Mixture of Experts

Robots in Disguise, 28th August 2024

Angus R. Williams
Senior Data Scientist
Public Policy, The Alan Turing Institute
arwilliams@turing.ac.uk

Overview

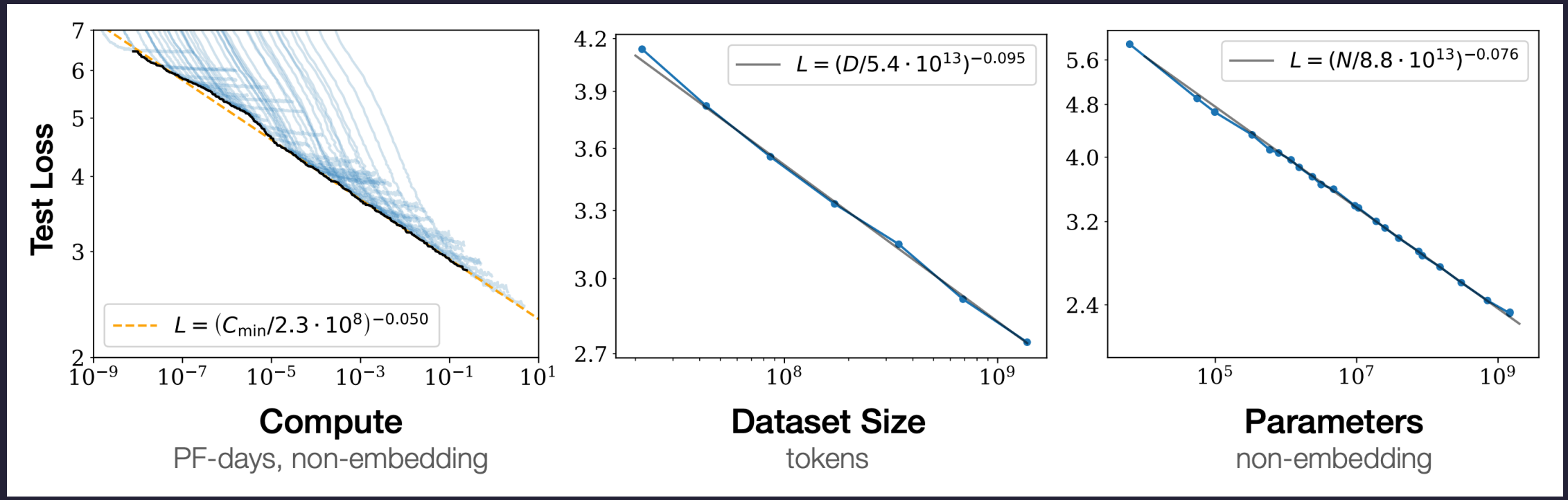
- Motivations for M.o.E
- How M.o.E works
- Challenges / Solutions
- Dynamics of experts

Motivation

- How do we scale models / train better models?
- **More parameters** (GPT3) vs. **more data** (Chinchilla) - *vs. better data (Llama 3)*
- Scaling parameters/data quantity → better models . . .
- . . . but scaling parameters/data quality → more compute

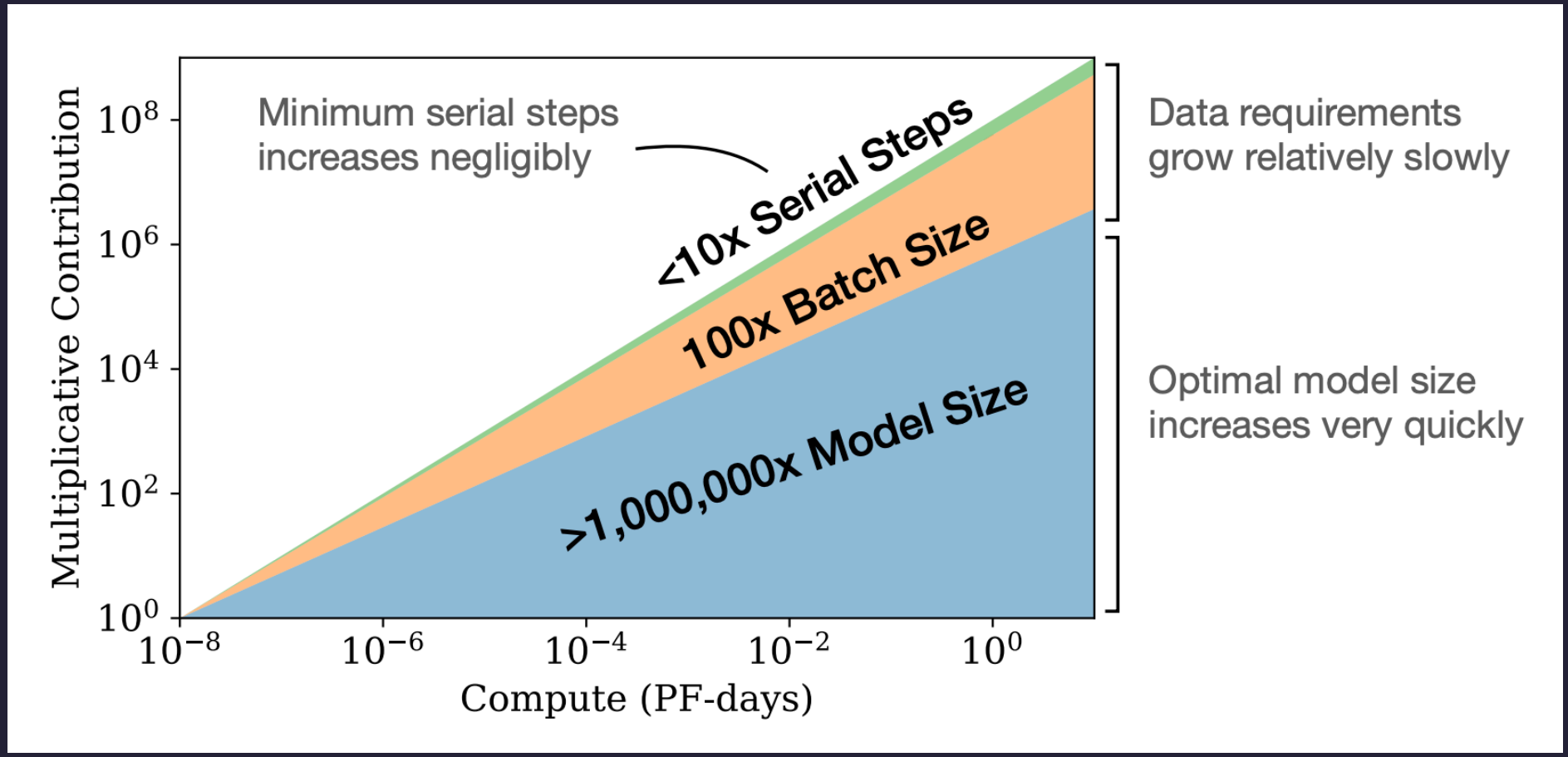
"Language Models are Few-Shot Learners", Brown et al., 2020,
"Training Compute-Optimal Large Language Models", Hoffman et al., 2022,
"The Llama 3 Herd of Models", Dubey et al., 2024

