Lecture 4

More details and some things about MoEs

CS336

Tatsu H

Outline and goals

- Finishing up – tokenizers, attention variants
- Mixture of experts

Tokenizers

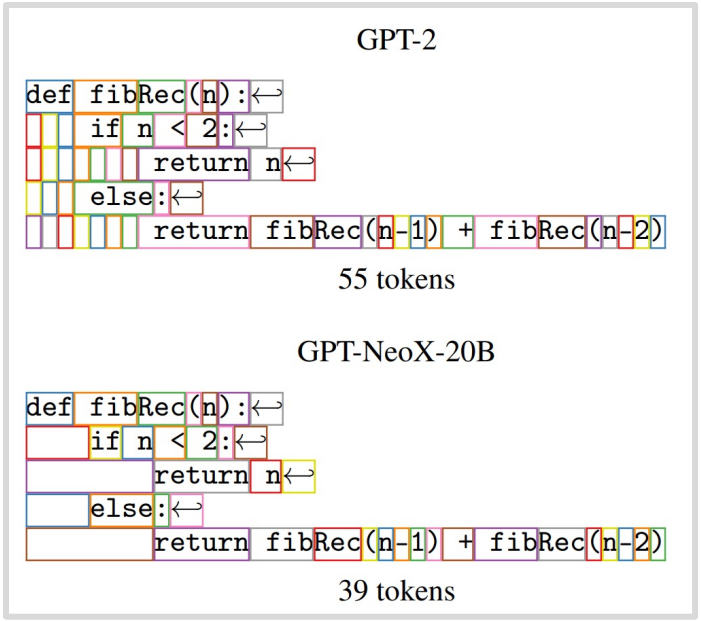
The non-google world uses BPE. Google uses the SentencePiece library, which (sometimes) refers to a non-BPE subword tokenizer

Model	Tokenizer
Original transformer	BPE
GPT 1/2/3	BPE
T5 / mT5 / T5v1.1	SentencePiece (Unigram)
Gopher/Chinchilla	SentencePiece (??)
PaLM	SentencePiece (??)
LLaMA	BPE

Important property – all of these tokenizers are *invertible*

Whitespace and number related hacks

Multi-whitespace tokenization (GPT-NeoX)



Individual digit tokenization (LLaMA/DeepSeek)

Tokenizer. We tokenize the data with the byte-pair encoding (BPE) algorithm (Sennrich et al., 2015), using the implementation from SentencePiece (Kudo and Richardson, 2018). Notably, we split all numbers into individual digits, and fallback to bytes to decompose unknown UTF-8 characters.

What are typical vocabulary sizes?

Monolingual models – 30-50k vocab

Model	Token count
Original transformer	37000
GPT	40257
GPT2/3	50257
T5/T5v1.1	32128
LLaMA	32000

Multilingual / production systems 100-250k

Model	Token count
mT5	250000
PaLM	256000
GPT4	100276
BLOOM	250680
DeepSeek	100000
Qwen 15B	152064
Yi	64000

Monolingual vocabs don’t need to be huge, but multilingual ones do

Tokenizer: summary

- Everyone uses invertible subword tokenizers (BPE, Unigram) for good reason
- NFKC normalization is a double edged sword (2^5) and many models don’t use it
- For math and code, careful manual handling of whitespace and numbers can help

Attention heads

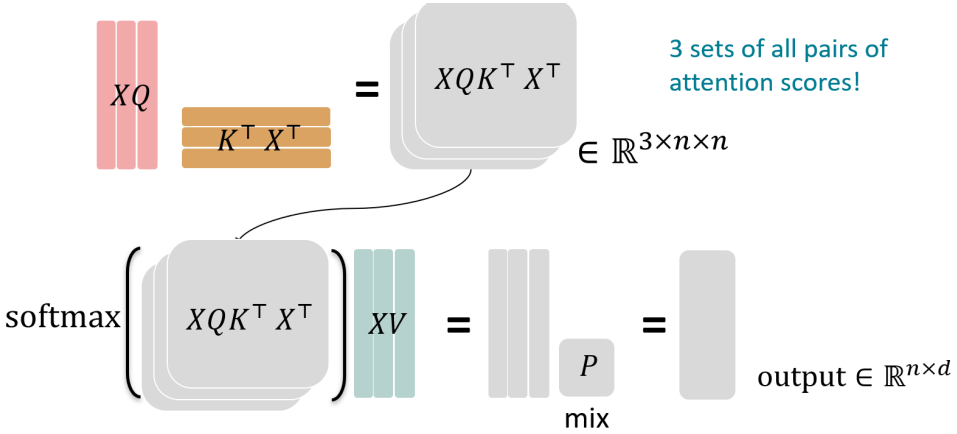
Most models don’t touch the attention head at all with a few minor exceptions..

GQA / MQA : Saving inference costs by reducing the number of heads

Sparse or sliding window attention (GPT4/Mistral): restricting the attention pattern to reduce compute cost

GQA/MQA – Reducing attention head cost

Let's think about the compute involved for attention



X softmax projection

Total arithmetic operations (bnd^2), **total memory accesses** $(bnd + bhn^2 + d^2)$

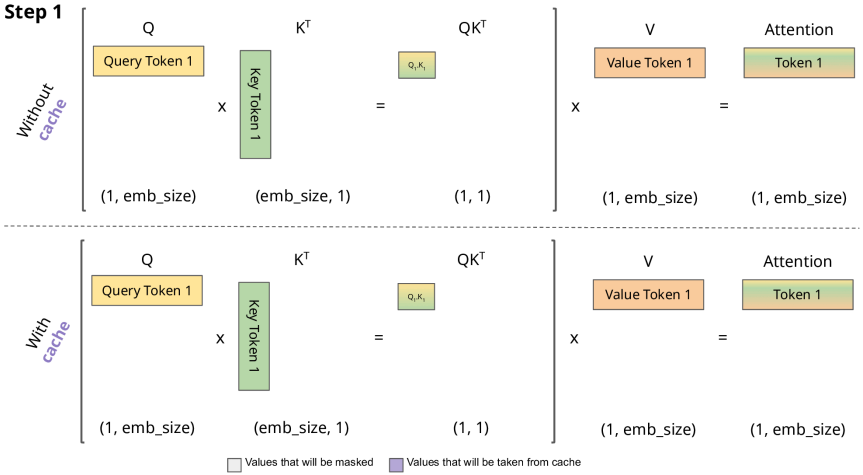
Arithmetic intensity is high $O\left(\left(\frac{1}{k} + \frac{1}{bn}\right)^{-1}\right)$ - we can keep our GPUs running

GQA/MQA – Reducing attention head cost

What about the *incremental* case when we generate text?

Key difference: can't parallelize the generation process – needs to be step by step

In this case – we need to incrementaly re-compute/update attention via the 'KV cache'



[Animation from https://medium.com/@joaolages/kv-caching-explained-276520203249]

GQA/MQA – Reducing attention head cost

What's the incremental arithmetic intensity?

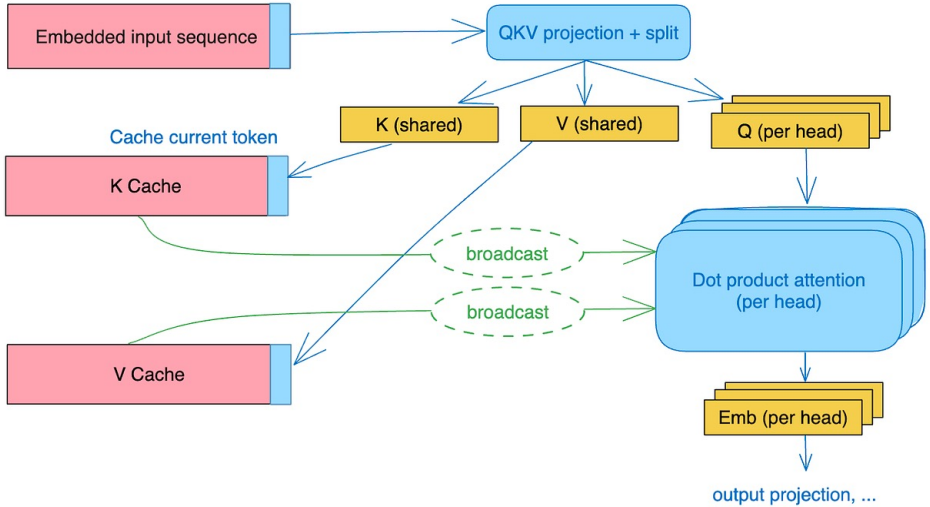
Total arithmetic operations (bnd^2), **total memory accesses** $(\overset{\text{X}}{bn^2d} + \overset{\text{projection}}{nd^2})$

Arithmetic intensity is not good $O\left(\left(\frac{n}{d} + \frac{1}{b}\right)^{-1}\right)$ - need large batches + short seq length (n) or big model dimensions (d)

Is there some way around this? The n/d term is difficult to reduce.

