ECE7115 ~~Multimodal VLM~~ LLM

5. Modern LLM Architecture - Mixture-of-Experts

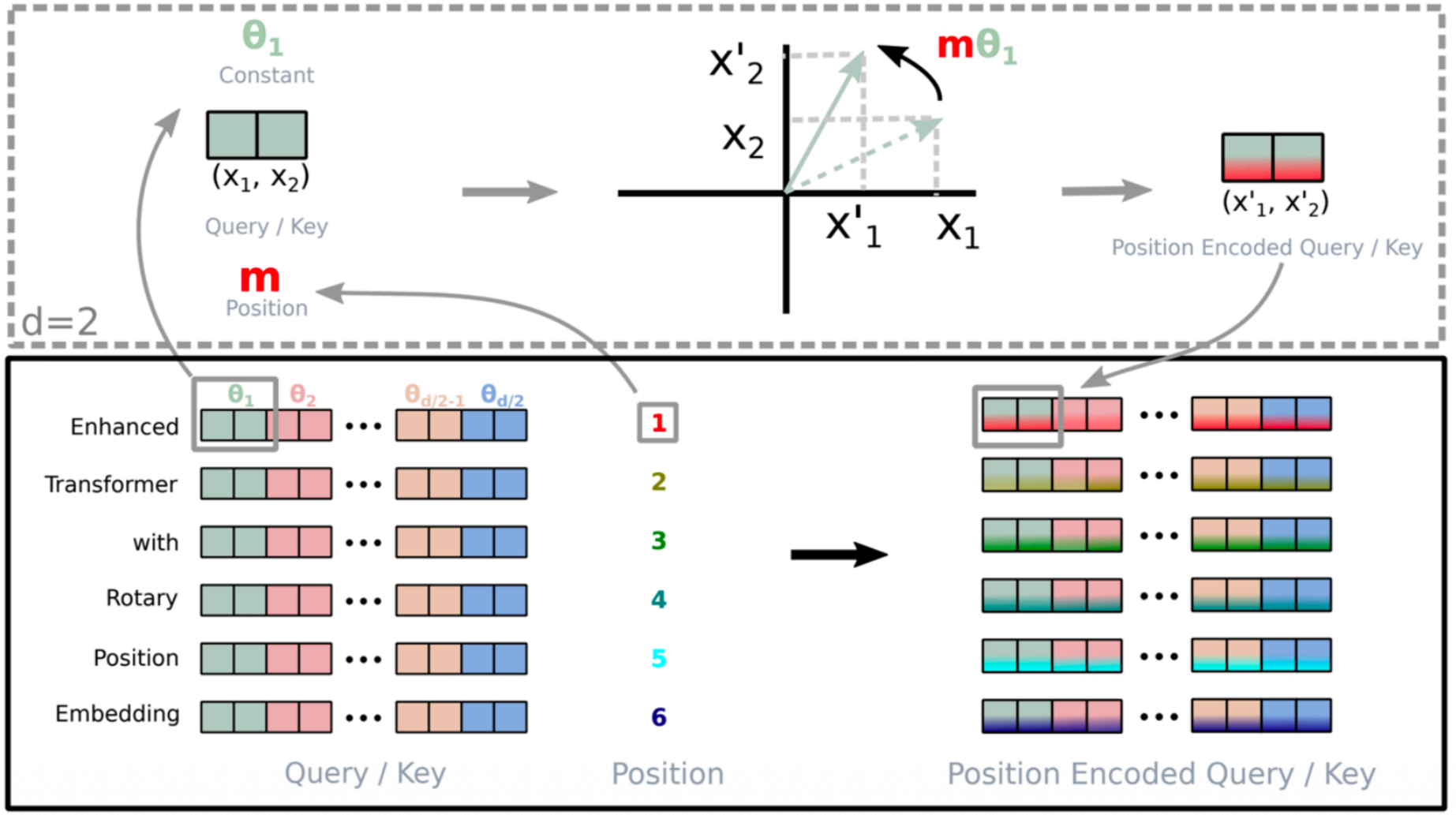
Spring 2026

Namhyuk Ahn, Inha University

Last Week: LLaMA-style Models

- Pre-vs-post norm
  - Everyone does pre-norm
- Layer vs RMSnorm
  - RMSnorm has clear compute wins, sometimes even performance
- Gating
  - GLUs seem generally better, though differences are small
- Serial vs parallel layers
  - No extremely serious ablations

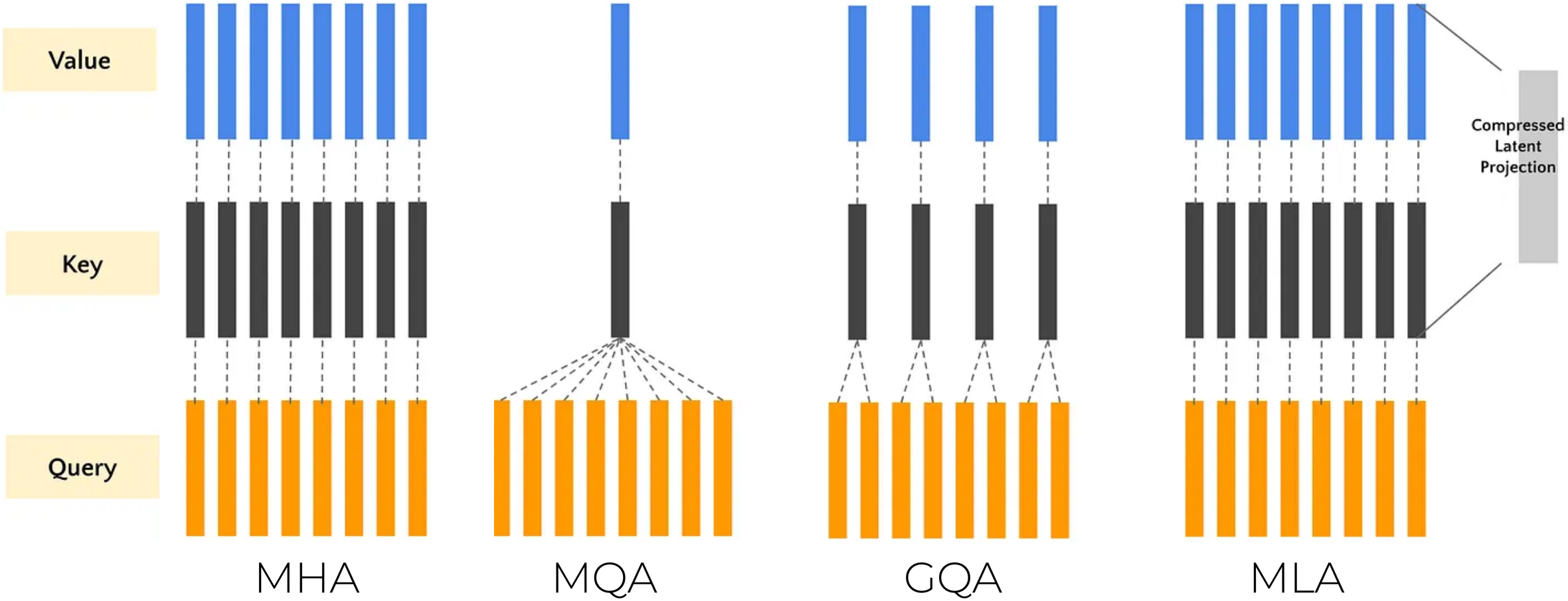
Last Week: RoPE



Last Week: Hyperparameters

- FFN Size: Factor-of-4 rule of thumb (8/3 for GLUs) is standard
- Head Dim: `d_head * num_head = d_model` is standard
- Aspect Ratio: Wide range of 'good' values (around 100-200)
- Regularization: You still 'regularize' LMs but its effects are primarily on optimization dynamics

Last Week: Reducing KV-Cache

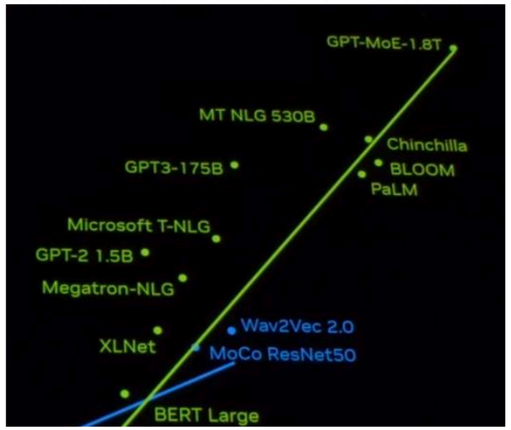


Lecture Overview

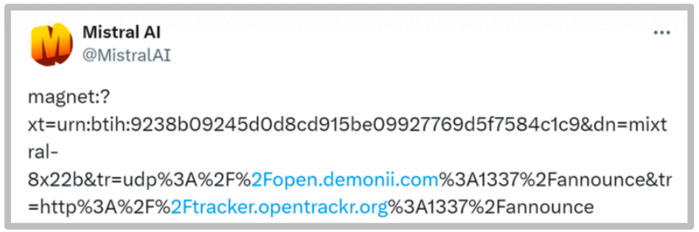
- Introduction to Mixture-of-Experts
- Deep Dives into Mixture-of-Experts

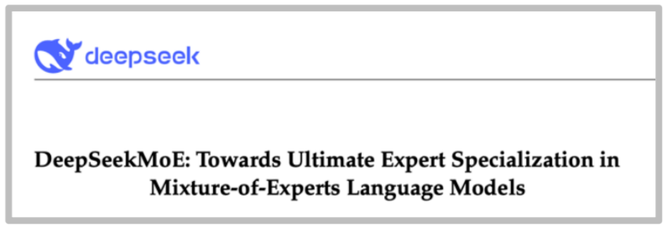
Mixture-of-Experts

- A new architectural trend since late 2024 and early 2025



GPT4 (?)

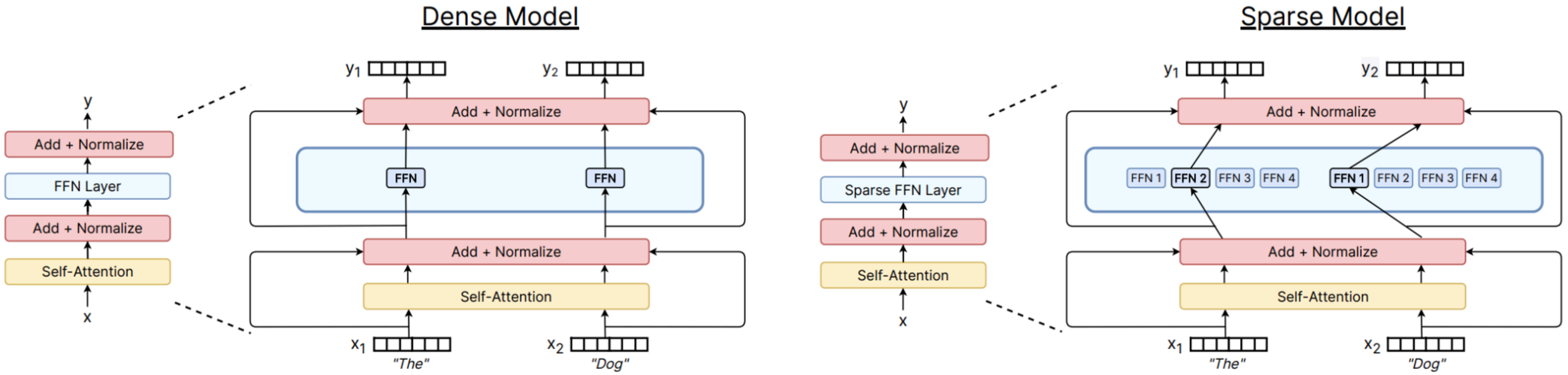




Mixture-of-Experts

- MoE is a technique that uses many different sub-models (or "experts") to improve the quality of LLMs

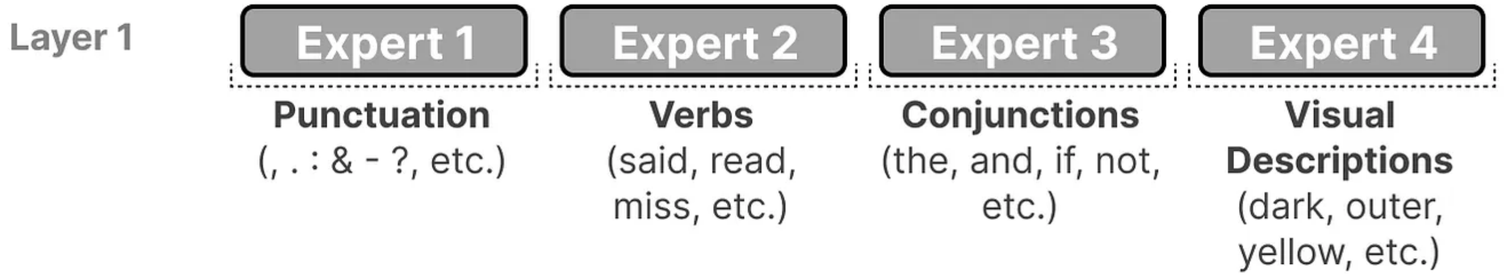
- Two main components:
  - Experts: Replace big FFN with many (but smaller) FFNs
  - Router: Determines which tokens are sent to which experts



