

# Mixtures of Experts

## Lecture 4

Anay Mehrotra

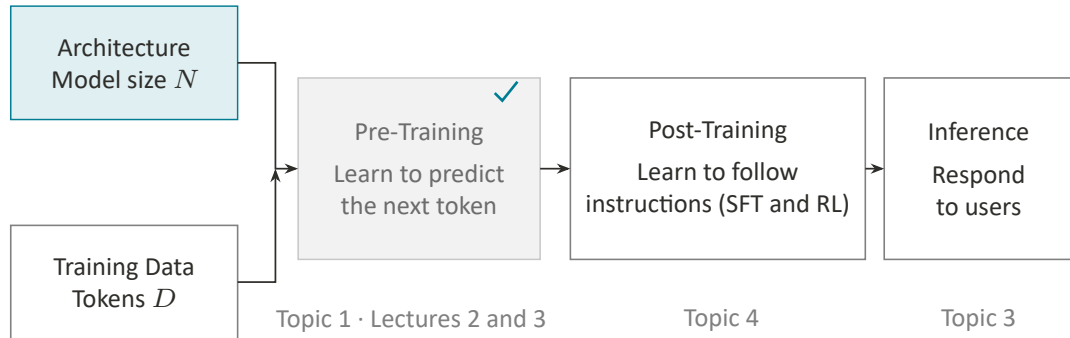
MS&E 319

Autumn 2026

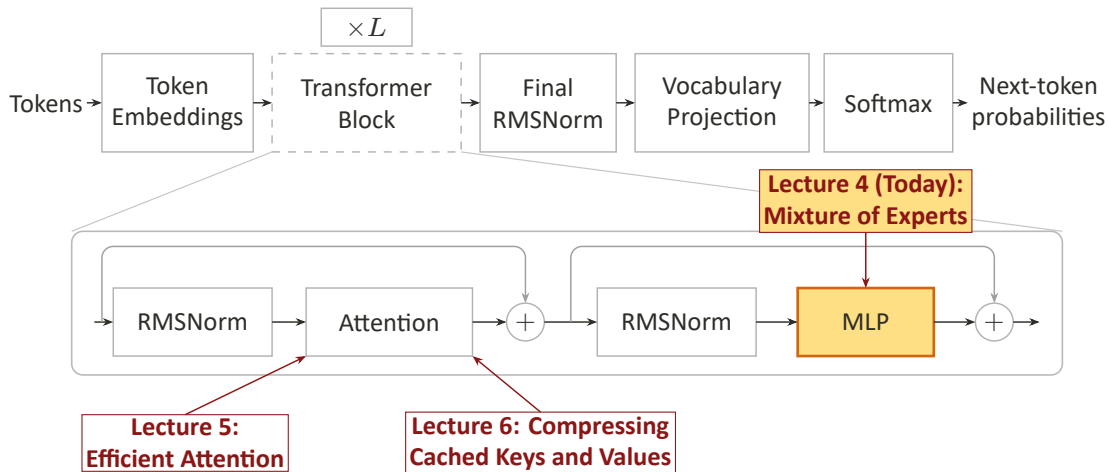


# The LLM Pipeline

## Topic 2 · Lectures 4–6



# Topic 2: Innovations in Architecture

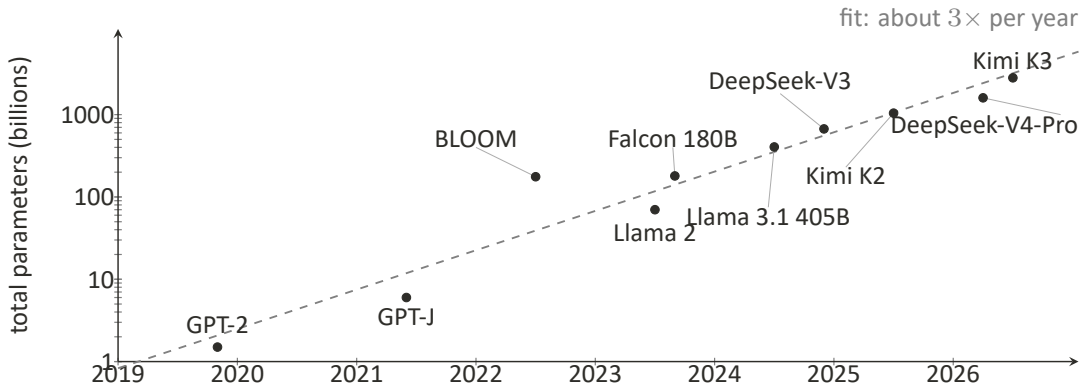


Keep quality fixed. Can each token cost less?



# Models Keep Growing

Open-weight models have grown about  $2,000\times$  since 2019.



Sources: various model releases and reports.  
MS&E 319 Lecture 4

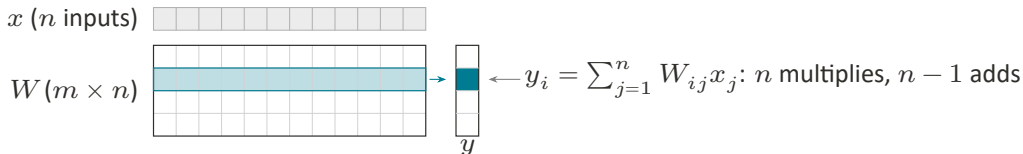


# Dense Models Use Every Weight

Larger models reach lower loss (Lecture 2).

In a dense model, every token uses every weight:

FLOPs per token  $\approx 2 \times$  parameters



So  $y = Wx$  costs about  $2mn$  FLOPs: two per weight.

A Transformer is mostly such matrices, each used once per token.

Not counted: attention's scores and weighted sums, which grow with context length; embedding lookups; nonlinearities.



# Dense Models Use Every Weight

Larger models reach lower loss (Lecture 2).

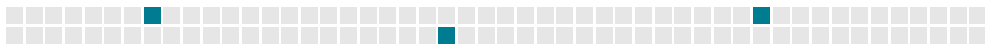
In a dense model, every token uses every weight:

$$\text{FLOPs per token} \approx 2 \times \text{parameters}$$

one multiply and one add per weight

So  $10\times$  the parameters costs  $10\times$  more per token, to train and to run.

Yet for each token, about 97% of T5-Base's MLP neurons output zero.



one token, one MLP layer (schematic): 3 of 100 neurons nonzero

Can a model store many parameters but use few per token?

Source: [Li et al. \(2023\)](#), abstract.

MS&E 319 Lecture 4

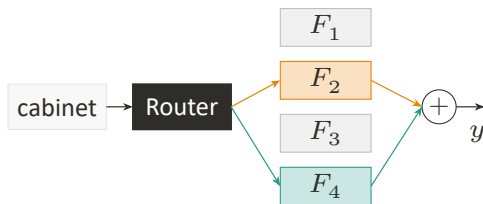
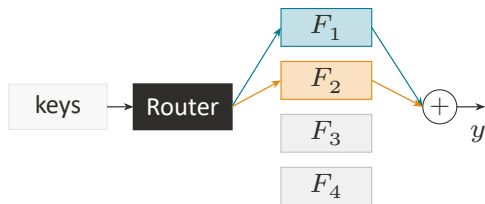


# Mixture of Experts (MoE)

Jacobs et al. (1991); Shazeer et al. (2017)

Replace the MLP with many MLPs, called “experts.”

A **router** sends each token to a few experts; only those run.



Do more experts help if the FLOPs per token stay fixed?