MQA – just have fewer key dimensions.

Key idea – have multiple queries, but just one dimension for keys and values



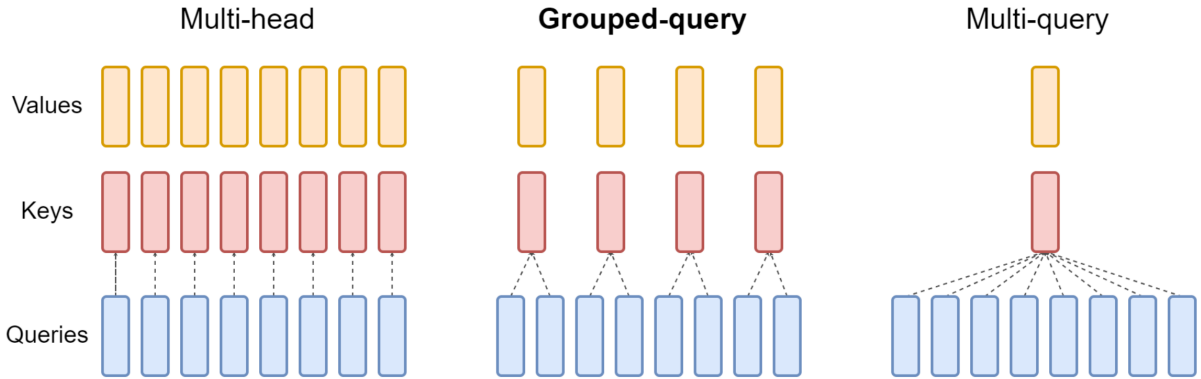
We have much fewer items to move in and out of memory (KV Cache)

Total memory access $(bnd + bn^2k + nd^2)$, **Arithmetic intensity** $O\left(\left(\frac{1}{d} + \frac{n}{dh} + \frac{1}{b}\right)^{-1}\right)$

[figure from https://blog.fireworks.ai/multi-query-attention-is-all-you-need-db072e758055]

Recent extension – GQA

Don’t go all the way to one dimension of KV – have fewer dims



Simple knob to control expressiveness (key-query ratio) and inference efficiency

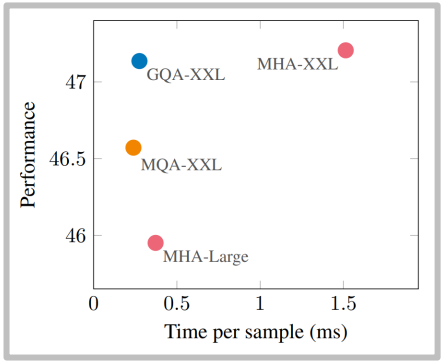
Does MQA hurt? Sometimes..

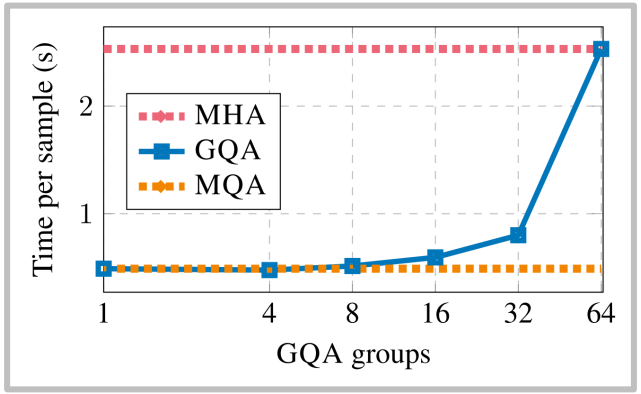
Small PPL hit w/ MQA [Shazeer 2019]

Table 3: Billion-Word LM Benchmark Results.

Attention	h	d_k, d_v	d_{ff}	dev-PPL
multi-head	8	128	8192	**29.9**
multi-query	8	128	9088	30.2
multi-head	1	128	9984	31.2
multi-head	2	64	9984	31.1
multi-head	4	32	9984	31.0
multi-head	8	16	9984	30.9

Low/no hit w/ GQA [Ainslie 2023]

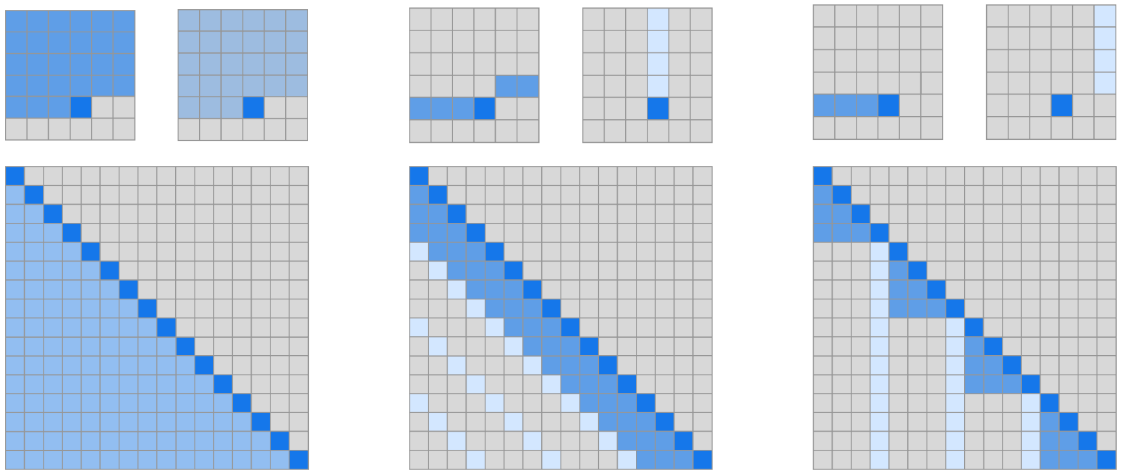




Sparse / sliding window attention

Attending to the entire context can be expensive (quadratic).