Experts

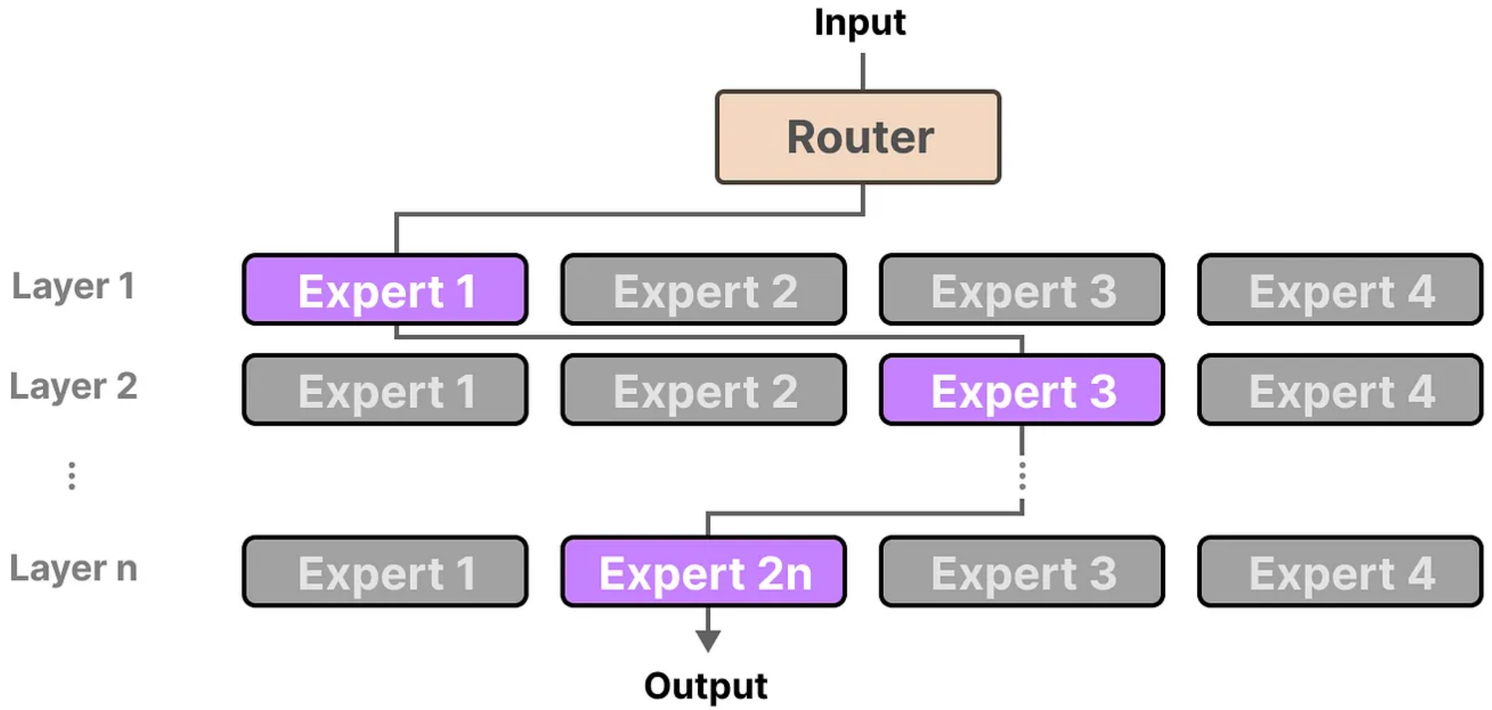
- In each FFN with MoE, we have (somewhat specialized) experts

- But experts are not specialized in a specific domain!
  - Rather it learns syntactic information on a word level instead



Router

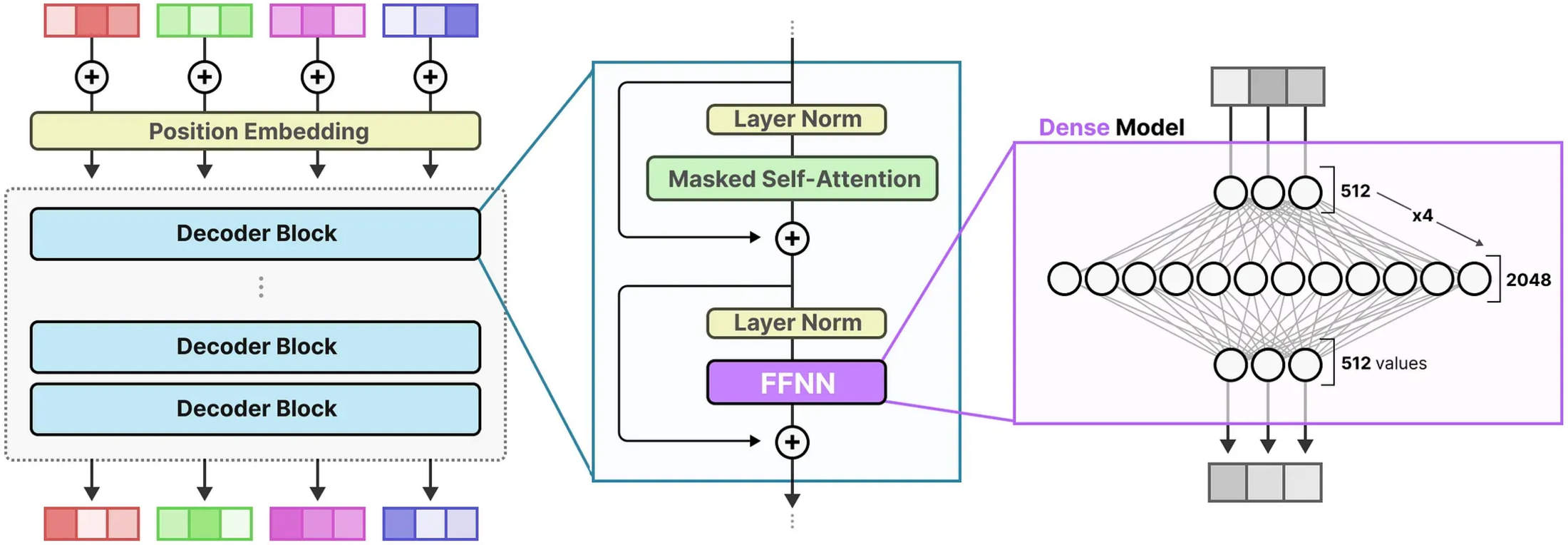
- The router selects the expert(s) best suited for a given input



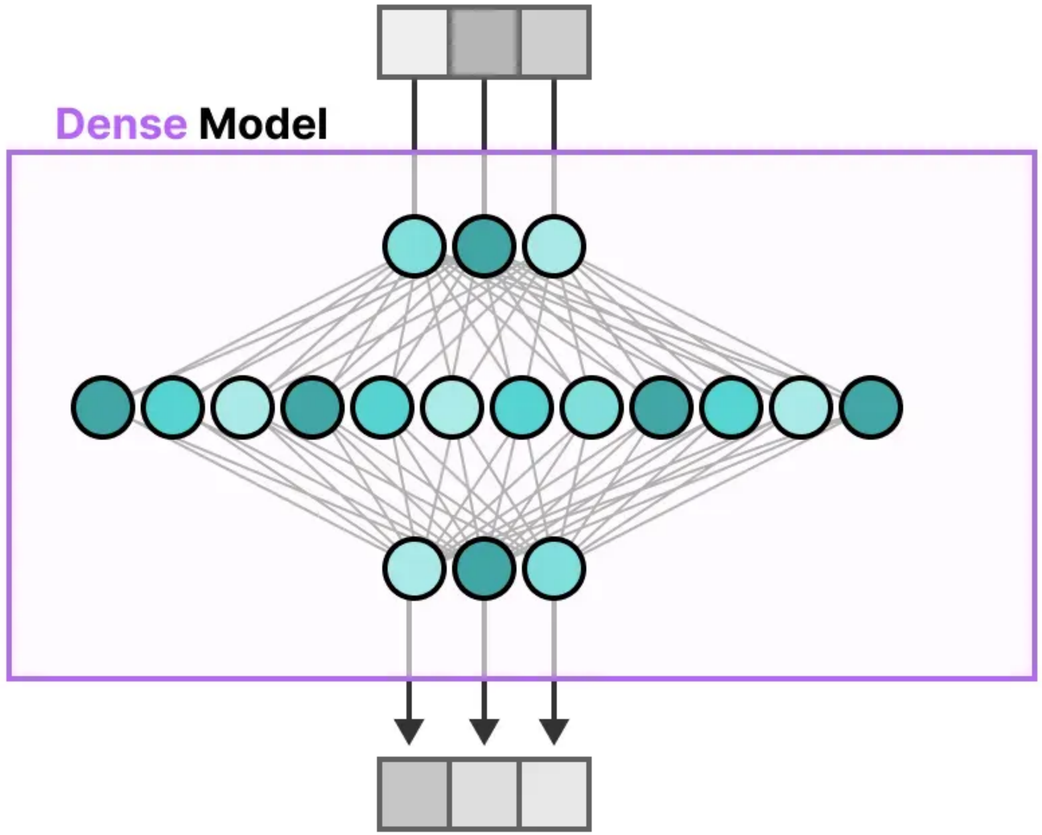
- Each expert is not an entire LLM but a submodel part of an LLM

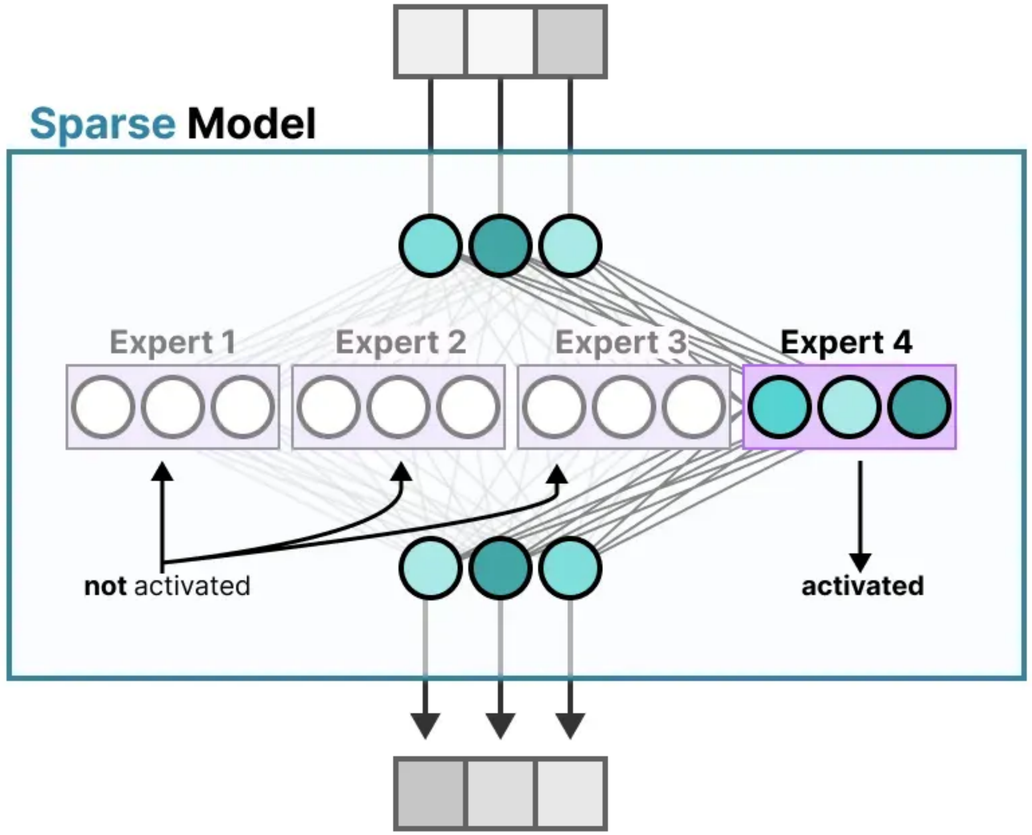
Dense FFN

- FFN without MoE is called a dense layer



Dense FFN to Sparse Layers (Experts)