Scaling Laws



"Scaling Laws for Neural Language Models", Kaplan et al., 2020

Scaling Laws

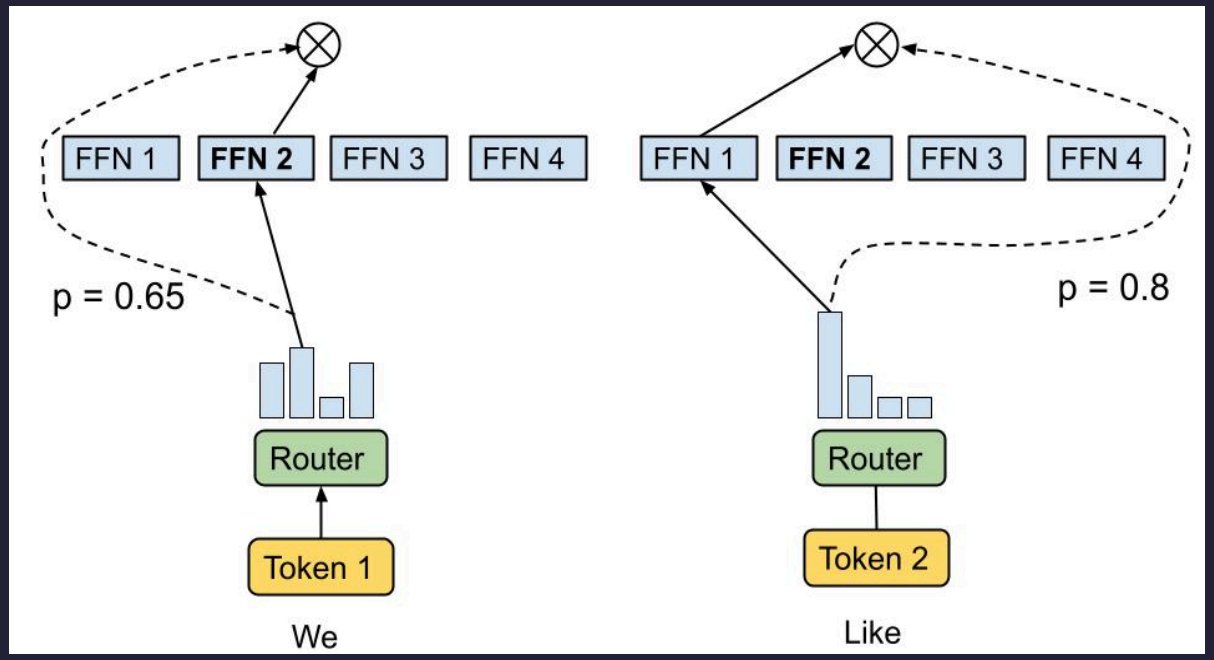


"Scaling Laws for Neural Language Models", Kaplan et al., 2020

How can we scale model size without scaling compute linearly?

Mixture of Experts

Replacing "dense" feed forward layers with multiple parallel "expert" networks, accompanied by a gating network or "router".



Mixture of Experts

- Each input (token) is routed to some number of "expert" subnetworks, outputs of chosen experts are combined.
- Gating layer / router is another set of trainable weights.
- Individual experts learn from different inputs.
- Experts deal with individual tokens, other layers (attention) handle sequence-level info.

Mixture of Experts

- Introducing **SPARSITY** - replacing dense FF layer with set of networks where only N components are active at any time.
- More trainable parameters + smaller number of active parameters.
- E.g. Mixtral 8 x 7B (i.e. a 56B* model) only ever requires 2 experts i.e. compute for 2x7=14B params (actually 12B because some parameters are shared)
- Decoupling computational cost and parameter count.

History

- First introduced in 1991 by Jacobs et al.
- MoEs as the whole model explored in SVMs, Gaussian Processes, etc.
- Applied as components of deep networks in 2013 by Eigen et al.
- Applied to LSTMs in 2017 by Shazeer et al., producing 137B parameter LSTM.
- Applied to Transformers in SwitchTransformers (Fedus et al., 2021) (Encoder-Decoder) and GLaM (Du et al, 2021) (Decoder only), producing >1T parameter models.
- GPT4 - 16 x 111B MoE model?
- Mixtral - open-source 8 x 7B MoE model (later released 8 x 22B)
- Gemini 1.5 - >1.5T param MoE model?

"Adaptive Mixtures of Local Experts", Jacobs et al., 1991; "Learning Factored Representations in a Deep Mixture of Experts", Eigen et al., 2013; "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer", Shazeer et al., 2017; "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity", Fedus et al., 2021; "GLaM: Efficient Scaling of Language Models with Mixture-of-Experts", Du et al., 2021; "Mixtral of Experts", Jiang et al., 2023

Fundamentals

- The output for a given input is the **sum of the outputs of the chosen experts (E), weighted by the value of the gating function (G) for each of the n experts.**

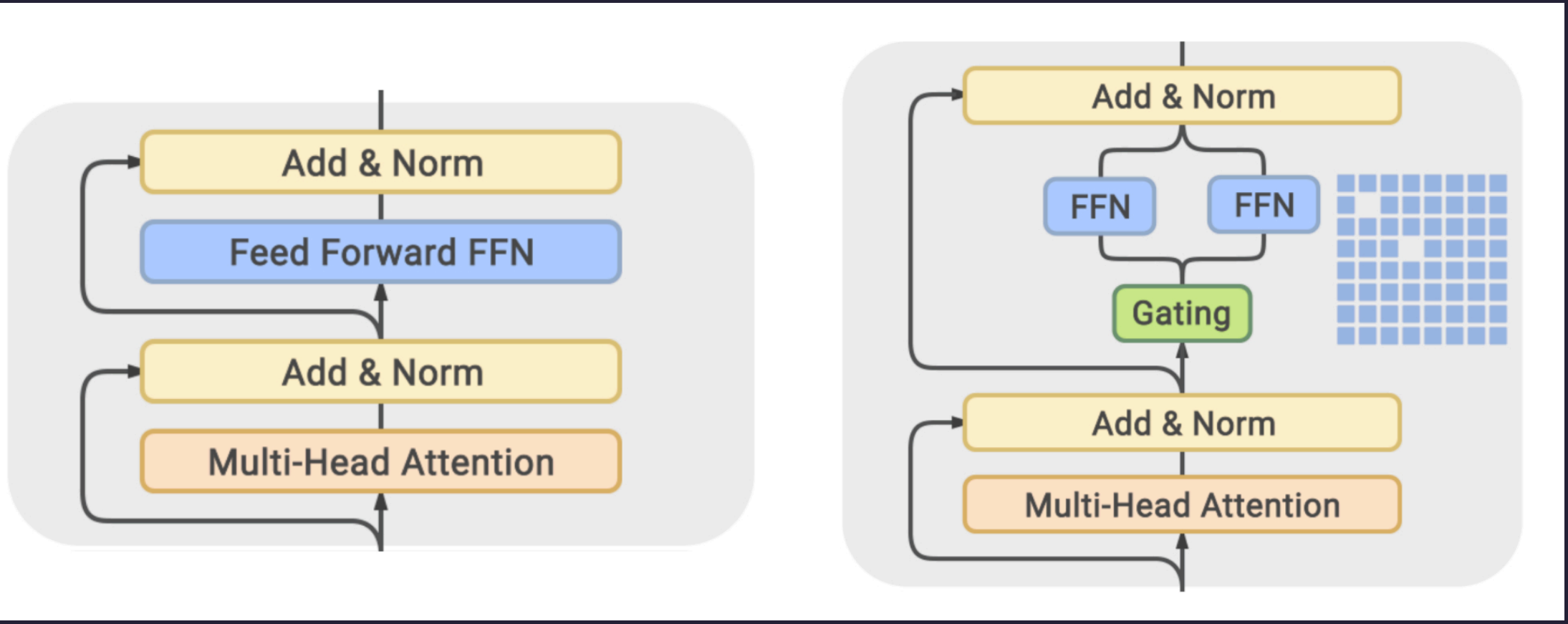
$$y = \sum_{i=1}^{n} G(x)_i E_i(x)$$

- In its most basic form, the gating function is a simple network that selects the top-K experts for a given token x, with a softmax function.

