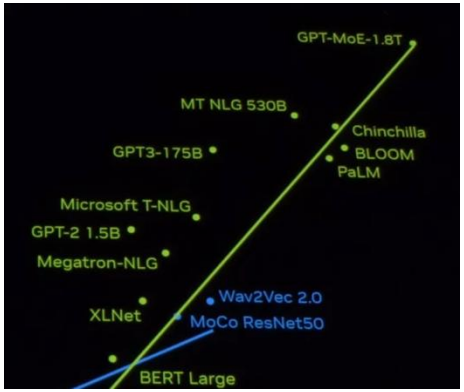
Lecture 4

Mixtures of Experts

CS336

Tatsu H

Mixture of experts



GPT4 (?)

Mistral AI
@MistralAI

magnet:?
xt=urn:btih:9238b09245d0d8cd915be09927769d5f7584c1c9&dn=mixtral-
8x22b&tr=udp%3A%2F%2Fopen.demonii.com%3A1337%2Fannounce&tr=http%3A%2F%2Ftracker.opentrackr.org%3A1337%2Fannounce

Grok

deepseek

DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models

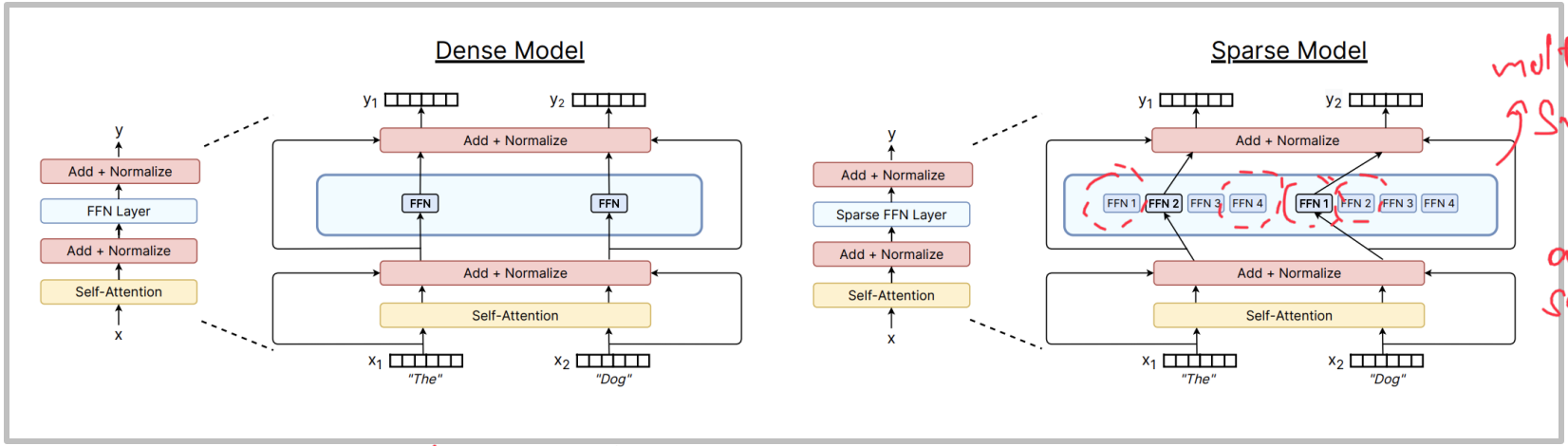
deepseek

DeepSeek-V3 Technical Report

OLMoE: Open Mixture-of-Experts Language Models

Niklas Muennighoff[ca] Luca Soldaini[a] Dirk Groeneveld[a] Kyle Lo[a] Jacob Morrison[a]
Sewon Min[a] Weijia Shi[w] Pete Walsh[a] Oyvind Tafjord[a] Nathan Lambert[a]
Yuling Gu[a] Shane Arora[a] Akshita Bhagia[a] Dustin Schwenk[a] David Wadden[a]
Alexander Wettig[ap] Binyuan Hui Tim Dettmers[a] Douwe Kiela[c] Ali Farhadi[aw]
Noah A. Smith[aw] Pang Wei Koh[aw] Amanpreet Singh[c] Hannaneh Hajishirzi[aw]
[a]Allen Institute for AI [c]Contextual AI [w]University of Washington [p]Princeton University
n.muennighoff@gmail.com hannah@allenai.org

What's a MoE?



multiple smaller FFNs & a selector layer.

sparsely selecting some model components. [Fedus et al 2022]

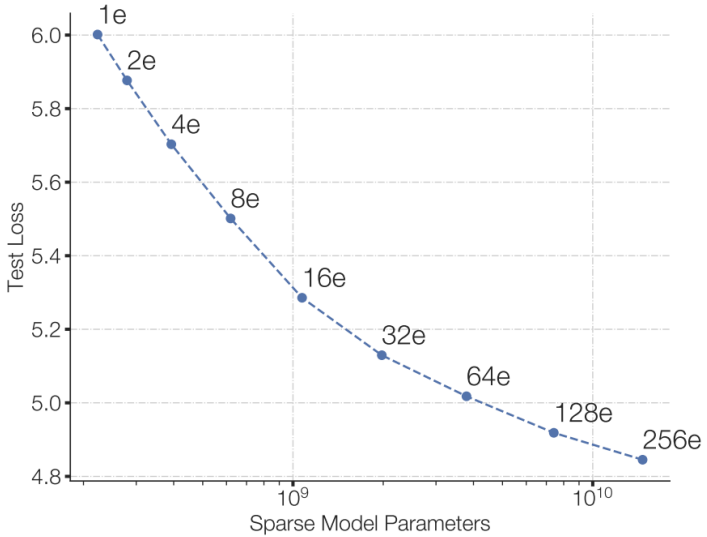
Replace big feedforward with (many) big feedforward networks and a selector layer

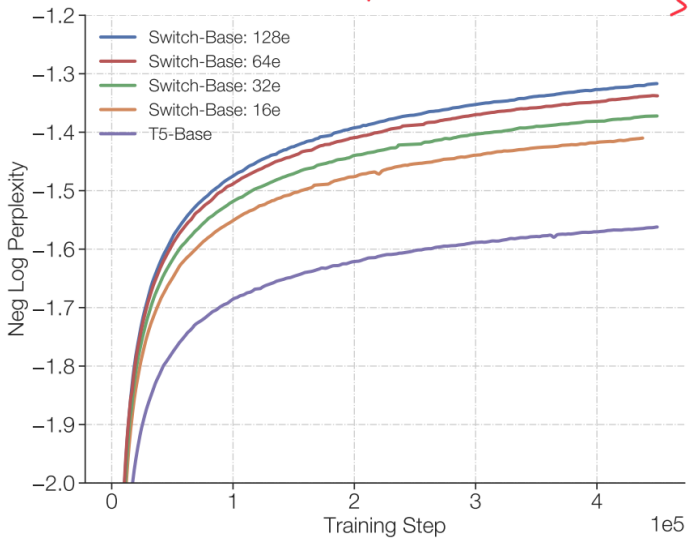
You can increase the # experts without affecting FLOPs

more parameters

Why are MoEs getting popular?

Same FLOP, more param does better



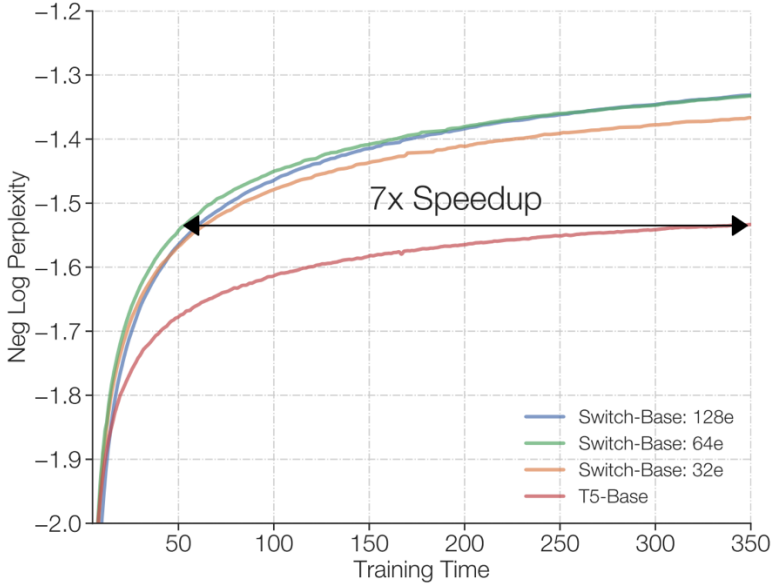


but MoEs will have more memory & systems complexity. infrastructure needs to be optimized to make it speed as load/unbal adds good wall clock!

[Fedus et al 2022]

Why are MoEs getting popular?

Faster to train MoEs



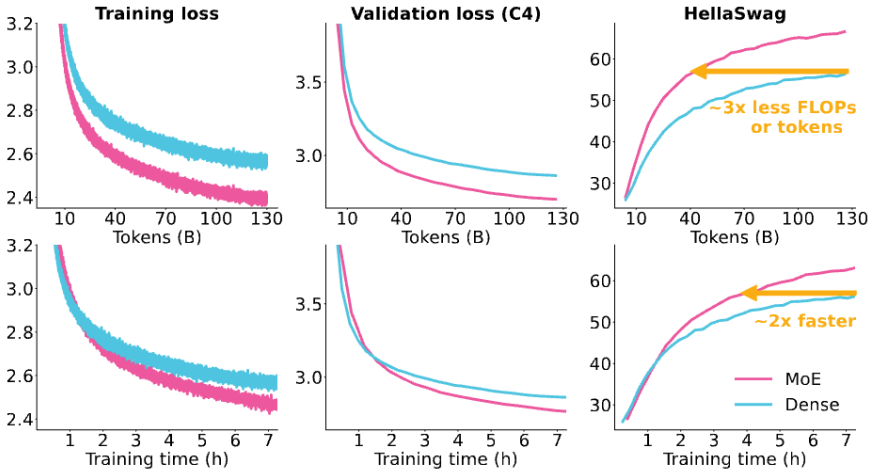
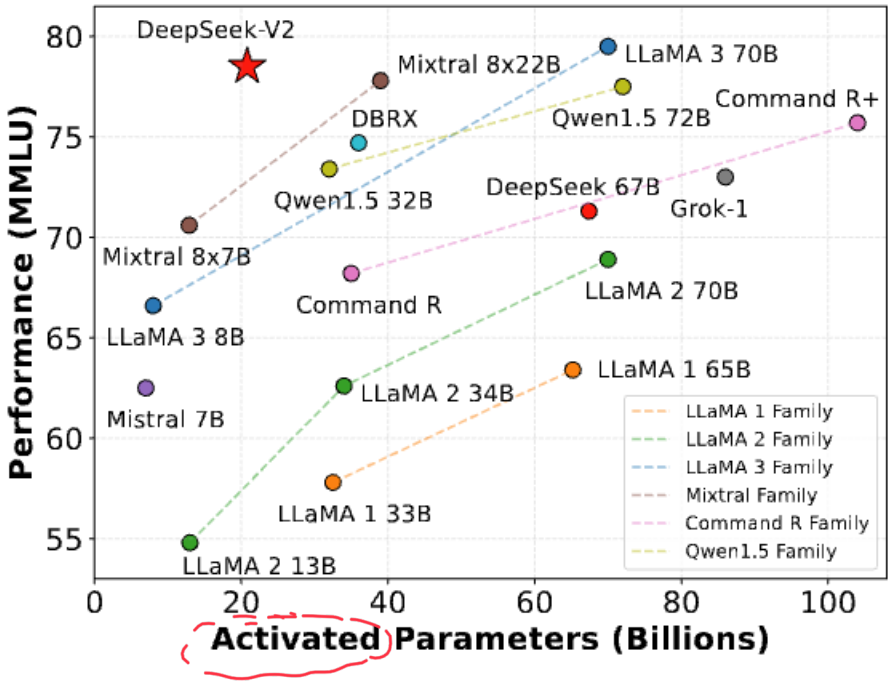


Figure 4: **MoE vs. Dense.** We train a 1.3B parameter dense model and a 1.3B active, 6.9B total parameter MoE model, each on 128 H100 GPUs. Apart from MoE-related changes, we train both

[OlMoE]

Why are MoEs getting popular?