Build sparse / structured attention that trades off expressiveness vs runtime (GPT3)

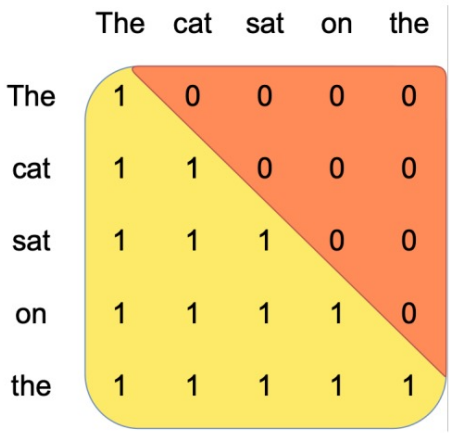


(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

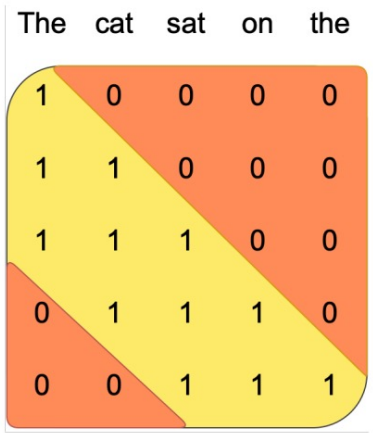
[Child et al 2019]

Sliding window attention

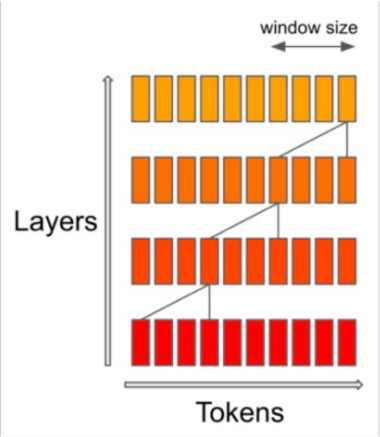
Another variation on this idea – sliding window attention



Vanilla Attention



Sliding Window Attention



Effective Context Length

Just use the main part of the strided pattern – let depth extend effective context (Mistral)

Architecture, hyperparams, and other details

Many aspects (arch, hparams) of transformers are in common across the big LMs

Name	Has pa...	Link	Year	Tokenizer type	Vocab count	Norm	Parallel Layer	Pre-norm	Position embedding	Activations	MoE	MLP factor	num_layers	model_dim
Original transformer	Yes	arxiv.org/abs...03762	2017	BPE	37000	LayerNorm	Serial	☐	Sine	ReLU	☐	4	6	
GPT	Yes	cdn.openai.com/res...er.pdf	2018	BPE	40257	LayerNorm	Serial	☐	Absolute	GeLU	☐	4	12	
GPT2	Yes	cdn.openai.com/bet...rs.pdf	2019	BPE	50257	LayerNorm	Serial	☑	Sine	GeLU	☐	4	48	
T5 (11B)	Yes	arxiv.org/abs...10683	2019	SentencePiece	32128	RMSNorm	Serial	☑	Relative	ReLU	☐	64	24	
GPT3 (175B)	Yes	arxiv.org/abs...14165	2020	BPE	50257	LayerNorm	Serial	☑	Sine	GeLU	☐	4	96	
mT5	Yes	arxiv.org/abs...11934	2020	SentencePiece	250000	RMSNorm	Serial	☑	Relative	GeGLU	☐	2.5	24	
T5 (XXL 11B) v1.1	Kind of	github.com/goo...d#t511	2020	SentencePiece	32128	RMSNorm	Serial	☑	Relative	GeGLU	☐	2.5	24	
Gopher (280B)	Yes	arxiv.org/abs...11446	2021	SentencePiece	32000	RMSNorm	Serial	☑	Relative	ReLU	☐	4	80	
Anthropic LM (not claude)	Yes	arxiv.org/abs...00861	2021	BPE	65536			☑			☐	4	64	
LaMDA	Yes	arxiv.org/abs...08239	2021	BPE	32000			☑	Relative	GeGLU	☐	8	64	
GPTJ	Kind of	huggingface.co/Ele...t-j-6b	2021	BPE	50257	LayerNorm	Parallel	☑	RoPE	GeLU	☐		28	
Chinchilla	Yes	arxiv.org/abs...15556	2022	SentencePiece	32000	RMSNorm	Serial	☑	Relative	ReLU	☐	4	80	
PaLM (540B)	Yes	arxiv.org/abs...02311	2022	SentencePiece	256000	RMSNorm	Parallel	☑	RoPE	SwiGLU	☐	4	118	
OPT (175B)	Yes	arxiv.org/abs...01068	2022	BPE	50272	LayerNorm	Serial	☑	Absolute	ReLU	☐	4	96	
BLOOM (175B)	Yes	arxiv.org/abs...05100	2022	BPE	250680	LayerNorm	Serial	☑	AliBi	GeLU	☐	4	70	
GPT-NeoX	Yes	arxiv.org/pdf...45.pdf	2022	BPE	50257	LayerNorm	Parallel	☑	RoPE	GeLU	☐	4	44	
GPT4 OPEN	Ad	arxiv.org/abs...08774	2023	BPE	100000			☐			☐			
LLaMA (65B)	Yes	arxiv.org/abs...13971	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	2.6875	80	
LLaMA2 (70B)	Yes	arxiv.org/abs...09288	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	3.5	80	
Mistral (7B)	Yes	arxiv.org/abs...06825	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	3.5	32	

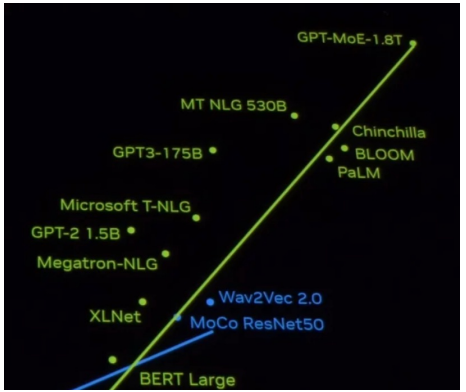
Major differences? Position embeddings, activations, tokenization

Mixture of experts

Mixtral of experts

A high quality Sparse Mixture-of-Experts.

DBRX



GPT4 (?)

Mistral AI
@MistralAI

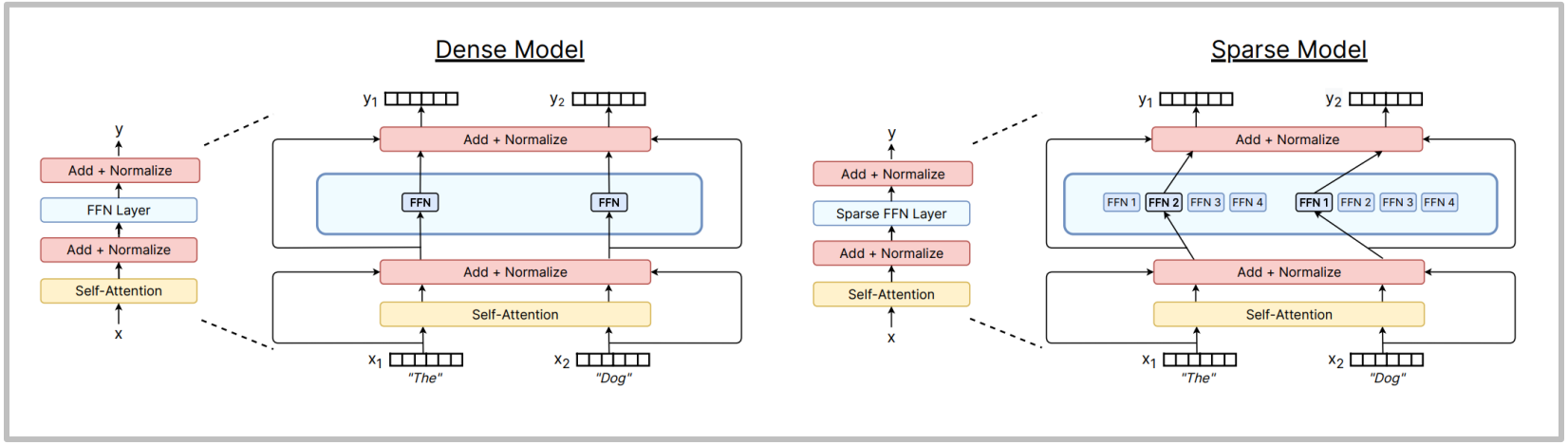
magnet:?xt=urn:btih:9238b09245d0d8cd915be09927769d5f7584c1c9&dn=mixtral-8x22b&tr=udp%3A%2F%2Fopen.demonii.com%3A1337%2Fannounce&tr=http%3A%2F%2Ftracker.opentrackr.org%3A1337%2Fannounce

Grok

deepseek

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

What's a MoE?