$$G(x) := Softmax(TopK(x \cdot W_g))$$

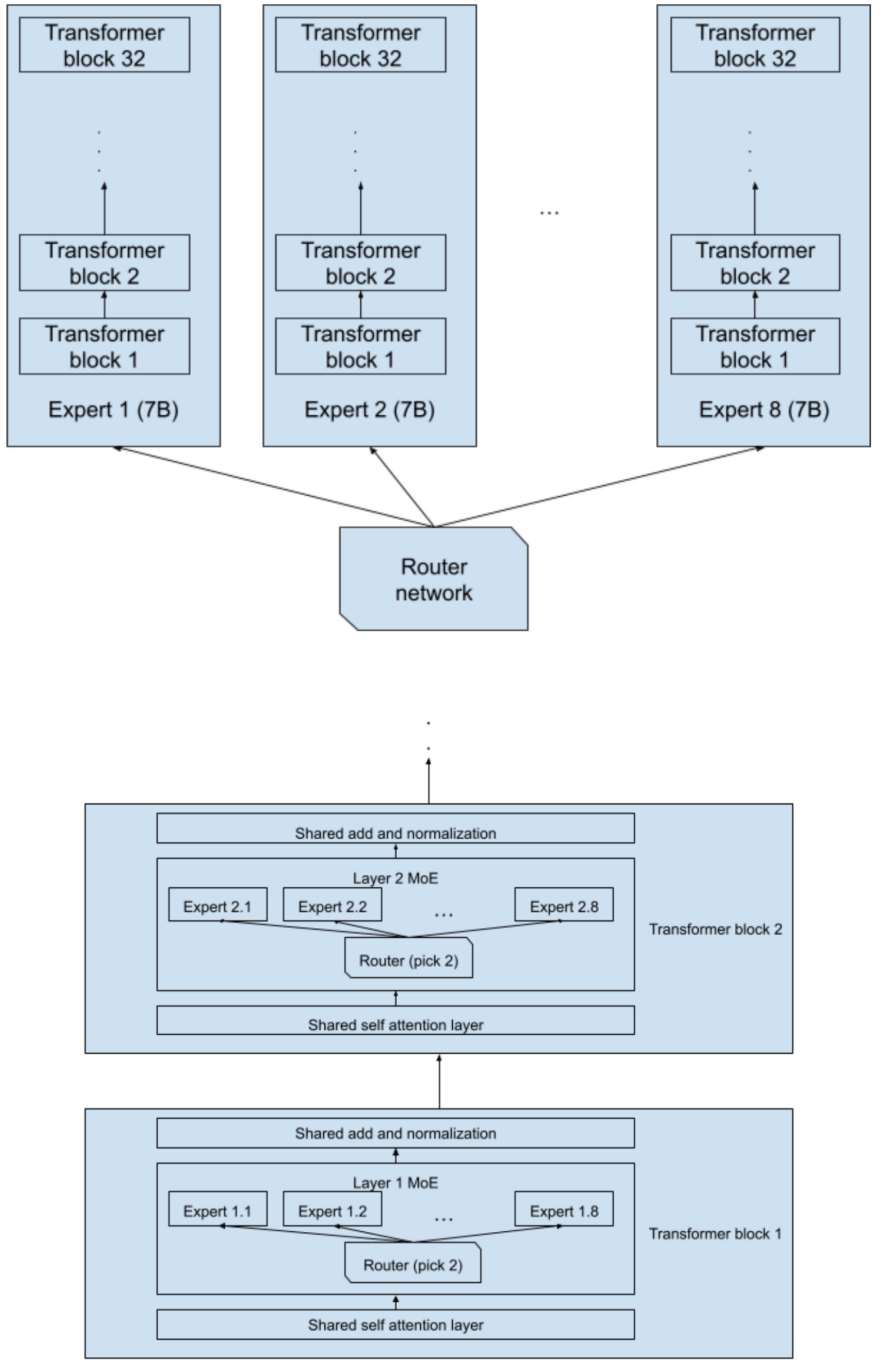
Transformer Architecture

- Replace FFN within transformer block with gating network and expert FFNs
- Some replace every transformer block (Mixtral), some every other (GLAM).



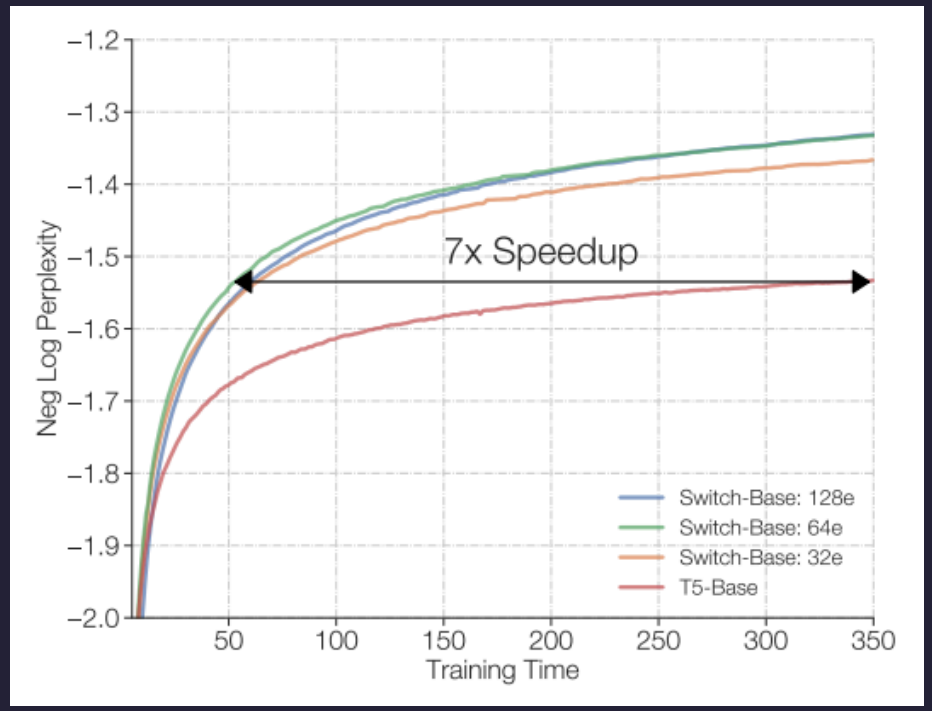
"Mixtral 8 x 7B"

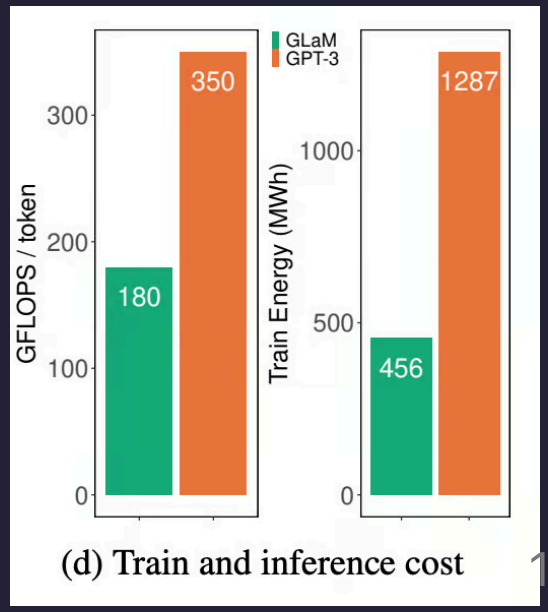
- **NOT 8 separate 7B models**
- "8 x 7B" refers to the fact that it's the same architecture as Mistral 7B but each FFN layer is replicated 8 times.
- 8 expert FFNs in each of 32 transformer blocks, i.e. 256 individual FFNs.
- 2 experts active per layer $\rightarrow$ 28 possible combinations per layer $\rightarrow$ 32 layers $\rightarrow$ effectively $2 \cdot 10^{46}$ unique models
- Token representations evolve throughout layers.



