15-442/15-642: Machine Learning Systems

Mixture of Experts

Spring 2024

Tianqi Chen and Zhihao Jia
Carnegie Mellon University

Outline

Mixture of Experts

Efficiently Compute Mixture of Experts

Outline

Mixture of Experts

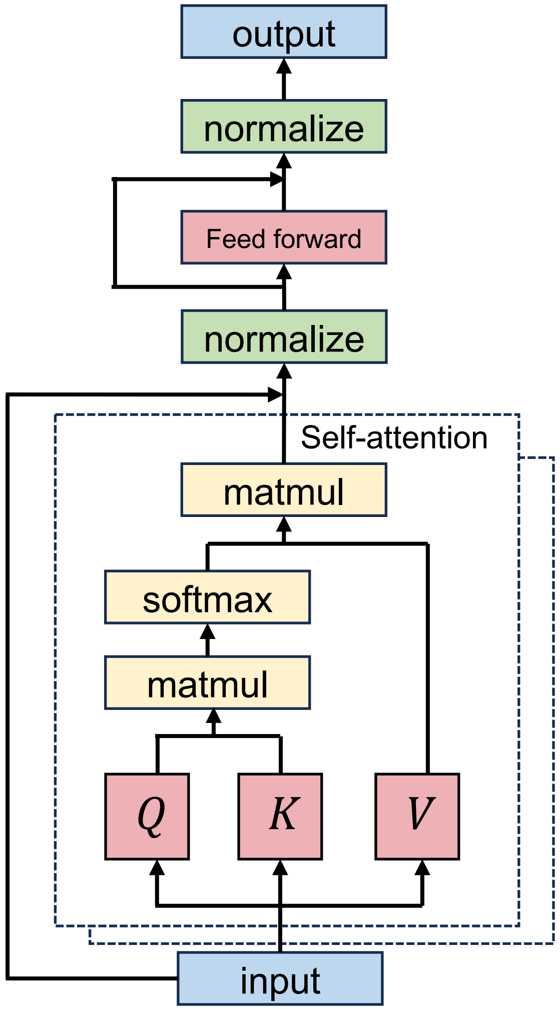
Efficiently Compute Mixture of Experts

Recap: Transformer Block

A typical transformer block

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

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



Normal Feed Forward Layer

Feed forward