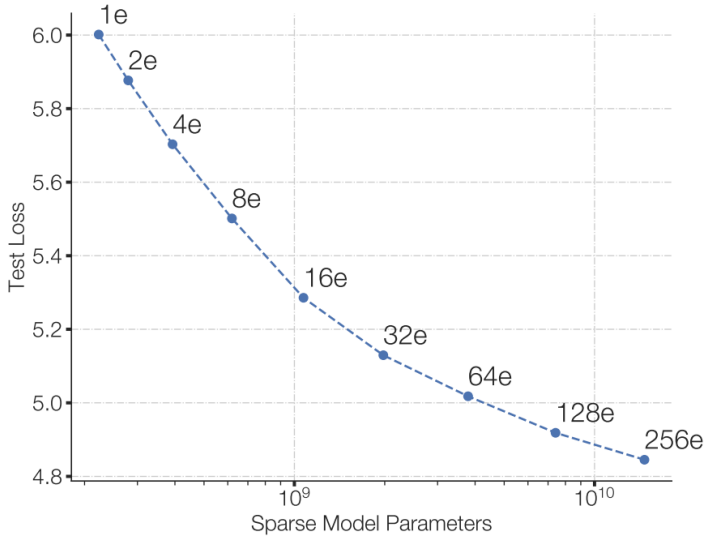
[Fedus et al 2022]

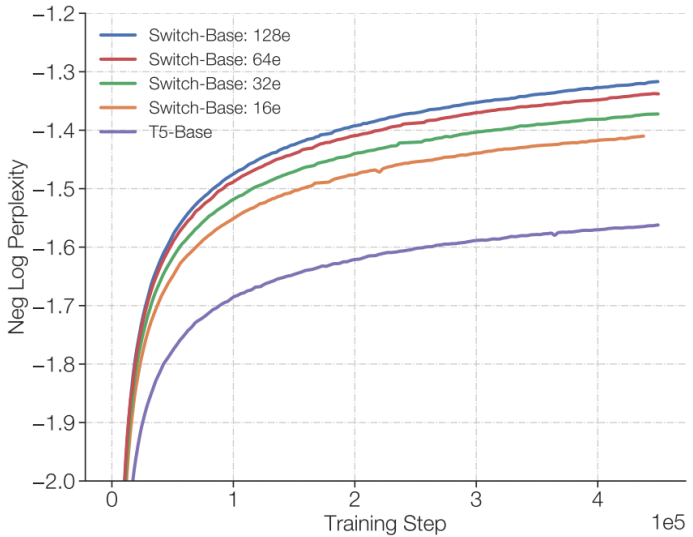
Replace big feedforward with (many) big feedforward networks and a selector layer

You can increase the # experts without affecting FLOPs

Why are MoEs getting popular?

Same FLOP, more param does better

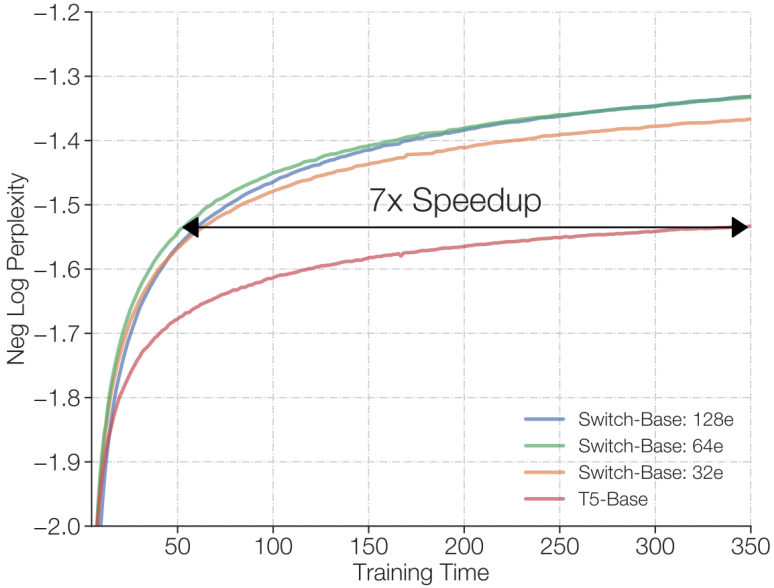




[Fedus et al 2022]

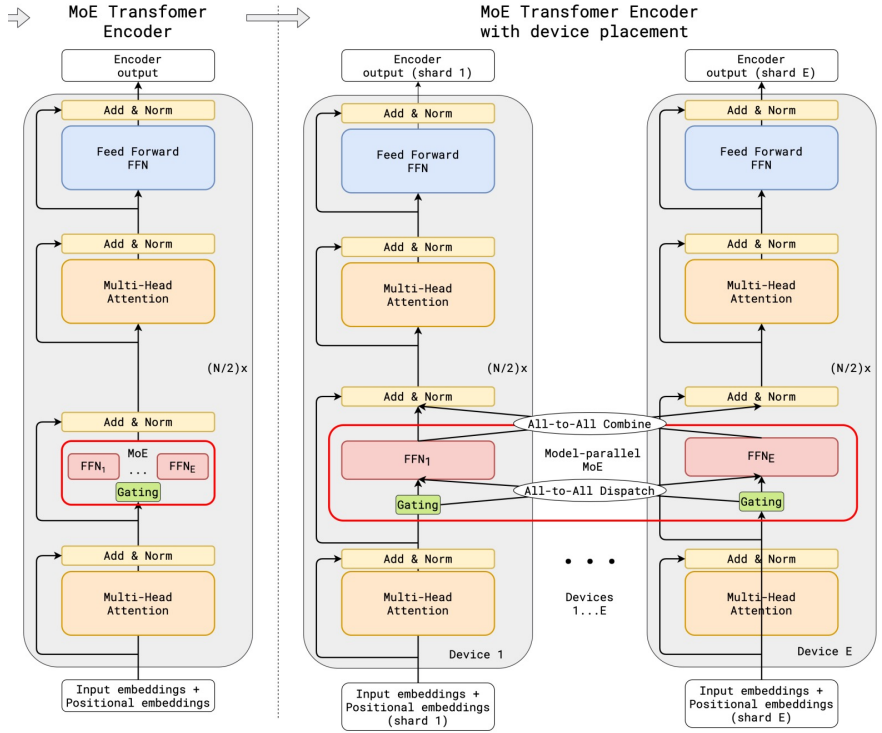
Why are MoEs getting popular?

Faster to train MoEs

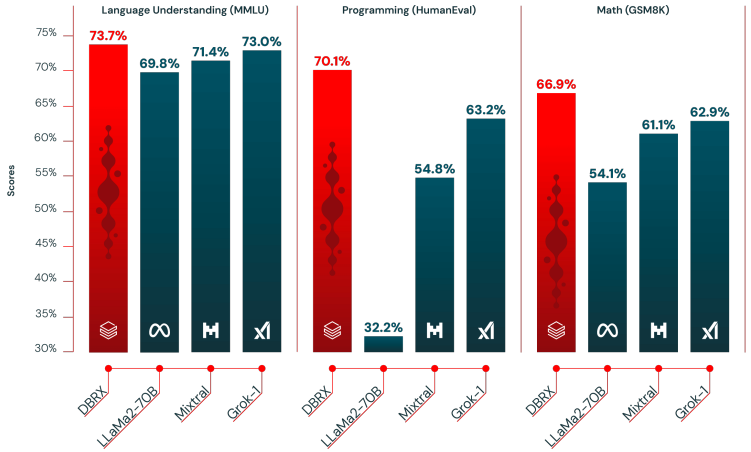


Why are MoEs getting popular?