Impact

- Sparsity, conditional computation
- Well suited for distributed computing
- 7 x speedup in pre-training vs. dense model (SwitchTransformers)
- GLaM outperforms GPT-3 on 6/7 benchmarks on 1/3 of the compute budget
- GPT4: ~280B params and ~560 TFLOPs in MoE architecture vs. ~1.8T params and ~3,700 TFLOP for dense equivalent
- Gemini 1 vs 1.5 (1.5 pro ~ 1.0 ultra, with less compute, 1.5 pro outperfoms 1.5 pro)
- Distilling from MoE models retains 30-40% of sparsity gains





"Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity", Fedus et al., 2021
"GLaM: Efficient Scaling of Language Models with Mixture-of-Experts", Du et al., 2021

Challenges

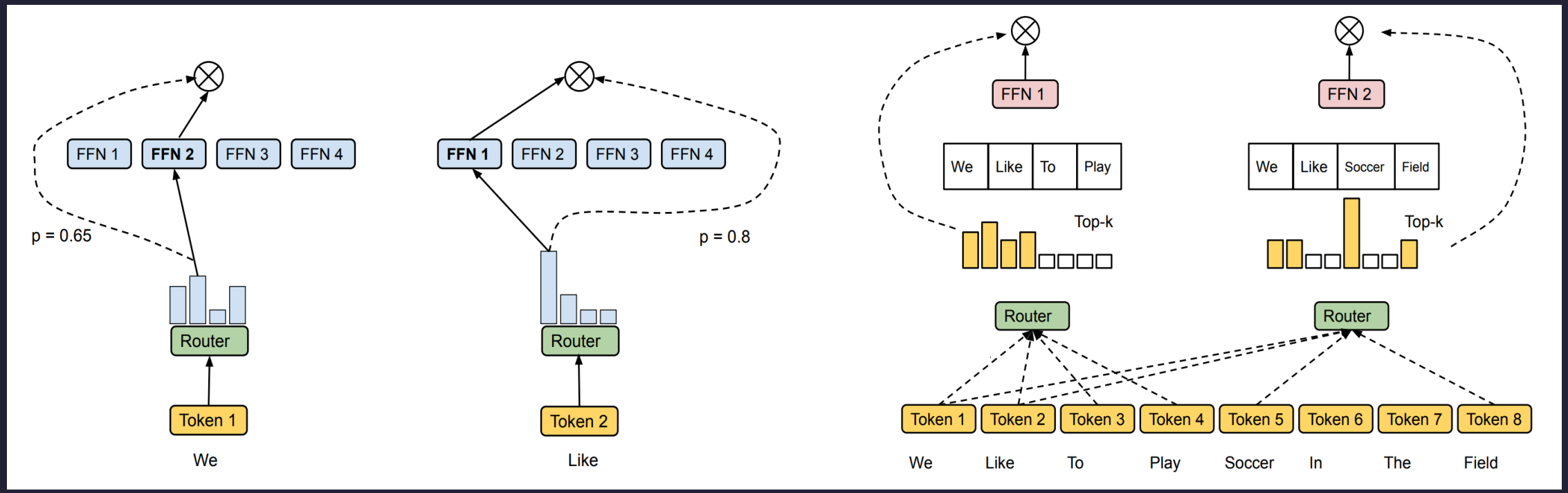
- (Still need to lead the whole model into memory)
- Load balancing
- Instability
- Overfitting
- Fine-tuning

Routing

Load Balancing: How do we stop the gating network from routing all tokens to the same expert(s)?

- **Expert Capacity**: threshold of how many tokens each expert can process. Any further tokens are sent to next layer / dropped
  - Sparse models are robust to dropping tokens: Dropping 10-15% ~ dropping < 1%
- **Auxiliary Loss**: loss added to encourage groups of tokens to evenly distribute across experts. Can contribute to instability
- **Z-Loss**: Penalise large logits entering gating network. Improve stability with no quality degradation
- **Expert Choice Routing**: Let experts pick top-k tokens. Enables tokens to be routed to **variable** number of experts

Routing

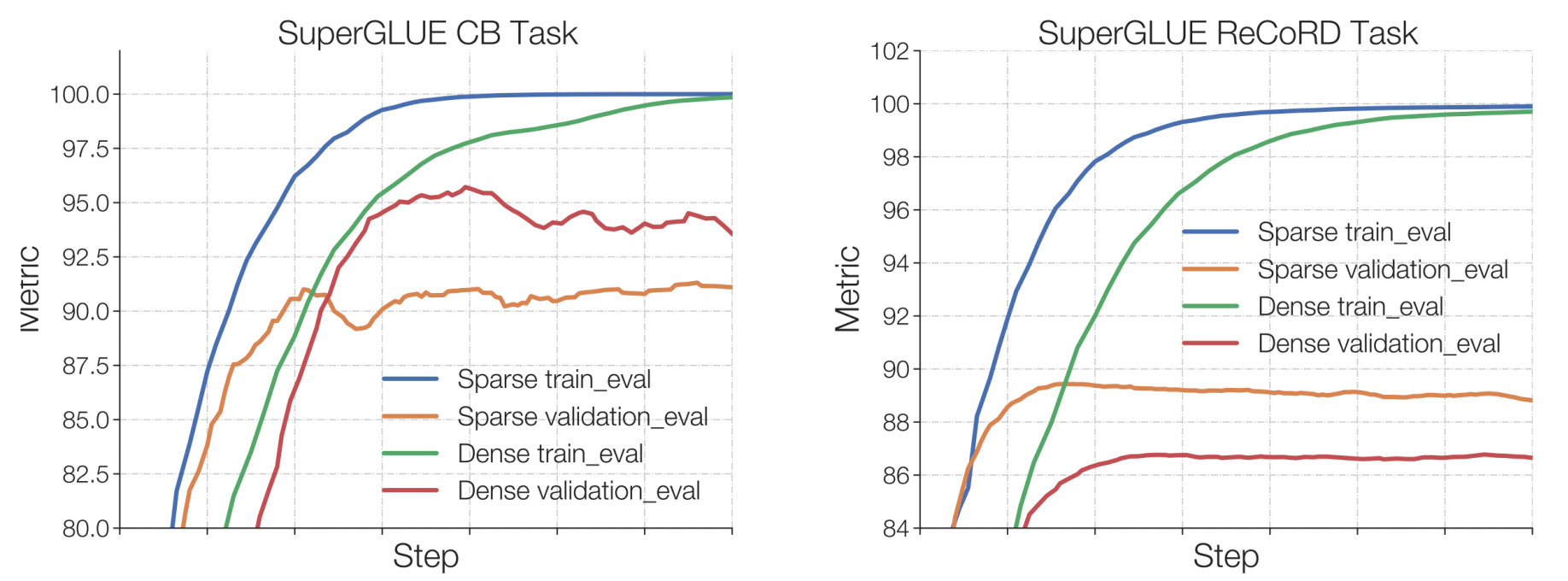

FFN 1
FFN 2
FFN 3
FFN 4
p = 0.65
Router
Token 1
We
FFN 1
FFN 2
FFN 3
FFN 4
p = 0.8
Router
Token 2
Like
FFN 1
We
Like
To
Play
Top-k
Router
Token 1
Token 2
Token 3
Token 4
Token 5
Token 6
Token 7
Token 8
We
Like
To
Play
Soccer
In
The
Field
FFN 2
We
Like
Soccer
Field
Top-k
Router

Training & Fine-tuning

- **Regularisation**: MoEs benefit from higher dropout in expert layers, lower dropout in shared layers.
- **Hyperparameter turing**: MoEs benefits from smaller batch sizes and higher learning rates $\rightarrow$ higher noise, better generalisation