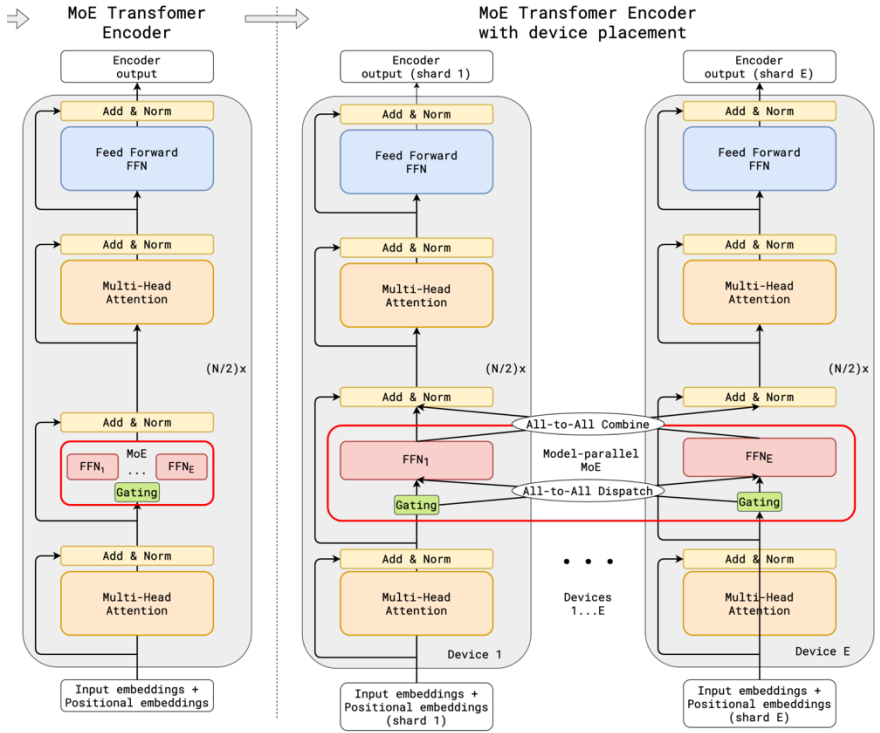
Highly competitive vs dense equivalents

Why are MoEs getting popular?

FFNs will be a natural point to shard!!

→ expert parallelism

Parallelizable to many devices



Some MoE results – from the west

Category Benchmark	Llama 4 Maverick	Gemini 2.0 Flash	DeepSeek v3.1	GPT-4o
Inference Cost Cost per 1M input & output tokens (3:1 blended)	$0.19-$0.49[5]	**$0.17**	$0.48	$4.38
Image Reasoning MMMU	**73.4**	71.7	No multimodal support	69.1
MathVista	**73.7**	73.1		63.8
Image Understanding ChartQA	**90.0**	88.3		85.7
DocVQA (test)	**94.4**	—		92.8
Coding LiveCodeBench (10/01/2024-02/01/2025)	43.4	34.5	**45.8/49.2**[3]	32.3[3]
Reasoning & Knowledge MMLU Pro	80.5	77.6	**81.2**	—
GPQA Diamond	**69.8**	60.1	68.4	53.6

MoEs are most of the highest-performance open models, and are quite quick

Earlier MoE results from Chinese groups – Qwen

Chinese LLM companies are also doing quite a bit of MoE work on the smaller end

Model	MMLU	GSM8K	HumanEval	Multilingual	MT-Bench
Mistral-7B	64.1	47.5	27.4	40.0	7.60
Gemma-7B	64.6	50.9	32.3	-	-
Qwen1.5-7B	61.0	62.5	36.0	45.2	7.60
DeepSeekMoE 16B	45.0	18.8	26.8	-	6.93
Qwen1.5-MoE-A2.7B	62.5	61.5	34.2	40.8	7.17

Model	#Parameters	#(Activated) Parameters
Mistral-7B	7.2	7.2
Qwen1.5-7B	7.7	7.7
Gemma-7B	8.5	7.8
DeepSeekMoE 16B	16.4	2.8
Qwen1.5-MoE-A2.7B	14.3	2.7

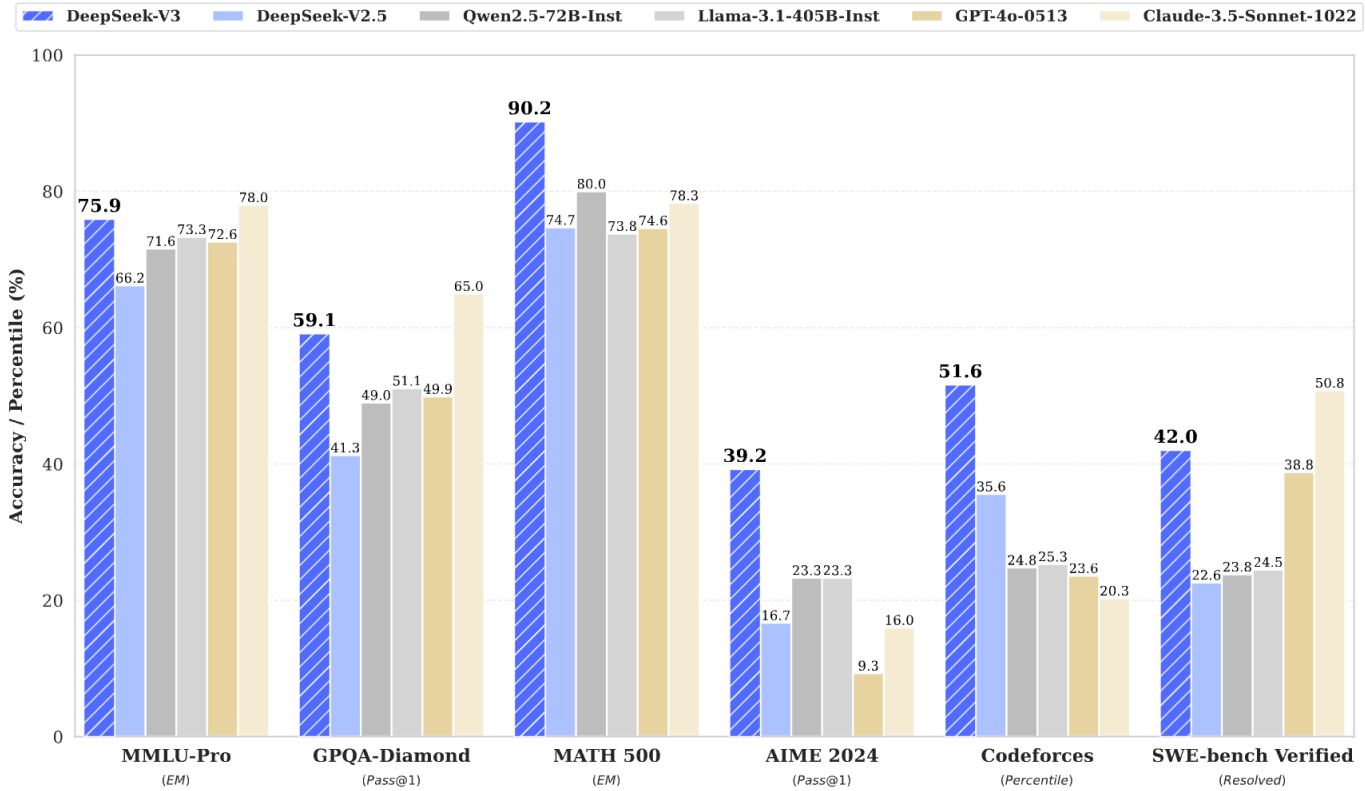
Earlier MoE results from Chinese groups - DeepSeek

There's also some good recent ablation work on MoEs showing they're generally good

dense MoE → naive routing → smart routing.

Metric	# Shot	Dense	Hash Layer	Switch
# Total Params	N/A	0.2B	2.0B	2.0B
# Activated Params	N/A	0.2B	0.2B	0.2B
FLOPs per 2K Tokens	N/A	2.9T	2.9T	2.9T
# Training Tokens	N/A	100B	100B	100B
Pile (Loss)	N/A	2.060	1.932	1.881
HellaSwag (Acc.)	0-shot	38.8	46.2	49.1
PIQA (Acc.)	0-shot	66.8	68.4	70.5
ARC-easy (Acc.)	0-shot	41.0	45.3	45.9
ARC-challenge (Acc.)	0-shot	26.0	28.2	30.2
RACE-middle (Acc.)	5-shot	38.8	38.8	43.6
RACE-high (Acc.)	5-shot	29.0	30.0	30.9
HumanEval (Pass@1)	0-shot	0.0	1.2	2.4
MBPP (Pass@1)	3-shot	0.2	0.6	0.4
TriviaQA (EM)	5-shot	4.9	6.5	8.9
NaturalQuestions (EM)	5-shot	1.4	1.4	2.5

Recent MoE results – DeepSeek v3



Why haven't MoEs been more popular?

→ if we do single node training → MoE doesn't make sense!

