Mixture of Experts for FFN

Karthik Inbasekar

12-11-2025

MoonMath.ai

Feed Forward Network (LLM)

- In diffusion transformers, FFN consists of generally a two layer MLP (Multi Layer Perceptron)

$$h(x) = x \cdot W_1 + b_1$$
$$F(x) = \sigma(h(x)) \cdot W_2 + b_2$$

$x \in \mathbb{R}^{d_{model}}$: input vector/token embedding

$b_1 \in \mathbb{R}^{d_{ff}}$:bias vector

$W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$: Weight matrix

$W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$: Weight matrix

$b_2 \in \mathbb{R}^{d_{model}}$:bias vector

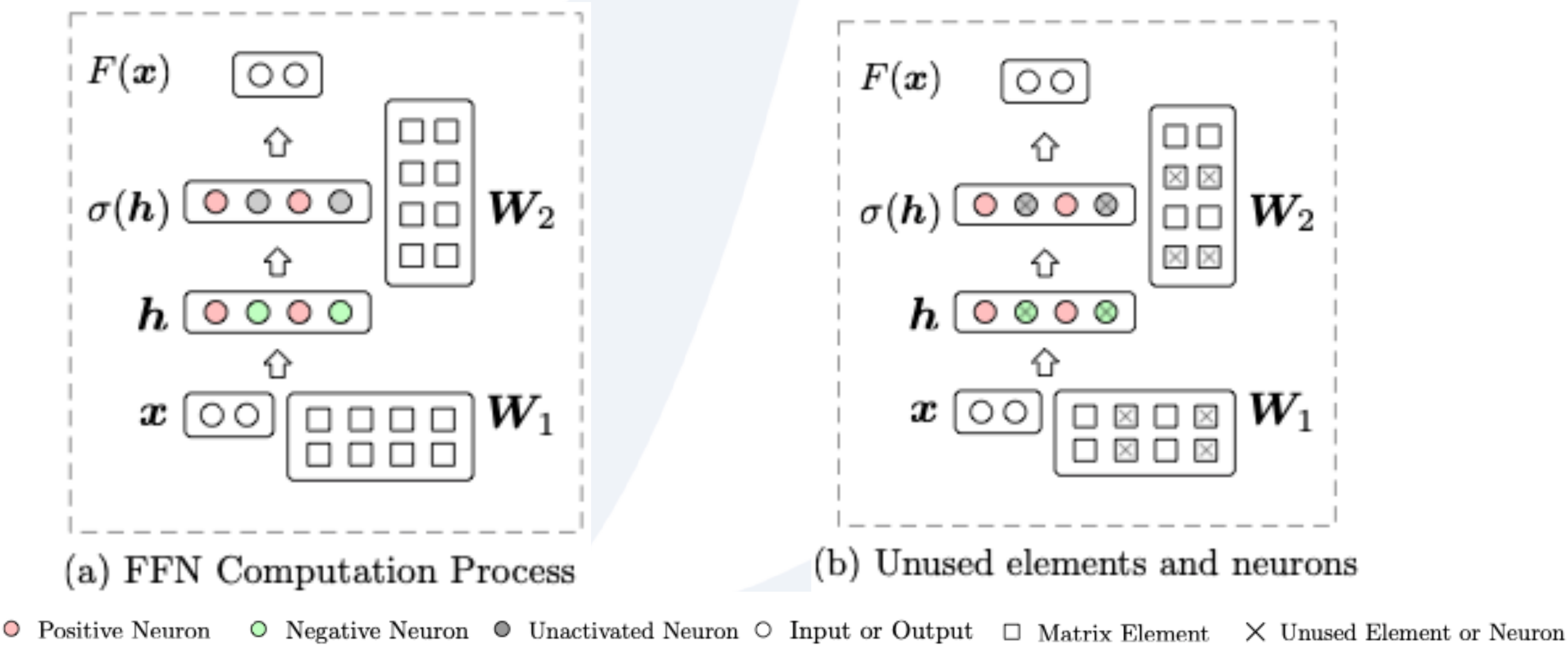
- $\sigma(\cdot)$: some non linear activation function: like RELU/GELU/SiLU