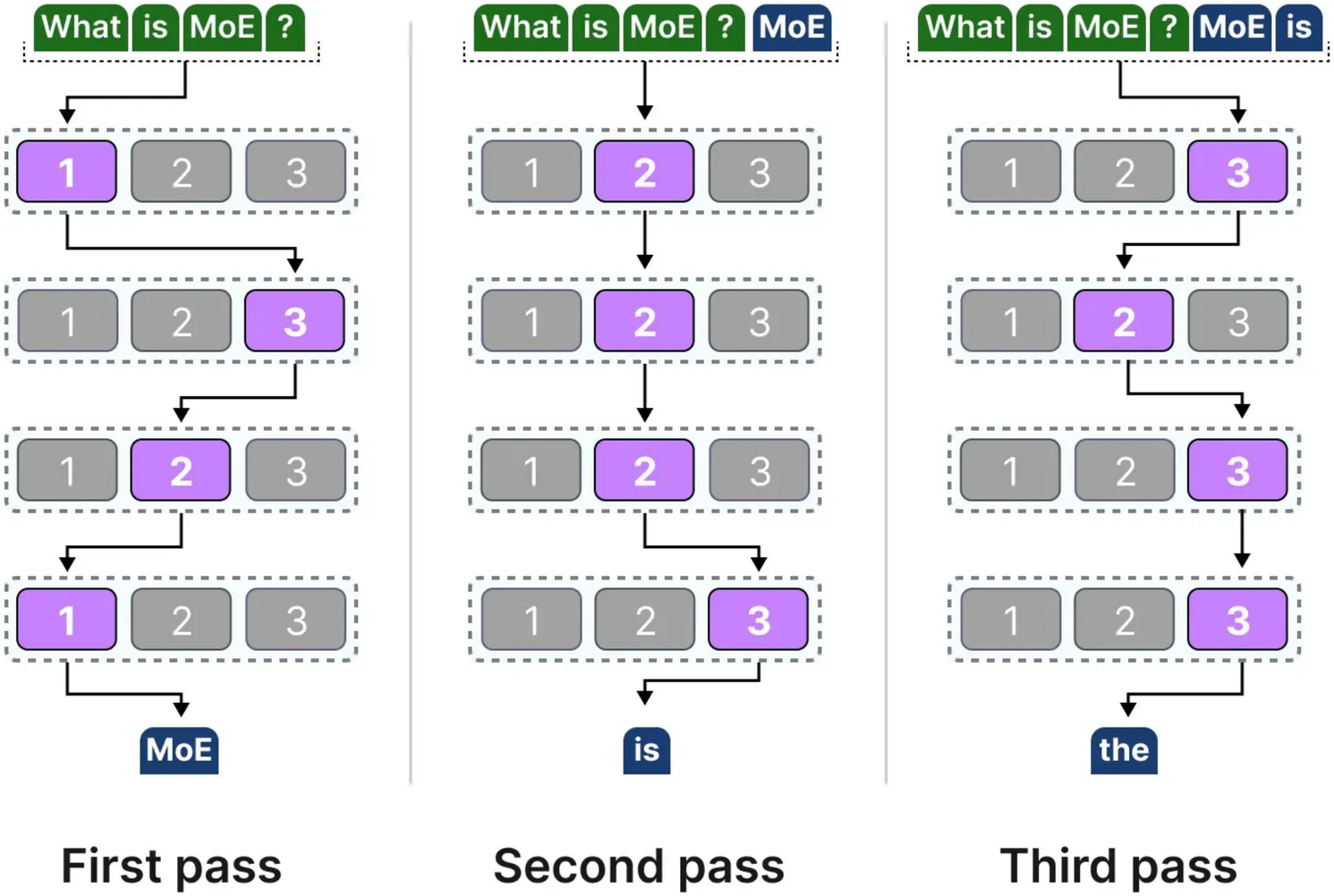
What Experts Learn?

Expert specialization	Expert position	Routed tokens
Punctuation	Layer 2	, , , , , , , , ,- , , , , ,).)
	Layer 6	, , , , , : . : , & , & & ? &-,,? ,,, .
Conjunctions and articles	Layer 3	The the the the the the the the The...
	Layer 6	a and and and and and and or and ...
Verbs	Layer 1	died falling identified fell closed left posted lost felt left said read miss place struggling falling signed died...
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
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

Mixtral example

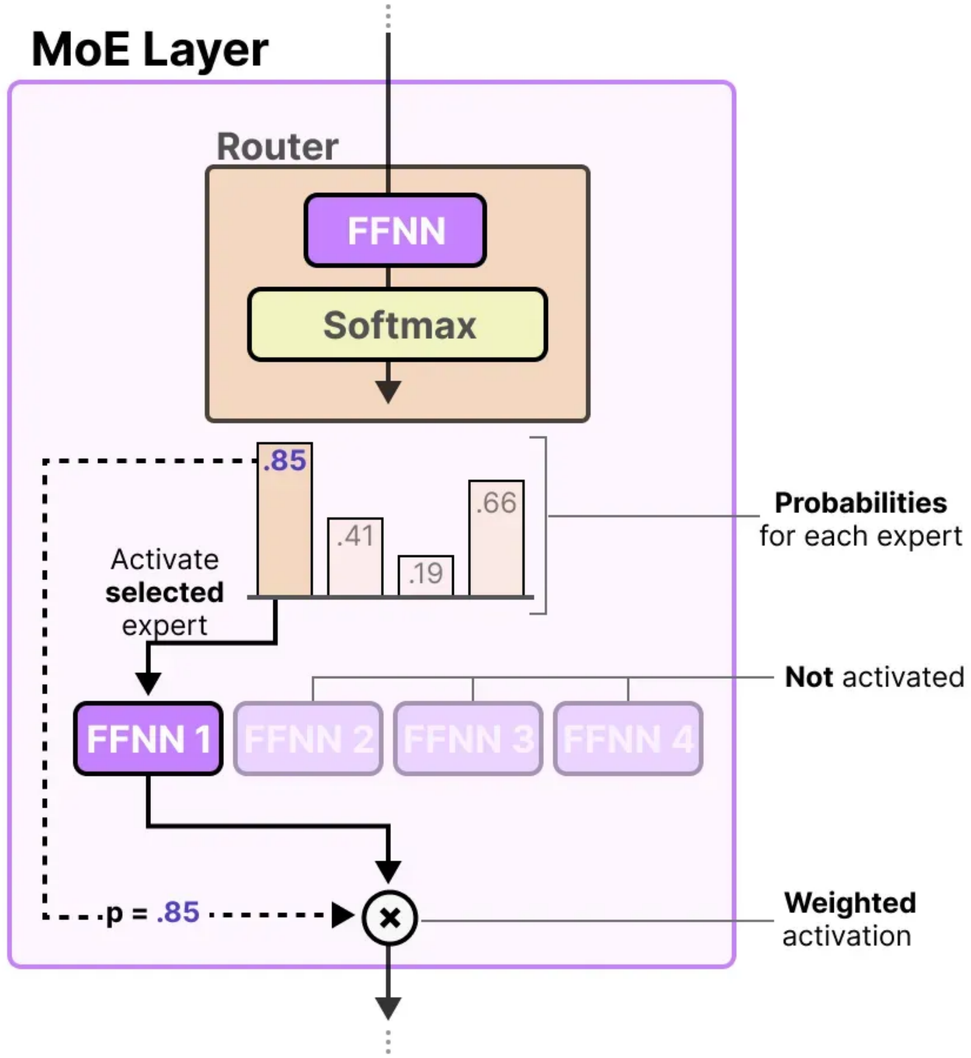
```
class MoeLayer(nn.Module):
    def __init__(self, experts: List[nn.Module],
        super().__init__()
        assert len(experts) > 0
        self.experts = nn.ModuleList(experts)
        self.gate = gate
        self.args = moe_args
```

Inference Example

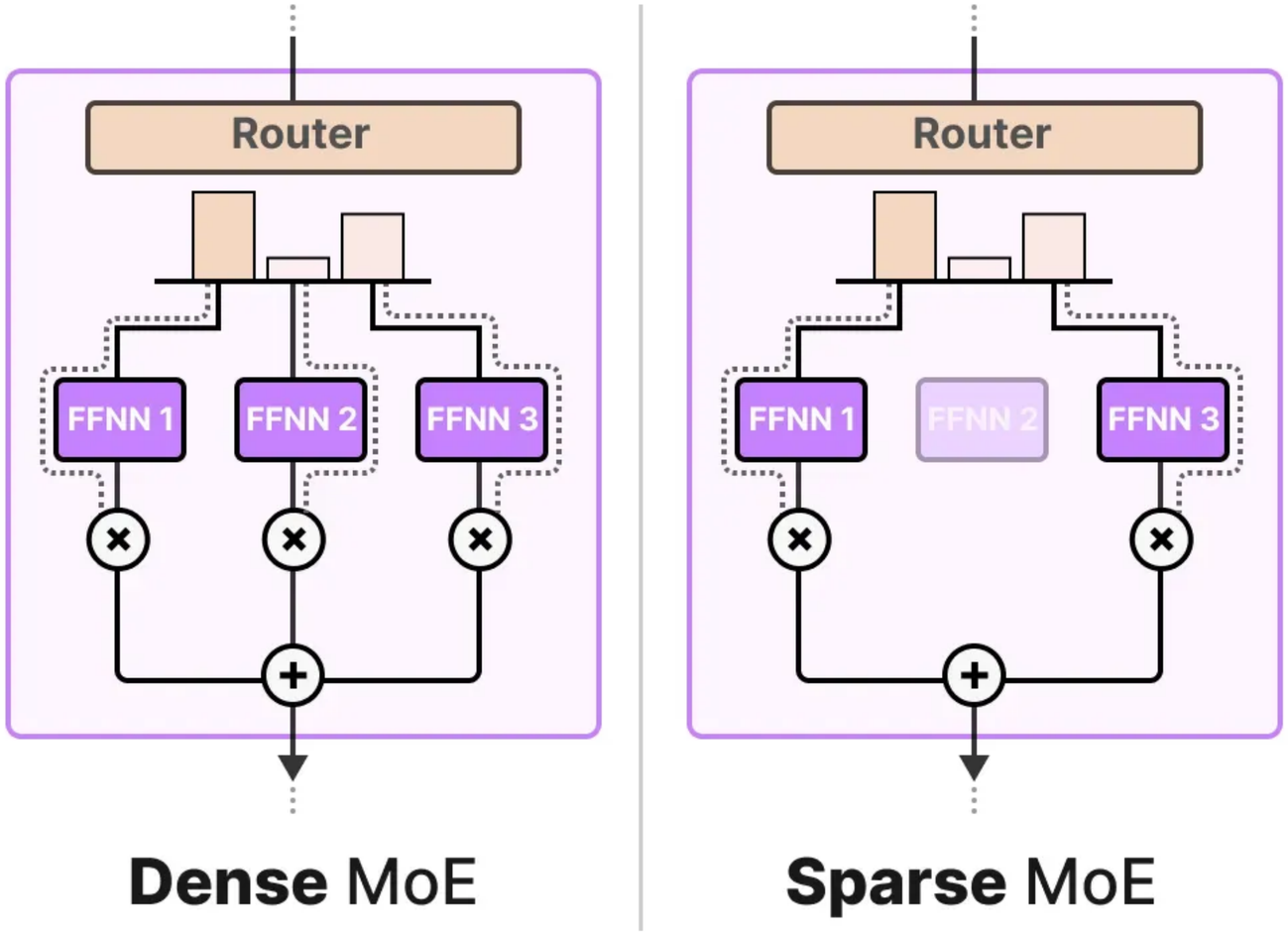


Router

- Router is used to choose the expert based on an input
- It outputs probabilities which it uses to select the best matching expert
- Router is also a FFN