Training & Fine-tuning

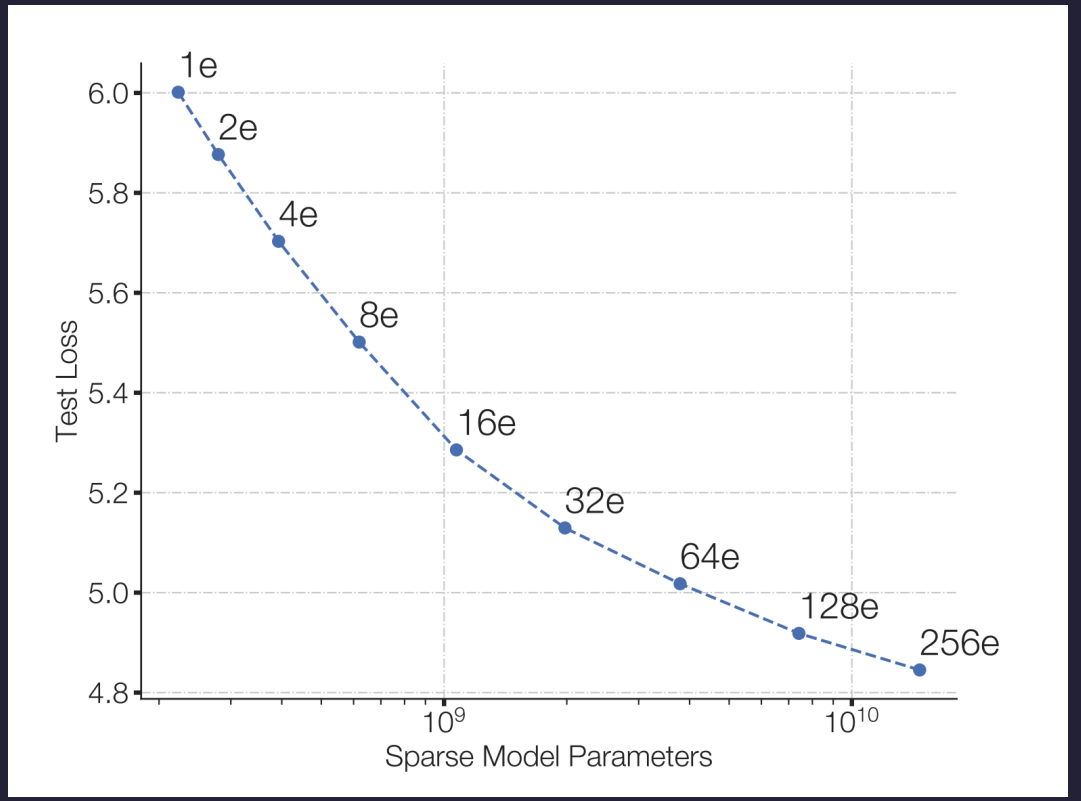
- MoEs learn more quickly than dense models, but also overfit more easily on smaller tasks / are "slower" at transferring knowledge.
- MoEs are generally **better at knowledge-heavy tasks** than dense models, worse at reasoning-heavy tasks.



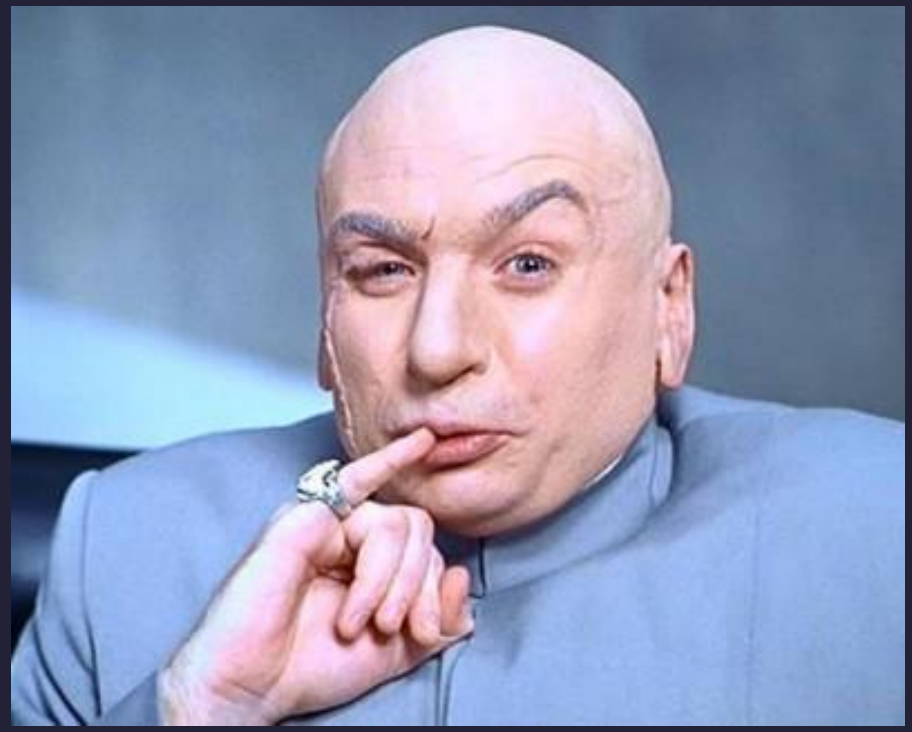
"ST-MoE: Designing Stable and Transferable Sparse Expert Models", Zoph et al., 2022

How many experts?

- Switch-C - 2,048/1
- GLaM - 64/2
- Mixtral - 8/2
- GPT4 - 16/2?
- Higher numbers of experts works better with hierarchical MoEs
- Diminishing returns after 256/512
- Properties of high number of experts consistent at smaller scale (2,4,8 experts)



"Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity", Fedus et al., 2021

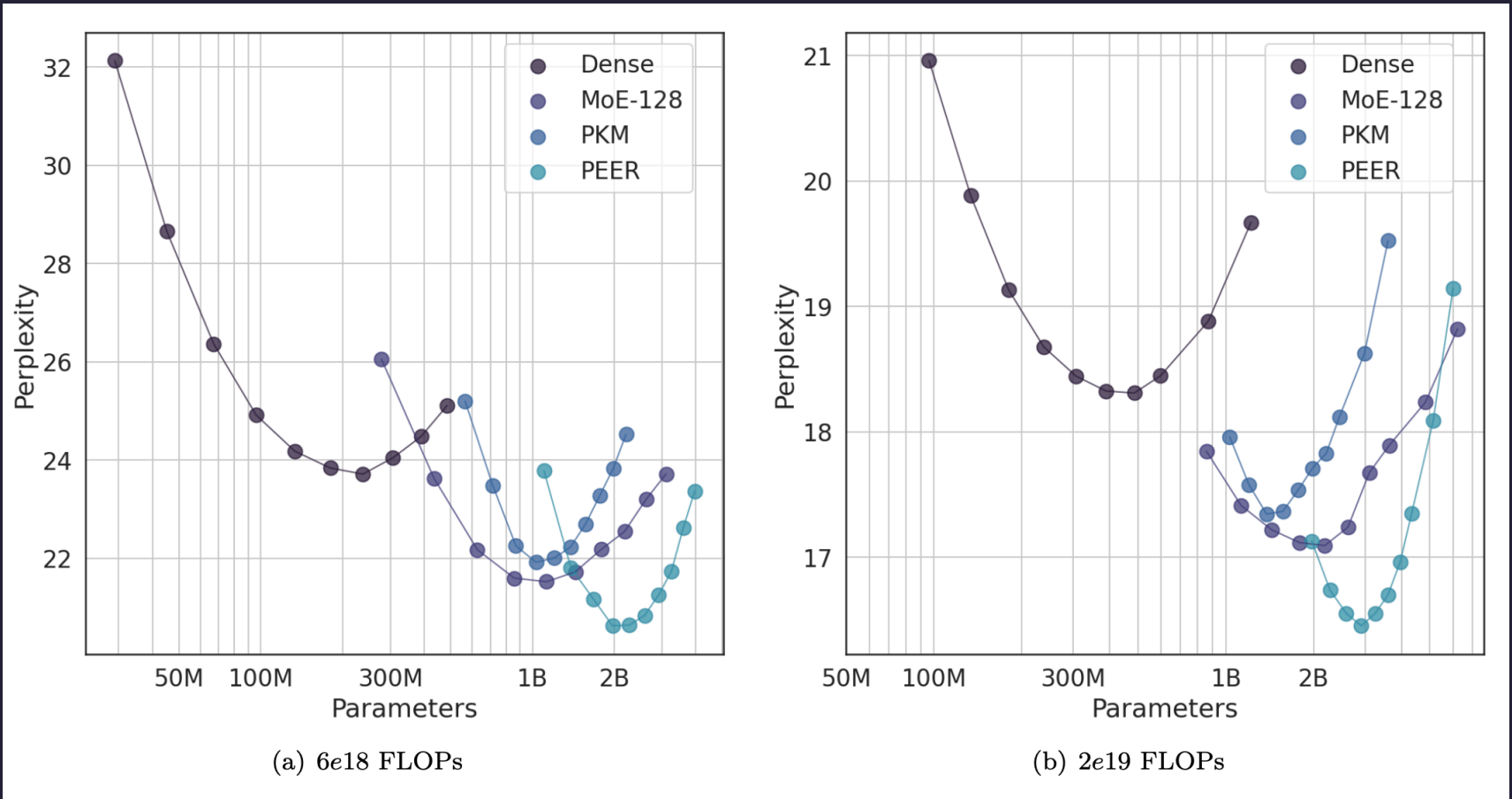


1 million experts

- We want to increase total number of trainable parameters but without increasing number of active parameters - so why not more, smaller experts?
- 1 million single neuron MLPs as experts
- Use multi-head retrieval to *"dynamically assembles an MLP with N neurons by aggregating N singleton MLPs retrieved from a shared repository"*
- Outperforms dense models and coarse MoE models on the same compute budget

"Mixture of A Million Experts", He.X, 2024

1 million experts

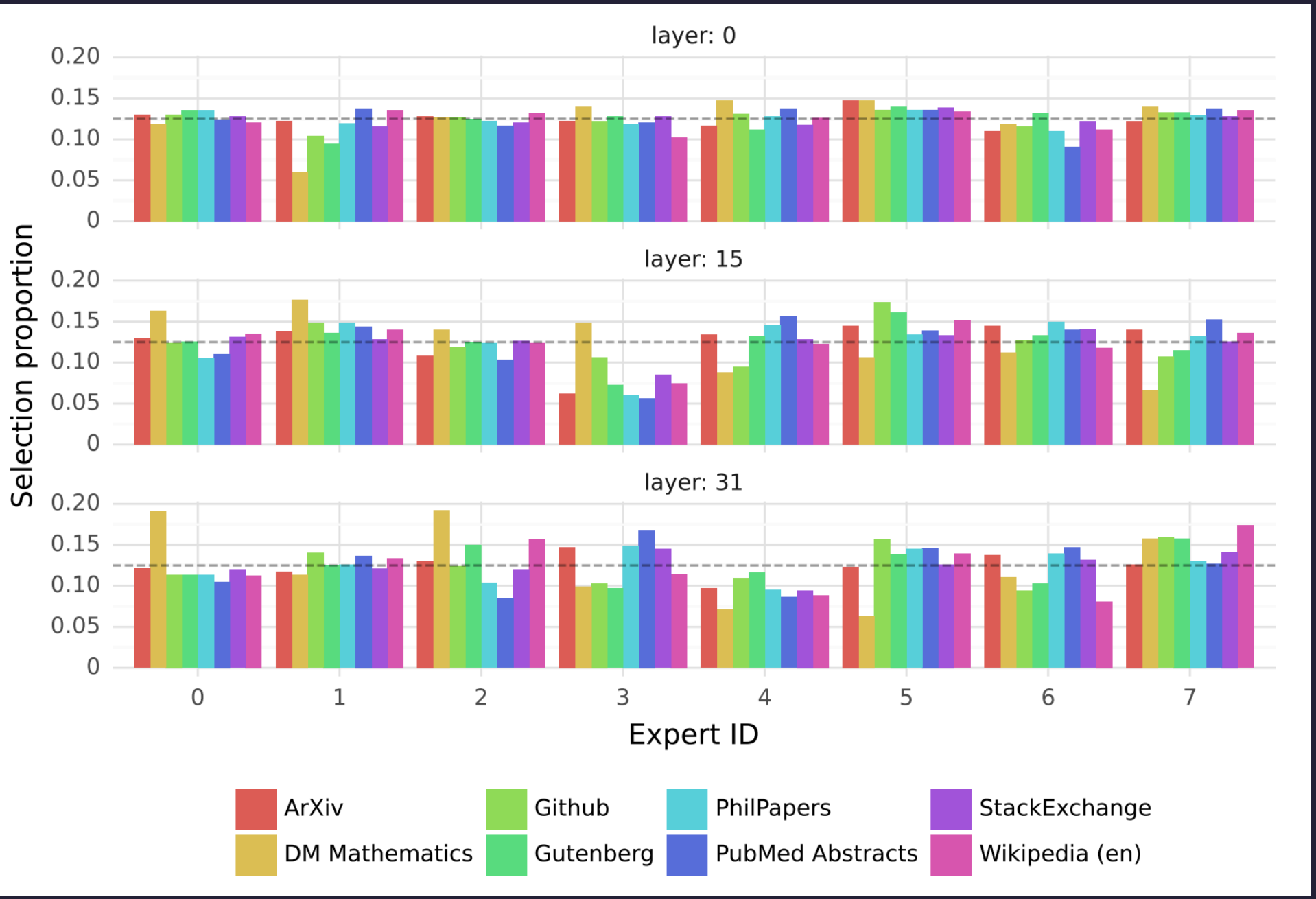


"Mixture of A Million Experts", He.X, 2024

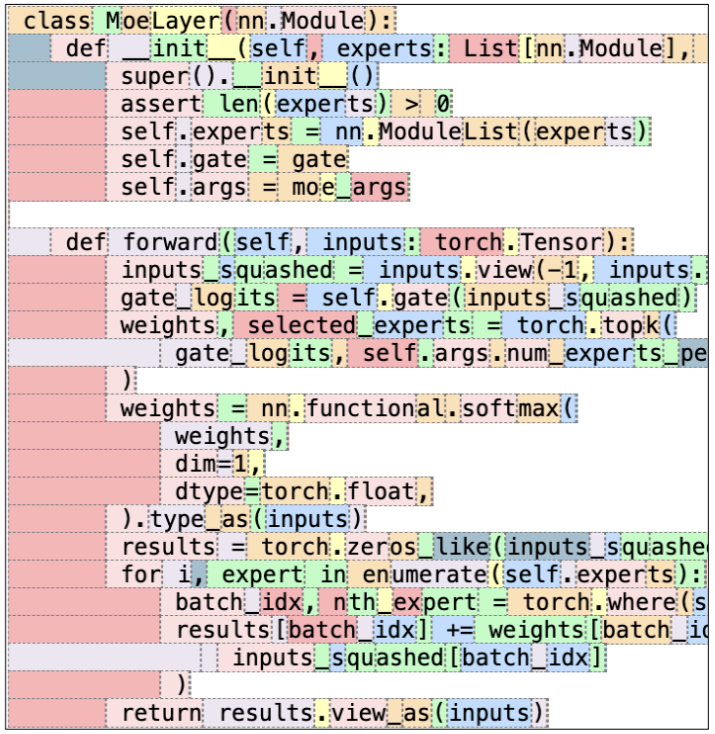
What are "experts" experts in?