- Computation complexity in FLOPs is dominated by W1, W2 mults

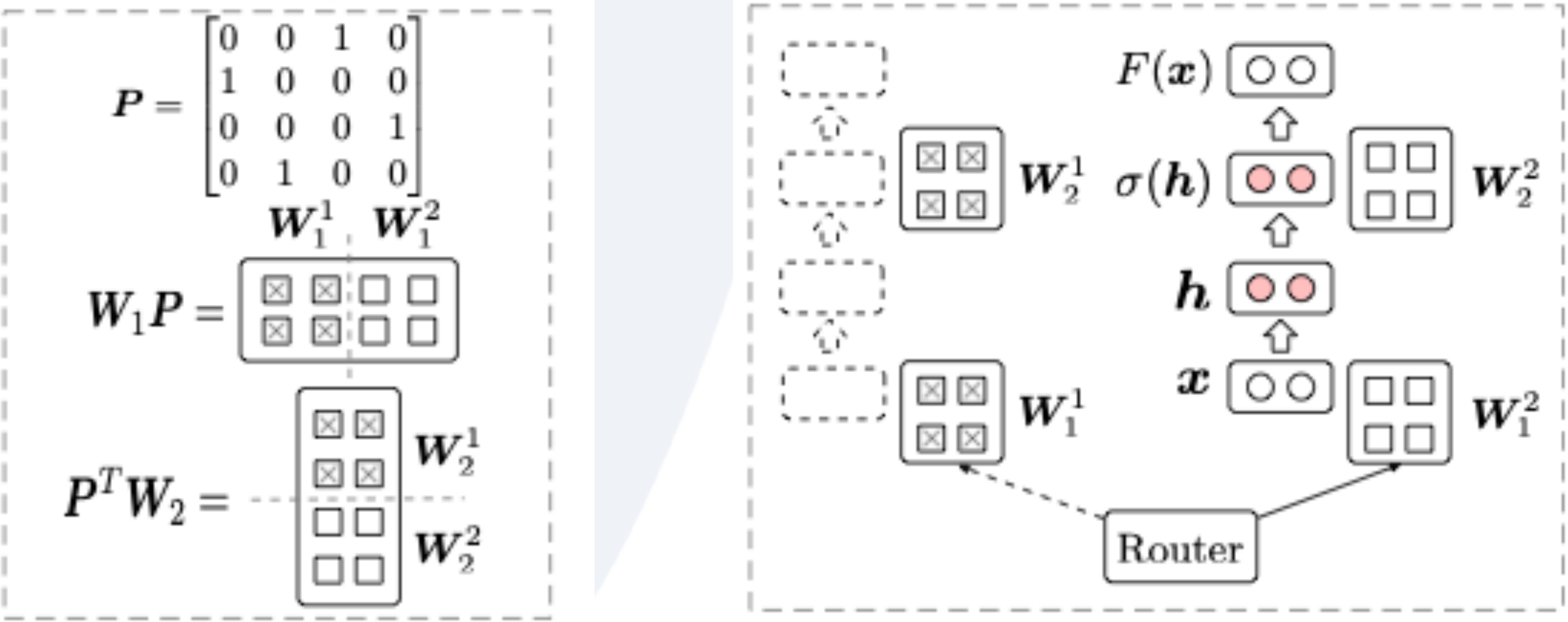
$$\mathcal{O}(A_{m \times p} \cdot B_{p \times n}) = 2 \cdot m \cdot p \cdot n \ \ FLOPS$$

$$\simeq \mathcal{O}(2 \cdot d_{ff} \cdot d_{model}) = 8d_{model}^2 \qquad d_{ff} = 4d_{model}$$

MOE: Feed Forward Network (LLM)

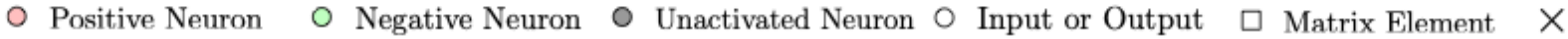


MOE: Feed Forward Network LLM



(c) Expert Construction

(d) FFN with MoE



Positive Neuron Negative Neuron Unactivated Neuron Input or Output Matrix Element Unused Element or Neuron

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

MOE: Feed Forward Network LLM

- Original FFN

$$h(x) = x \cdot W_1 + b_1$$

$$F(x) = \sigma(h(x)) \cdot W_2 + b_2$$

- Permutation $PP^T = I$

$$\begin{aligned} F(x) &= \sigma(h(x)) \cdot PP^T \cdot W_2 + b_2 \\ &= \sigma(h(x)P) \cdot P^T \cdot W_2 + b_2 \\ &= \sigma(x \cdot W_1 \cdot P + b_1 \cdot P) \cdot P^T \cdot W_2 + b_2 \end{aligned}$$

$$P_{d_{ff} \times d_{ff}}$$

- Split into N parts

$$[W_1^1, W_1^2, \ldots, W_1^N] = W_1 P$$

$$b_1^1 \oplus b_1^2 \oplus \ldots \oplus b_1^N = b_1 P$$

$$[(W_2^1)^T, (W_2^2)^T, \ldots, (W_2^N)^T] = (P^T W_2)^T$$

MOE: Feed Forward Network LLM

- Split into N parts using a permutation matrix

$$F(x) = \sum_{i=1}^{N} g_i(x) \cdot F_i(x)$$

- Idea is to train a router (gating network) $g_i(x)$ weights for expert i, based on input x

- Problem is there are many combinations of experts, and how to pick the best combination.