Infrastructure is complex / advantages on multi node

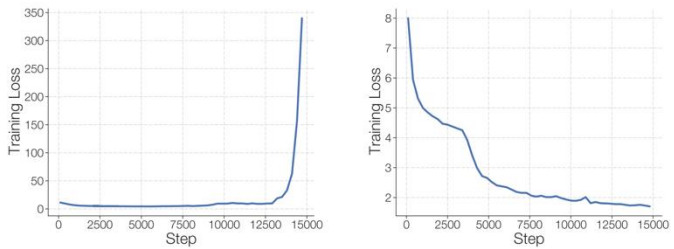
> At a high level, sparsity is good when you have many accelerators (e.g. GPU/TPU) to host all the additional parameters that comes when using sparsity. Typically models are trained using data-parallelism where different machines will get different slices of the training/inference data. The machines used for operating on the different slices of data can now be used to host many more model parameters. Therefore, sparse models are good when training with data parallelism and/or have high throughput while serving: training/serving on many machines which can host all of the parameters.

[Fedus et al 2022]

Training objectives are somewhat heuristic (and sometimes unstable)

so, lot of engineering needed

Sparse models often suffer from training instabilities (Figure 1) worse than those observed in standard densely-activated Transformers.

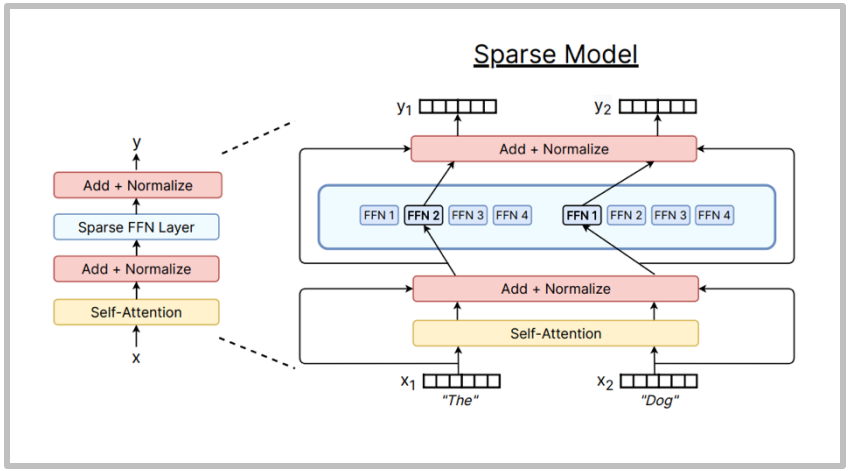


routing to which expert is a tricky difficult to optimize problem. (we need differentiable fns)

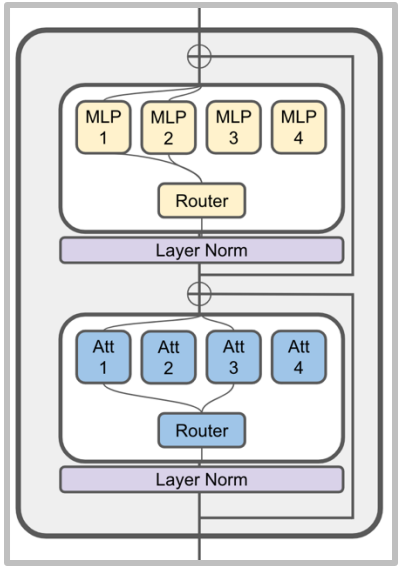
[Zoph et al 2022]

What MoEs generally look like

Typical: replace MLP with MoE layer



Less common: MoE for attention heads



[ModuleFormer, JetMoE]

MoE – what varies?

- Routing function → how to route
- Expert sizes → how many experts
  → how big each expert should be
- Training objectives → how to train router
  generally non-differentiable objective!!

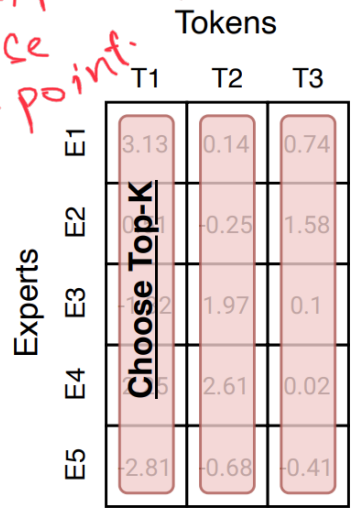
Routing function - overview

Many of the routing algorithms boil down to 'choose top k'

token will be picked by best expert

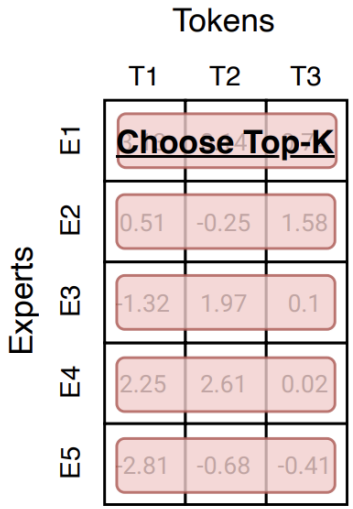
makes sense from acc. point.

experts are used well balanced - utilization from systems point.



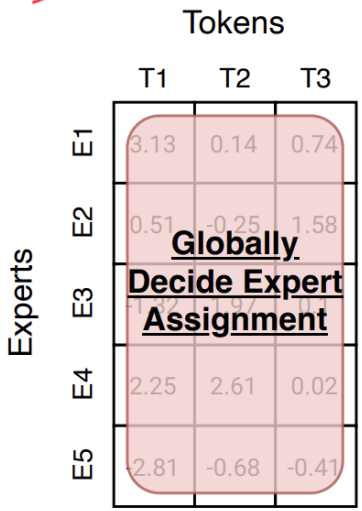
Token chooses expert

(choose top-k experts for a token)



Expert chooses token

(choose top-k tokens for an expert)



Global routing via optimization

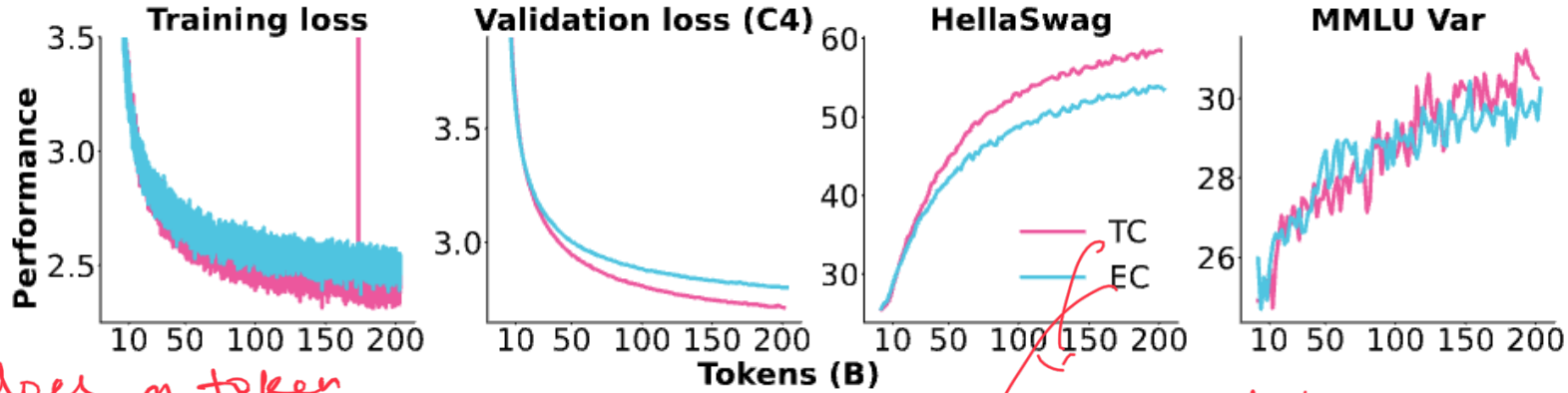
jointly optimize both.

[Fedus et al 2022]

Routing type

each token choses k experts.

Almost all the MoEs do a standard 'token choice topk' routing. Some recent ablations



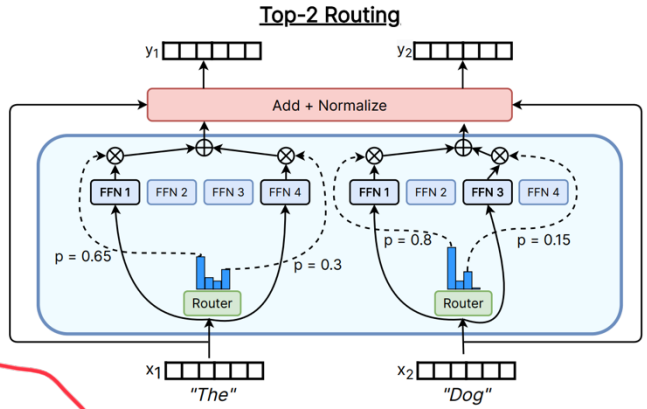
how does a token know which expert it should go?

Token choice
expert choice

token vector (x) -> some weight (w) -> softmax -> scores telling how good this token is for all experts

Common routing variants in detail

Top-k



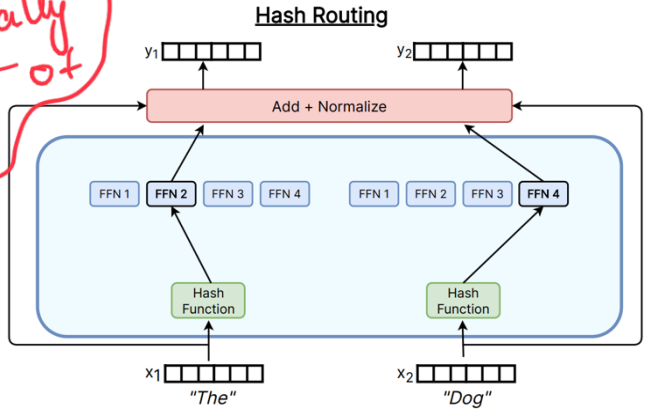
Used in *most* MoEs

Switch Transformer (k=1)
Gshard (k=2), Grok (2), Mixtral (2),
Qwen (4), DBRX (4),
DeepSeek (7)

router is basically a softmax over vector matrix (embed) multiplication & picking the best possible FFNs.

a pure random router totally independent of i/ps is worse

Hashing



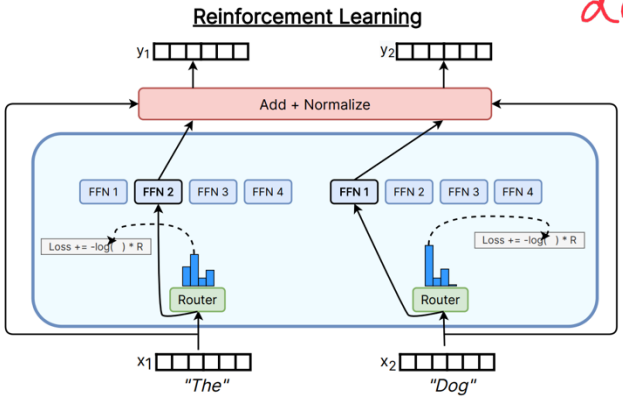
Common baseline

some experts might learn common words etc; [Fedus et al 2022]

Other routing methods

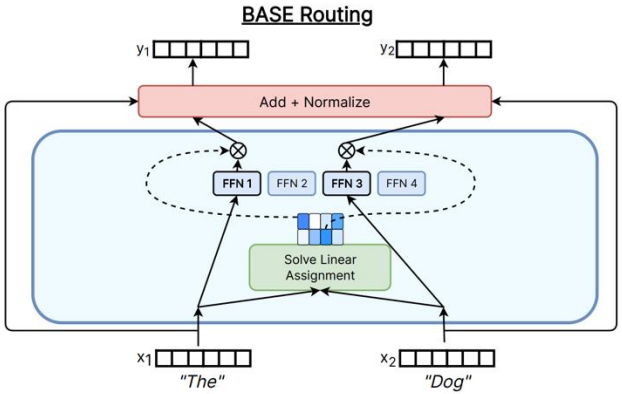
This is better to learn discrete decisions. But too much engineering for too little gain.

