Mixture of experts LMs

↳ bigger models + more data
= lower loss ⇒ better next-token prediction

how to add params?

↳ more layers
↳ layers ≈ "blocks"

↳ increase dimensionality of embeddings / hidden states

↳ increase # of heads

↳ last time: dense model

↳ each parameter is used in computations for every token

↳ today: sparse models

↳ what if we have a ton of params, but only use a small % for each token?

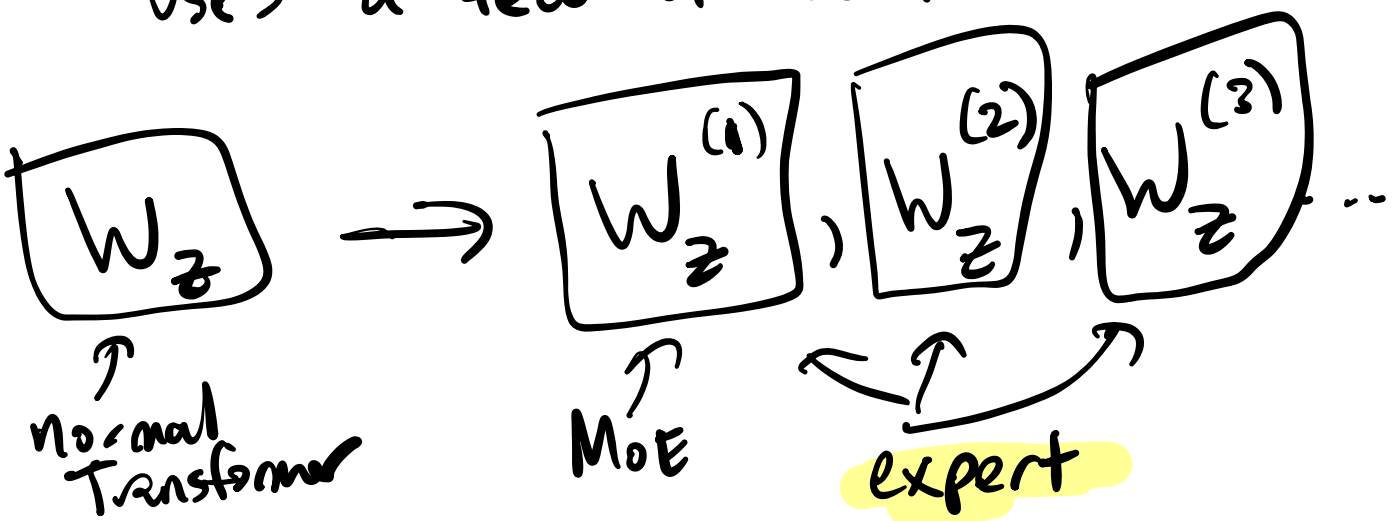
↳ MoE models:

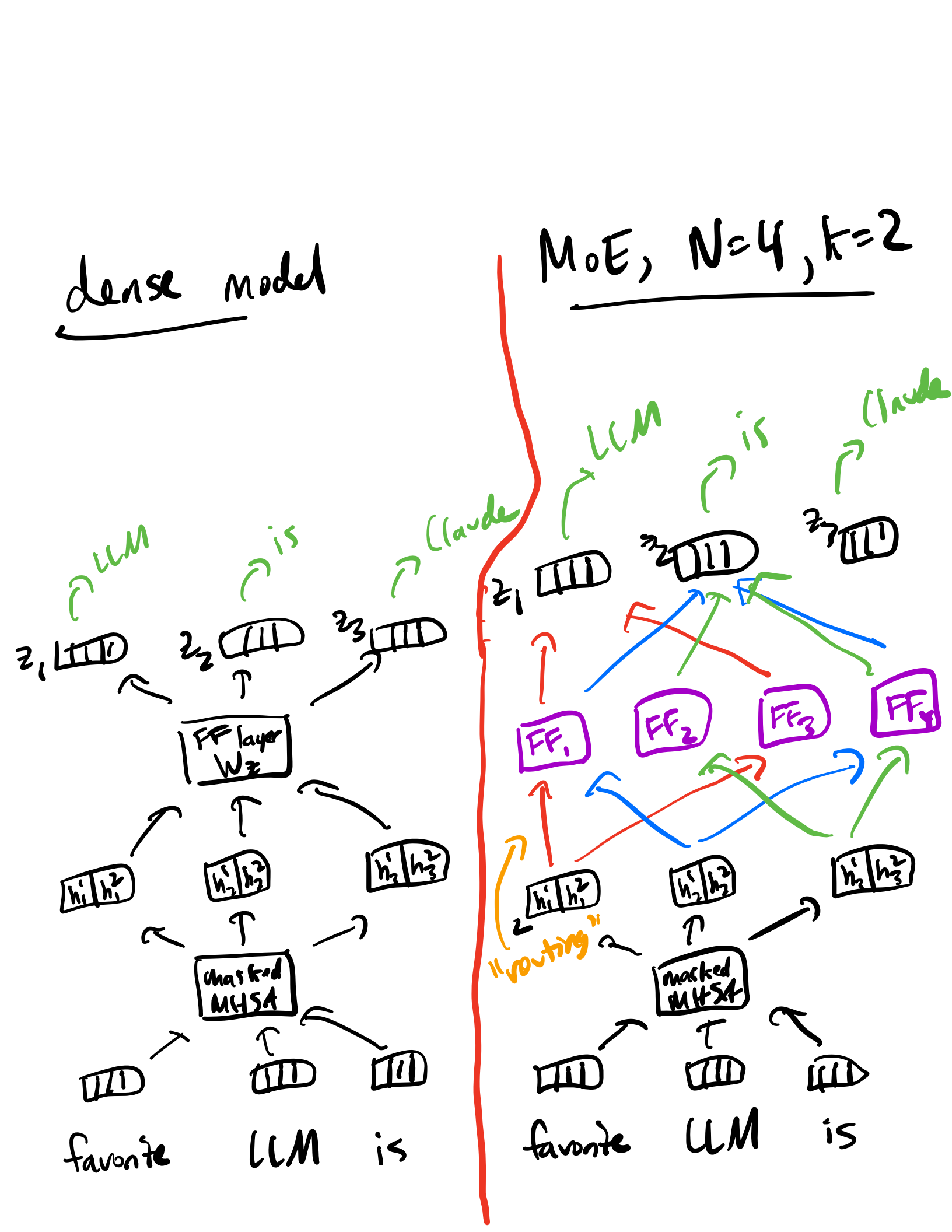
↳ W_z: feed-forward layer where most of the model params live

$$z_t^L = f\left(W_z\left[h_t^1, h_t^2\right]\right) + z_t^{L-1}$$

residual

↳ what if we have many W_z's per layer, and each token only uses a few of them




dense model
MoE, N=4, k=2
favorite
LLM
is
Claude
FF layer
masked MHSA
"routing"
FF1
FF2
FF3
FF4

↳ parameters of MoE to grow w/ number of experts N, compute per token grows with the number of active experts k

---

how do we train the router?

↳ input to router at timestep t

$$x_t = [h_t^1, h_t^2]$$

↳ output: one score per expert

$$s_t = W_r x_t$$

W_r: $N \times d$ → dim of x_t
↳ # of experts

↳ gates $g_t = \text{softmax}(s_t)$

↳ given g_t, we want to select the top-k gates. we select T_t: set of active experts at t

↳ MoE eq:

$$z_t^L = z_t^{L-1} + \sum_{i \in T_t} g_{t,i} \cdot f(W_z^{(i)} x_t)$$

each expert opts its own W_z

↳ the $N-k$ inactive experts are never run for timestep t

↳ dense: MoE w/ $N=1, k=1$

↳ softmax usage in LLMs

1. attention mechanism
2. router
3. output layer

---

training MoE:

↳ $g_{t,i}$ scores are the only

↳ scores for active gates

contribution from the router to the next-token prediction

↳ router has weight matrix W_r

↳ top-k function is not differentiable!