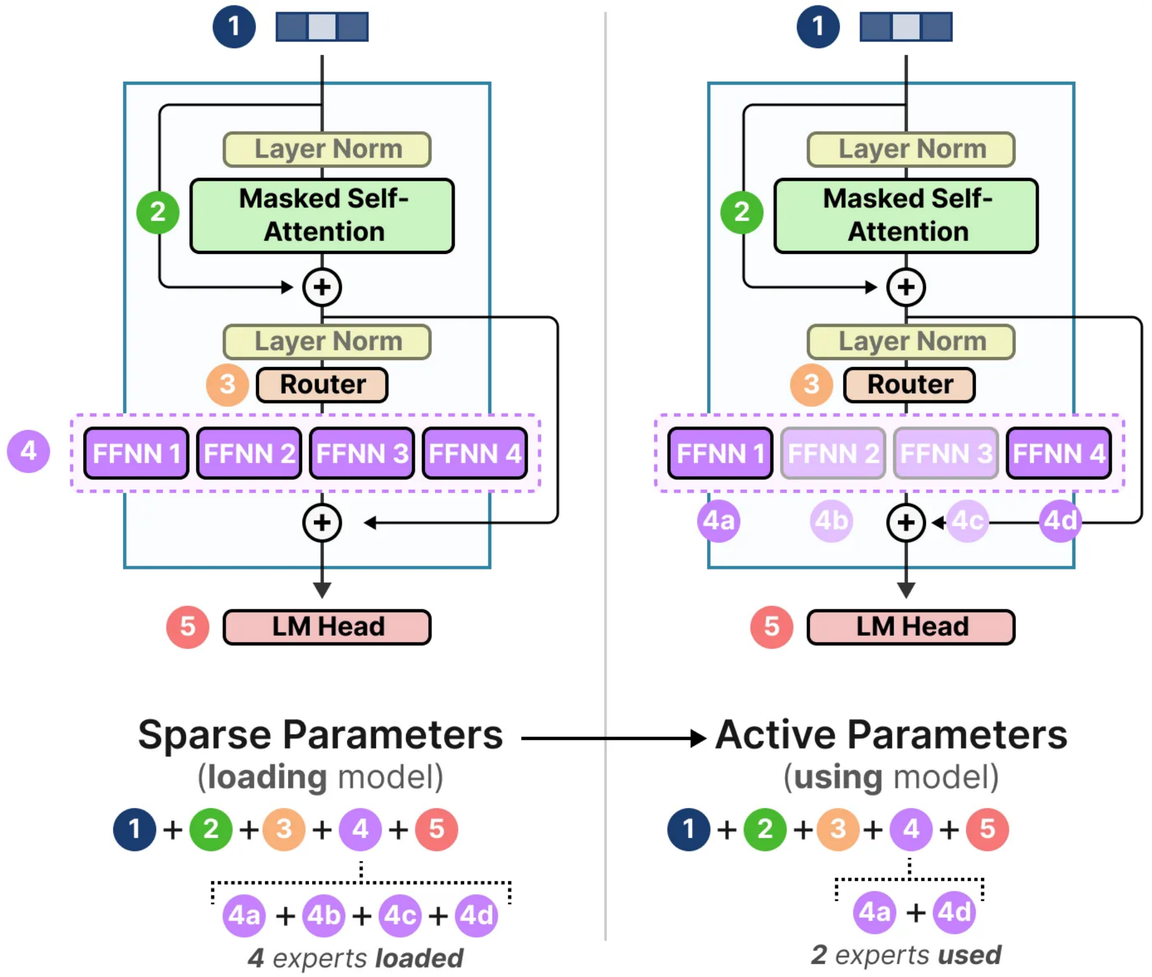
Router


Router
FFNN 1
FFNN 2
FFNN 3
Dense MoE
Router
FFNN 1
FFNN 2
FFNN 3
Sparse MoE

Why MoEs?

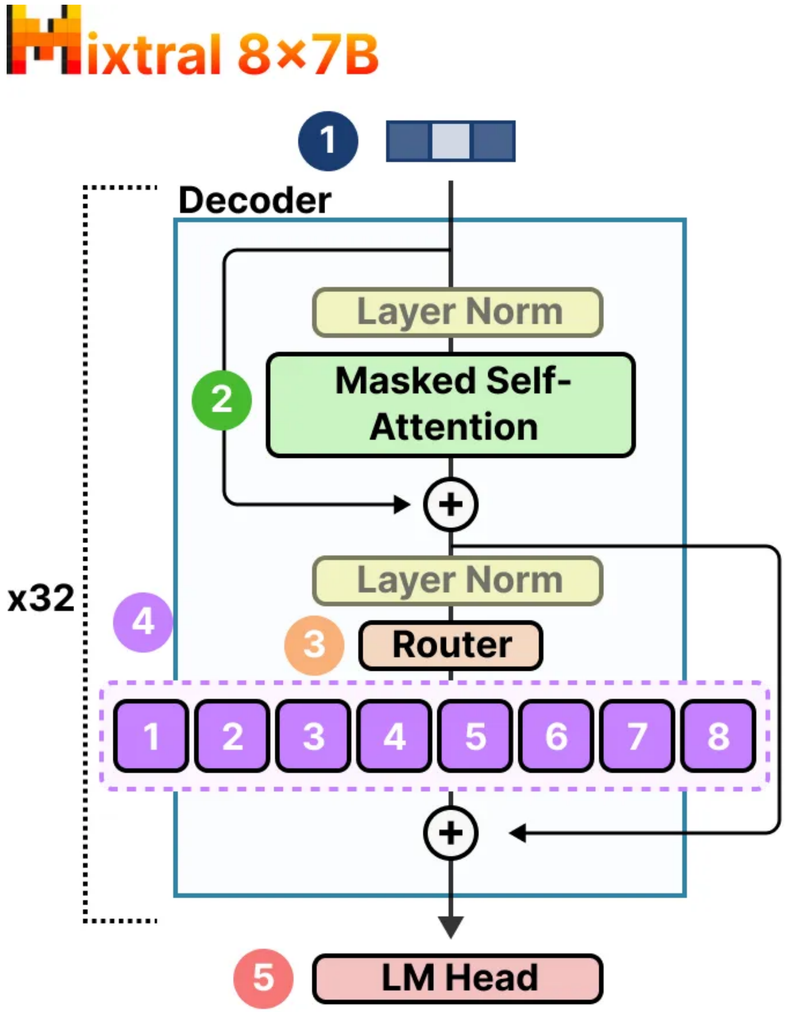
- MoE is used because of its computational requirements
  - Since only a subset of experts are used, we have access to more parameters than we are using

- MoE has more parameters to load than dense models (sparse params)
- Fewer are activated since only some experts are activated at inference (active params)



Why MoEs?

- Parameter calculation



1 Embeddings

32000 × 4096 = **131.072.000** (**shared** parameters)

- 4096: embedding size

2 Attention

32 × 41.943.040 = **1.342.177.280** (**shared** parameters)

- 32: repeated decoder blocks
- 41.943.040: (q, k, v)

3 Router

8 × 4096 = **32.768** (**shared** parameters)

- 8: # experts

4 Experts

8 × 5.637.144.576 = **45.097.156.608** (**total** parameters)

- 8: # experts

2 × 5.637.144.576 = **11.274.289.152** (**active** parameters)

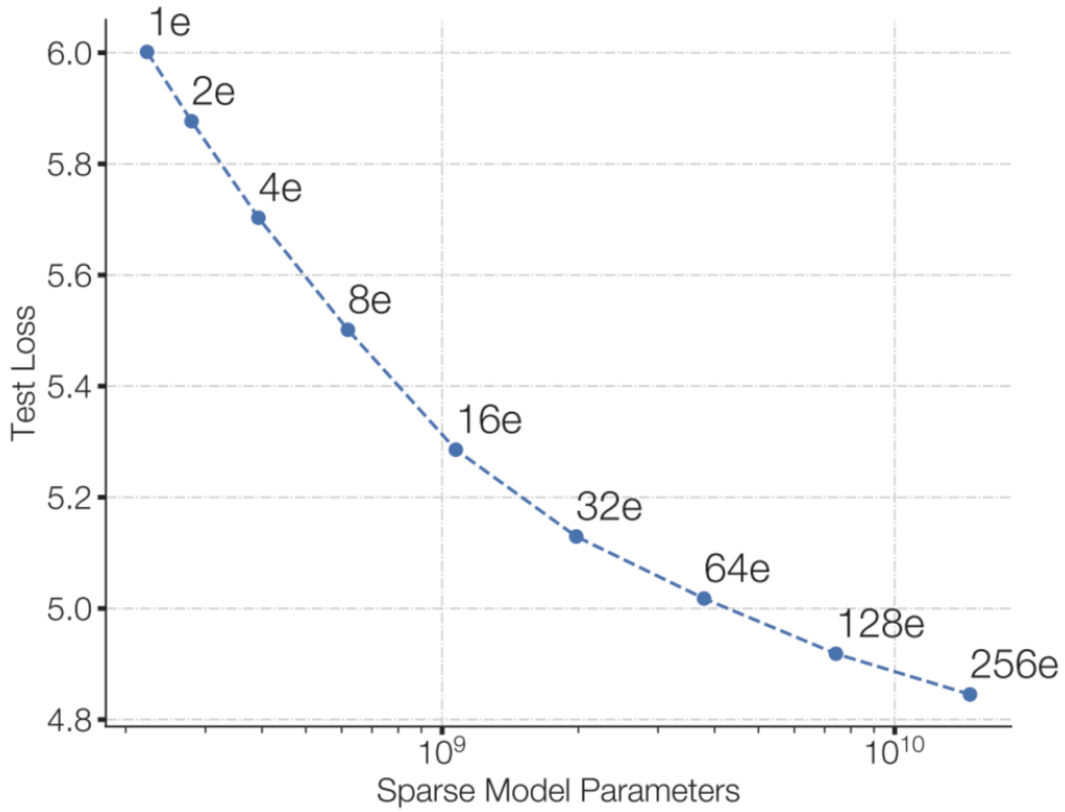
- 2: # experts
- 5.637.144.576: expert size

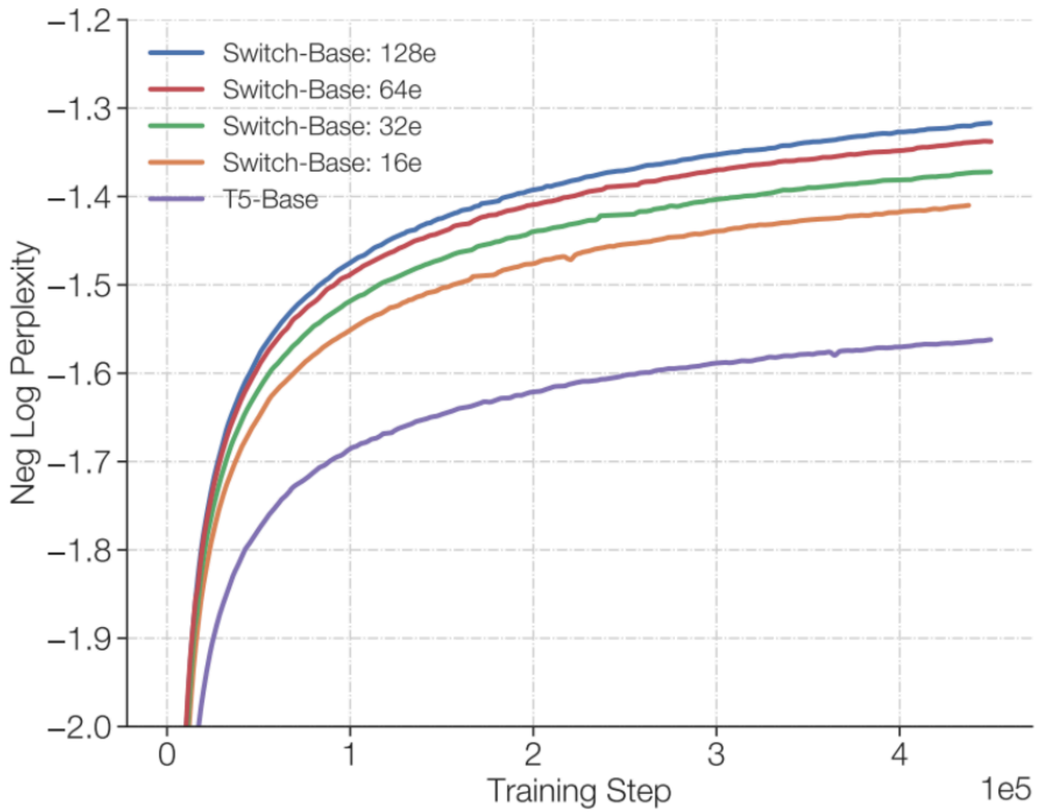
5 LM Head

32000 × 4096 = **131.072.000** (**shared** parameters)

Why MoEs?