RL to learn routes



Used in some of the earliest work
Bengio 2013, not common now

Solve a matching problem



Linear assignment for routing
Used in various papers like Clark ‘22

[Fedus et al 2022]

Top-K routing in detail.

'e' vectors is just used to pick the experts.

Most papers do the old and classic top-k routing. How does this work?

Gating

$$\mathbf{h}_t^l = \sum_{i=1}^{N} \left(g_{i,t}\, \mathrm{FFN}_i\left(\mathbf{u}_t^l\right) \right) + \mathbf{u}_t^l,$$

$$g_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \mathrm{Topk}(\{s_{j,t} | 1 \leqslant j \leqslant N\}, K), \\ 0, & \text{otherwise}, \end{cases}$$

$$s_{i,t} = \mathrm{Softmax}_i\left(\mathbf{u}_t^{l\,T} \mathbf{e}_i^l \right),$$

Gates selected by a logistic regressor

e is separate from FFN.

this helps systems wise as every forward pass in training & inference, we pick only top-k FFNs.

parameters for routers

residual input

expert vectors. (learnable parameters, one for each expert)

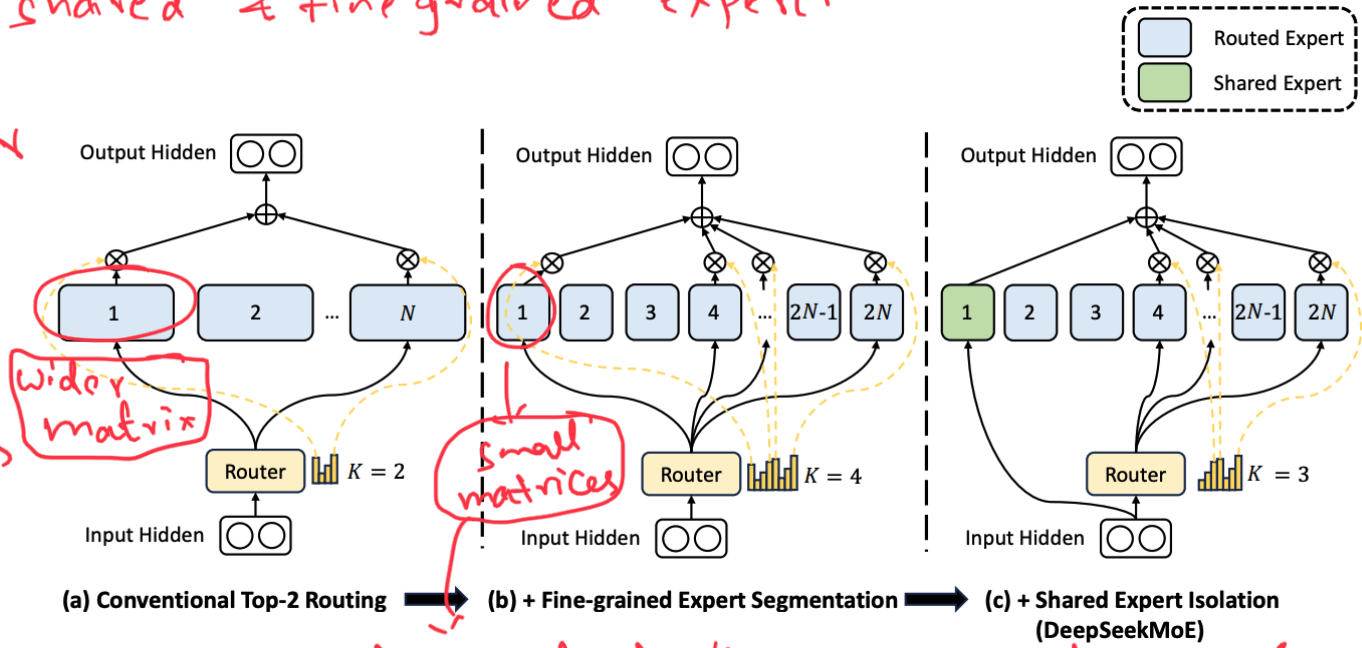
This is the DeepSeek (V1-2) router (Grok, Qwen do this too)

Mixtral, DBRX, DeepSeek v3 softmaxes after the TopK

[Dai et al 2024]

Recent variations from DeepSeek and other Chinese LMs

idea of shared & fine grained experts.



The router function 'e' is simple as systems come into play kinda tradeoff!

wider matrix

small matrices

allows to have more experts for same parameter count.

Smaller, larger number of experts + a few shared experts that are always on.

(Used in DeepSeek / Qwen, originally from DeepSpeed MoE)

Various ablations from the DeepSeek paper

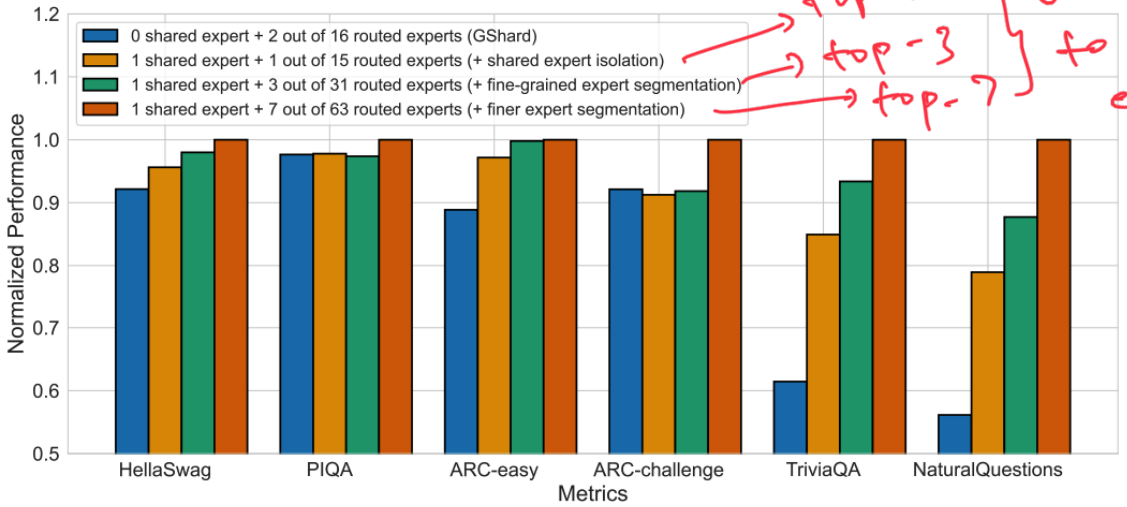
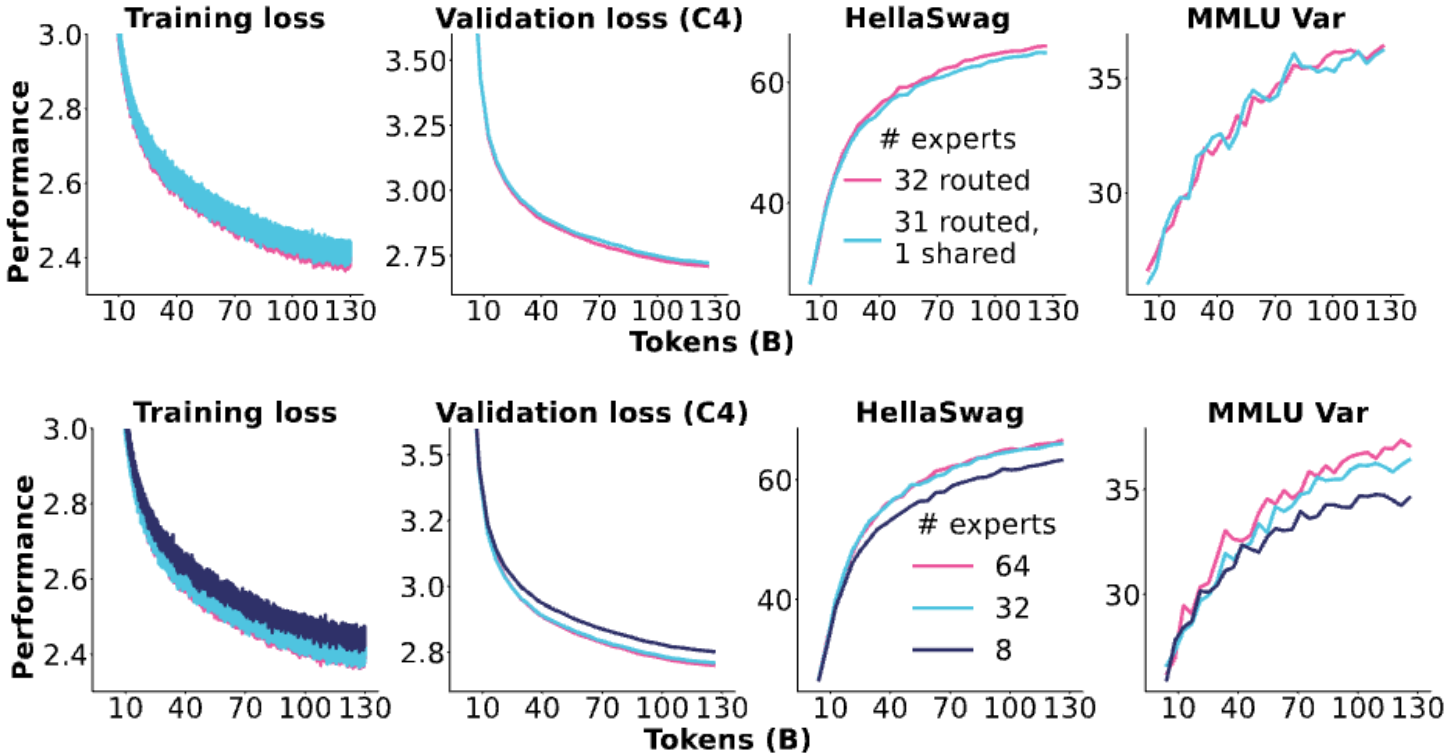


Figure 3 | Ablation studies for DeepSeekMoE. The performance is normalized by the best performance for clarity in presentation. All compared models have the same number of parameters and activated parameters. We can find that fine-grained expert segmentation and shared expert isolation both contribute to stronger overall performance.