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

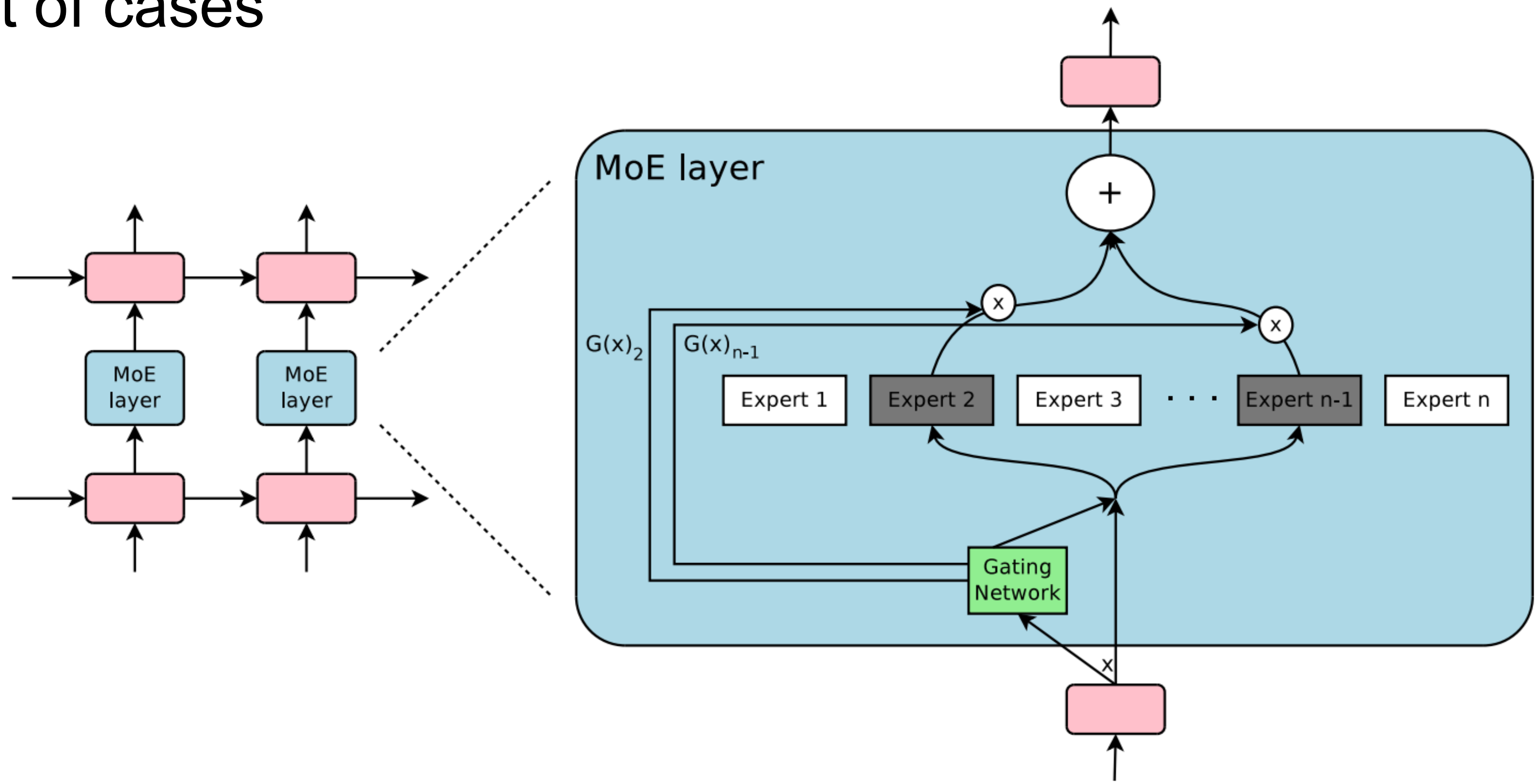
$W_1 \in R^{n \times m}$

Increasing feature size will increase compute quadratically

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

Mixture-of-Experts

Key idea: make each expert focus on predicting the right answer for a subset of cases



In practice, each expert here is an FFN

A Closer Look at Mixture-of-Experts

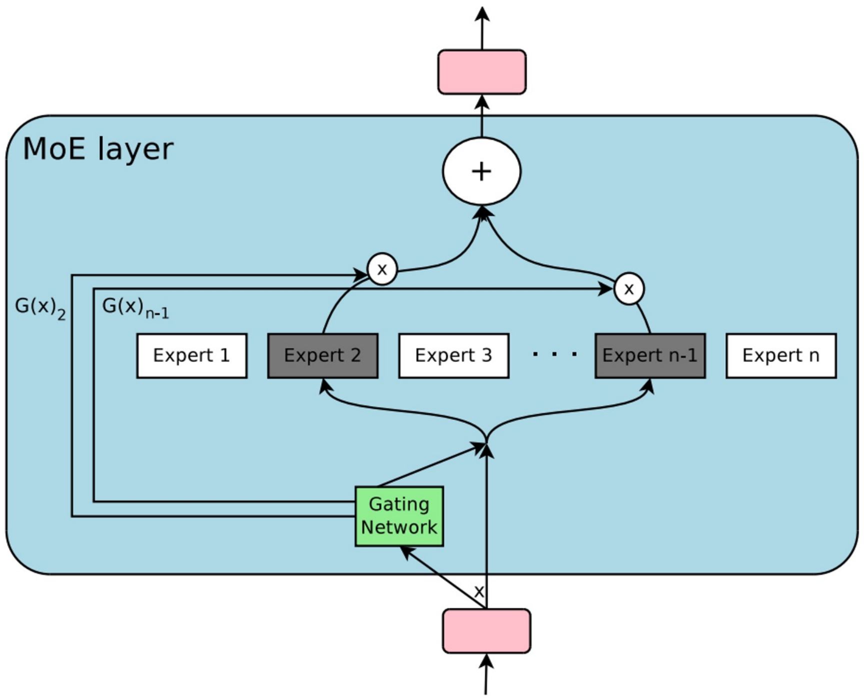
A typical MoE layer (assume single instance and activate two experts)