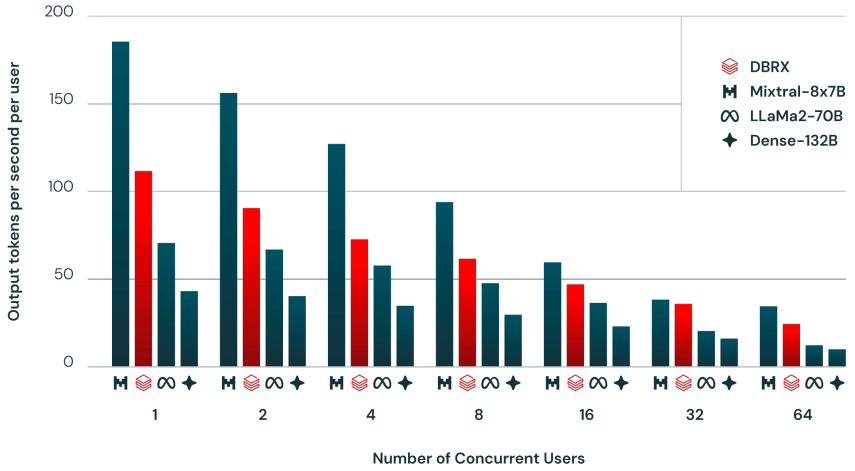
Parallelizable to many devices



Some recent MoE results – Mixtral / DBRX / Grok



Performance



Inference throughput

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

Recent MoE results – 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

Recent MoE results - DeepSeek

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

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

Why haven't MoEs been more popular?

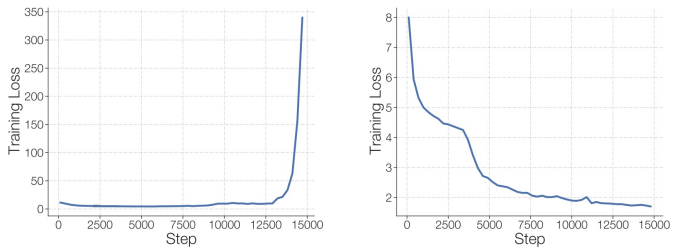
Infrastructure is complex / advantages on multi node

> 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)

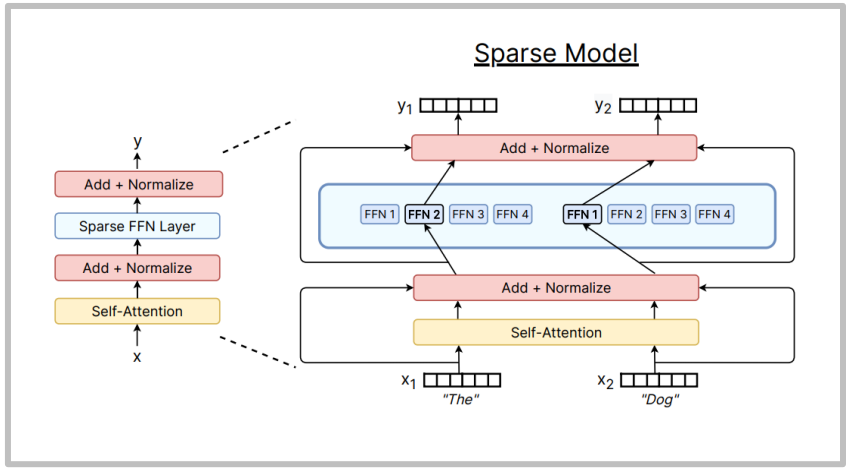
Sparse models often suffer from training instabilities (Figure 1) worse than those observed in standard densely-activated Transformers.



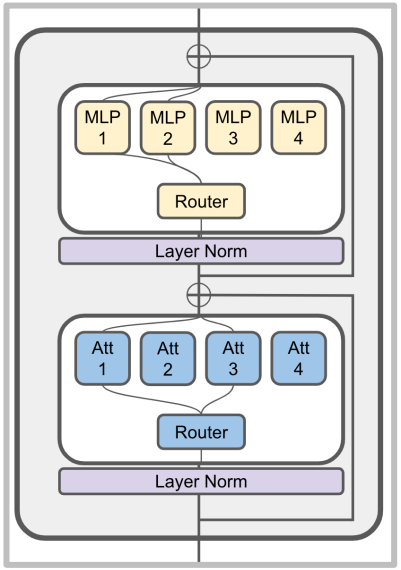
[Zoph et al 2022]

What MoEs generally look like

Typical: replace MLP with MoE layer



Less common: MoE for attention heads



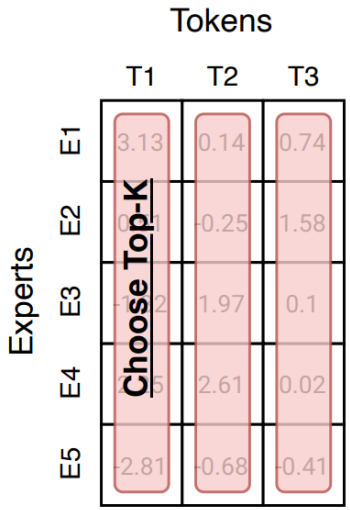
[ModuleFormer, JetMoE]

MoE – what varies?

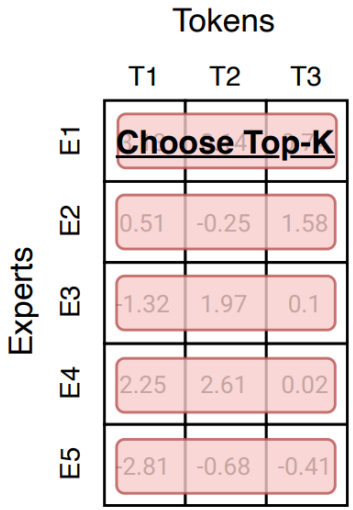
- Routing function
- Expert sizes
- Training objectives

Routing function - overview

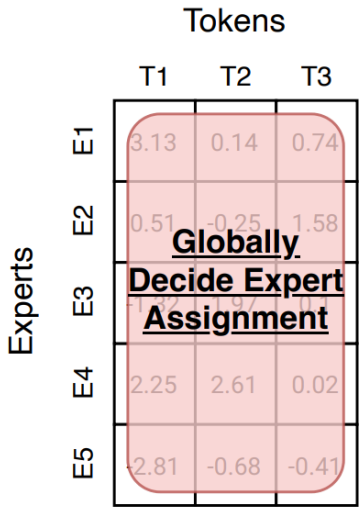
Many of the routing algorithms boil down to ‘choose top k’



Token chooses expert



Expert chooses token

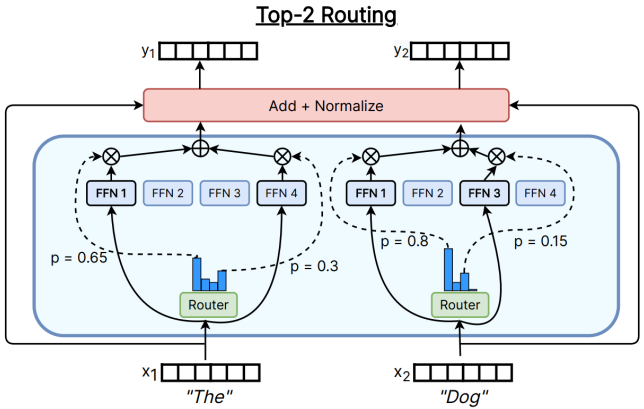


Global routing via optimization

[Fedus et al 2022]

Common routing variants in detail

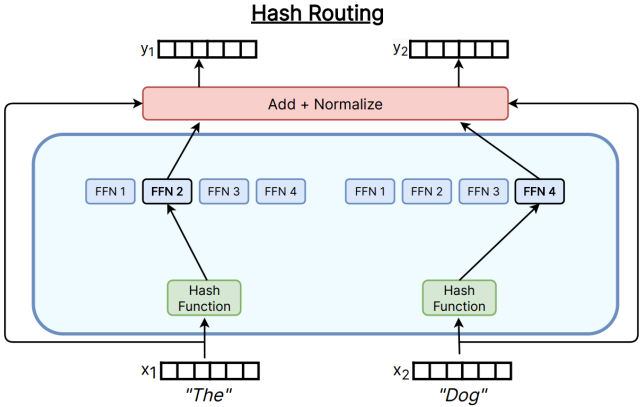
Top-k



Used in *most* MoEs

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

Hashing

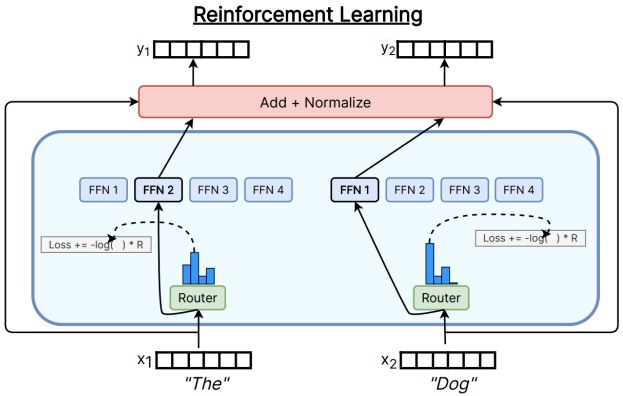


Common baseline

[Fedus et al 2022]