- Same FLOPs, more param do better

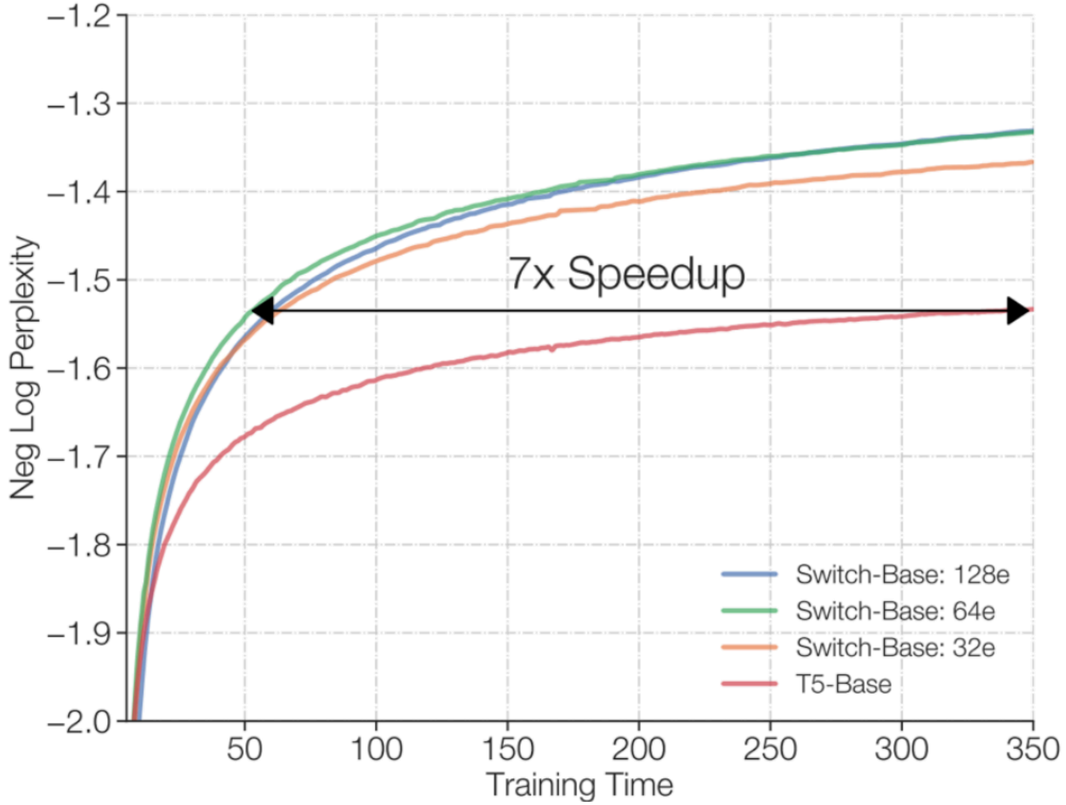




[Fedus+ 2022]

Why MoEs?

- Faster to train MoEs



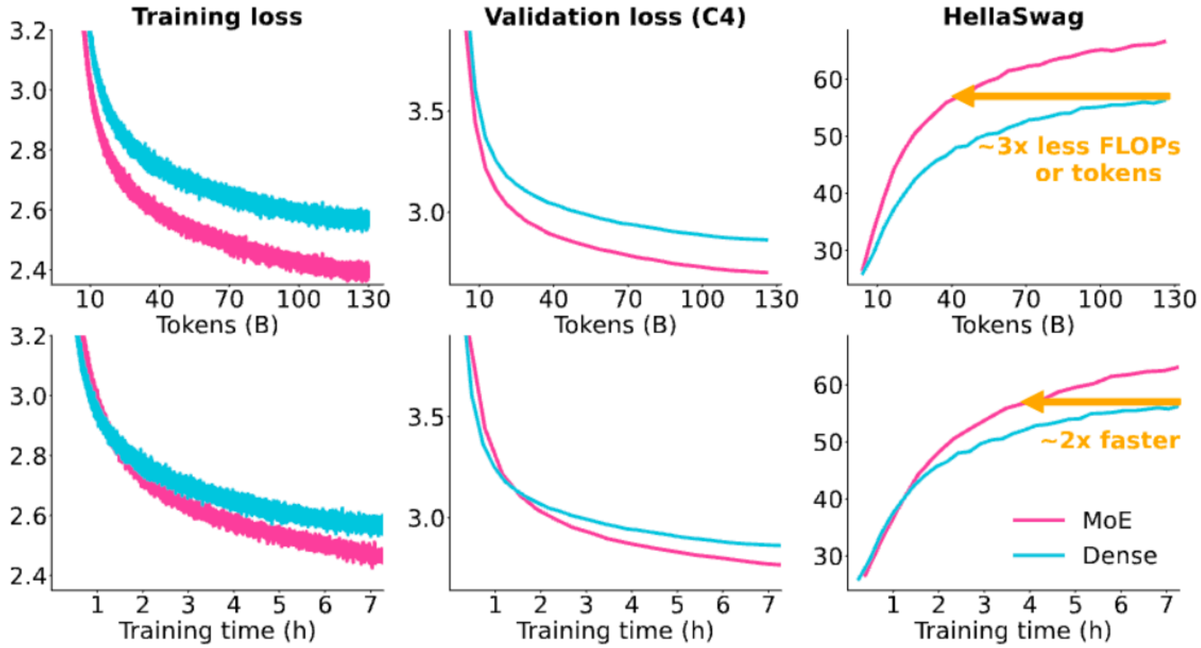
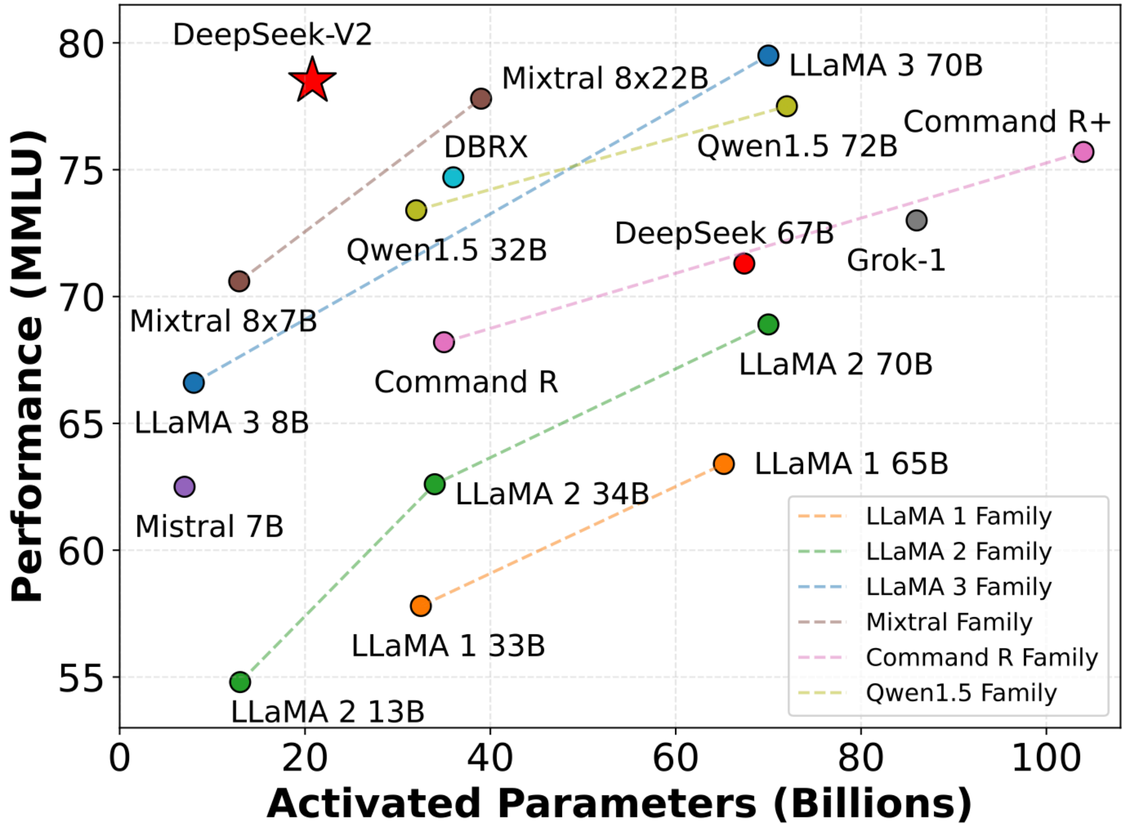


Figure 4: **MoE vs. Dense.** We train a 1.3B parameter dense model and a 1.3B active, 6.9B total parameter MoE model, each on 128 H100 GPUs. Apart from MoE-related changes, we train both

OLMoE [Muennighoff+ 2024]

Why MoEs?

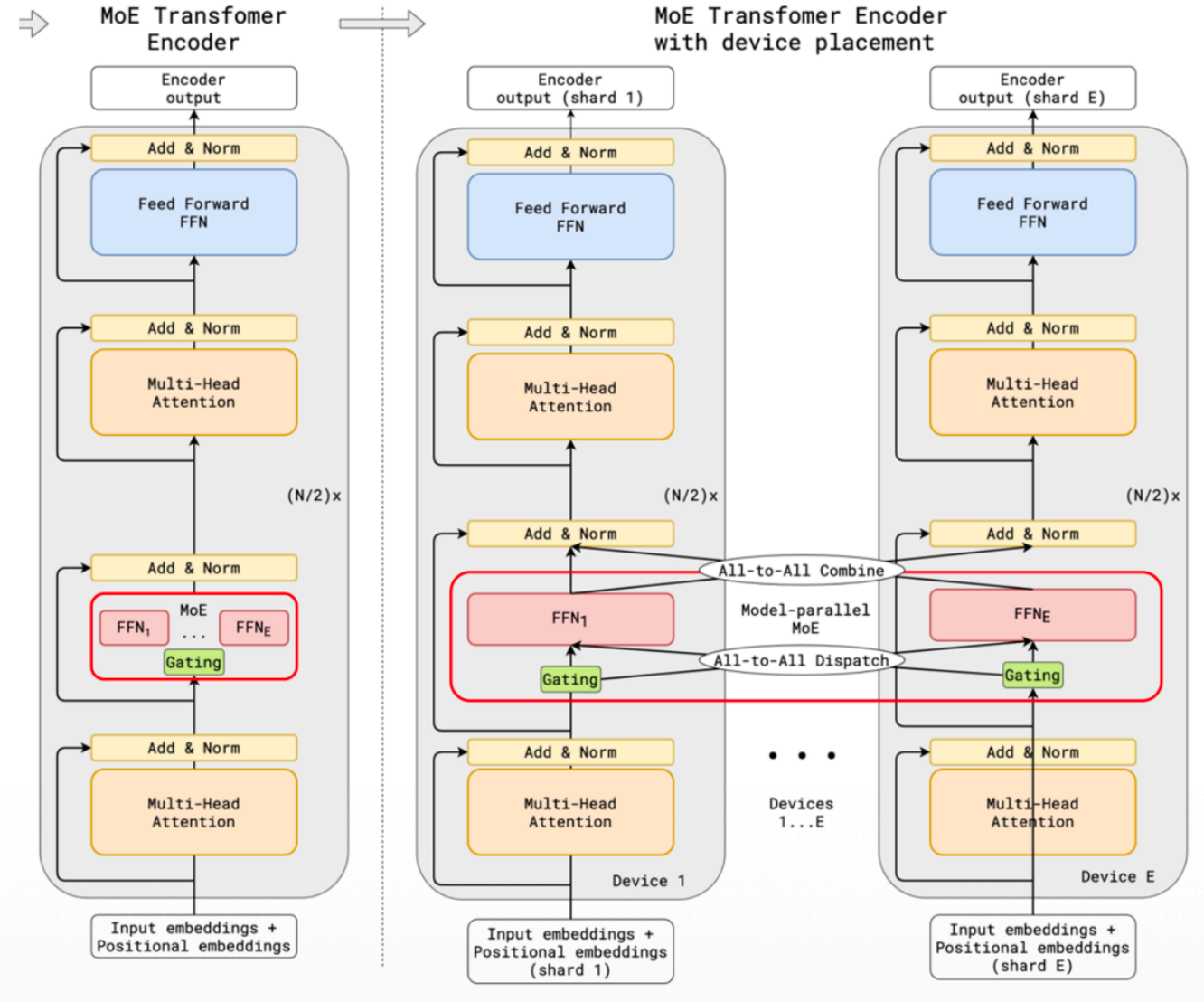
- Highly competitive vs. dense models



DeepSeek V2 [DeepSeek-AI 2024]

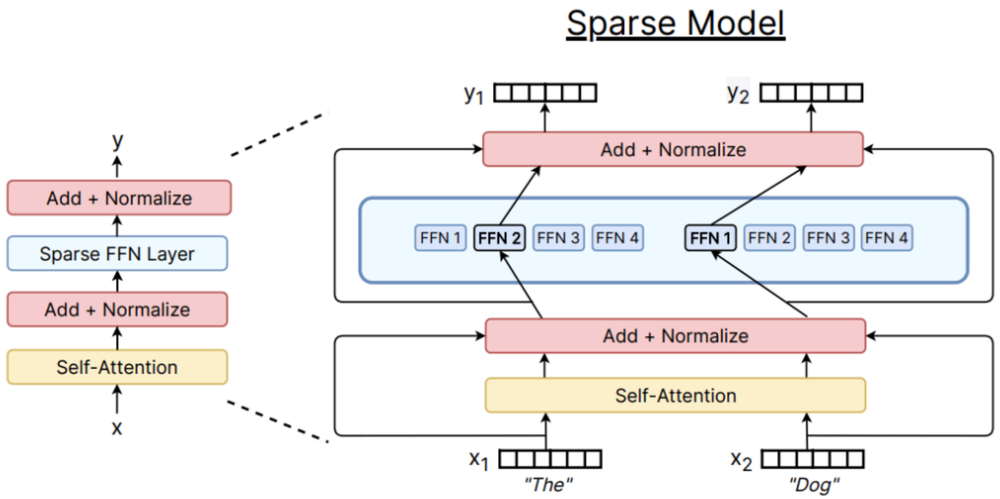
Why MoEs?