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

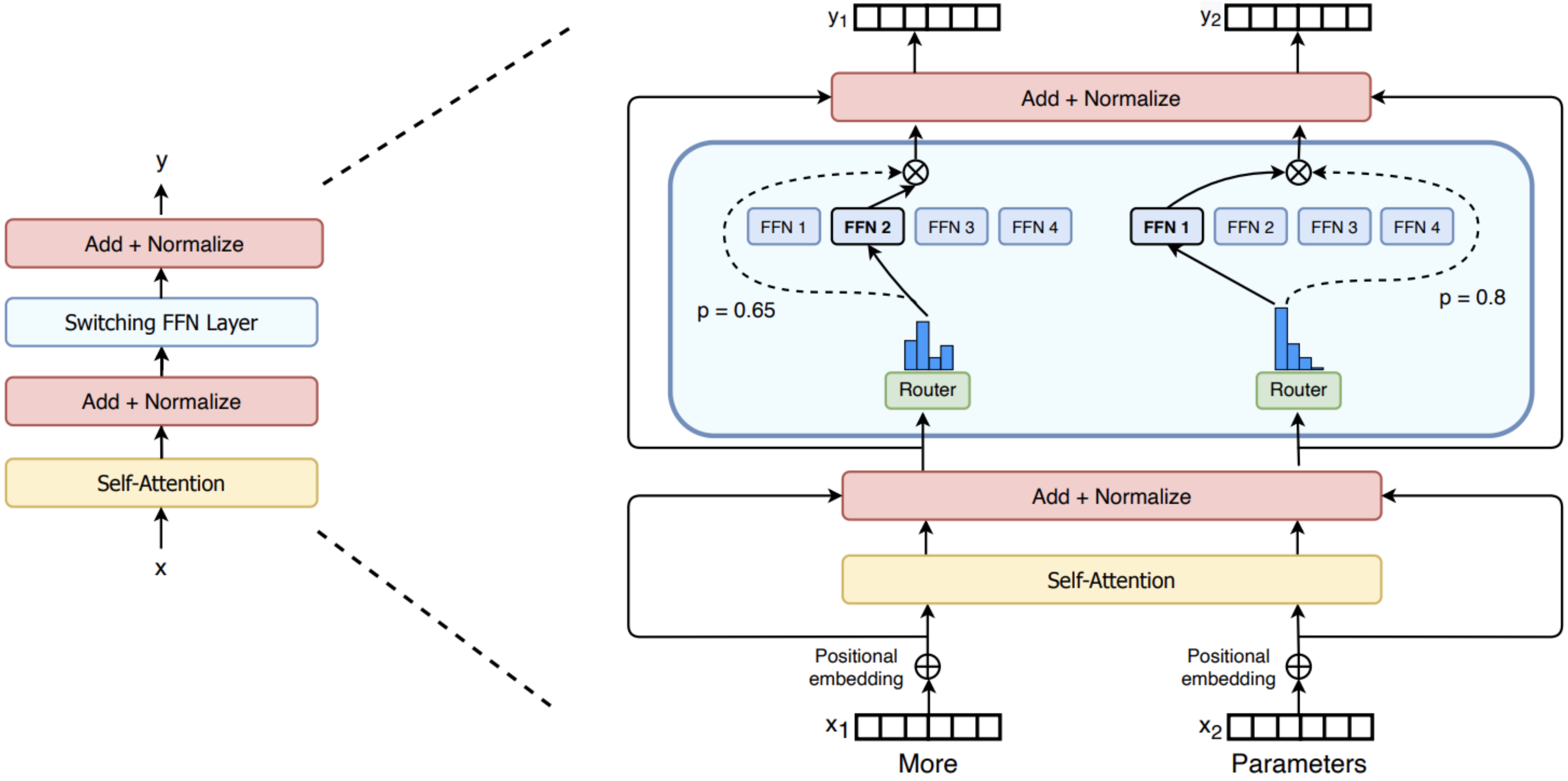
Different models may have different FFN configurations, usually contains multiple linear layers and some non-linear mixing