More experts, shared experts all seem to generally help

Ablations from OlMoE

Gains from fine-grained experts, none from shared experts.



Expert routing setups for recent MoEs

ratio b/w original router & fine grained routers.

Model	Routed	Active	Shared	Fine-grained ratio
GShard	2048	2	0	
Switch Transformer	64	1	0	
ST-MOE	64	2	0	
Mixtral	8	2	0	
DBRX	16	4	0	
Grok	8	2	0	
DeepSeek v1	64	6	2	1/4
Qwen 1.5	60	4	4	1/8
DeepSeek v3	256	8	1	1/14
OlMoE	64	8	0	1/8
MiniMax	32	2	0	~1/4
Llama 4 (maverick)	128	1	1	1/2

(↑) in experts by fine graining

total expert

something like top-k.

How do we train MoEs?

Major challenge: we need sparsity for training-time efficiency…

But sparse gating decisions are not differentiable!

for training, we can't turn on all experts. so, we need train-time sparsity!

Solutions?

1. Reinforcment learning to optimize gating policies
2. Stochastic perturbations
3. Heuristic 'balancing' losses.

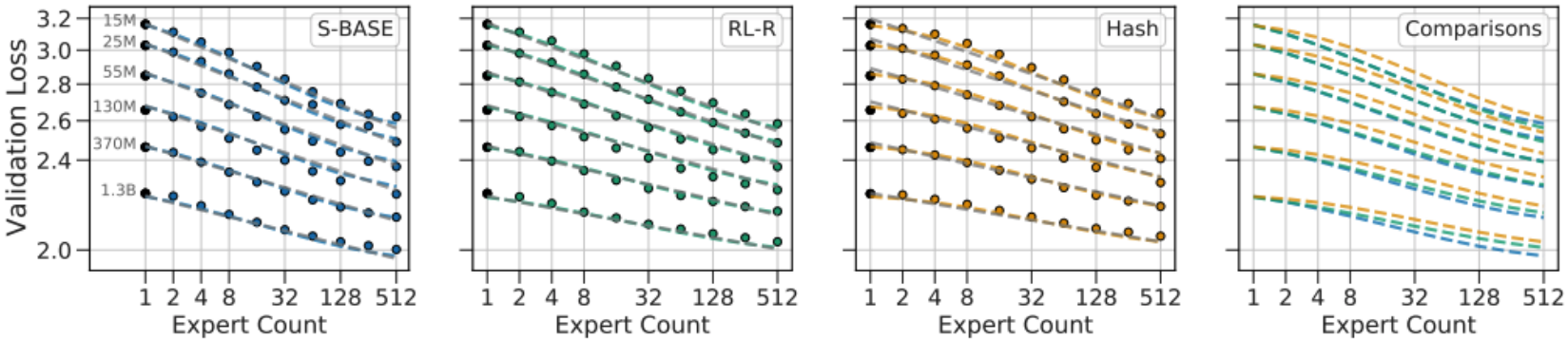
↘ complex, not ~~too~~ many gains

Guess which one people use in practice?

RL for MoEs

(principled, but complex)

RL via REINFORCE does work, but not so much better that it's a clear win



(REINFORCE baseline approach, Clark et al 2020)

RL is the 'right solution' but gradient variances and complexity means it's not widely used

Stochastic approximations

$$G(x) = Softmax(KeepTopK(H(x), k))$$

$$H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i)$$

$$KeepTopK(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases}$$

From Shazeer et al 2017 – routing decisions are *stochastic* with gaussian perturbations.

1. This naturally leads to experts that are a bit more robust.
2. The softmax means that the model learns how to rank K experts

Stochastic approximations

```
if is_training:
    # Add noise for exploration across experts.
    router_logits += mtf.random_uniform(shape=router_logits.shape, minval=1-eps, maxval=1+eps)

# Convert input to softmax operation from bfloat16 to float32 for stability.
router_logits = mtf.to_float32(router_logits)

# Probabilities for each token of what expert it should be sent to.
router_probs = mtf.softmax(router_logits, axis=-1)
```

Stochastic jitter in Fedus et al 2022. This does a uniform multiplicative perturbation for the same goal of getting less brittle experts. This was later removed in Zoph et al 2022

Method	Fraction Stable	Quality (↑)
Baseline	4/6	**-1.755** ±0.02
Input jitter (10^{-2})	3/3	-1.777 ±0.03
Dropout (0.1)	3/3	-1.822 ±0.11

Heuristic balancing losses

Another key issue – systems efficiency requires that we use experts evenly..

For each Switch layer, this auxiliary loss is added to the total model loss during training. Given N experts indexed by $i = 1$ to N and a batch $\mathcal{B}$ with T tokens, the auxiliary loss is computed as the scaled dot-product between vectors f and P,

$$\text{loss} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \cdot P_i \tag{4}$$

where f_i is the fraction of tokens dispatched to expert i,

$$f_i = \frac{1}{T} \sum_{x \in \mathcal{B}} \mathbb{1}\{\text{argmax } p(x) = i\} \tag{5}$$

and P_i is the fraction of the router probability allocated for expert i, [2]

$$P_i = \frac{1}{T} \sum_{x \in \mathcal{B}} p_i(x). \tag{6}$$

From the Switch Transformer [Fedus et al 2022]

The derivative with respect to $p_i(x)$ is $\frac{\alpha N}{T^2} \sum 1_{argmax\ p(x)=i}$,
so more frequent use = stronger downweighting

Example from deepseek (v1-2)

Per-expert balancing – same as the switch transformer

$$\mathcal{L}_{\text{ExpBal}} = \alpha_1 \sum_{i=1}^{N'} f_i P_i, \tag{12}$$

$$f_i = \frac{N'}{K'T} \sum_{t=1}^{T} \mathbb{1}(\text{Token } t \text{ selects Expert } i), \tag{13}$$

$$P_i = \frac{1}{T} \sum_{t=1}^{T} s_{i,t}, \tag{14}$$

→ expert group

each batch expert gets even number of tokens. So, if u pick an expert more freq (P(↑)), the no. of tokens u send (f) will be less.

Per-device balancing – the objective above, but aggregated by device.

$$\mathcal{L}_{\text{DevBal}} = \alpha_2 \sum_{i=1}^{D} f'_i P'_i, \tag{15}$$

$$f'_i = \frac{1}{|\mathcal{E}_i|} \sum_{j \in \mathcal{E}_i} f_j, \tag{16}$$

$$P'_i = \sum_{j \in \mathcal{E}_i} P_j, \tag{17}$$

→ device group

same as above, measure which tokens go to which device for even systems gains!!

DeepSeek v3 variation – per-expert biases

Set up a per-expert bias (making it more likely to get tokens) and use online learning

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} + b_i \in \text{Topk}(\{s_{j,t} + b_j | 1 \leqslant j \leqslant N_r\}, K_r), \\ 0, & \text{otherwise.} \end{cases}$$

use b_i's only to make routing decisions.

if expert i gets more tokens, reduce b_i. if expert i gets less tokens, ↑ b_i so it'll come in topk.

They call this '**auxiliary loss free balancing**'

Complementary Sequence-Wise Auxiliary Loss. Although DeepSeek-V3 mainly relies on the auxiliary-loss-free strategy for load balance, to prevent extreme imbalance within any single sequence, we also employ a complementary sequence-wise balance loss:

$$\mathcal{L}_{\text{Bal}} = \alpha \sum_{i=1}^{N_r} f_i P_i, \tag{17}$$

$$f_i = \frac{N_r}{K_r T} \sum_{t=1}^{T} \mathbb{1}\left(s_{i,t} \in \text{Topk}(\{s_{j,t} | 1 \leqslant j \leqslant N_r\}, K_r)\right), \tag{18}$$

$$s'_{i,t} = \frac{s_{i,t}}{\sum_{j=1}^{N_r} s_{j,t}}, \tag{19}$$

$$P_i = \frac{1}{T} \sum_{t=1}^{T} s'_{i,t}, \tag{20}$$

(but the approach is not fully aux loss free..)

What happens when removing load balancing losses?

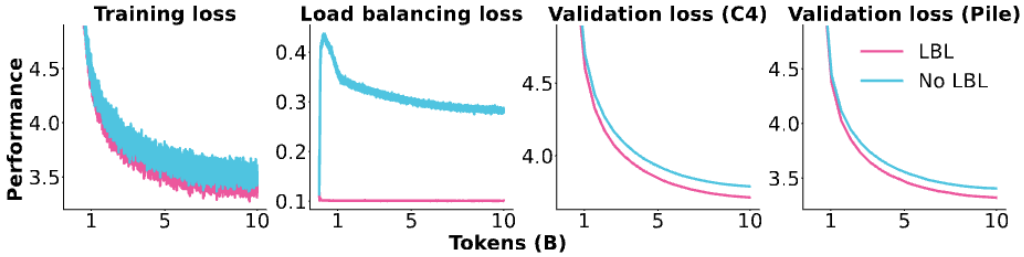


Figure 9: **Impact of applying a load balancing loss (LBL).** The training loss plot excludes the load balancing loss for both models. More results, logs, and configurations: https://wandb.ai/ai2-llm/olmoe/reports/Plot-LBL-vs-No-LBL--Vmlldzo4OTkyNDg4

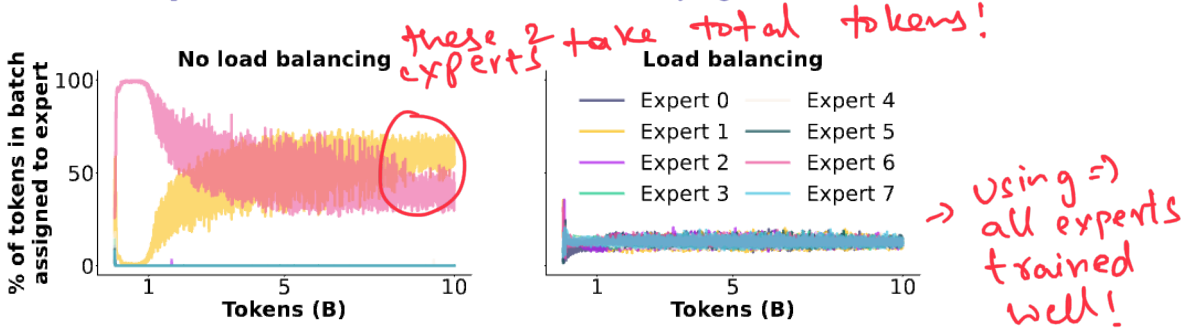
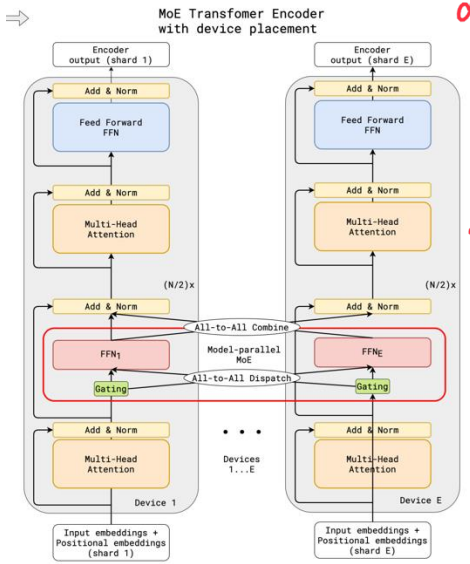