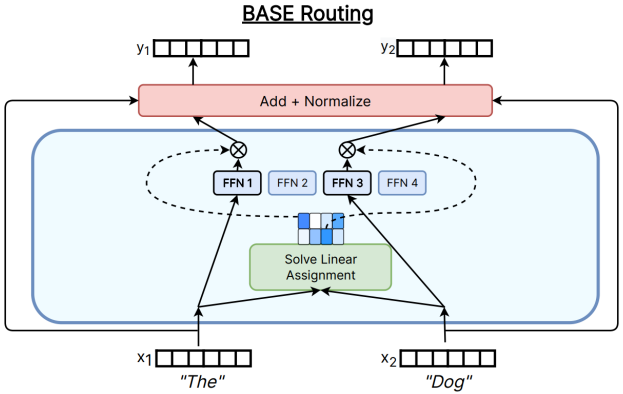
Other routing methods

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.

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

Gating

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

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

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

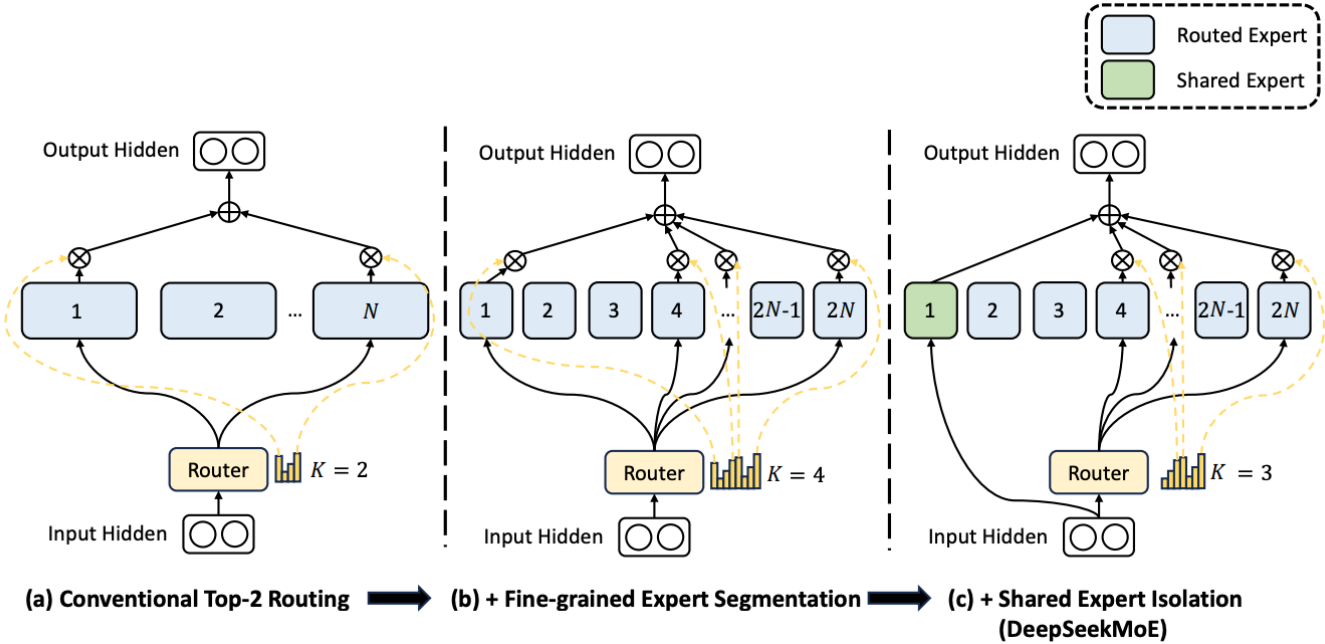
Gates selected by a logistic regressor

This is the DeepSeek router (Grok does this too)

Mixtral and DBRX softmaxes after the TopK

[Dai et al 2024]

Recent variations from Chinese LMs



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

(Used in DeepSeek / Qwen)

Various ablations from the DeepSeek paper

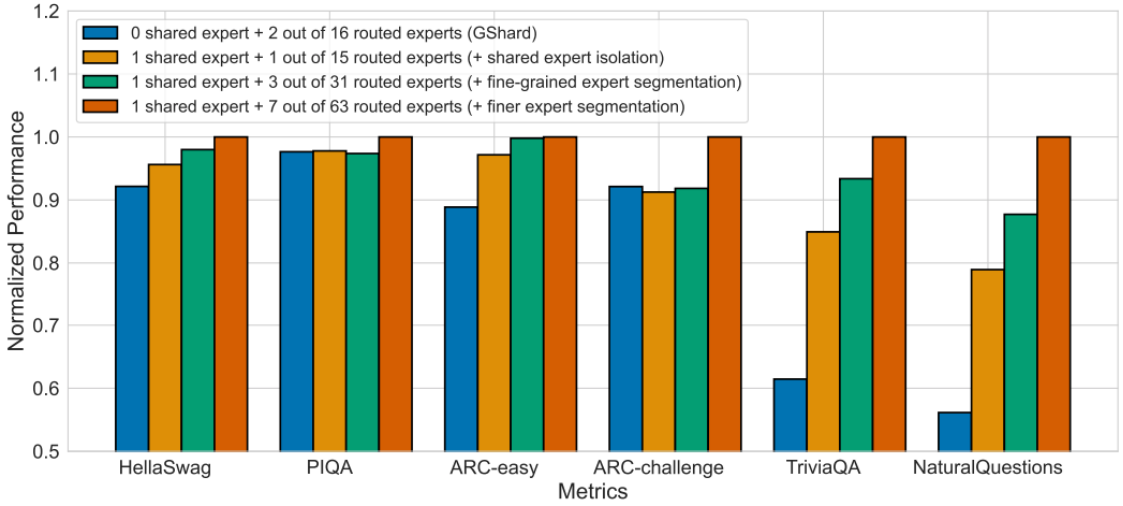


Figure 3 | Ablation studies for DeepSeekMoE. The performance is normalized by the best performance for clarity in presentation. All compared models have the same number of parameters and activated parameters. We can find that fine-grained expert segmentation and shared expert isolation both contribute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

More experts, shared experts all seem to generally help

How do we train MoEs?

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

But sparse gating decisions are not differentiable!

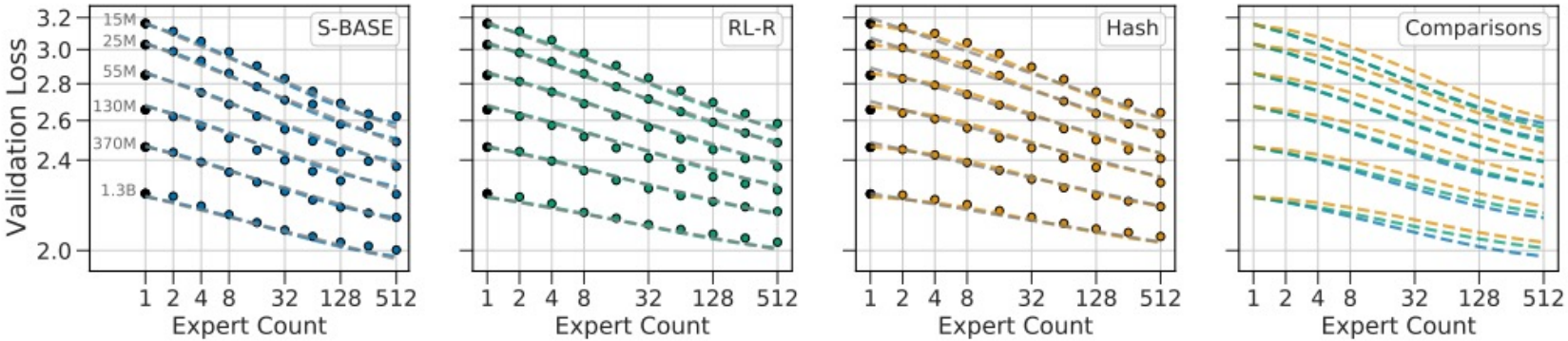
Solutions?