Discussions

What are the advantage of using Mixture of Experts vs Linear layers

Transformers + Mixture of Experts

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



Outline

Mixture of Experts

Efficiently Compute Mixture of Experts

Discussions

What are opportunities and challenges in accelerating mixture of expert layers?

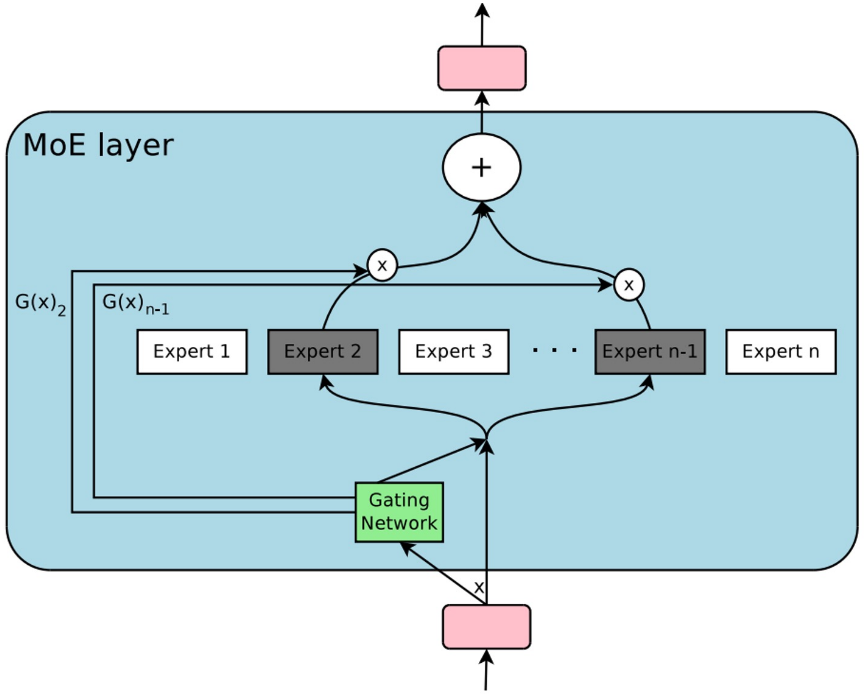
Single Batch Setting

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