Figure 10: **Expert assignment during training when using or not using a load balancing loss for the first MoE layer.** More results, logs, and configurations: https://wandb.ai/ai2-llm/olmoe/reports/Plot-LBL-vs-No-LBL--Vmlldzo4OTkyNDg4

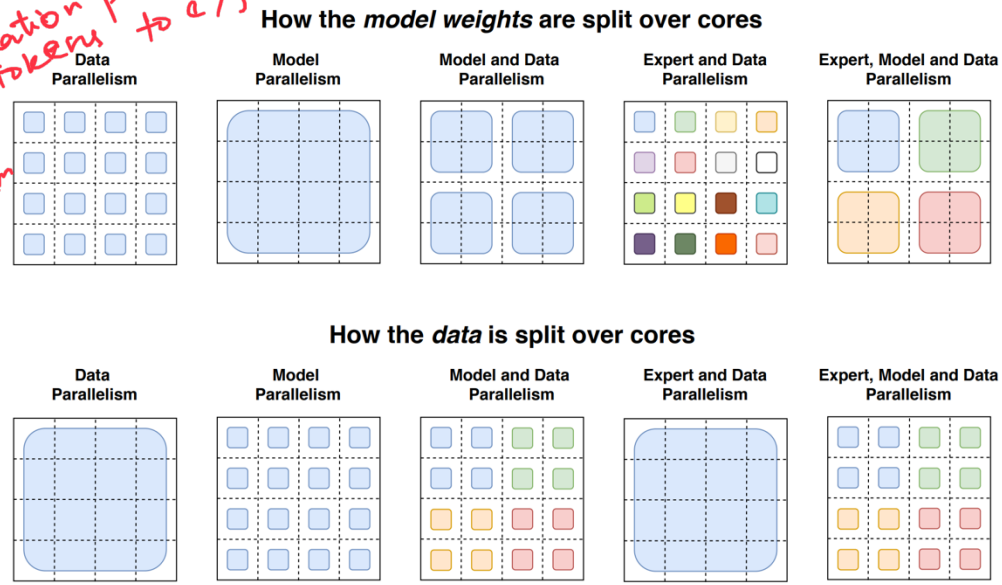
Training MoEs – the systems side

MoEs parallelize nicely – Each FFN can fit in a device



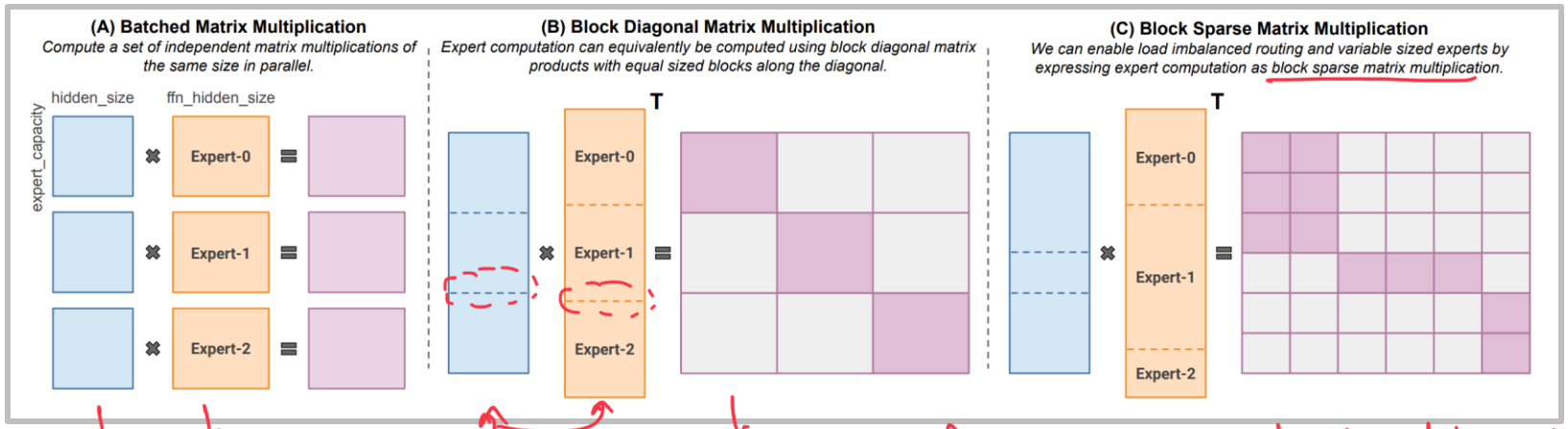
All-all dispatch & all-all combine
↓
Communication protocols to send tokens to expert FFN & collect from them

Enables additional kinds of parallelism



Training MoEs – the systems side

MoE routing allows for parallelism, but also some complexities

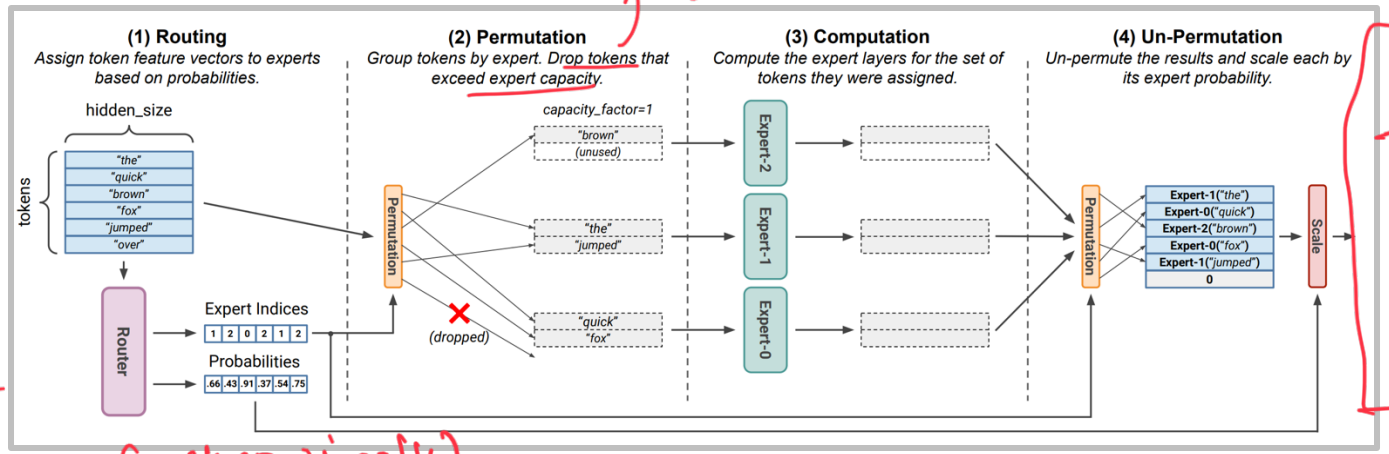


Modern libraries like MegaBlocks (used in many open MoEs) use smarter sparse MMs

Fun side issue – stochasticity of MoE models

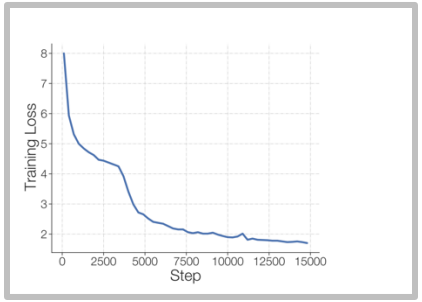
There was speculation that GPT-4's stochasticity was due to MoE..

Why would a MoE have additional randomness?



Token dropping from routing happens at a *batch* level – this means that other people's queries can drop your token!

Issues with MoEs - stability



[7]Exponential functions have the property that a small input perturbation can lead to a large difference in the output. As an example, consider inputting 10 logits to a softmax function with values of 128 and one logit with a value 128.5. A roundoff error of 0.5 in `bfloat16` will alter the softmax output by 36% and incorrectly make all logits equal. The calculation goes from $\frac{\exp(0)}{\exp(0)+10\cdot\exp(-0.5)} \approx 0.142$ to $\frac{\exp(0)}{\exp(0)+10\cdot\exp(0)} \approx 0.091$. This occurs because the max is subtracted from all logits (for numerical stability) in softmax operations and the roundoff error changes the number from 128.5 to 128. This example was in `bfloat16`, but analogous situations occur in `float32` with larger logit values.

[Zoph 2022]

e-vectors.

Solution: Use Float 32 just for the expert router (sometimes with an aux z-loss)

$$L_z(x) = \frac{1}{B}\sum_{i=1}^{B}\left(\log\sum_{j=1}^{N} e^{x_j^{(i)}}\right)^2 \quad (5)$$

→ this normalizes the values near 1 → nice for stability

softmax easily disrupts training as small disturbances shoot up.