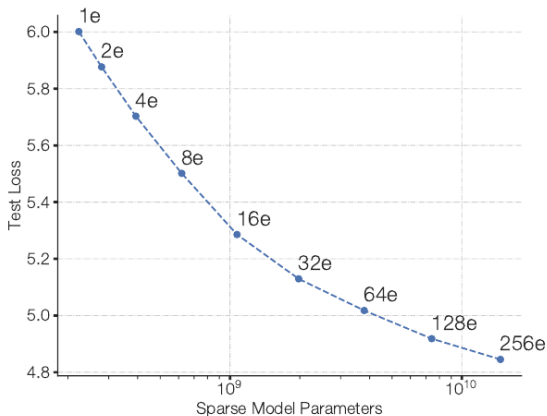


# Do More Experts Help? Yes.

Start: a dense 223M-parameter Transformer (1e).

In alternating Transformer layers, replace the MLP with 2 to 256 experts.

Each token uses one expert, so FLOPs per token match the dense model.

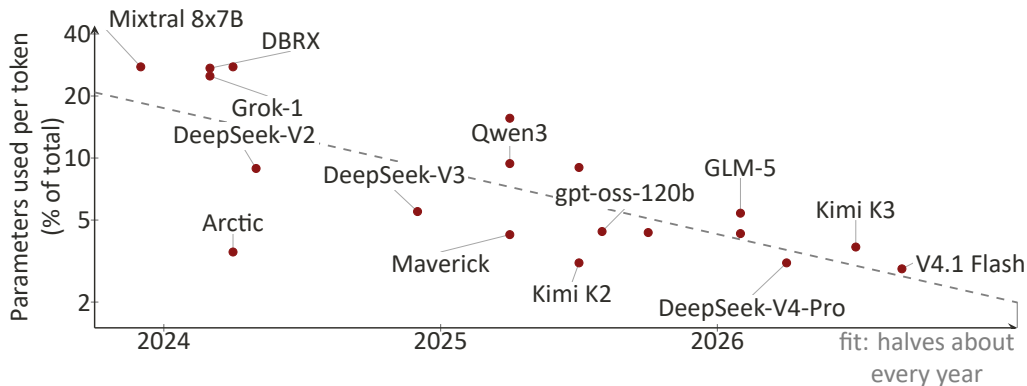


All models train for the same number of steps.

Source: [Fedus, Zoph, and Shazeer \(2022\)](#), Fig. 1 (left) and §3.  
MS&E 319 Lecture 4



# Released Models Increasingly Use MoE



Closed frontier models too, e.g., Gemini 1.5 Pro and 2.5, are MoE.

Sources: various model reports and configurations.

MS&E 319 Lecture 4



# Lecture Outline

MoE trades memory for compute: many parameters, few per token.

**1 Routing:** picks are discrete. How does the router learn?

Shazeer et al. (2017)

**2 Balance:** tokens can crowd onto a few experts.

Shazeer et al. (2017)

**3 Many devices:** store every expert; move tokens to them.

data and expert parallelism; GShard (2020)

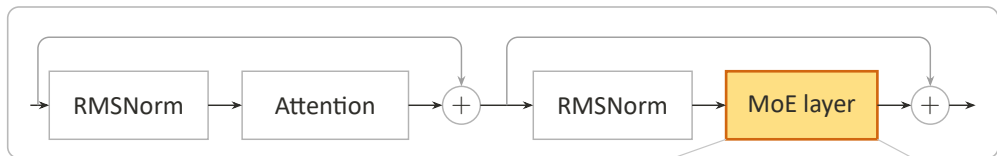
**4 Later designs:** new routing, experts, and balancing.

Switch (2022); DeepSeekMoE (2024); router biases (2024)

How do we route tokens, and train the router?

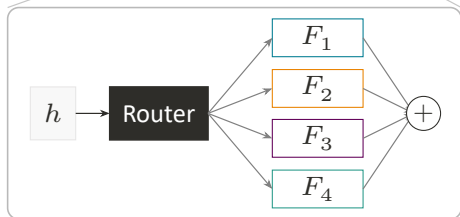


# MoE Layers Replace the MLP



Transformer MoEs replace the MLP with an MoE layer:

GShard in every other block;  
DeepSeek-V3 in all but the first 3.

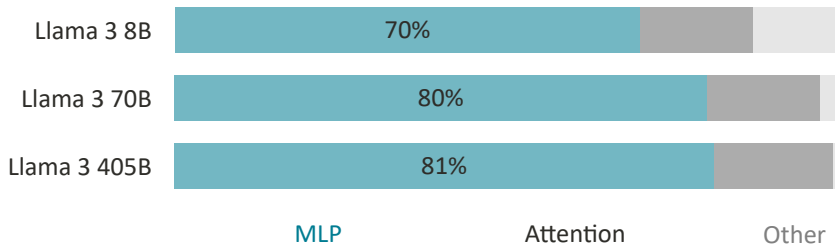


Shazeer et al. (2017) used LSTMs: one MoE layer between two LSTM layers.



# Why Put Experts in the MLP?

1. The MLP holds most of the weights.



2. The MLP works on each token separately, so each token can use its own MLPs.

Source: [Llama Team \(2024\)](#), Table 3; counts from layer dimensions.



# MoE Implementation

## A simplified router

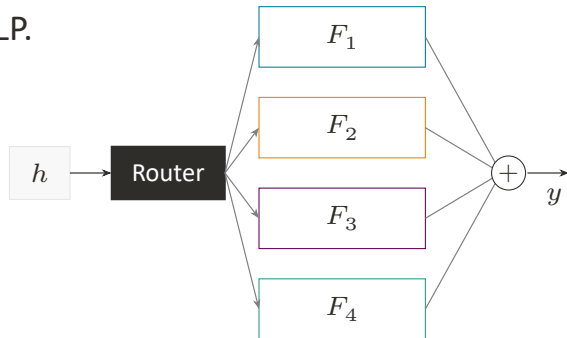
$E$  experts  $F_1, \dots, F_E$ , each an MLP.

For a token vector  $h \in \mathbb{R}^d$ :

$$y := \sum_{e \in S(h)} w_e(h) F_e(h).$$

the  $k$  experts  
picked for  $h$

the weight on  
expert  $e$



Parameters grow with  $E$ ; FLOPs per token grow with  $k$ .



# MoE Implementation

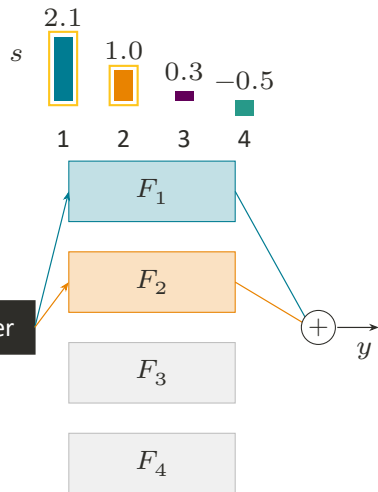
A simplified router

Score every expert:

$$s := W_r h, \quad W_r \in \mathbb{R}^{E \times d}.$$

$$S(h) := \text{TopK}_k(s), \quad \leftarrow \text{keep the } k \text{ largest scores}$$

$$w_e := \frac{e^{s_e}}{\sum_{j \in S(h)} e^{s_j}}. \quad \leftarrow \begin{array}{l} \text{take a softmax over them:} \\ \text{here } w_1 \approx 0.75, w_2 \approx 0.25 \end{array}$$

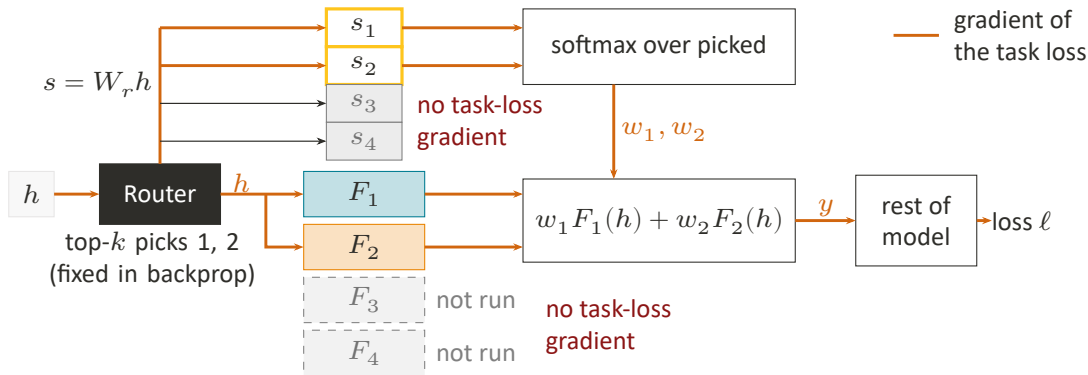


Shazeer et al. also add noise; we will see why.