- Parallelizable to many devices

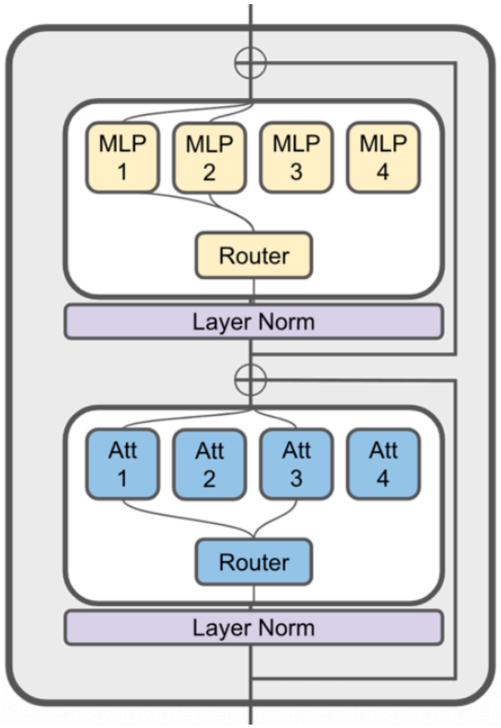


Typical MoEs

Typical: Replace dense FFN with MoE



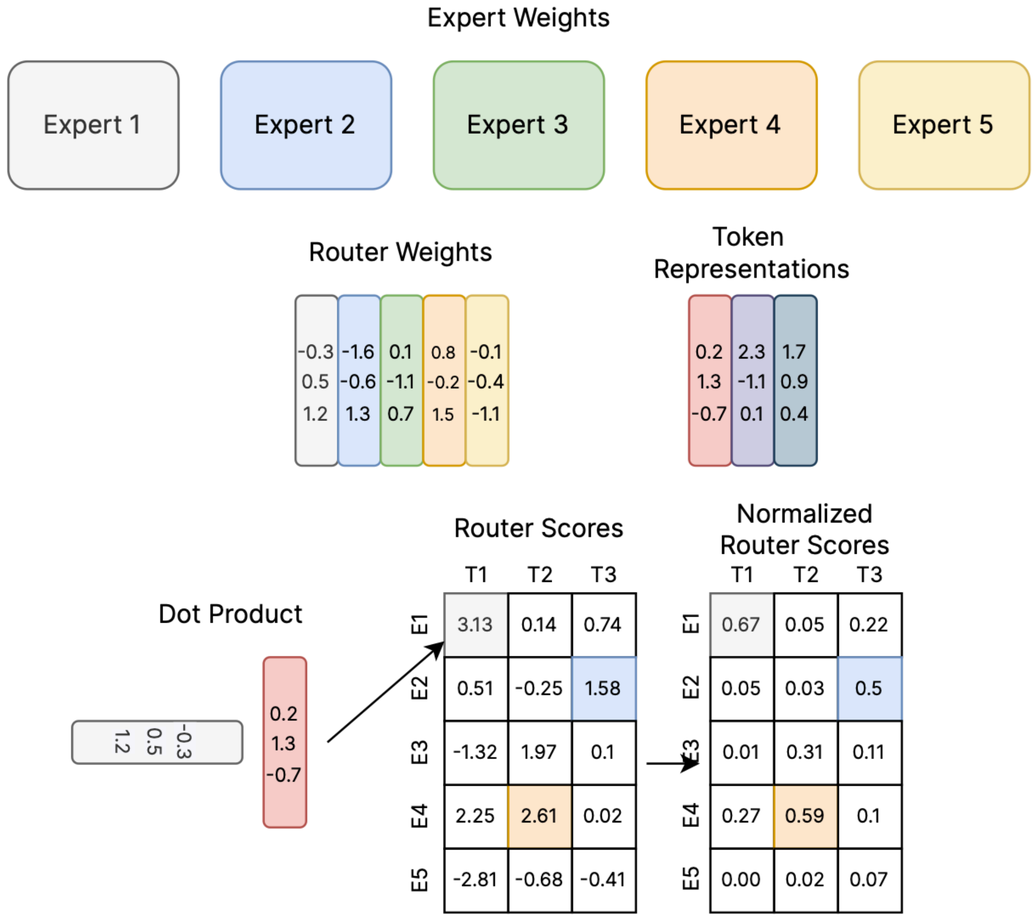
Less common: MoE for attention



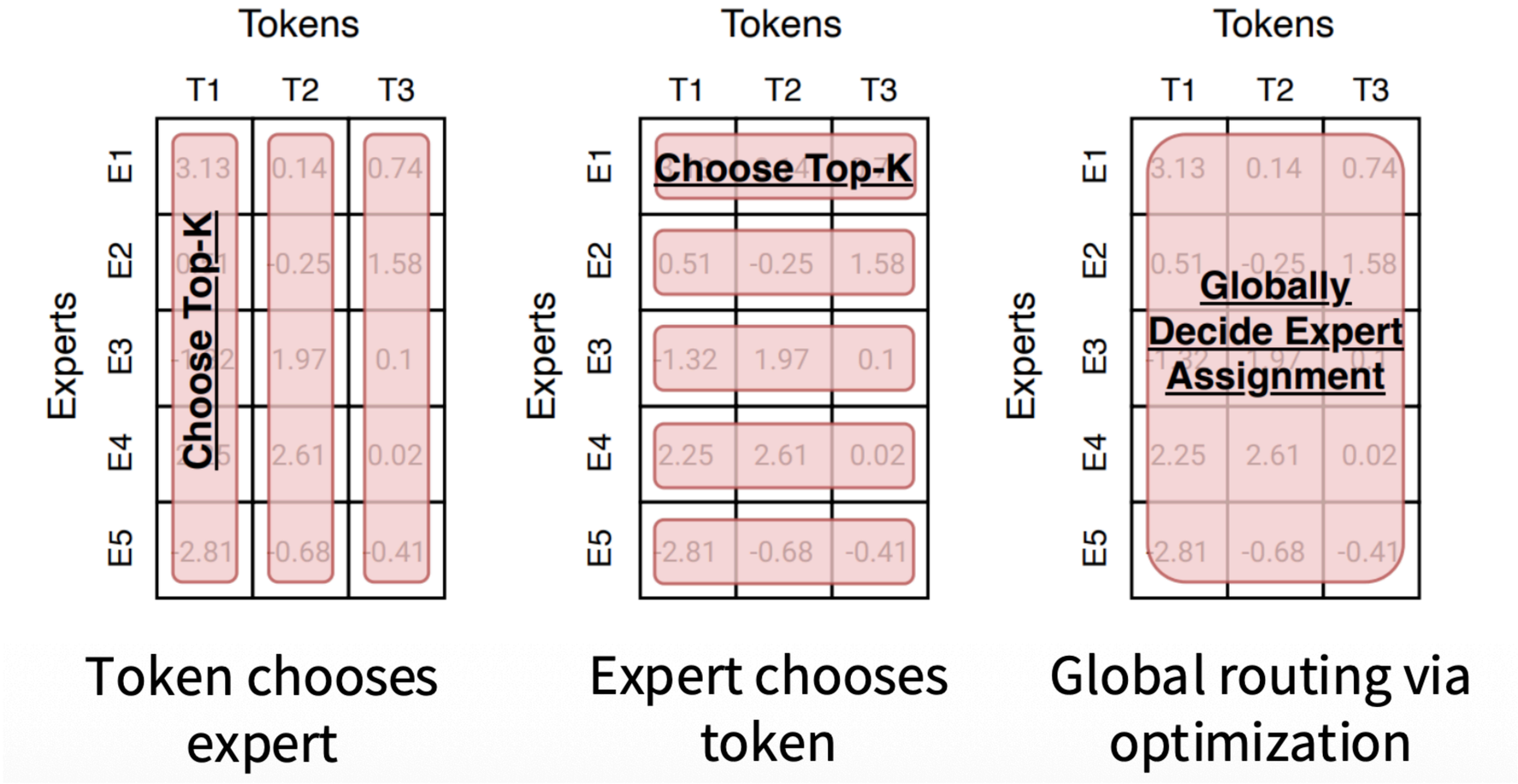
Deep Dives into MoEs

Router

- Many of the routing algorithms to choose top-k



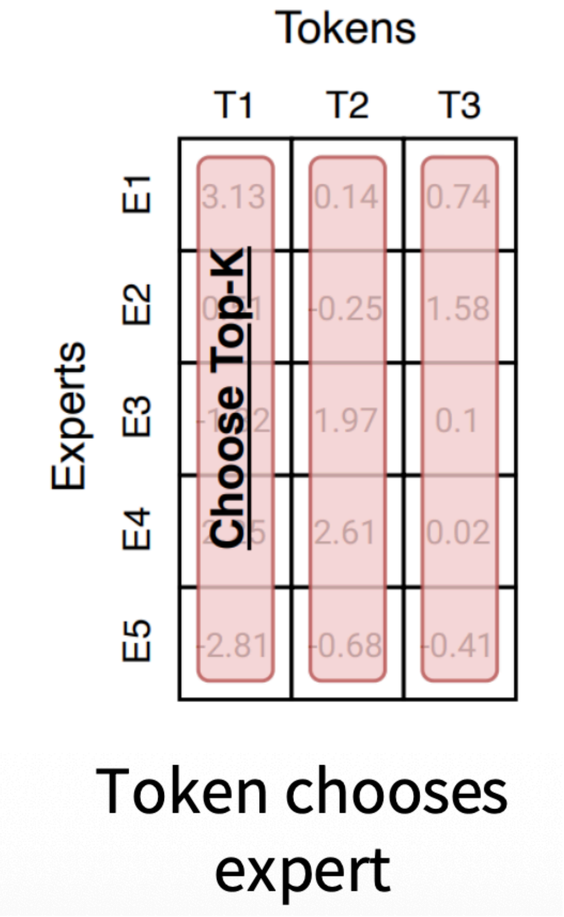
Router Type


Tokens
T1
T2
T3
Experts
E1
E2
E3
E4
E5
3.13
0.14
0.74
0.51
-0.25
1.58
-1.32
1.97
0.1
2.25
2.61
0.02
-2.81
-0.68
-0.41
Choose Top-K
Choose Top-K
Globally Decide Expert Assignment
Token chooses expert
Expert chooses token
Global routing via optimization

Router Type: Token Choose

- For each input token, it chooses expert

- 👍 Straightforward implementation
- 👍 Good performance
- 👎 Load balancing
  - Unbalanced expert selection
  - Requires auxiliary load balancing loss