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

Guess which one people use in practice?

RL for MoEs

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

Per-expert balancing – same as the switch transformer

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

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

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

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

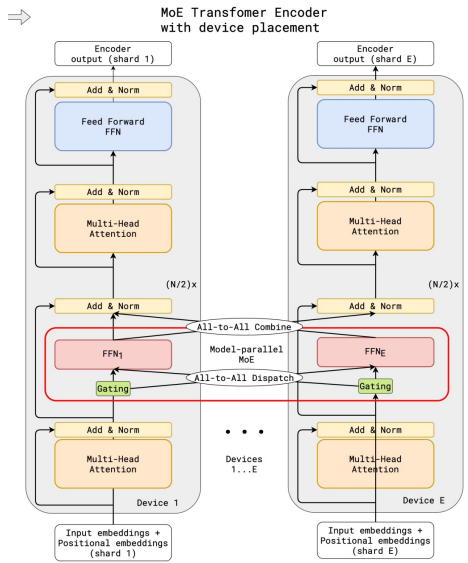
$$\mathcal{L}_{\text{DevBal}} = \alpha_2 \sum_{i=1}^{D} f'_i P'_i, \tag{15}$$

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

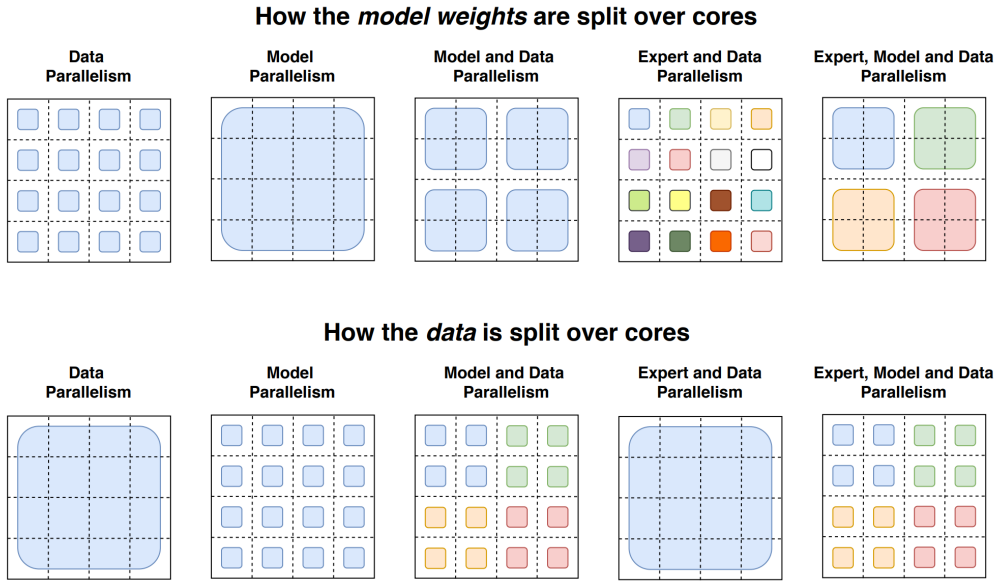
$$P'_i = \sum_{j \in \mathcal{E}_i} P_j, \tag{17}$$