# MoE Implementation

## How the router learns



From the task loss, the router learns only about the experts it picked.

This holds for softmax over the picked experts; Switch changes it.



# Lecture Outline

**1 Routing:** picks are discrete. How does the router learn?

Shazeer et al. (2017)



**2 Balance:** tokens can crowd onto a few experts.

Shazeer et al. (2017)

**3 Many devices:** store every expert; move tokens to them.

data and expert parallelism; GShard (2020)

**4 Later designs:** new routing, experts, and balancing.

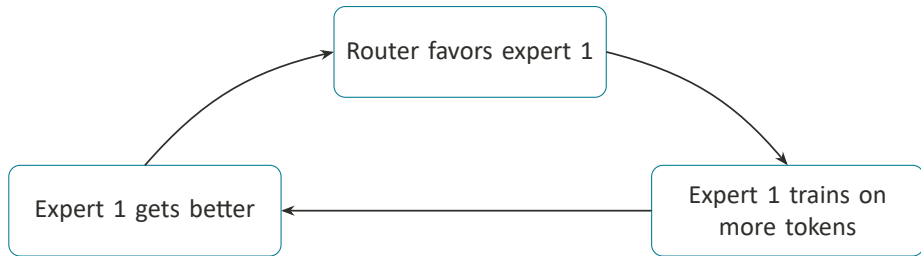
Switch (2022); DeepSeekMoE (2024); router biases (2024)

What happens if the router favors a few experts?



# Balancing Experts

## Why imbalance happens



Shazeer et al.: the imbalance is “self-reinforcing.”

How can we balance the experts?

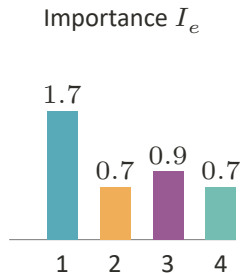
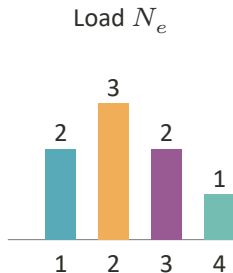
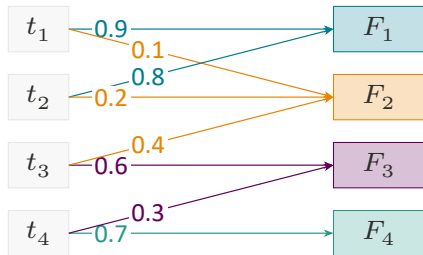


# Balancing Experts

## Two measures of expert use

First, measure how each expert is used.

$T = 4$  tokens in this batch



**Load**  $N_e$ : number of tokens sent to expert  $e$ .

**Importance**  $I_e := \sum_h w_e(h)$ : total weight on expert  $e$ .



# Balancing Experts

What perfect balance means

Load  $N_e$  controls how long expert  $e$  runs.

Importance  $I_e$  controls how much expert  $e$  learns.

Each token picks  $k$  experts, with weights summing to 1, so always

$$\sum_e N_e = kT \quad \text{and} \quad \sum_e I_e = T.$$

**Perfect balance:**  $N_e = kT/E$  and  $I_e = T/E$  for every expert  $e$ .

Should we balance the load  $N_e$  or the importance  $I_e$ ?

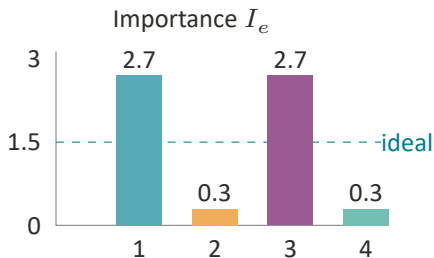
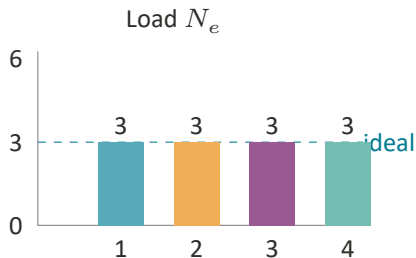


# Balancing Experts

Equal loads, unequal importance

Tokens 1–3 pick experts 1, 2 with weights 0.9, 0.1.

Tokens 4–6 pick experts 3, 4 with weights 0.9, 0.1.



Experts 2 and 4's outputs get  $9\times$  smaller gradients than experts 1 and 3's.

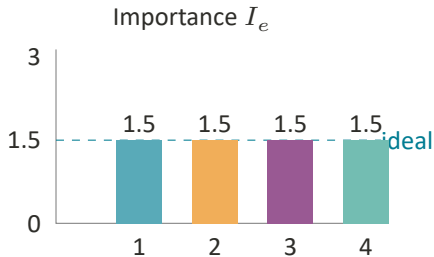
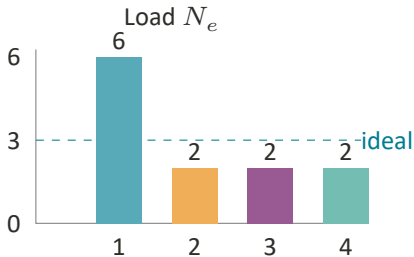


# Balancing Experts

Equal importance, unequal loads

Each token picks expert 1 (0.25) and one other (0.75):

tokens 1–2 add expert 2, tokens 3–4 expert 3, tokens 5–6 expert 4.



Expert 1 does  $3\times$  the work of each other expert.



# Balancing Experts

Shazeer et al.'s solution: penalize both imbalances

Squared coefficient of variation, for  $v = (v_1, \dots, v_E)$ :

$$\text{CV}^2(v) := \frac{\text{Var}(v)}{\text{mean}(v)^2}, \text{ zero exactly when all } v_e \text{ are equal.}$$

**Claim.** For  $\lambda_I, \lambda_N > 0$ :  $\lambda_I \text{CV}^2(I) + \lambda_N \text{CV}^2(N) = 0$   
if and only if  $N_e = kT/E$  and  $I_e = T/E$  for every expert  $e$ .

**Proof.** Both terms are nonnegative, so each is zero: all  $N_e$  are equal and all  $I_e$  are equal.



# Balancing Experts

Load has no gradient

A natural choice is to add  $\lambda_I \text{CV}^2(I) + \lambda_N \text{CV}^2(N)$  to the objective.

$I_e$  is differentiable: with the picks fixed,  
it is smooth in the scores.

$N_e$  is not differentiable: it changes  
only when the picks change.

Load is not differentiable. How can we train to balance it?

Idea: add noise, then differentiate the probability of being picked.