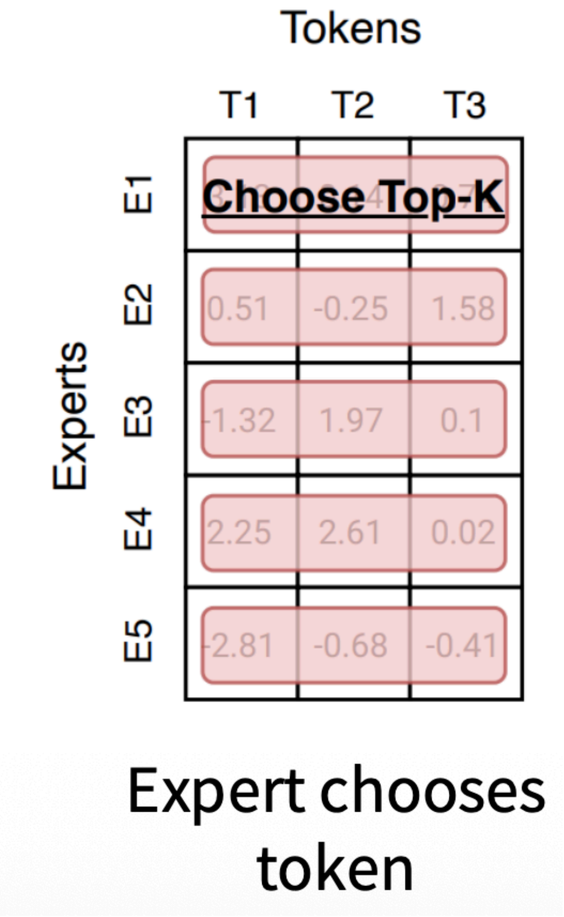
Token chooses expert

Router Type: Expert Choose

- For each expert, it chooses token

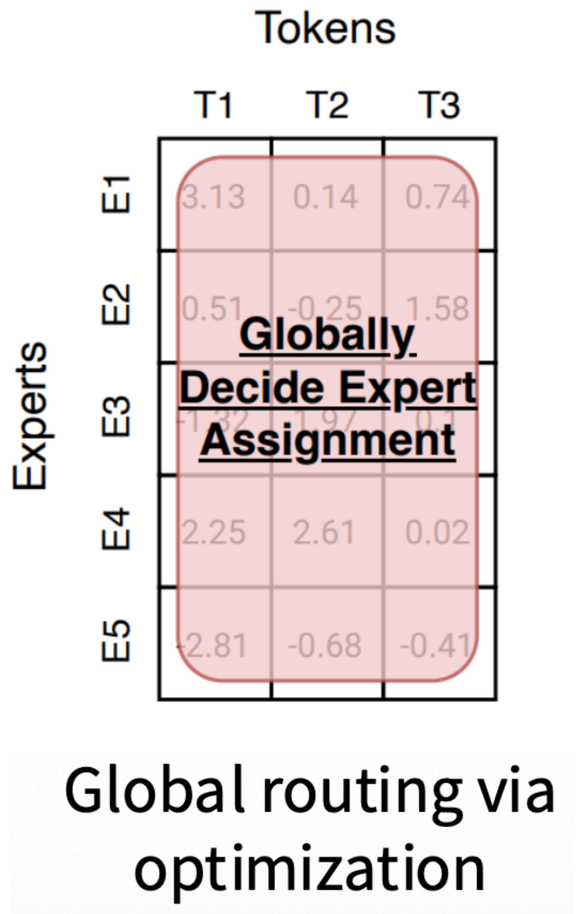
- 👍 Good load balancing
- 👍 Variable token computation
  - Multiple experts compute important tokens

- 👎 Harder to implement
  - Expert should see all the input tokens
  - Difficult to adapt into autoregressive models



Router Type: Global Routing

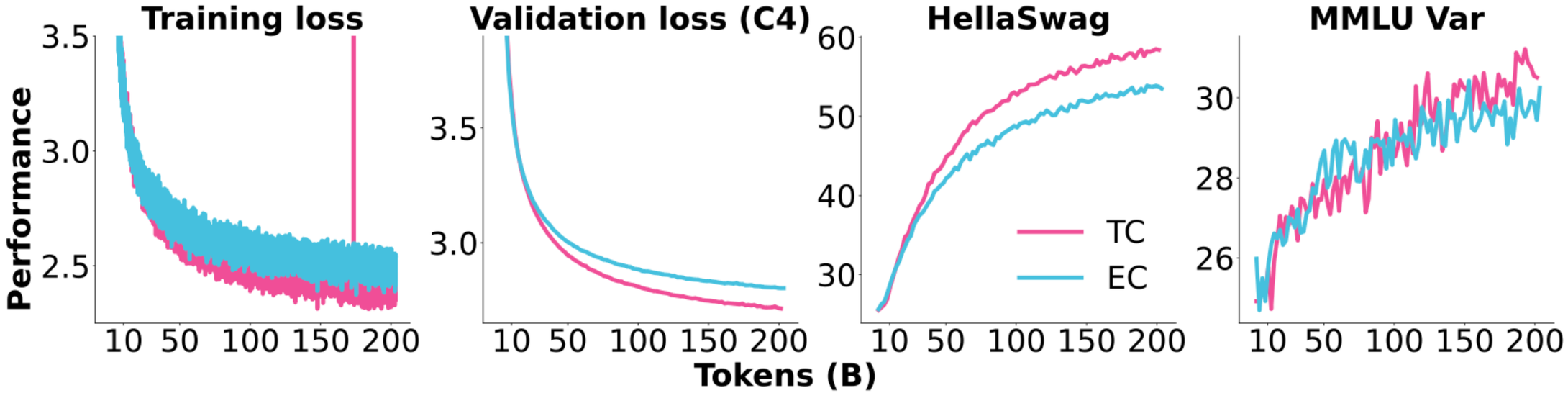
- Solve this as an optimization problem
- 👍 Optimal case
- 👎 Computational overhead



Global routing via optimization

Router Type

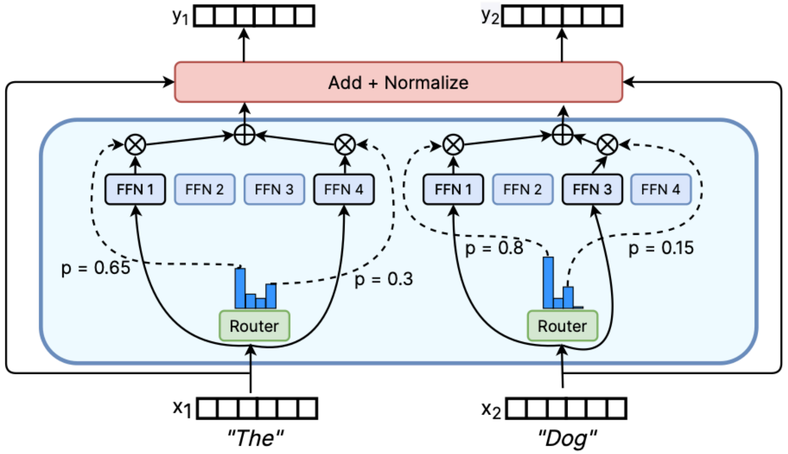
- Almost all the MoEs do standard token choice top-k routing
- Some recent ablations from OLMoE



OLMoE [Muennighoff+ 2024]

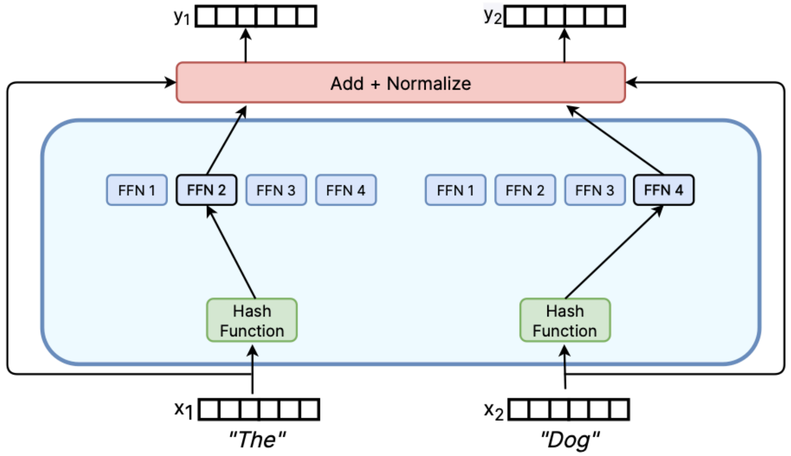
Common Routing Variants

Top-K



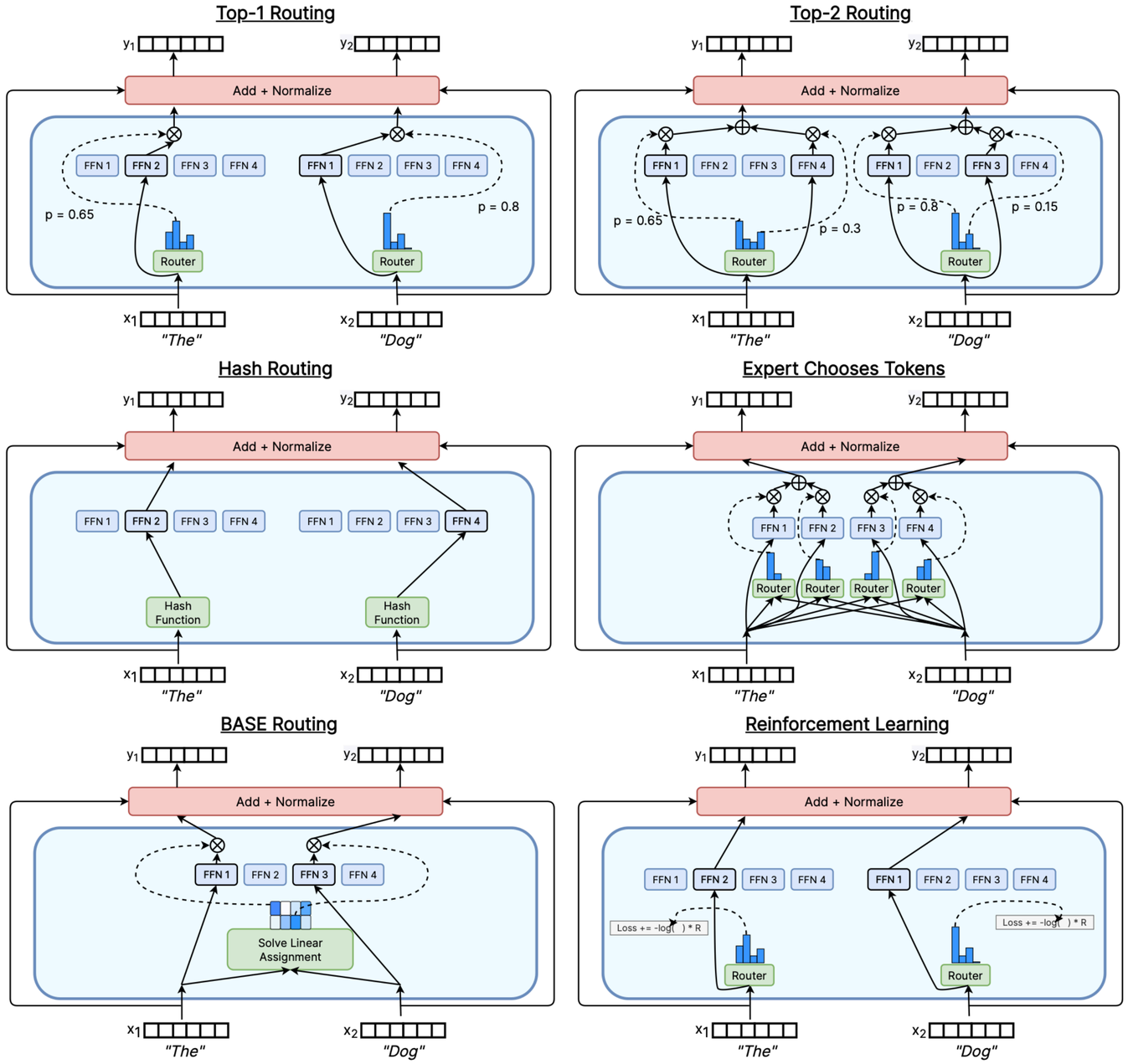
- Used in most MoEs
- Different models use different k

Hashing



- Common baseline

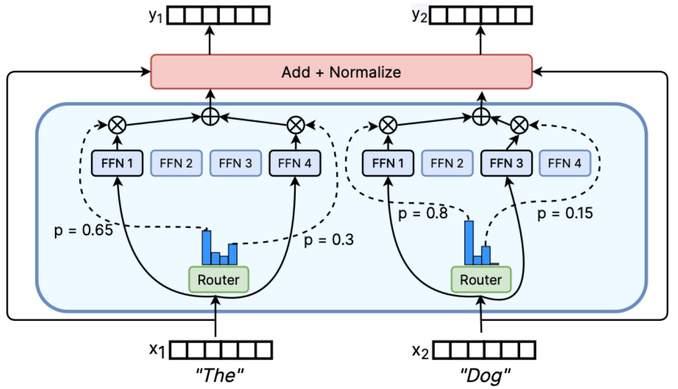
Routing Variants



[Fedus+ 2022]

Top-K Routing in Detail

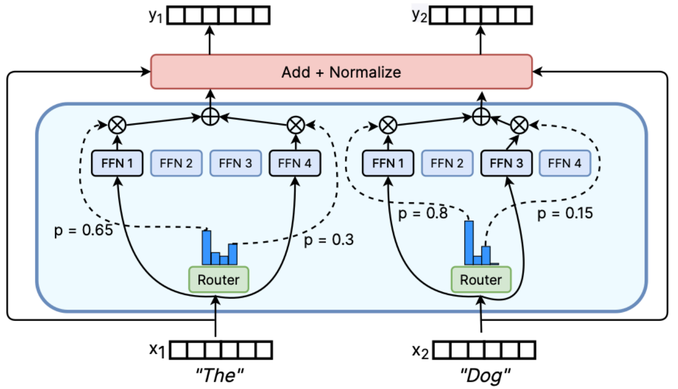
- Most papers do the old and classic top-k routing



$$\mathbf{h}_t^l = \sum_{i \in \mathcal{A}_t} g_{i,t} \mathrm{FFN}_i(\mathbf{u}_t^l) + \mathbf{u}_t^l \qquad z_{i,t} = (\mathbf{u}_t^l)^\top \mathbf{e}_i^l$$