- "Experts" conjures ideas of high-level specialisation
- The reality is **shallow** specialisation - token routing, load balancing prevents hi-level specialisation
- Expert selection more closely aligned with syntax than domain
- Levels of specialisation tends to differ across layers
- Consecutive tokens often assigned to the same expert - router is learning something about sequence information?

Expert specialization	Expert position	Routed tokens
Sentinel tokens	Layer 1	been <extra_id_4><extra_id_7>floral to <extra_id_10><extra_id_12><extra_id_15> <extra_id_17><extra_id_18><extra_id_19>...
	Layer 4	<extra_id_0><extra_id_1><extra_id_2> <extra_id_4><extra_id_6><extra_id_7> <extra_id_12><extra_id_13><extra_id_14>...
	Layer 6	<extra_id_0><extra_id_4><extra_id_5> <extra_id_6><extra_id_7><extra_id_14> <extra_id_16><extra_id_17><extra_id_18>...
Punctuation	Layer 2	, , , , , , , , , - , , , , ,).)
	Layer 6	, , , , , : . : , & , & & ? & - , , ? , , , . <extra_id_27>
Conjunctions and articles	Layer 3	The the the the the the the the the The the the the the the The the the the
	Layer 6	a and and and and and and and or and a and . the the if ? a designed does been is not
Verbs	Layer 1	died falling identified fell closed left posted lost felt left said read miss place struggling falling signed died falling designed based disagree submitted develop
Visual descriptions *color, spatial position*	Layer 0	her over her know dark upper dark outer center upper blue inner yellow raw mama bright bright over open your dark blue
Proper names	Layer 1	A Mart Gr Mart Kent Med Cor Tri Ca Mart R Mart Lorraine Colin Ken Sam Ken Gr Angel A Dou Now Ga GT Q Ga C Ko C Ko Ga G
Counting and numbers *written and numerical forms*	Layer 1	after 37 19. 6. 27 I I Seven 25 4, 54 I two dead we Some 2012 who we few lower each


layer: 0
layer: 15
layer: 31
Selection proportion
0.20
0.15
0.10
0.05
0
0
1
2
3
4
5
6
7
Expert ID
ArXiv
DM Mathematics
Github
Gutenberg
PhilPapers
PubMed Abstracts
StackExchange
Wikipedia (en)

Layer 0

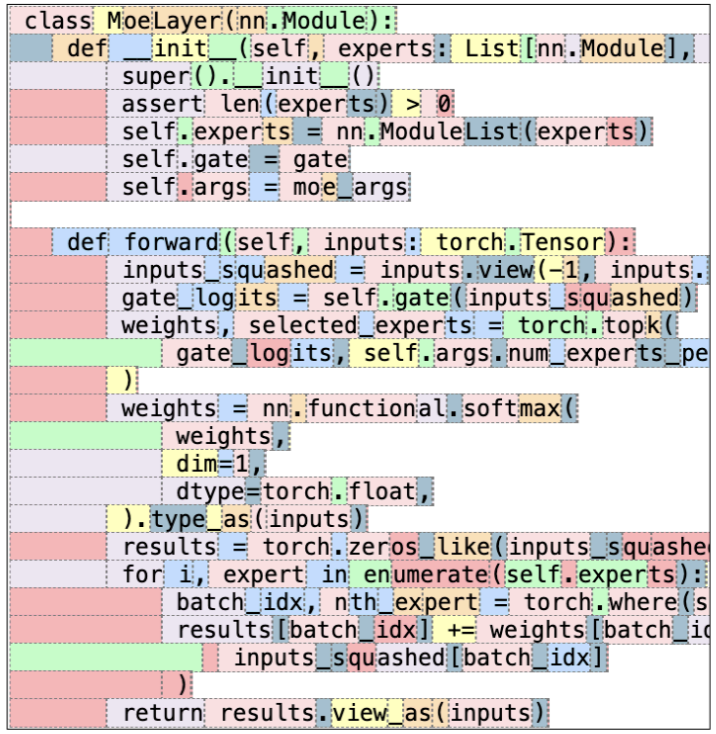


```
class MoeLayer(nn.Module):
    def __init__(self, experts: List[nn.Module],
        super().__init__()
        assert len(experts) > 0
        self.experts = nn.ModuleList(experts)
        self.gate = gate
        self.args = moe_args

    def forward(self, inputs: torch.Tensor):
        inputs_squashed = inputs.view(-1, inputs.
        gate_logits = self.gate(inputs_squashed)
        weights, selected_experts = torch.topk(
            gate_logits, self.args.num_experts_pe
        )
        weights = nn.functional.softmax(
            weights,
            dim=1,
            dtype=torch.float,
        ).type_as(inputs)
        results = torch.zeros_like(inputs_squashe
        for i, expert in enumerate(self.experts):
            batch_idx, nth_expert = torch.where(s
            results[batch_idx] += weights[batch_id
                inputs_squashed[batch_idx]
            )
        return results.view_as(inputs)
```

```
Question: Solve -42*r + 27*c = -1167 and 130*r
Answer: 4

Question: Calculate -841880142.544 + 411127.
Answer: -841469015.544

Question: Let x(g) = 9*g + 1. Let q(c) = 2*c +
Answer: 54*a - 30
```

```
A model airplane flies slower when flying into th
wind and faster with wind at its back. When launch
right angles to the wind, a cross wind, its ground
compared with flying in still air is
(A) the same (B) greater (C) less (D) either grea
or less depending on wind speed
```

Layer 15



```
class MoeLayer(nn.Module):
    def __init__(self, experts: List[nn.Module],
        super().__init__()
        assert len(experts) > 0
        self.experts = nn.ModuleList(experts)
        self.gate = gate
        self.args = moe_args

    def forward(self, inputs: torch.Tensor):
        inputs_squashed = inputs.view(-1, inputs.
        gate_logits = self.gate(inputs_squashed)
        weights, selected_experts = torch.topk(
            gate_logits, self.args.num_experts_pe
        )
        weights = nn.functional.softmax(
            weights,
            dim=1,
            dtype=torch.float,
        ).type_as(inputs)
        results = torch.zeros_like(inputs_squashe
        for i, expert in enumerate(self.experts):
            batch_idx, nth_expert = torch.where(s
            results[batch_idx] += weights[batch_id
                inputs_squashed[batch_idx]
            )
        return results.view_as(inputs)
```

```
Question: Solve -42*r + 27*c = -1167 and 130*r
Answer: 4

Question: Calculate -841880142.544 + 411127.
Answer: -841469015.544

Question: Let x(g) = 9*g + 1. Let q(c) = 2*c +
Answer: 54*a - 30
```

```
A model airplane flies slower when flying into th
wind and faster with wind at its back. When launch
right angles to the wind, a cross wind, its ground
compared with flying in still air is
(A) the same (B) greater (C) less (D) either grea
or less depending on wind speed
```