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

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



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

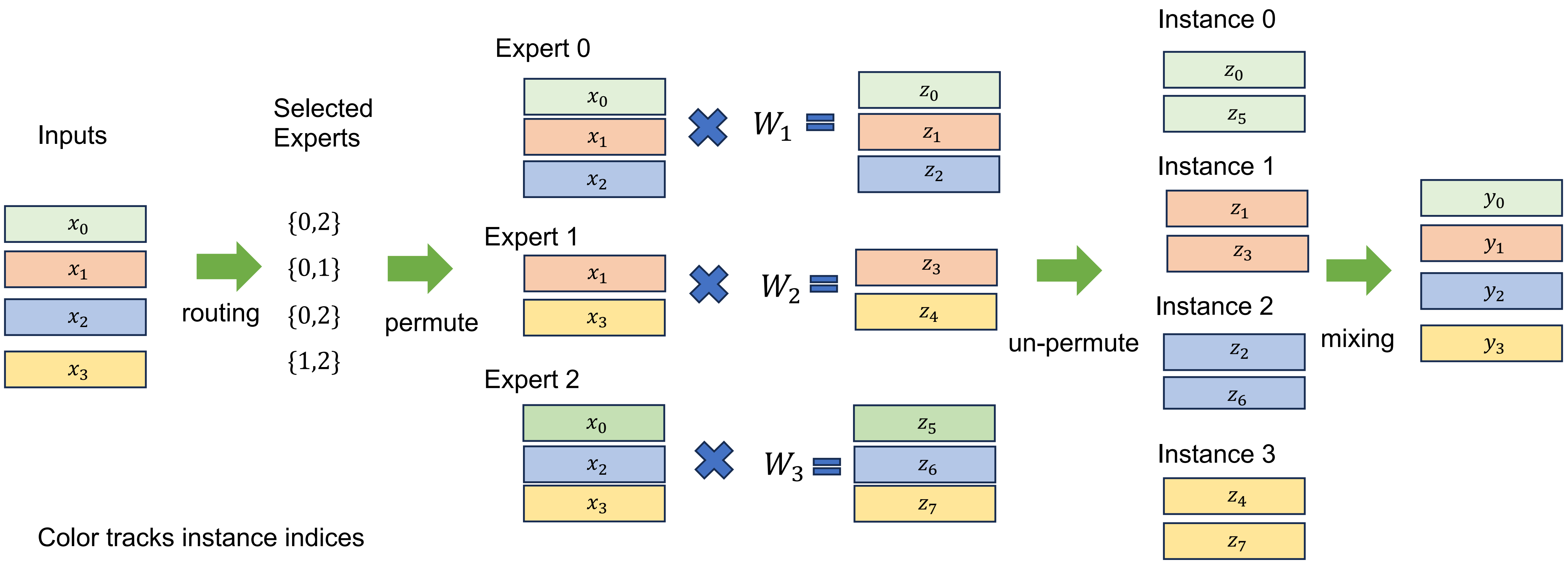
$Z = X\,W, X \in \mathbb{R}^{b \times n}, W \in \mathbb{R}^{n \times m}$

b is the batch size

When b becomes larger, we get better compute efficiency

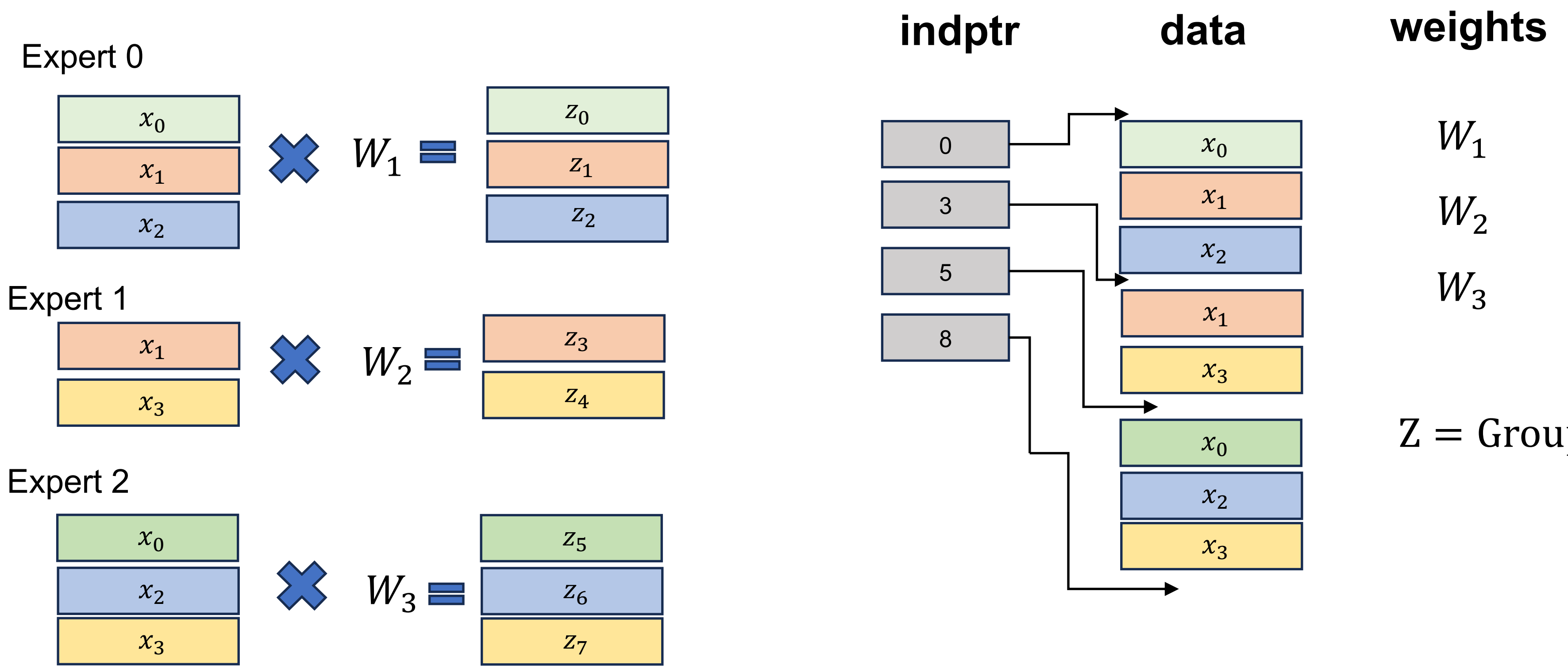
due to memory load reuse in matrix multiply and hardware specialization via TensorCore