# Balancing Experts

## Noisy top- $k$ routing

Add learned noise before picking the top  $k$ :

$$\tilde{s}_e := s_e + \sigma_e \varepsilon_e, \quad \varepsilon_e \sim \mathcal{N}(0, 1) \text{ independently.}$$

$$\sigma_e := \text{softplus}((W_{\text{noise}} h)_e).$$



$$\text{softplus}(z) = \log(1 + e^z) > 0$$

Pick the top  $k$  of  $\tilde{s}$ ; softmax their noisy scores.



# Balancing Experts

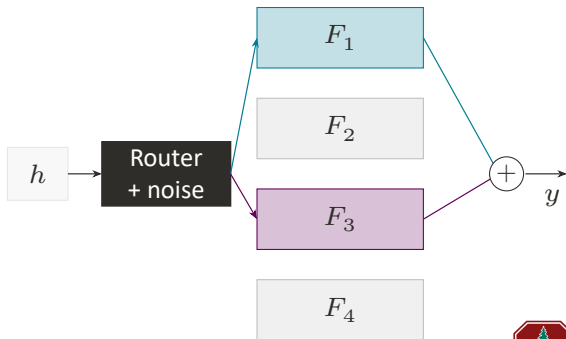
## Noisy top- $k$ routing: an example

The router's scores  $s$  become noisy scores  $\tilde{s}$ :

$$s = (2.1, 1.0, 0.3, -0.5),$$

$$\tilde{s} = (2.2, 0.5, 0.9, -0.3).$$

Pick experts 1 and 3.



Noise lets the router try experts it would skip.



# Balancing Experts

A smooth estimate of load

Fix the other experts' noisy scores; let  $\kappa_e$  be their  $k$ -th largest.

Expert  $e$  is picked if  $s_e + \sigma_e \varepsilon_e > \kappa_e$ .

standard normal CDF

$$q_e(h) := \Pr[s_e + \sigma_e \varepsilon_e > \kappa_e] = \Phi\left(\frac{s_e - \kappa_e}{\sigma_e}\right).$$

Estimated load  $\hat{N}_e := \sum_h q_e(h)$  is smooth in  $s_e$ .



# Balancing Experts

Shazeer et al.'s training loss

$$\ell := \ell_{\text{task}} + \lambda_I \text{CV}^2(I) + \lambda_N \text{CV}^2(\hat{N}).$$

Each MoE layer adds its own two penalties.

The penalties balance experts on average, not in every batch.



# Balancing Experts

Do the penalties help?

| $\lambda_I$ | $\lambda_N$ | Perplexity | Max / mean load |
|-------------|-------------|------------|-----------------|
| 0           | 0           | 39.8       | 17.8            |
| 0.2         | 0           | 35.6       | 1.5             |
| 0           | 0.2         | 35.7       | 1.2             |
| 0.1         | 0.1         | 35.6       | 1.1             |

Without penalties, the peak load is  $18\times$  the mean.

Source: [Shazeer et al. \(2017\)](#), Appendix A, Table 6.

MS&E 319 Lecture 4



# Lecture Outline

**1 Routing:** picks are discrete. How does the router learn?

Shazeer et al. (2017)



**2 Balance:** tokens can crowd onto a few experts.

Shazeer et al. (2017)



**3 Many devices:** store every expert; move tokens to them.  
data and expert parallelism; GShard (2020)

**4 Later designs:** new routing, experts, and balancing.

Switch (2022); DeepSeekMoE (2024); router biases (2024)

How do we train one model on many devices?



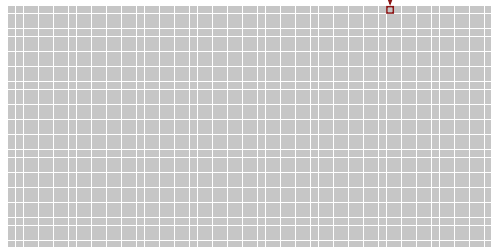
# Multi-Device Training

Why many devices

Device: a GPU or TPU core with its own memory.

**Too much data**

1 GPU: 318 years

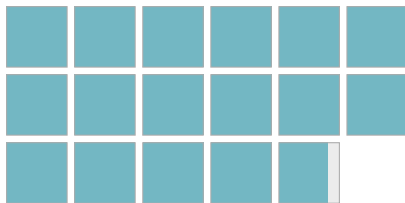


2,048 GPUs: under 2 months

DeepSeek-V3: 2.788M H800 GPU-hours

**Too many weights**

80 GB / H800



671B parameters  $\times$  2 bytes = 1.34 TB

the weights alone fill 17 GPUs

So we split both the data and the weights across devices.



# Multi-Device Training

A simpler case: smaller experts

For now, suppose all model weights fit on one GPU.

## Batch activations



Too large for one GPU

## Model weights



Fit on one GPU

Each square represents one GPU's memory.

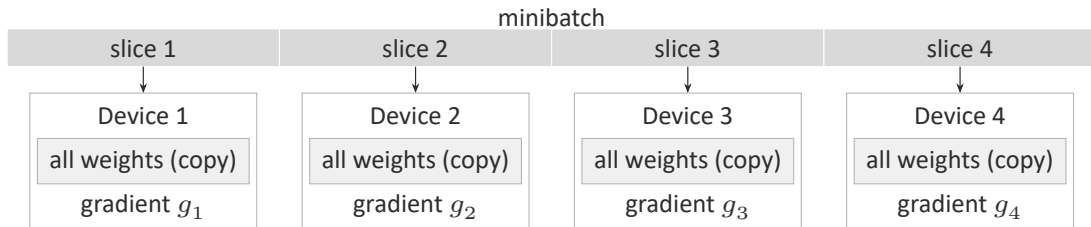
Split the batch across devices.



# Multi-Device Training

Data parallelism: split the batch

Suppose the weights fit on one device.



Each device computes the gradient  $g_i$  of the loss on its own slice.

All devices must take the same step.

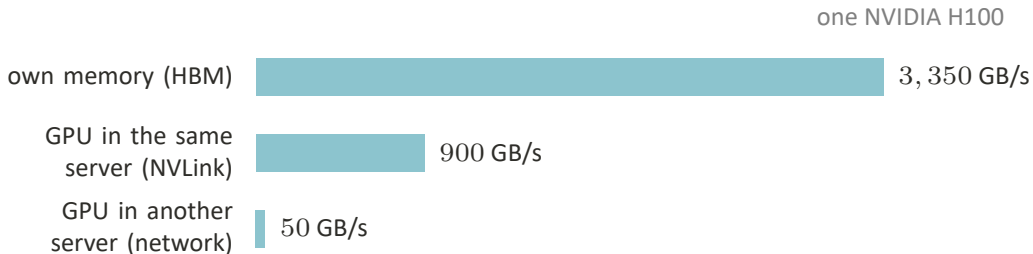
How do they combine  $g_1, \dots, g_4$ ?



# Multi-Device Training

Communication is slow

Moving data between devices is much slower than working on one.



One byte crosses the network  $\approx 20,000$  FLOPs on one H100.

Idea: compute locally and exchange rarely, in large blocks.



# Multi-Device Training

## One all-reduce per step

Each device computes its whole gradient first. Then, exchange once:

**Communication: all-reduce.** Each device  $i$  starts with a vector  $g_i$ . Every device ends with the sum  $\sum_i g_i$ .



Every device takes the *same* step.



# Multi-Device Training

## Back to the large experts

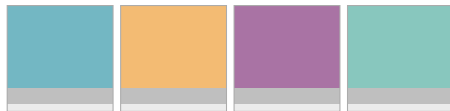
Data parallelism still copies all weights onto every GPU.