Training MoEs – the systems side

MoEs parallelize nicely – Each FFN can fit in a device

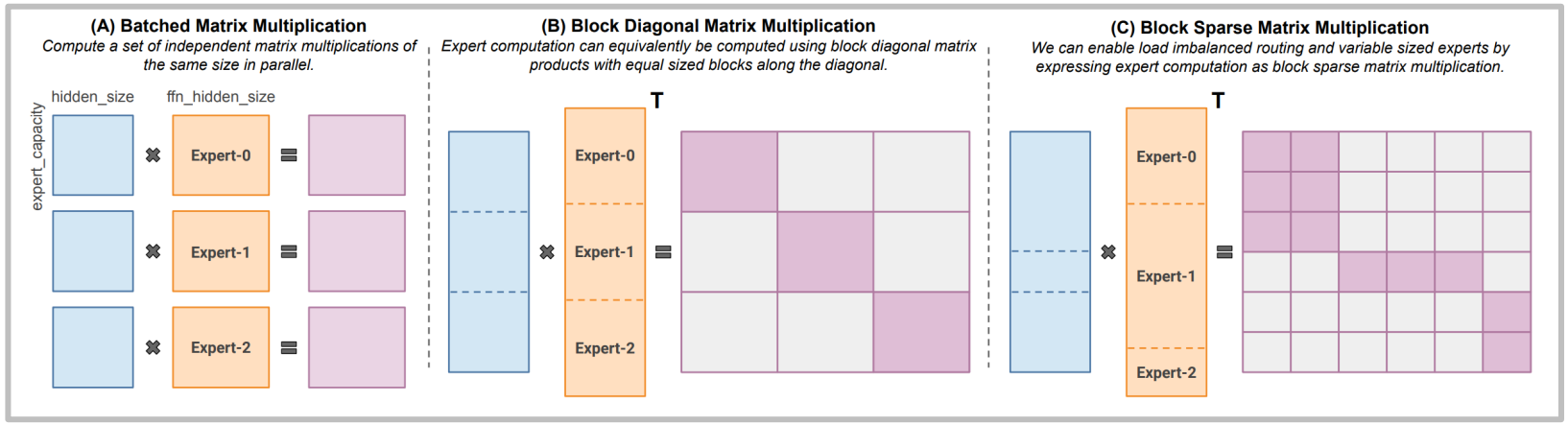


Enables additional kinds of parallelism



Training MoEs – the systems side

MoE routing allows for parallelism, but also some complexities

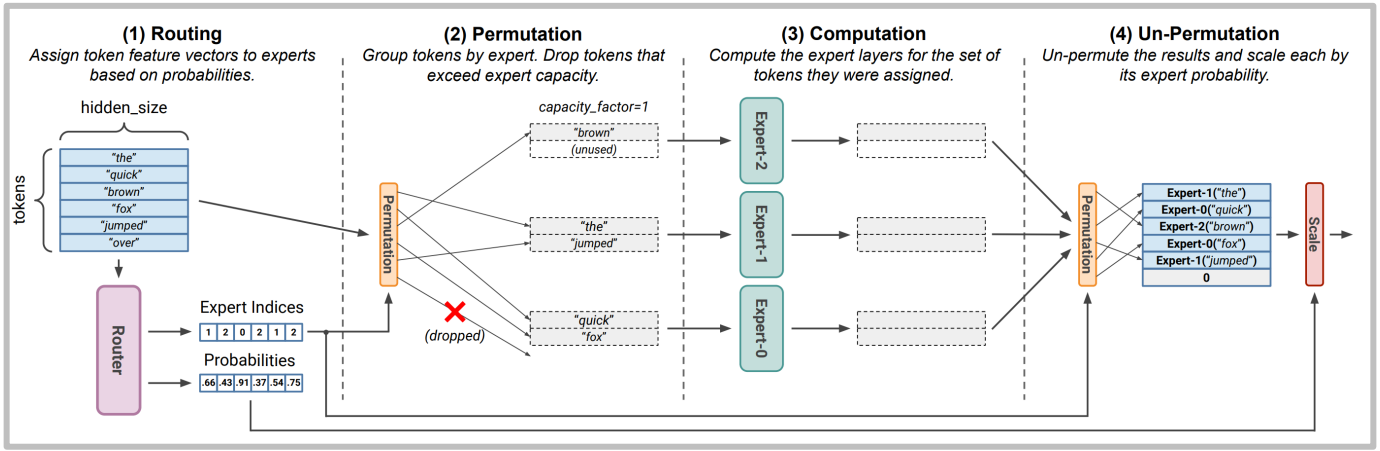


Modern libraries like MegaBlocks (used in many open MoEs) use smarter sparse MMs

Fun side issue – stochasticity of MoE models

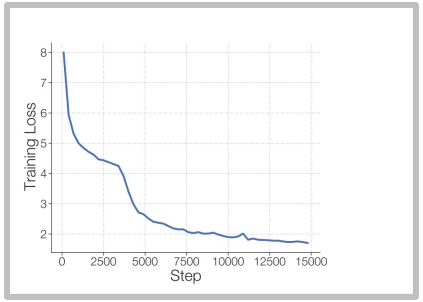
There was speculation that GPT-4's stochasticity was due to MoE..

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.

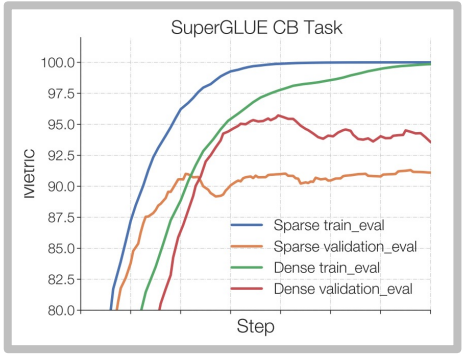
[Zoph 2022]

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

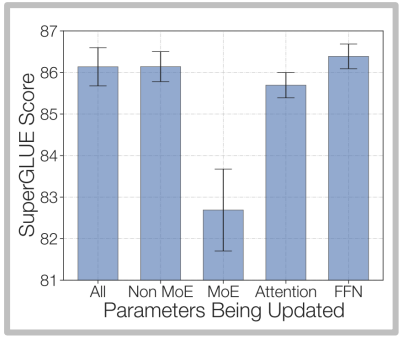
$$L_z(x) = \frac{1}{B}\sum_{i=1}^{B}\left(\log\sum_{j=1}^{N} e^{x_j^{(i)}}\right)^2 \tag{5}$$

Issues with MoEs – fine-tuning

Sparse MoEs can overfit on smaller fine-tuning data



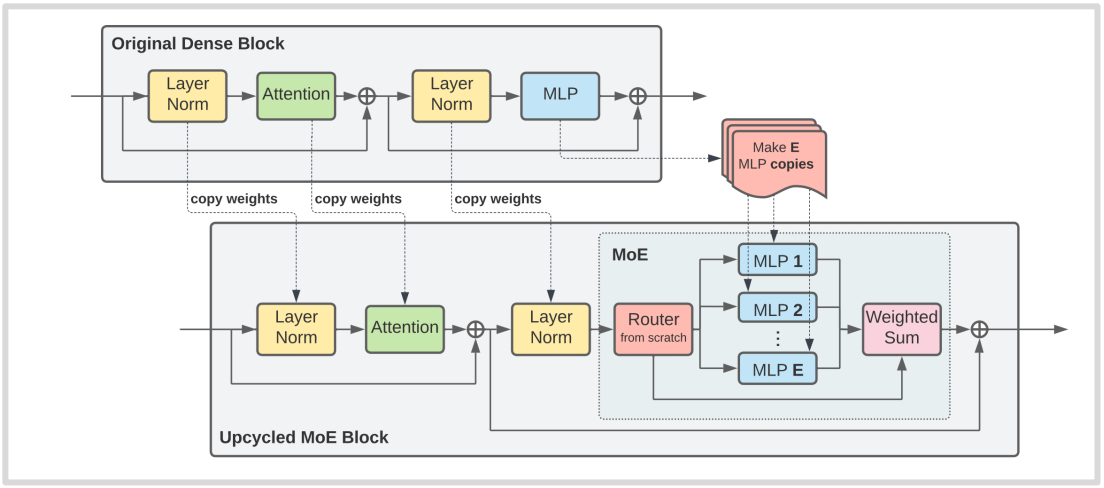
Zoph et al solution – finetune non-MoE MLPs

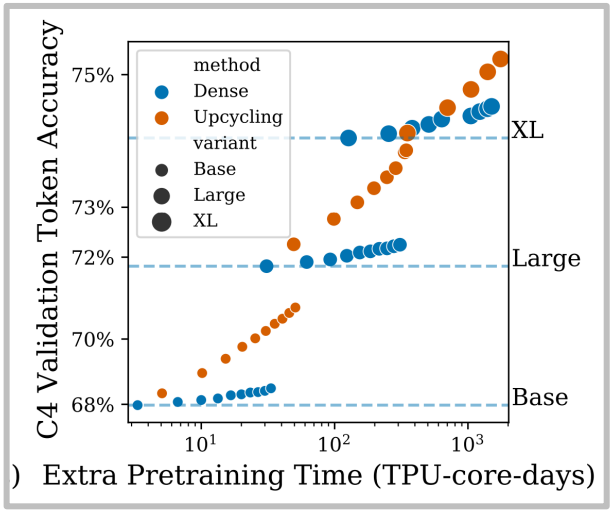


DeepSeek solution – use lots of data 1.4M SFT

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





Can we use a pre-trained LM to initialize a MoE?

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

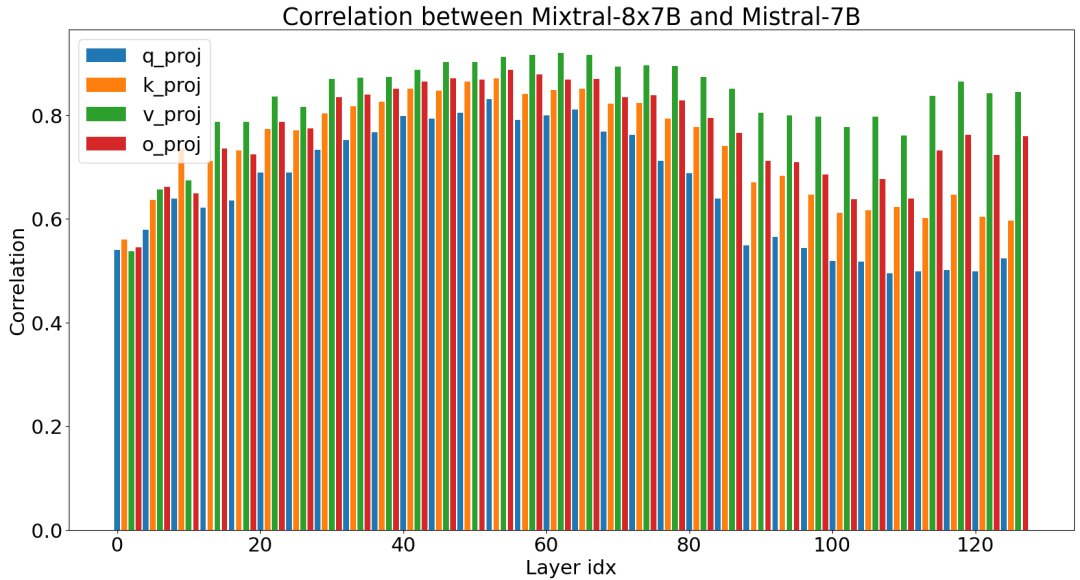
Qwen MoE – Initialized from the Qwen 1.8B model top-k=4, 60 experts w/ 4 shared.

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

Upcycling example (?) Mixtral

Some people think Mixtral may also be upcycled



… but since Mixtral is only open weights and not open training code, we don't really know..

[Source: https://twitter.com/tianle_cai/status/1734188749117153684]

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