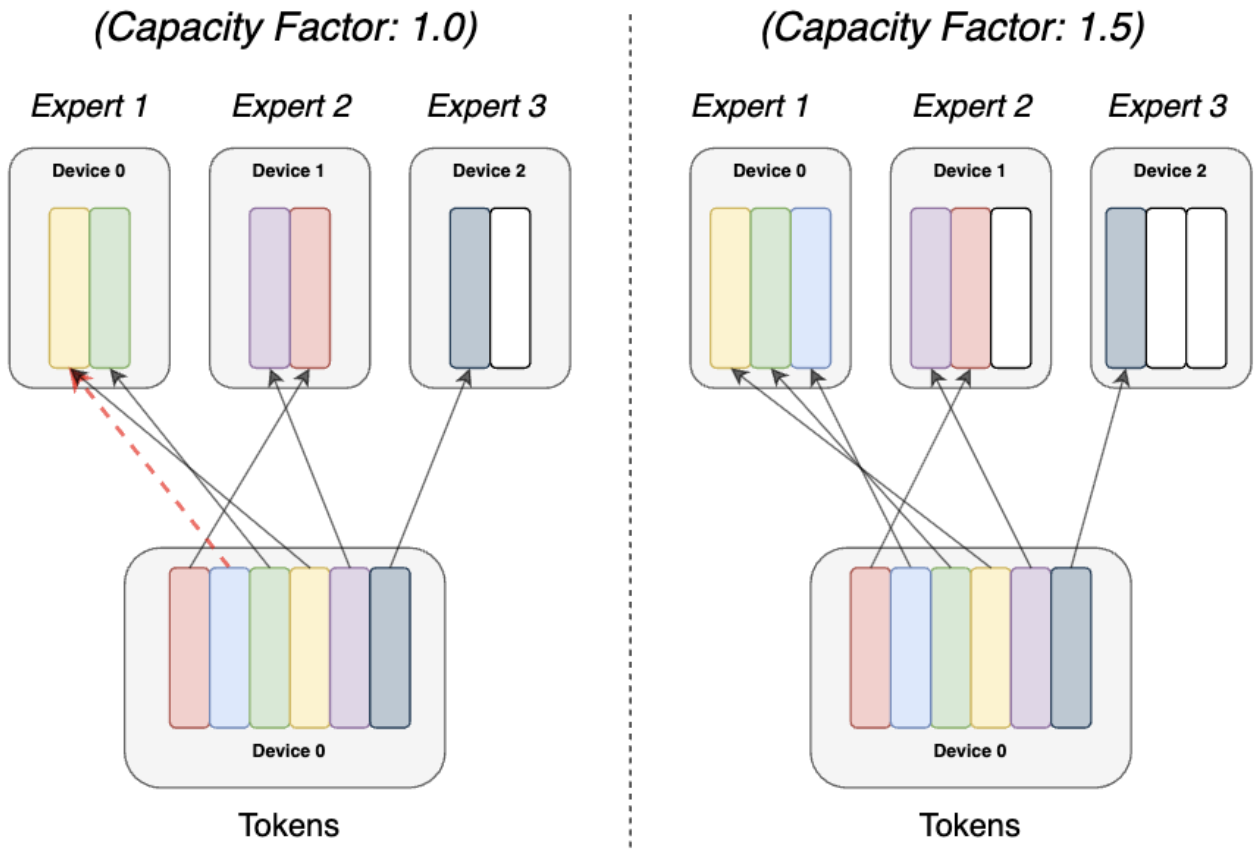
How to parallelize MoE models?

How to Parallelize MoE Model Training/Inference?