### Batch activations



Too large for one GPU

### Model weights



Too large for one GPU

Each square represents one GPU's memory.

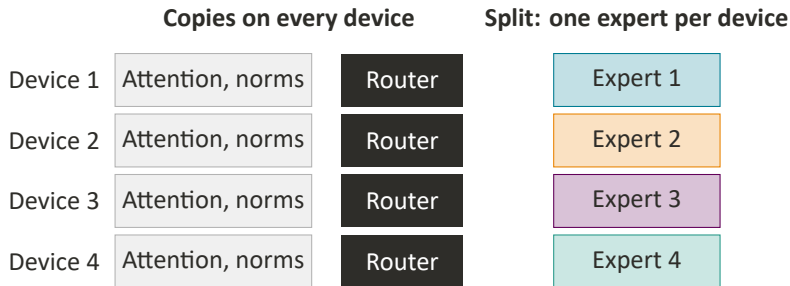
We also need to spread the experts across devices.



# Experts on Many Devices

Where the weights live

The experts are too large for one device. How do we split them?



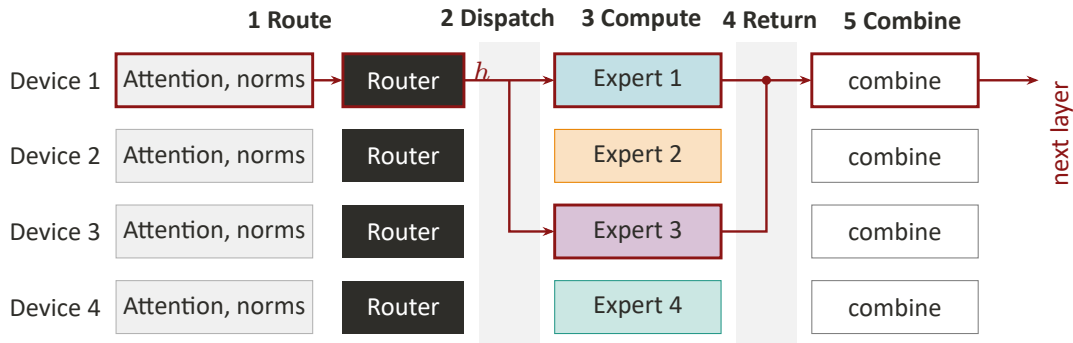
Each device also keeps its own slice of the batch.

Each expert gets tokens from every device: a larger batch.



# Experts on Many Devices

One token's path through an MoE layer



Example: a token on device 1 picks experts 1 and 3.

Two rounds of communication per layer: steps 2 and 4.

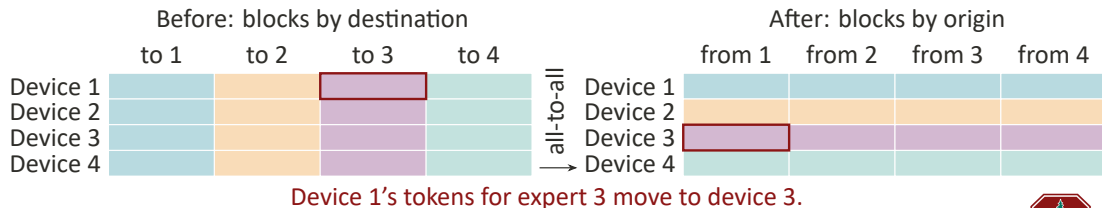


# Experts on Many Devices

How is dispatch implemented?

Every device dispatches all its tokens at the same time.

**Communication: all-to-all.** Each device cuts its data into  $D$  blocks and sends block  $j$  to device  $j$ . Every device ends with one block from each device. Nothing is summed.

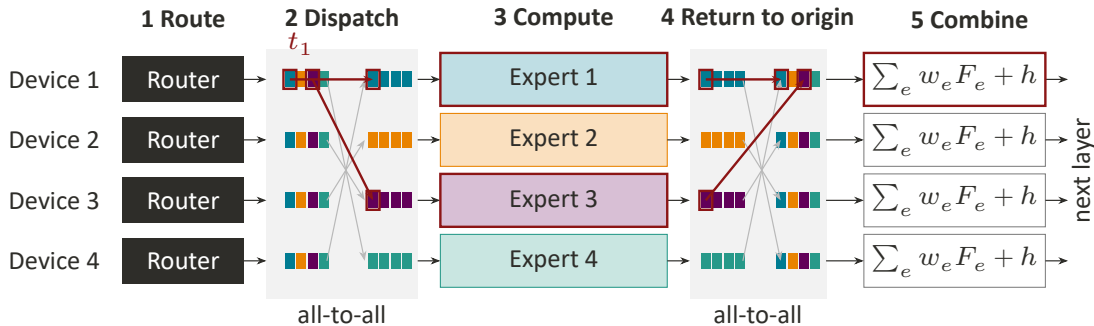


Steps 2 and 4 are all-to-alls.



# Experts on Many Devices

One MoE layer, all tokens



each square: one token sent to one expert, in that expert's color  
(top-2: two squares per token)

Devices run in lockstep: each all-to-all waits for every device.



# Lecture Outline

**1 Routing:** picks are discrete. How does the router learn?

Shazeer et al. (2017)



**2 Balance:** tokens can crowd onto a few experts.

Shazeer et al. (2017)



**3 Many devices:** store every expert; move tokens to them.

data and expert parallelism; **GShard (2020)**

**4 Later designs:** new routing, experts, and balancing.

Switch (2022); DeepSeekMoE (2024); router biases (2024)

GShard scales this arrangement to 600B parameters.



# GShard

## Four changes

Lepikhin et al. (2020)

---

| Change                  | What it does                          |
|-------------------------|---------------------------------------|
| 1. Expert capacity      | Caps each expert's work               |
| 2. Local groups         | Routes groups of tokens independently |
| 3. Random second choice | Sends fewer second choices            |
| 4. A balancing loss     | Balances experts without score noise  |

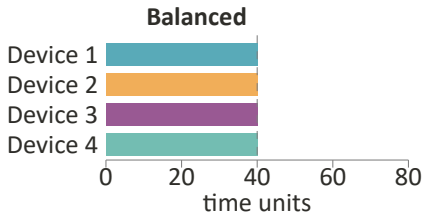
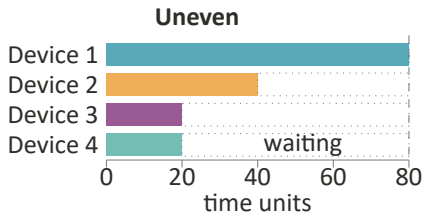
---



# GShard

## Problem 1: the busiest expert sets the pace

Recall: devices run in lockstep, so step 3 ends with the busiest expert.



Shazeer et al.'s penalties are soft:

- they need time to work, and balance only on average;
- any single batch can still overload one expert.

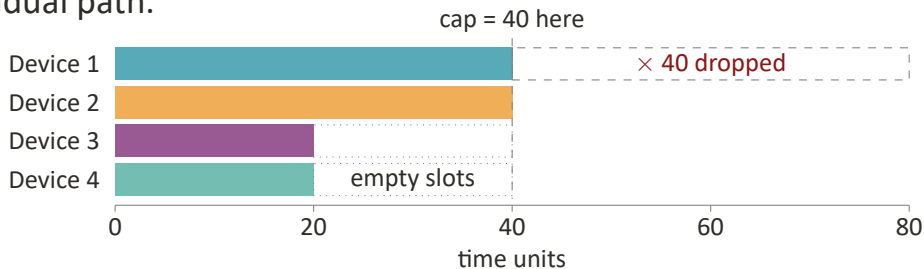


# GShard

## 1. Expert capacity

Each expert takes at most **cap** tokens; the rest are dropped.