Batching MoE computation



Gale et.al MegaBlocks: Efficient Sparse Training with Mixture-of-Experts

Batched Expert Compute



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

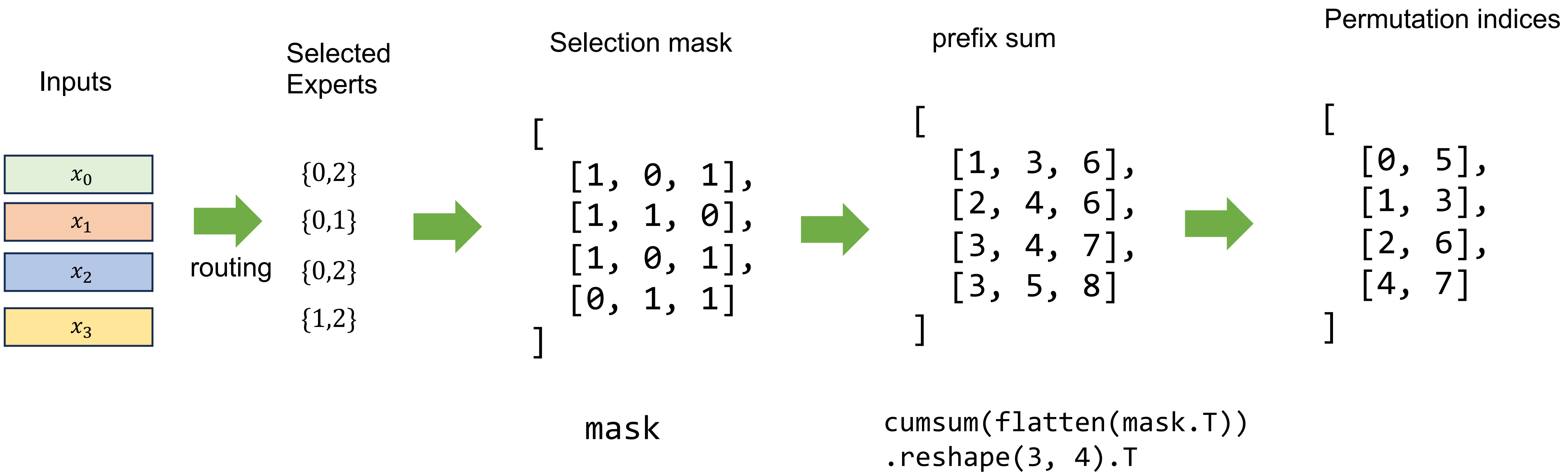
In practice can be computed by one GPU kernel, benefit from batching