Z-loss stability for the router

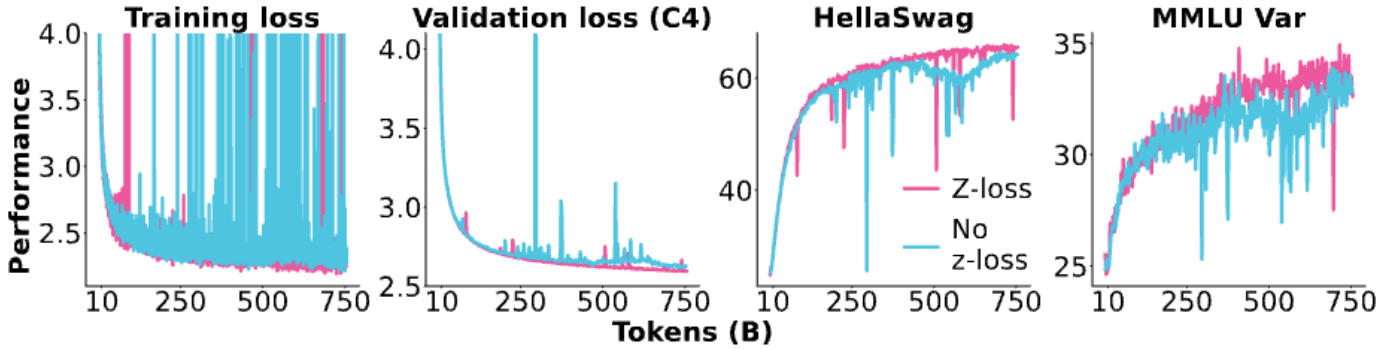


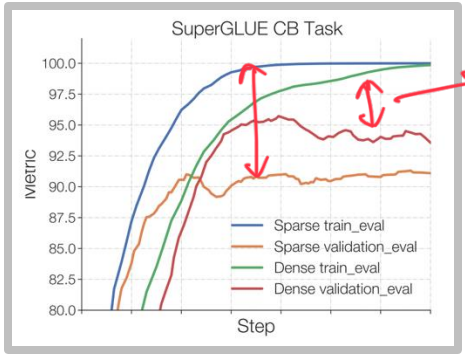
Figure 11: **Router z-loss.** We compare adding router z-loss with a loss weight of 0.001 versus no additional z-loss. More results, logs, and configurations: https://wandb.ai/ai2-llm/olmoe/reports/Plot-Zloss-vs-none--Vmlldzo4NDM4NjUz

What happens when we remove the z-loss?

Issues with MoEs – fine-tuning

this is a problem for MoE + RLHF as RLHF data is not easy to get (limited)

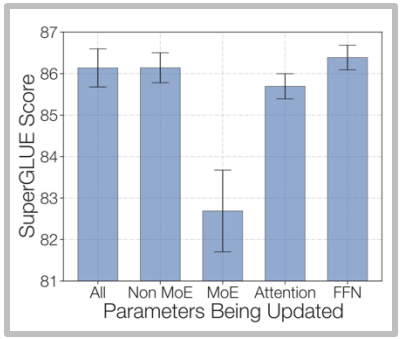
Sparse MoEs can overfit on smaller fine-tuning data



→ dense (less train-test gap)

→ b/c2 these are too huge models.

Zoph et al solution – finetune non-MoE MLPs

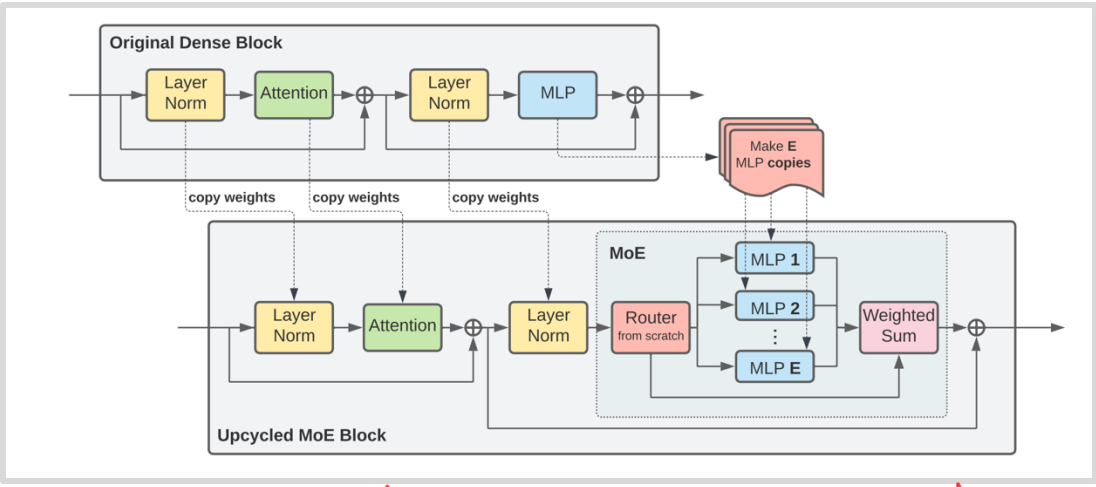


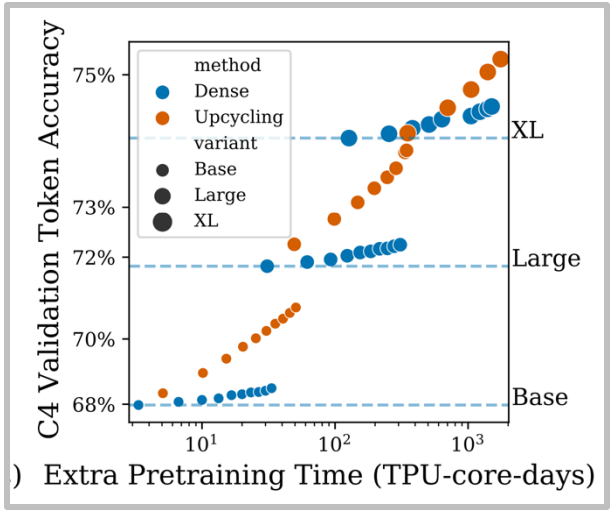
DeepSeek solution – use lots of data 1.4M SFT

if overfitting is problem, use lots of data

Training Data. For training the chat model, we conduct supervised fine-tuning (SFT) on our in-house curated data, comprising 1.4M training examples. This dataset spans a broad range of categories including math, code, writing, question answering, reasoning, summarization, and more. The majority of our SFT training data is in English and Chinese, rendering the chat model versatile and applicable in bilingual scenarios.

Other training methods - upcycling





start with MLPs as experts & finetune (perturb) a bit

Can we use a pre-trained LM to initialize a MoE? → Yes & cost-effective.

Upcycling example - MiniCPM

Uses the MiniCPM model (topk=2, 8 experts, ~ 4B active params).

Model	C-Eval	CMMLU	MMLU	HumanEval	MBPP	GSM8K	MATH	BBH
Llama2-34B	-	-	62.6	22.6	33.0†	42.2	6.24	**44.1**
Deepseek-MoE (16B)	40.6	42.5	45.0	26.8	39.2	18.8	4.3	-
Mistral-7B	46.12	42.96	**62.69**	27.44	45.20	33.13	5.0	41.06
Gemma-7B	42.57	44.20	60.83	38.41	50.12	47.31	6.18	39.19
MiniCPM-2.4B	51.13	51.07	53.46	50.00	47.31	53.83	10.24	36.87
MiniCPM-MoE (13.6B)	**58.11**	**58.80**	58.90	**56.71**	**51.05**	**61.56**	**10.52**	39.22

Table 6: Benchmark results of MiniCPM-MoE. † means evaluation results on the full set of MBPP, instead of the hand-verified set (Austin et al., 2021). The evaluation results of Llama2-34B and Qwen1.5-7B are taken from their technical reports.

Simple MoE, shows gains from the base model with ~ 520B tokens for training

Upcycling example – Qwen MoE

Qwen MoE – Initialized from the Qwen 1.8B model top-k=4, 60 experts w/ 4 shared.

Model	#Parameters	#(Activated) Parameters	MMLU	GSM8K	HumanEval	Multilingual	MT-Bench
Mistral-7B	7.2	7.2	64.1	47.5	27.4	40.0	7.60
Qwen1.5-7B	7.7	7.7	64.6	50.9	32.3	-	-
Gemma-7B	8.5	7.8	61.0	62.5	36.0	45.2	7.60
DeepSeekMoE 16B	16.4	2.8	45.0	18.8	26.8	-	6.93
Qwen1.5-MoE-A2.7B	14.3	2.7	62.5	61.5	34.2	40.8	7.17

Similar architecture / setup to DeepSeekMoE, but one of the first (confirmed) upcycling successes

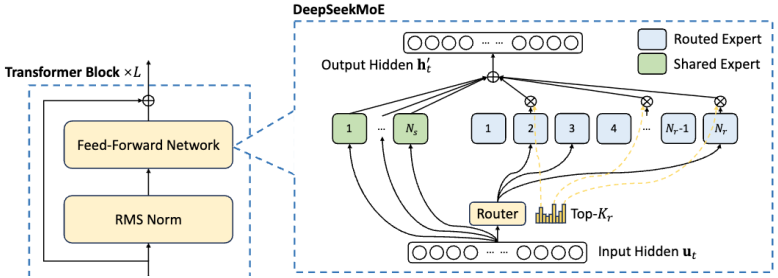
DeepSeek MoE v1-v2-v3

To wrap up, we'll walk through the DeepSeek MoE architecture.

V1 (16B – 2.8 active):

they had both dense & MoE model at this point

Shared (2) + Fine-grained (64/4) experts → so, total will be 6 active experts.



Standard, top-k routing

$$\mathbf{h}'_t = \mathbf{u}_t + \sum_{i=1}^{N_s} \mathrm{FFN}_i^{(s)}(\mathbf{u}_t) + \sum_{i=1}^{N_r} g_{i,t}\,\mathrm{FFN}_i^{(r)}(\mathbf{u}_t),$$

$$g_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \mathrm{Topk}(\{s_{j,t} | 1 \leqslant j \leqslant N_r\}, K_r), \\ 0, & \text{otherwise}, \end{cases}$$

$$s_{i,t} = \mathrm{Softmax}_i\left(\mathbf{u}_t^T \mathbf{e}_i\right),$$

Standard Aux-loss balancing (Expert + Device)

$$\mathcal{L}_{\mathrm{ExpBal}} = \alpha_1 \sum_{i=1}^{N_r} f_i P_i,$$

$$f_i = \frac{N_r}{K_r T} \sum_{t=1}^{T} \mathbb{1}(\text{Token } t \text{ selects Expert } i),$$

$$P_i = \frac{1}{T} \sum_{t=1}^{T} s_{i,t},$$

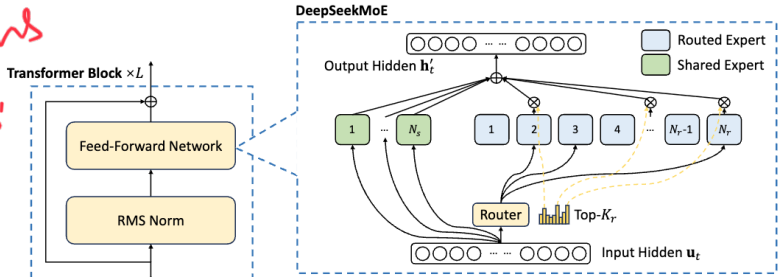
DeepSeek MoE v2

why can't I have 1024 finegrained or 2048 finegrained experts!!

V2 (236B – 21 active):

naively picking top-k experts need to send tokens to lots of devices

Shared (2) + Fine-grained (160/10) experts, 6 active



more experts means we need to route to all of them so communication costs (↑)

New things:

Top-M device routing

> For DeepSeek-V2, beyond the naive top-K selection of routed experts, we additionally ensure that the target experts of each token will be distributed on at most M devices. To be specific, for each token, we first select M devices that have experts with the highest affinity scores in them. Then, we perform top-K selection among experts on these M devices. In practice, we find that when $M \geqslant 3$, the device-limited routing can achieve a good performance roughly aligned with the unrestricted top-K routing.