A dropped assignment skips that expert; the token still takes the residual path.

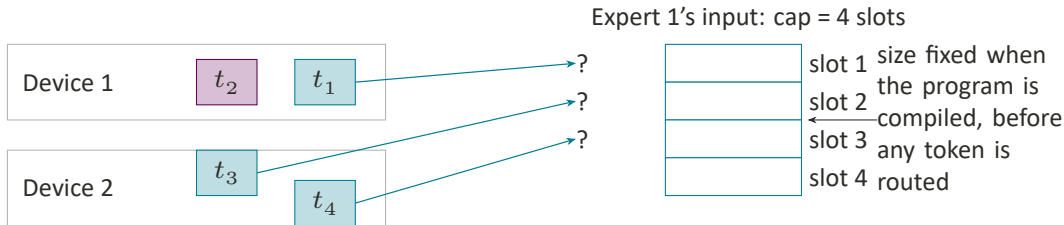


**Cost: dropped assignments get no expert.**



# GShard

## Problem 2: which slot does a token take?



Expert 1 runs its MLP on this array as one matrix: one row (slot) per token.

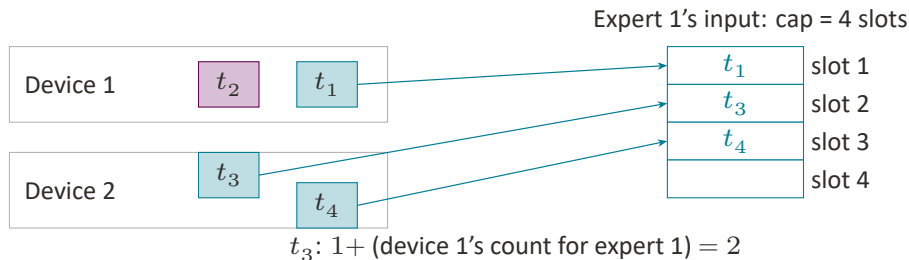
Device 1 does not know how many tokens device 2 sends to expert 1, and vice versa.

Agreeing on slots takes communication,  
and communication is slow.



# GShard

## Problem 2: a slot depends on other devices



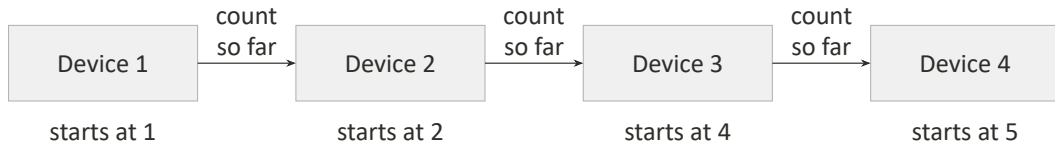
Rule: slot =  $1 +$  the number of earlier tokens sent to the same expert, counting device 1's tokens before device 2's.

Device 2 cannot place  $t_3$  until it knows device 1's count.



# GShard

## Problem 2: counting makes devices wait



Each device needs the counts of every device before it.

So devices exchange counts and wait, before every dispatch, in every MoE layer.

GShard: about a million tokens and 2,048 devices; global counting would leave devices idle.



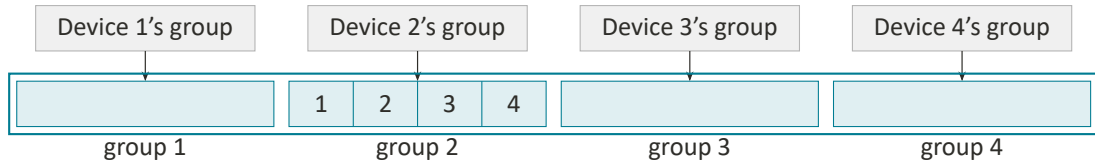
# GShard

## 2. Local groups: count slots within a group

Split the batch into groups, for example one per device.

Each group gets its own share of every expert's buffer:

$$\text{cap} := \frac{2T_g}{E} \text{ slots per expert, per group of } T_g \text{ tokens.}$$



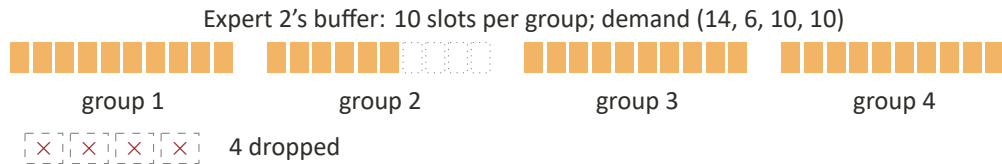
Expert 1's input, on device 1, split by source group



# GShard

## 2. Local groups: the cost

Example:  $E = 4$  experts and  $T_g = 20$  tokens per group, so cap = 10.



Group 1 drops 4 tokens while group 2 leaves 4 slots empty.

Cost: a group cannot borrow another group's empty slots.



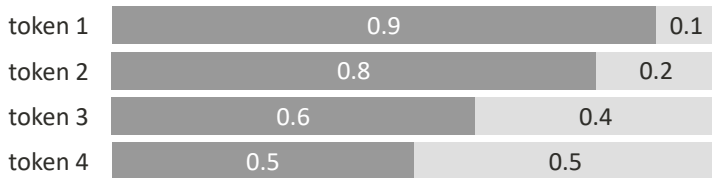
# GShard

## 3. Random second choice: why

Top-2 sends every token to two experts: twice the assignments, filling slots twice as fast.

Often the second expert gets little weight:

$$y = \underbrace{w_a}_{w_a} F_a(h) + \underbrace{w_b}_{w_b} F_b(h), \quad \text{with } w_b \text{ small.}$$



Can we skip second choices that add little?



# GShard

## 3. Random second choice: the rule

Send the token to its first choice  $a$ , if  $a$  has a free slot.

Send it to  $b$  with probability  $2w_b$ . This is at most 1, since  $w_b \leq 1/2$ .

|         | $w_a$ | $w_b$ | send to $b$ |
|---------|-------|-------|-------------|
| token 1 | 0.9   | 0.1   | 20%         |
| token 2 | 0.8   | 0.2   | 40%         |
| token 3 | 0.6   | 0.4   | 80%         |
| token 4 | 0.5   | 0.5   | 100%        |

These 4 tokens need 8 slots under top-2; 6.4 on average here.

If  $b$  is skipped,  $y = w_a F_a(h)$ : the weights are not renormalized.

Low-weight second choices are usually skipped, saving slots.



# GShard

## 4. A balancing loss: what it measures

GShard measures expert use like Shazeer et al., with two changes:

---

| Shazeer et al. (Slide 16) | GShard |
|---------------------------|--------|
|---------------------------|--------|

---

|                                                  |                                                                                           |                      |
|--------------------------------------------------|-------------------------------------------------------------------------------------------|----------------------|
| Load $N_e$ : all picks                           | $f_e = N_e^{(1)} / T_g$ : fraction of the group's tokens whose <b>first</b> choice is $e$ | a count: no gradient |
| Importance $I_e$ : weights of the picked experts | $\bar{p}_e$ : average of $p_e(h)$ , a softmax over <b>all</b> $E$ experts                 | smooth               |

---

$$\ell_{\text{aux}} := \frac{1}{E} \sum_{e=1}^E f_e \bar{p}_e.$$



# GShard

Why this form?