Example input in compressed row(CSR) format

A Closer Look at Permutation

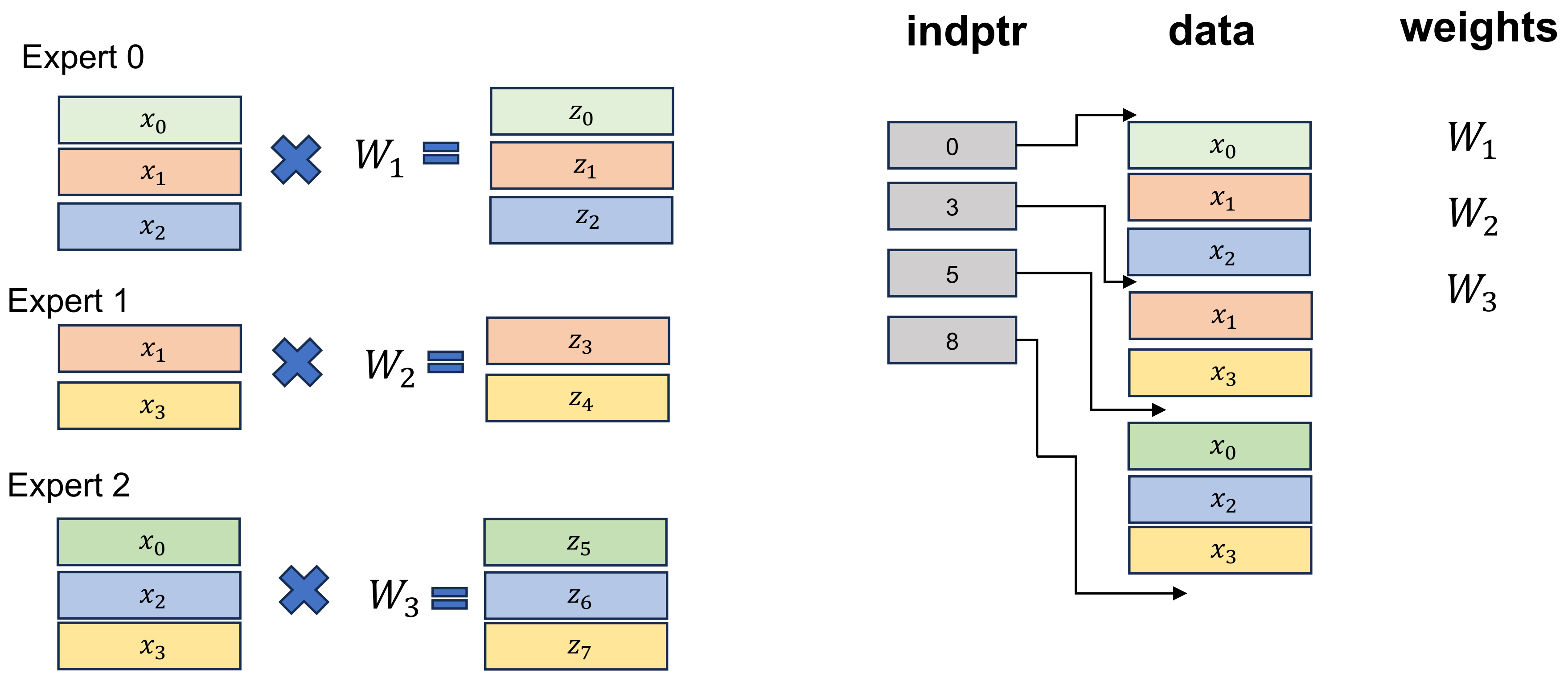
How to get the permutation indices efficiently in GPU?

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?

Discussions

What are opportunities and challenges in parallelizing mixture of expert layers?

Outline

Mixture of Experts

Efficiently Compute Mixture of Experts