So, pick top-M devices. restrict picking top-k experts from these

Communication balancing loss – balancing both communication in *and* out

$$\mathcal{L}_{\text{CommBal}} = \alpha_3 \sum_{i=1}^{D} f''_i P''_i, \tag{29}$$

$$f''_i = \frac{D}{MT} \sum_{t=1}^{T} \mathbb{1}(\text{Token } t \text{ is sent to Device } i), \tag{30}$$

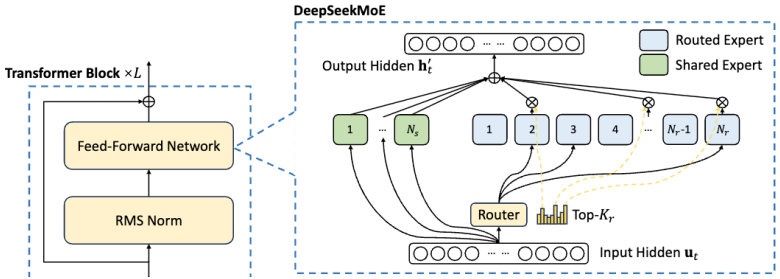
$$P''_i = \sum_{j \in \mathcal{E}_i} P_j, \tag{31}$$

where α_3 is a hyper-parameter called communication balance factor. The device-limited routing mechanism operates on the principle of ensuring that each device transmits at most MT hidden states to other devices. Simultaneously, the communication balance loss is employed to encourage each device to receive around MT hidden states from other devices. The communication balance loss guarantees a balanced exchange of information among devices, promoting efficient communications.

DeepSeek MoE v3

V2 (671B – 37 active):

Shared (1) + Fine-grained (258) experts, 8 active



training time, it's fine not to have it. test time we don't know what sequences we'll get, so we don't want the risk to overload an expert!

New things

Sigmoid+Softmax topK + topM

$$\mathbf{h}'_t = \mathbf{u}_t + \sum_{i=1}^{N_s} \mathrm{FFN}_i^{(s)}(\mathbf{u}_t) + \sum_{i=1}^{N_r} g_{i,t}\,\mathrm{FFN}_i^{(r)}(\mathbf{u}_t),$$

$$g_{i,t} = \frac{g'_{i,t}}{\sum_{j=1}^{N_r} g'_{j,t}},$$

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \mathrm{Topk}(\{s_{j,t} | 1 \leqslant j \leqslant N_r\}, K_r), \\ 0, & \text{otherwise}, \end{cases}$$

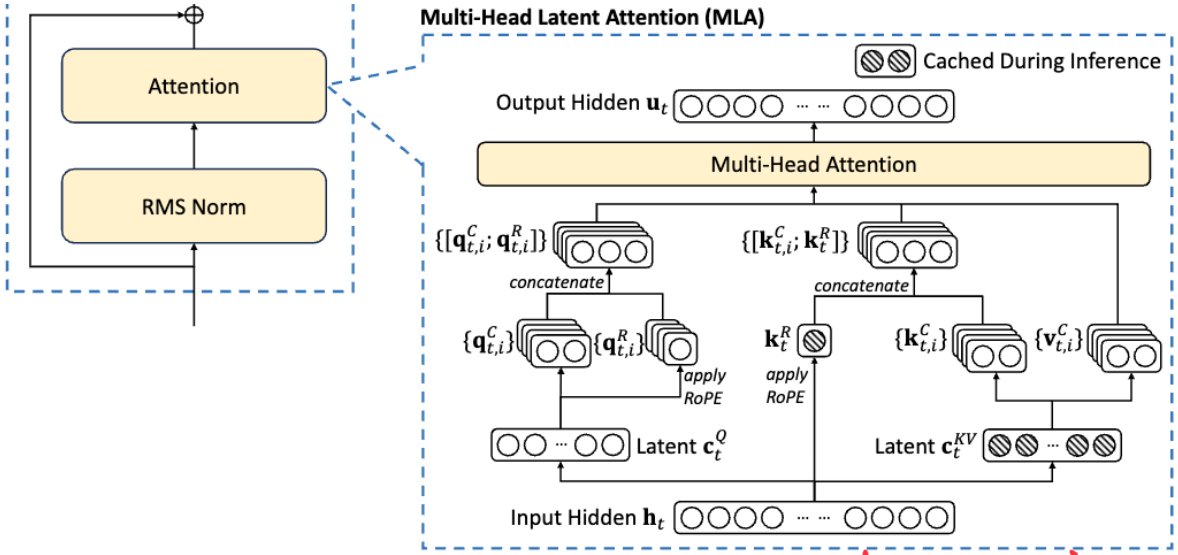
$$s_{i,t} = \mathrm{Sigmoid}\left(\mathbf{u}_t{}^T \mathbf{e}_i\right),$$

Aux-loss-free + seq-wise aux

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} + b_i \in \mathrm{Topk}(\{s_{j,t} + b_j | 1 \leqslant j \leqslant N_r\}, K_r), \\ 0, & \text{otherwise}. \end{cases}$$

Bonus: What else do you need to make DeepSeek MoE v3?

MLA : Multihead, latent attention



lower dimension h to c.

Basic idea: express the Q, K, V as functions of a lower-dim, 'latent' activation

save c's & use it for KV caching.

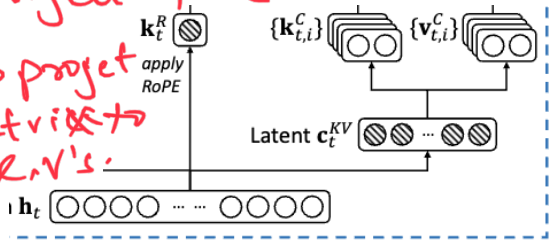
What else do you need to make DeepSeek MoE v3?

Basic idea: express the Q, K, V as functions of a lower-dim, 'latent' activation

$$\mathbf{c}_t^{KV} = W^{DKV}\mathbf{h}_t,$$
$$\mathbf{k}_t^C = W^{UK}\mathbf{c}_t^{KV},$$
$$\mathbf{v}_t^C = W^{UV}\mathbf{c}_t^{KV},$$

down project to C

up project matrices to K, V's.



Benefits: when KV-caching, we only need to store c_t^{KV}, which can be much smaller.

W^{UK} can be merged into the Q projection → so no additional mat muls. FLOPs.

$$\mathbf{c}_t^Q = W^{DQ}\mathbf{h}_t,$$
$$\mathbf{q}_t^C = W^{UQ}\mathbf{c}_t^Q,$$

(they also compress queries, for memory savings during training)

QK^T c_t^{KV} W^{UK} W^{QK} q^K c_t

Complexity: rope conflicts with MLA-style caching.

combining matrices so FLOPs will same.

Without RoPE – $\langle Q, K\rangle = \langle hW^Q, W^{UK}c_t^{KV}\rangle = \langle h\, W^Q W^{UK}, c_t^{KV}\rangle$

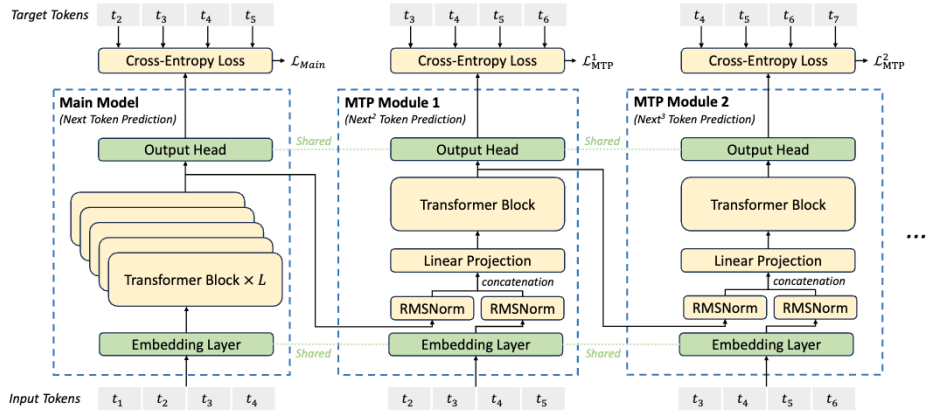
With RoPE - $\langle QR_q, R_k K\rangle = \langle hW^Q R_q, R_k W^{UK}c_t^{KV}\rangle = \langle h\, W^Q R_q R_k W^{UK}, c_t^{KV}\rangle$

The solution – Have a few non-latent key dimensions that can be rotated !!

What else do you need to make DeepSeek MoE v3?

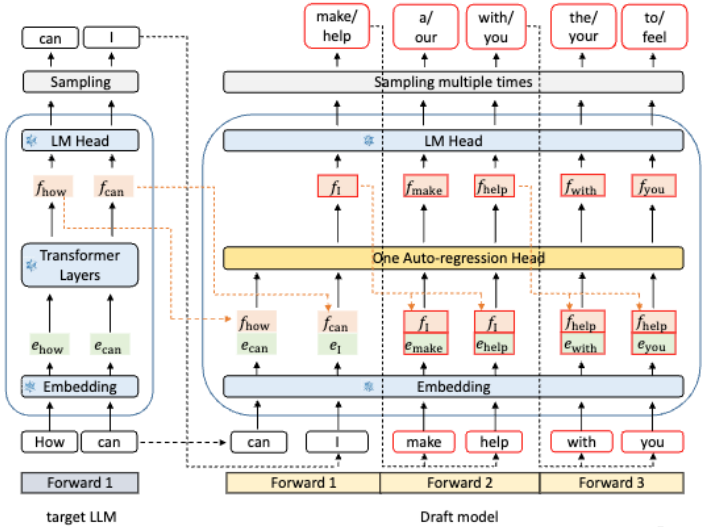
MTP: Have small, lightweight models that predict multiple steps ahead

multiple tokens in 1 fwd.



(But they only do MTP with one token ahead)

[Deepseek v3]



[EAGLE]

just take hidden state & pass it to light-weight transformer

$$\mathbf{h}_i'^k = M_k[\text{RMSNorm}(\mathbf{h}_i^{k-1}); \text{RMSNorm}(\text{Emb}(t_{i+k}))],$$

$$\mathbf{h}_{1:T-k}^k = \text{TRM}_k(\mathbf{h}_{1:T-k}'^k),$$

$$P_{i+k+1}^k = \text{OutHead}(\mathbf{h}_i^k).$$

(See paper for ablations)

MoE summary

- MoEs take advantage of sparsity – not all inputs need the full model
  - Discrete routing is hard, but top-k heuristics seem to work
- Lots of empirical evidence now that MoEs work, and are cost-effective