But  $f_e$  has no gradient. Replace one factor with a smooth one:

$$\frac{1}{E} \sum_e f_e f_e \longrightarrow \frac{1}{E} \sum_e \underset{\substack{\uparrow \\ \text{held fixed}}}{f_e} \underset{\substack{\uparrow \\ \text{smooth: gradients flow here}}}{\bar{p}_e}$$

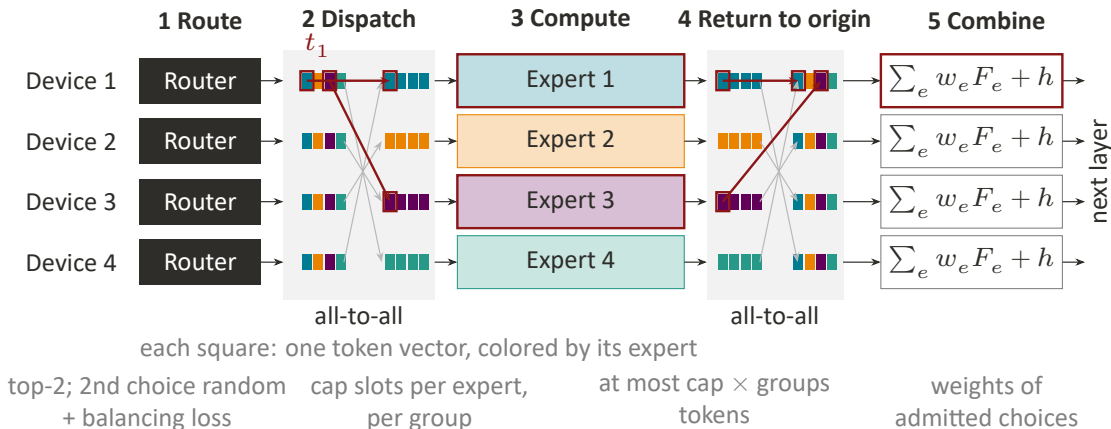
Shazeer et al. used noise to make load smooth; GShard uses full-softmax probabilities.

The gradient pushes probability toward less-loaded experts, even experts a token currently skips.



# GShard

## The complete MoE layer



# GShard

## Results

Translating to English, with 2,048 experts per MoE layer:

|                    | Parameters | BLEU gain | TPU v3 core-years |
|--------------------|------------|-----------|-------------------|
| MoE, 2,048 experts | 600B       | +13.5     | 22                |
| Dense, 96 layers   | 2.3B       | +6.1      | 235               |

BLEU: a translation-quality score; gains are over bilingual baselines.

Core-years: number of cores  $\times$  years of training.

Higher translation quality with about  $10\times$  less training compute.

Source: [Lepikhin et al. \(2020\)](#), Table 3.



# Lecture Outline

**1 Routing:** picks are discrete. How does the router learn?

Shazeer et al. (2017)



**2 Balance:** tokens can crowd onto a few experts.

Shazeer et al. (2017)



**3 Many devices:** store every expert; move tokens to them.

data and expert parallelism; GShard (2020)



**4 Later designs:** new routing, experts, and balancing.

Switch (2022); DeepSeekMoE (2024); router biases (2024)

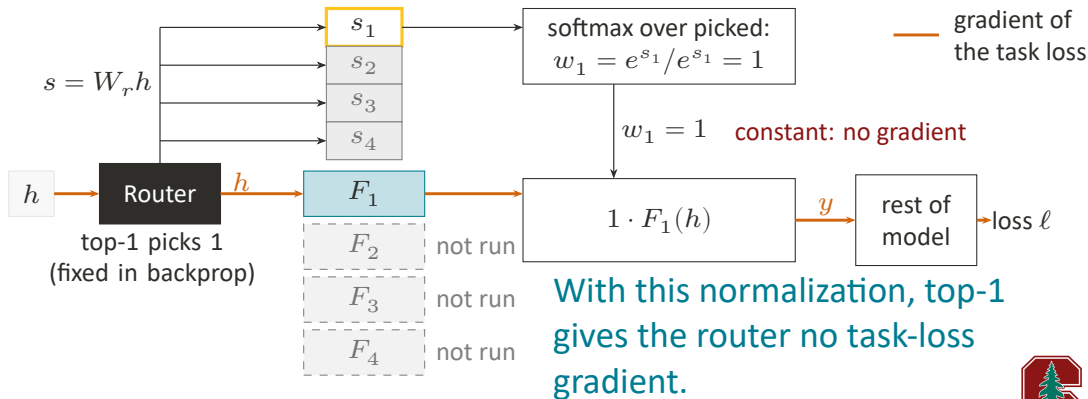
GShard sends each token to two experts. Is one enough?



# Switch

Can one expert be enough? Fedus, Zoph, and Shazeer (2022)

One expert per token  $\approx$  half the assignments, less communication.

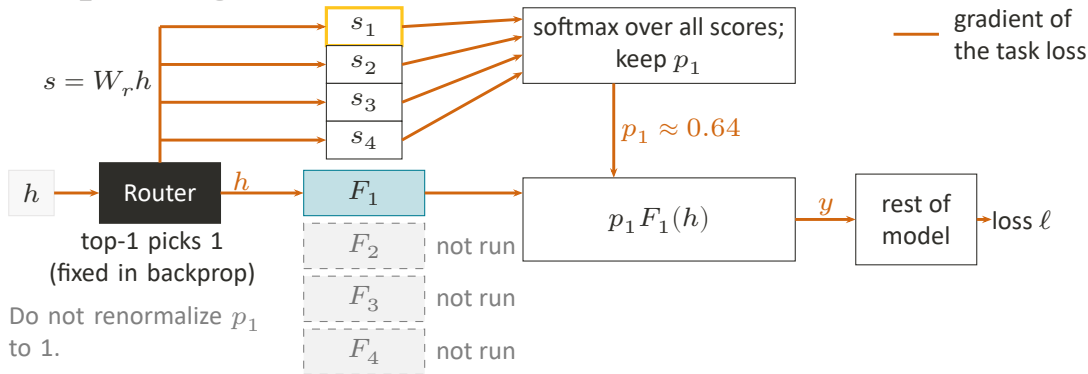


The balancing loss still trains the router to spread tokens.  
MS&E 319 Lecture 4



# Switch

# Keep a weight that can learn



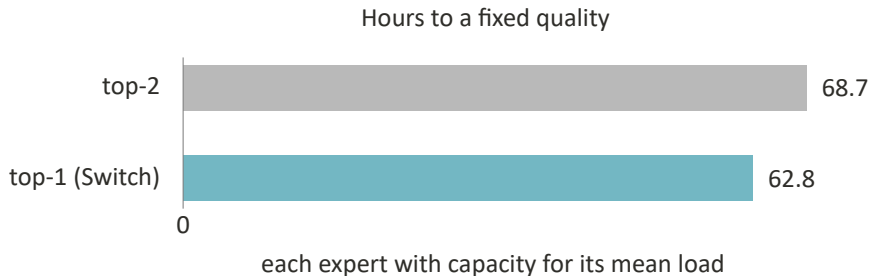
Even with one expert, the router still receives gradient.



# Switch

Does one expert work?

Stable: router in float32, the rest in bfloat16.



Switch reaches the target quality 9% faster in this comparison.



# Lecture Outline

**1 Routing:** picks are discrete. How does the router learn?

Shazeer et al. (2017)



**2 Balance:** tokens can crowd onto a few experts.

Shazeer et al. (2017)



**3 Many devices:** store every expert; move tokens to them.

data and expert parallelism; GShard (2020)



**4 Later designs:** new routing, experts, and balancing.

Switch (2022); DeepSeekMoE (2024); router biases (2024)

How can we make each active parameter more useful?