↳ example:

g_t: $\langle 0.61, 0.14, 0.22, 0.03 \rangle$

select expert 1, 3

↳ do gradient descent and it tells me to make

g_3: 0.21

↳ if I increase g_2 to 0.15, T_t is unchanged,

if I increase g_2 to above 0.22, T_t becomes $\{1,2\}$ from $\{1,3\}$

top-k f_R is like a switch

↳ Switch transformer

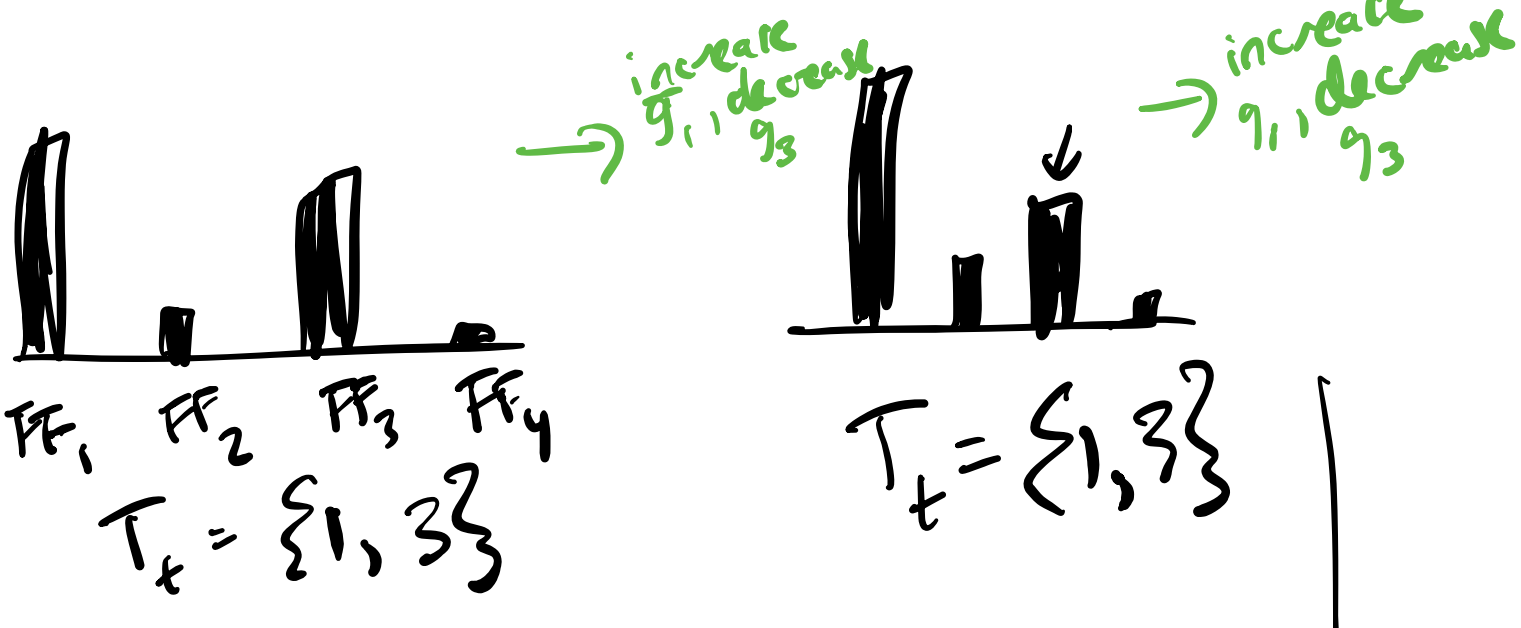
↳ gradient only flows thru experts that were picked

So how do we actually train the router to make good choices?

↳ trick: the top-k choice is not differentiable, but the scores $g_{t,i}$ are

↳ top-k choice gives a ranking

↳ if we change the scores enough, ranking changes





$T_t = \{1, 2\}$

---

total vs. active params:

total: stored in memory

active: what's used at each token
⇒ compute

⇒ model naming:

Qwen3 -30B- A3B

test-time scenarios

batch size	Qwen-30B-A3B	Qwen-30B
1 seq	much faster	
1000s of seq	much faster	
100 seq	tie	tie