- The goal of the gating network is to compute raw scores for ALL the N experts

$$h_i(x) = x \cdot W_g + b_g$$

$W_g \in \mathbb{R}^{d_{model} \times N}$: Weight matrix $\quad$ $b_g \in \mathbb{R}^N$: bias vector $\quad$ Parameters of the gate

- And use top-k selection mechanism to select the top k scores out of N in h

MOE: top k softmax

- If $S \equiv \{i_1, i_2, \ldots i_k\}$ is the set of $k < N$ experts that are selected

- Create a sparsity mask

$$M_i = \begin{cases} 1 & \text{If } i \in S(h) \\ 0 & \text{Otherwise} \end{cases}$$

- Compute masked logits by setting unwanted experts to large negative weights

$$z = h \circ M + (1 - M) \cdot (-\infty)$$

- Gate router weight

$$g_i(x) = softmax(z)_i = \frac{e^{z_i}}{\sum_{i=1}^{N} e^{z_i}}$$

$$\sum_{i=1}^{N} g_i(x) = 1 \qquad \text{Only } k < N \text{ of them are non zero}$$

Choosing k < N : compute vs capacity

- Raw score computation

$$h_i(x) = x \cdot W_g + b_g \qquad \simeq \mathcal{O}(N \cdot d_{model})$$

$W_g \in \mathbb{R}^{d_{model} \times N}$: Weight matrix $\quad b_g \in \mathbb{R}^N$: bias vector $\quad$ Parameters of the gate

- Expert forward pass

$$\begin{aligned} h^i(x) &= x \cdot W_1^i + b_1^i \\ F^i(x) &= \sigma(h^i(x)) \cdot W_2^i + b_2^i \end{aligned} \qquad \simeq \mathcal{O}(k \cdot 2 \cdot d_{expert} \cdot d_{model})$$

- Compute Complexity scaling

$$\frac{\mathcal{O}(MOE)}{\mathcal{O}(dense)} = \frac{k \cdot FLOPS_{expert}}{FLOPS_{dense}} \simeq \frac{k \cdot d_{expert}}{d_{ff}}$$

Choosing k < N : compute vs capacity

- Increasing k increases the number of experts per token, and improves performance and accuracy, while decreasing k makes fewer combinations available and limits the model ability to specialize.
- K experts per token is a general measure of the cost
- If there are batches of 64 tokens, then it involves 64k forward passes per batch
- Inference cost increases linearly with k, (k is often proprietary) per token
    - Mixtral 8x7b: N = 8, k = 2
    - Gemini 1.5 pro N (very large unknown), k = 2 - 4 dynamically selected
    - Chatgpt4 (hypothesized) N = 8 or 16, k = 2

Parameter scaling heuristic

$$\frac{P_{MOE}}{P_{dense}} \simeq \frac{2 \cdot d_{model} \cdot d_{expert} N}{2 \cdot d_{model} \cdot d_{ff}} = \frac{N \cdot d_{expert}}{d_{ff}} \qquad \frac{\mathcal{O}(MOE)}{\mathcal{O}(dense)} = \frac{k \cdot FLOPS_{expert}}{FLOPS_{dense}} \simeq \frac{k \cdot d_{expert}}{d_{ff}}$$

- Max capacity (model scaling) $d_{expert} = d_{ff}$

$$P_{MOE} \simeq N \cdot P_{Dense} \qquad \mathcal{O}(MOE) \simeq k \cdot \mathcal{O}(dense)$$

Scaling the total model size as the number of experts with the slight increase in compute complexity

- Constant compute $d_{expert} = \dfrac{d_{ff}}{k}$

$$P_{MOE} = \frac{N}{k} \cdot P_{Dense} \qquad \mathcal{O}(MOE) \simeq \mathcal{O}(dense)$$

Keeping the compute complexity approx the same, moderate improvement in parameters

Parameter scaling heuristic

$$\frac{P_{MOE}}{P_{dense}} \simeq \frac{2 \cdot d_{model} \cdot d_{expert} N}{2 \cdot d_{model} \cdot d_{ff}} = \frac{N \cdot d_{expert}}{d_{ff}}$$

$$\frac{\mathcal{O}(MOE)}{\mathcal{O}(dense)} = \frac{k \cdot FLOPS_{expert}}{FLOPS_{dense}} \simeq \frac{k \cdot d_{expert}}{d_{ff}}$$

- Inference cost saving scaling $d_{expert} = \dfrac{d_{ff}}{N}$

$$P_{MOE} \simeq P_{Dense}$$

$$\mathcal{O}(MOE) \simeq \frac{k}{N} \cdot \mathcal{O}(dense)$$

Keeping the original model size for the architecture, maximum performance push