# DeepSeekMoE

Dai et al. (2024)

## Two problems with standard experts

**Bundled skills.** A large expert learns many unrelated skills. Picking it activates all of them.

**Repeated knowledge.** Most tokens need some common knowledge, so many experts each learn their own copy.

Can we fix both at the same compute per token?



# DeepSeekMoE

## Change 1: shared experts

Some processing helps almost every token (e.g., basic grammar).

Without sharing, many experts learn the same transformation.

A **shared expert**  $F_0$  is an MLP that every token uses:

$$y := F_0(h) + \sum_{e \in S(h)} w_e(h) F_e(h).$$

↑  
one fewer routed expert, to keep compute fixed

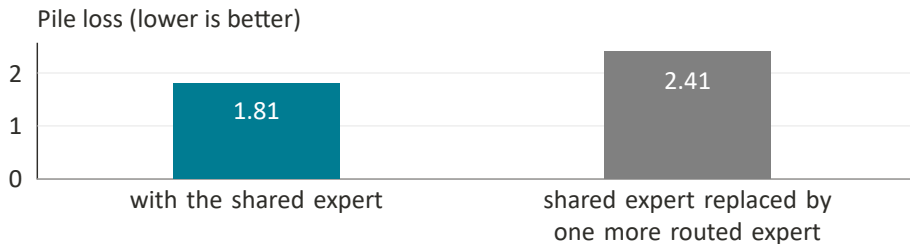
Common knowledge lives in  $F_0$ , so routed experts can specialize.



# DeepSeekMoE

## Does the shared expert matter?

In the trained model, disable the shared expert and pick one more routed expert instead: same compute.



Pile loss: cross-entropy on held-out text from the Pile.

The loss rises by about a third: routed experts cannot replace the shared expert.



# DeepSeekMoE

Change 2: smaller experts, more of them

Split each expert into  $m$  smaller ones, and pick  $m$  times as many.

Same total parameters, same compute per token.

16 experts, pick 2:  $\binom{16}{2} = 120$  combinations

64 experts, pick 8:  $\binom{64}{8} \approx 4.4 \times 10^9$  combinations

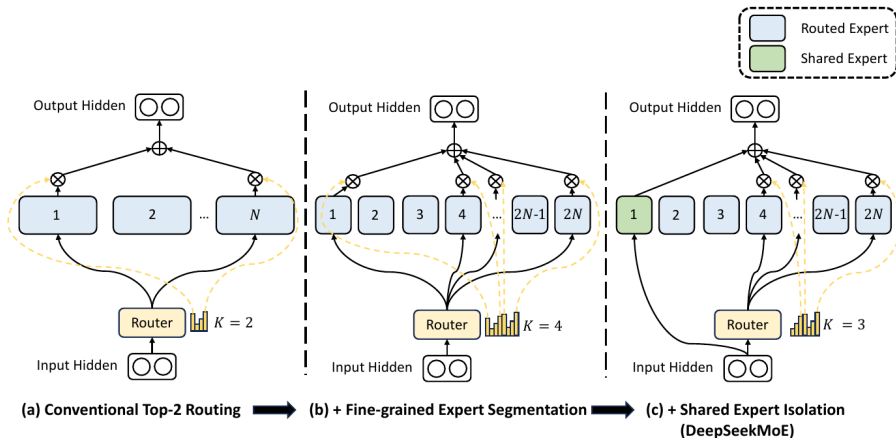
Finer pieces let the router assemble the skills a token needs.

DeepSeek-V3: 1 shared expert and 8 of 256 routed experts.



# DeepSeekMoE

Both ideas together



Source: Dai et al. (2024), Fig. 2.

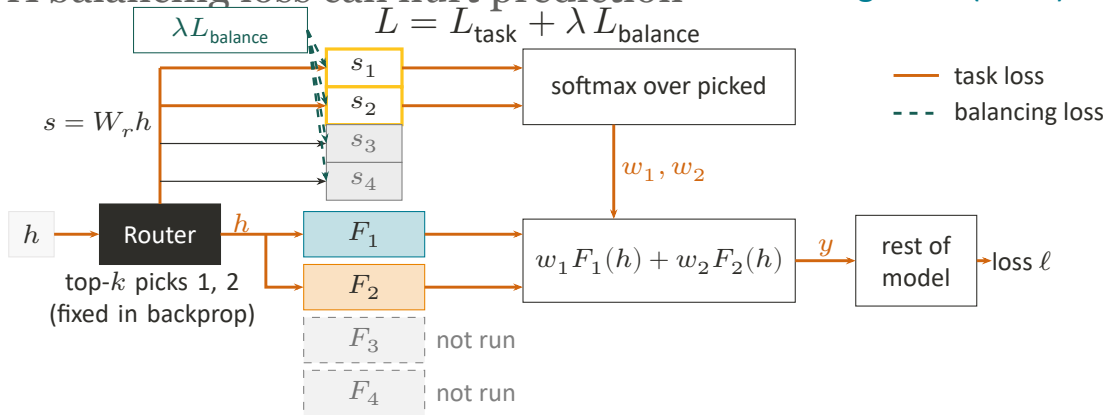
MS&E 319 Lecture 4



# Balancing with Router Biases

A balancing loss can hurt prediction

Wang et al. (2024)



Small  $\lambda$ : experts become overloaded.

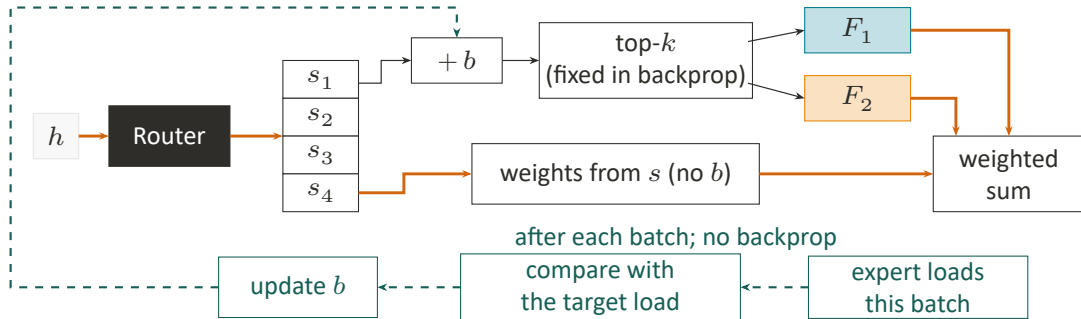
Large  $\lambda$ : balance improves, but prediction gets worse.

The router gets two objectives whose gradients can conflict.



# Balancing with Router Biases

Balance with biases, not a loss



$b_e \leftarrow b_e - \gamma$  if overloaded;  $b_e \leftarrow b_e + \gamma$  if underloaded.

Think of  $-b_e$  as a price on busy experts.

Gradient descent learns useful scores; load feedback adjusts selection.



# Summary

MoE: old idea (Jacobs et al., 1991), now in most frontier open models.

Core idea: store many parameters, use a few per token.

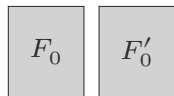
Making it work at LLM scale:

- **Routing:** noisy top- $k$ , full-softmax weights
- **Balance:** penalties, capacity, biases
- **Many devices:** split data and experts, all-to-all
- **Better experts:** shared and fine-grained

2 shared (DeepSeekMoE)

16 of 896: top- $k$  (Shazeer et al.)

Kimi K3



...



896 routed

experts spread over devices (Shazeer et al.; GShard)

2.8T stored, 104B used per token (3.7%); 92 of 93 layers are MoE.



# Questions?

Next: Lecture 5, Efficient Attention.