Layer 31

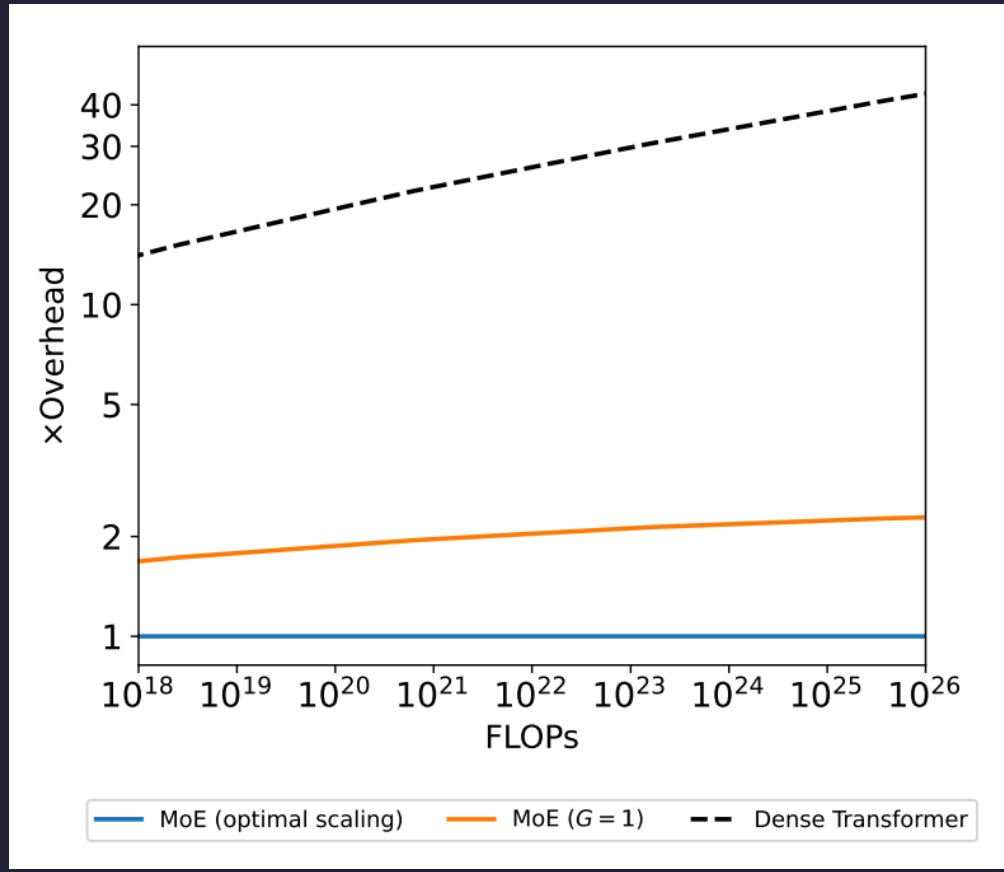
```
class MoeLayer(nn.Module):
    def __init__(self, experts: List[nn.Module],
        super().__init__()
        assert len(experts) > 0
        self.experts = nn.ModuleList(experts)
        self.gate = gate
        self.args = moe_args

    def forward(self, inputs: torch.Tensor):
        inputs_squashed = inputs.view(-1, inputs.
        gate_logits = self.gate(inputs_squashed)
        weights, selected_experts = torch.topk(
            gate_logits, self.args.num_experts_pe
        )
        weights = nn.functional.softmax(
            weights,
            dim=1,
            dtype=torch.float,
        ).type_as(inputs)
        results = torch.zeros_like(inputs_squashe
        for i, expert in enumerate(self.experts):
            batch_idx, nth_expert = torch.where(s
            results[batch_idx] += weights[batch_id
                inputs_squashed[batch_idx]
            )
        return results.view_as(inputs)
```

```
Question: Solve -42*r + 27*c = -1167 and 130*r
Answer: 4

Question: Calculate -841880142.544 + 411127.
Answer: -841469015.544

Question: Let x(g) = 9*g + 1. Let q(c) = 2*c +
Answer: 54*a - 30
```

```
A model airplane flies slower when flying into th
wind and faster with wind at its back. When launch
right angles to the wind, a cross wind, its ground
compared with flying in still air is
(A) the same (B) greater (C) less (D) either grea
or less depending on wind speed
```

Scaling Laws

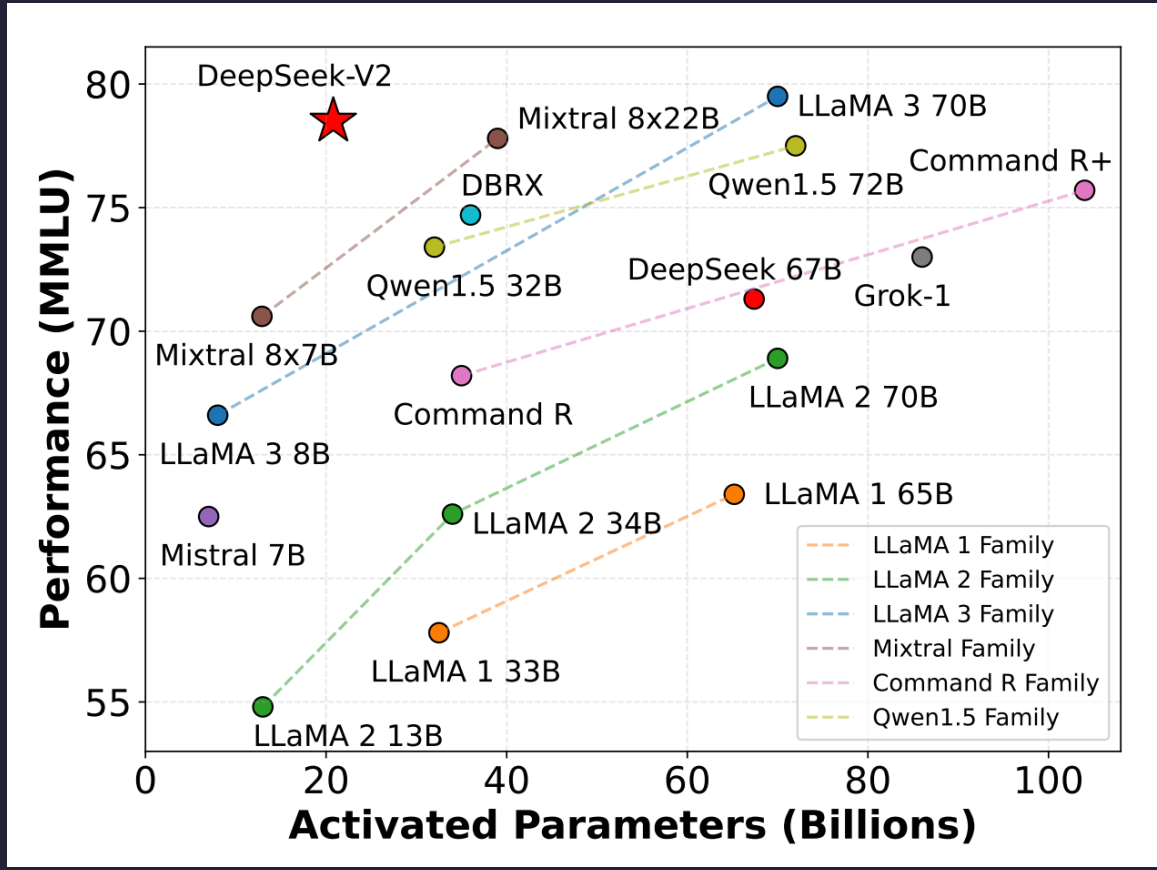
- MoEs are **always** more efficient than dense models
- Efficiency gap between dense and MoE models widens as we scale up the model size and training budget
- Performance declines when number of parameters in the routing mechanism **exceeds active parameters in actual experts**



"Scaling Laws for Fine-Grained Mixture-of-Experts Models", Krajewski et al., 2024

Looking forward

- **Lifelong learning** - freezing / adding new experts to deal with distributional shift over time while preserving existing knowledge
- Disentangling terminology: Mixture of Experts vs. Ensembles/Modular LLMs
- Dense models still relevant (Llama 3)
- There will always be another way to make gains - data quantity, data quality, compute efficiency



Thank you!

- Huggingface MoE Blog
- Nvidia MoE Blog
- Applying MoE in PyTorch
- Scaling Laws