Expert Parallelism

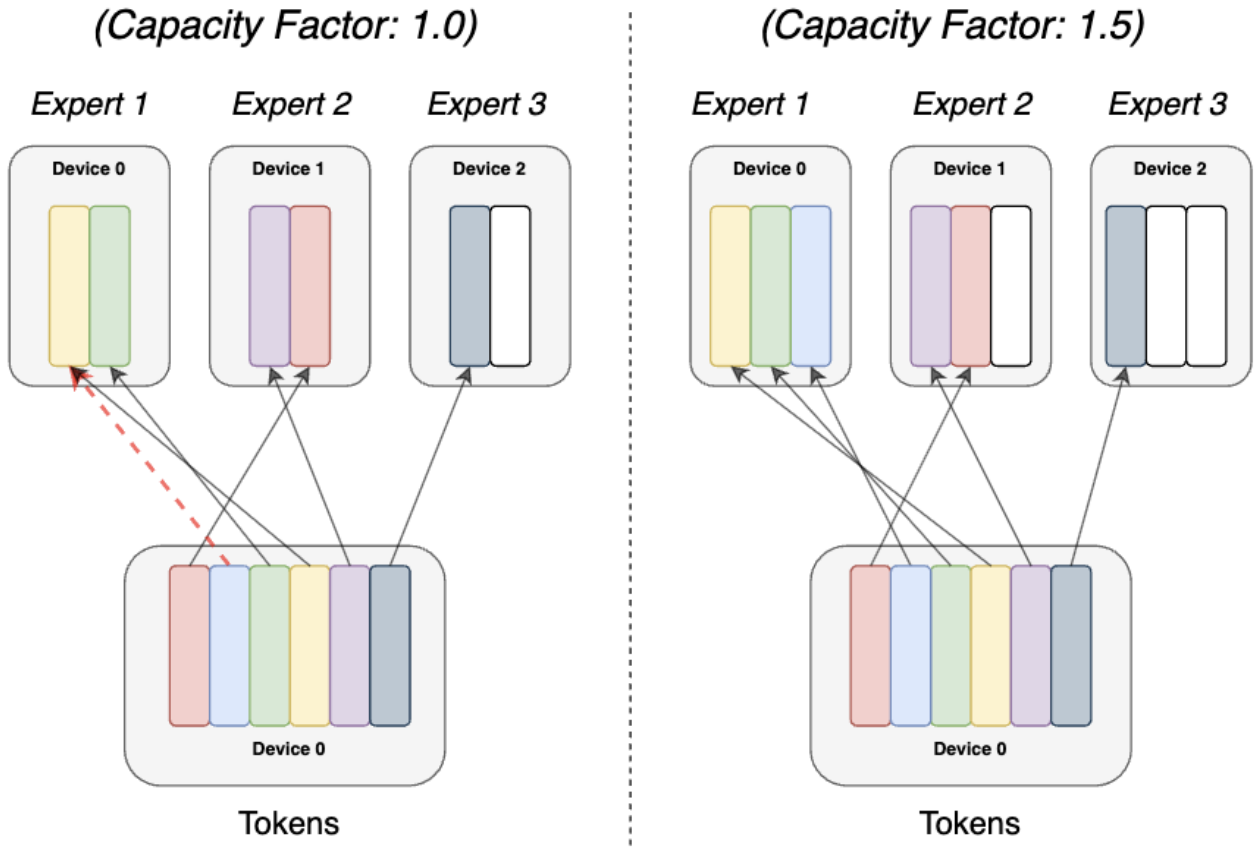
- Each expert is assigned to a GPU (similar to model parallelism)
- **Expert capacity**: batch size of each expert: tokens_per_batch / num_experts * capacity_factor
- Tokens exceeding expert capacity are ignored



Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity

Expert Parallelism: Design Tradeoffs

- If the capacity factor is too small, overflow tokens are directly passed to the next layer via residual
- If the capacity factor is too large, GPUs will be underutilized



Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity

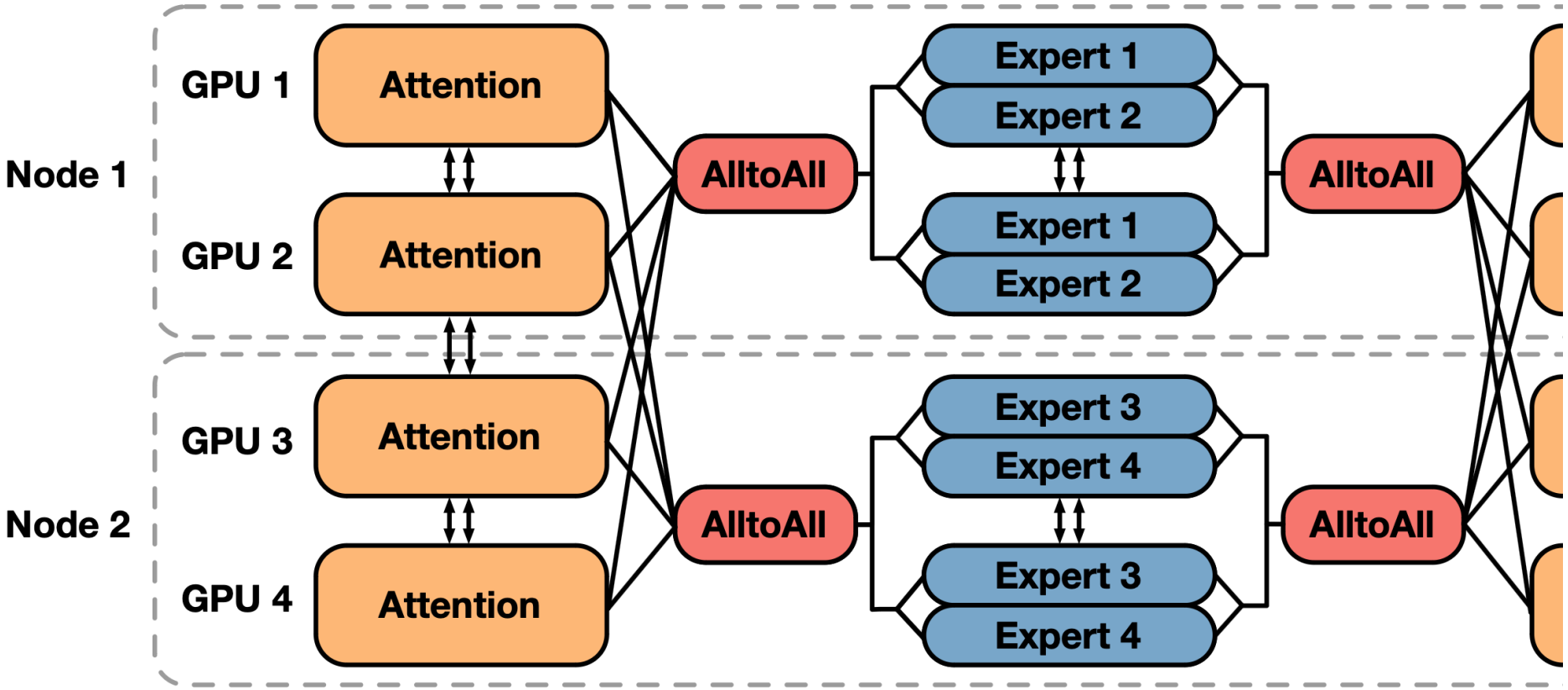
Expert Parallelism: Load Balancing

- **Problem**: most tokens are dispatched to a small number of experts
  - Popular experts tend to get more tokens -> more accurate and popular
  - Other experts do not get sufficient training data

- **Solution**: introduce load balance term to the loss
  - Encourage distributed load evenly across experts within each batch
  - $L_{LB} = \sum_{e=1}^{E} f_e P_e$
  - f_e is the fraction of tokens routed to expert e
  - P_e is the average gating probability for expert e

Hybrid Parallelism for MoE Training

- Data and model parallelism for attention layers: AllReduce communication
- Expert parallelism for MoE layers: AlltoAll communication



Enhanced MoE Parallelism, Open-source MoE Model Training Can Be 9 Times More Efficient