Top-K Routing in Detail

- Most papers do the old and classic top-k routing



$$\mathbf{h}_t^l = \sum_{i \in \mathcal{A}_t} g_{i,t} \mathrm{FFN}_i(\mathbf{u}_t^l) + \mathbf{u}_t^l \qquad z_{i,t} = (\mathbf{u}_t^l)^\top \mathbf{e}_i^l$$

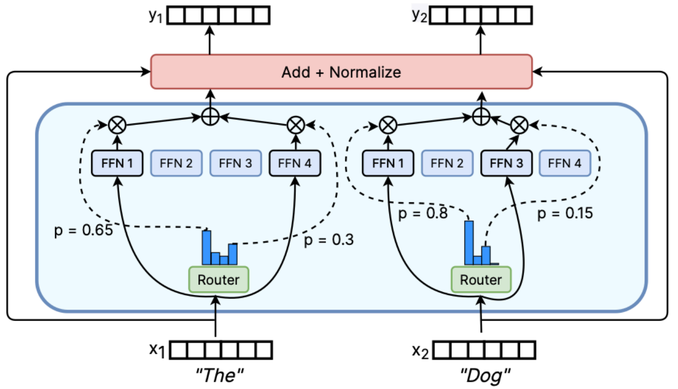
top-k FFN

gating

routing logit

router weight

Top-K Routing in Detail



- Most papers do the old and classic top-k routing

$$\mathbf{h}_t^l = \sum_{i \in \mathcal{A}_t} g_{i,t} \mathrm{FFN}_i(\mathbf{u}_t^l) + \mathbf{u}_t^l \qquad z_{i,t} = (\mathbf{u}_t^l)^\top \mathbf{e}_i^l$$

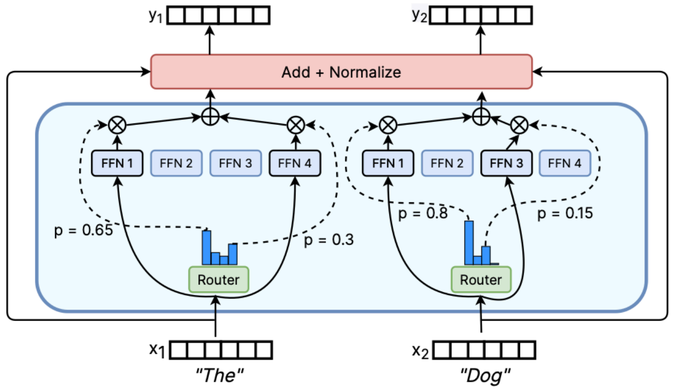
- Pre-Softmax (DeepSeek v1/2, Grok, Qwen)

$$g_{i,t}^{\mathrm{pre}} = \begin{cases} s_{i,t}, & \text{if } i \in \mathcal{A}_t^{\mathrm{pre}} \\ 0, & \text{otherwise} \end{cases} \qquad s_{i,t} = \frac{\exp(z_{i,t})}{\sum_{j=1}^{N} \exp(z_{j,t})}$$

$$\mathcal{A}_t^{\mathrm{pre}} = \mathrm{TopK}(\{s_{j,t}\}_{j=1}^{N}, K)$$

Note: Sum of $g_{i,t}$ is not 1

Top-K Routing in Detail



- Most papers do the old and classic top-k routing

$$\mathbf{h}_t^l = \sum_{i \in \mathcal{A}_t} g_{i,t} \mathrm{FFN}_i(\mathbf{u}_t^l) + \mathbf{u}_t^l \qquad z_{i,t} = (\mathbf{u}_t^l)^\top \mathbf{e}_i^l$$

- Post-Softmax (Mixtral, DeepSeek v3)

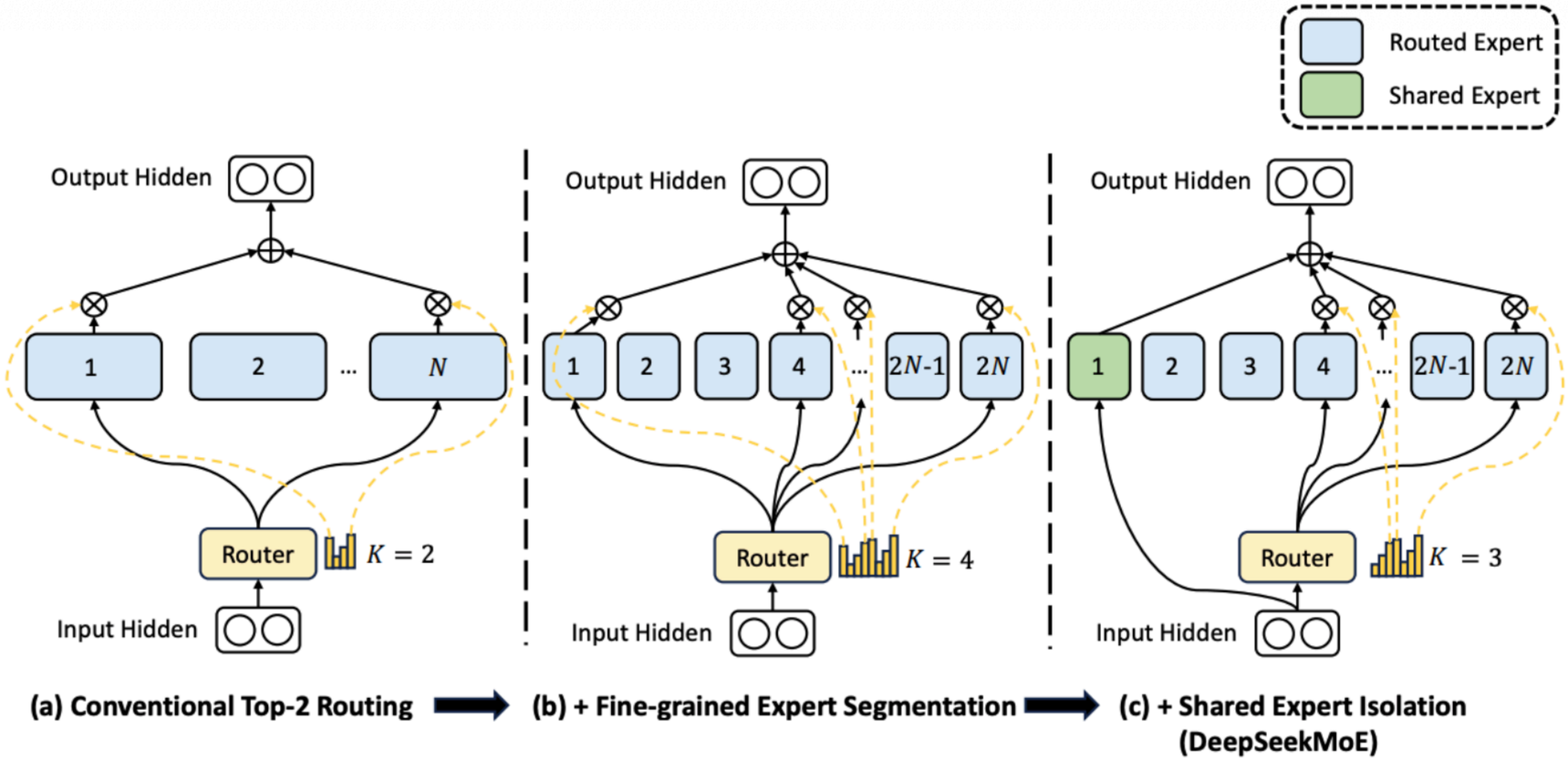
$$g_{i,t}^{\mathrm{post}} = \begin{cases} \frac{\exp(z_{i,t})}{\sum_{j \in \mathcal{A}_t^{\mathrm{post}}} \exp(z_{j,t})}, & \text{if } i \in \mathcal{A}_t^{\mathrm{post}} \\ 0, & \text{otherwise} \end{cases}$$

$$\mathcal{A}_t^{\mathrm{post}} = \mathrm{TopK}(\{z_{j,t}\}_{j=1}^N, K)$$

Note: Now Sum of $g_{i,t}$ is 1

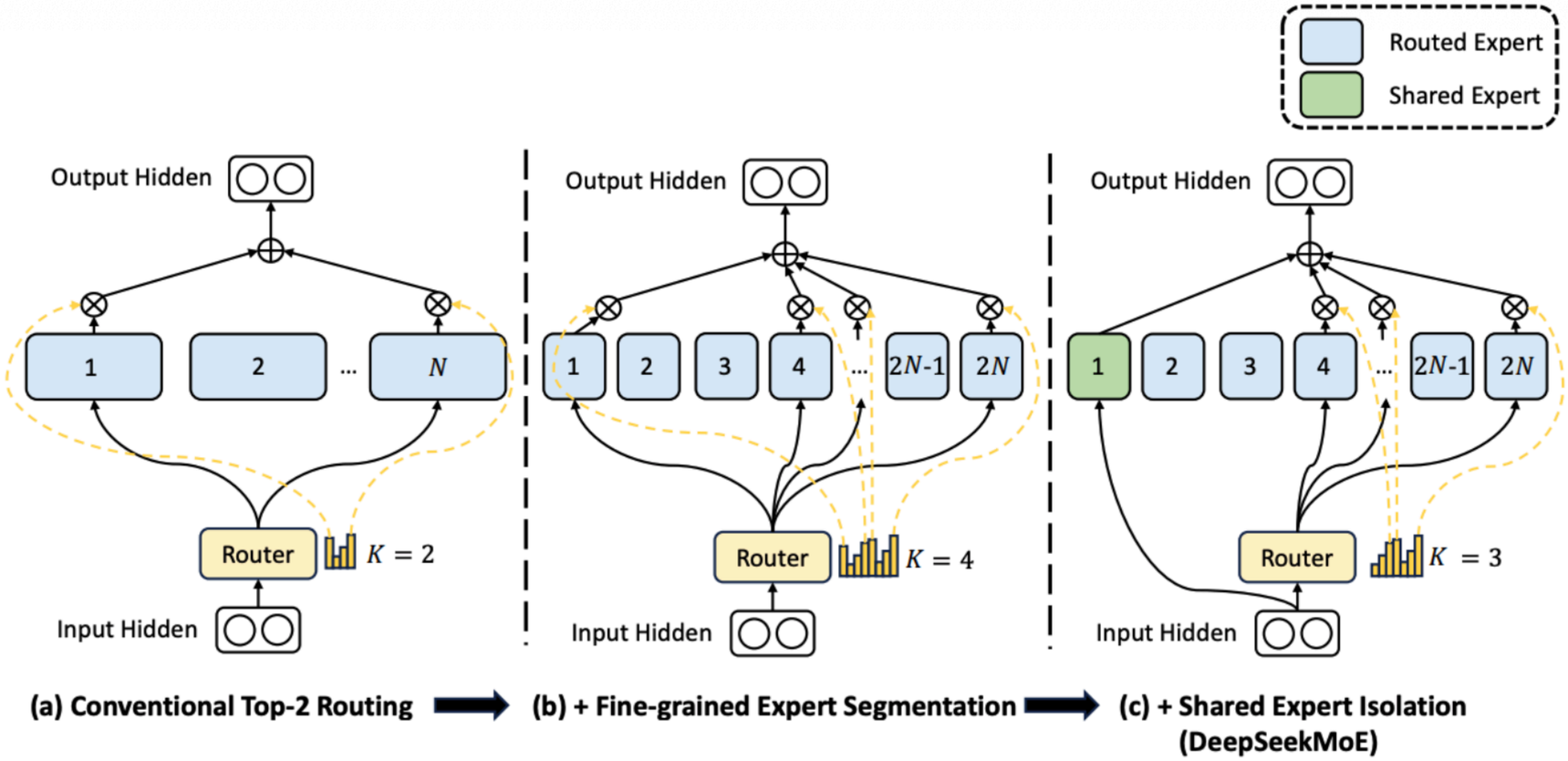
Fine-Grained Routed + Shared Experts

- Coarse-grained: Size of dense FFN == MoE FFN (e.g. Mixtral 8x7B)
- Fine-grained: Segment MoE FFN by m times



Fine-Grained Routed + Shared Experts

- Recent trends of Chinese MoE models (e.g. DeepSeek, Qwen, etc)
  - Fine-grained experts + a few shared experts



Fine-Grained Routed + Shared Experts

- Ablation from the DeepSeek paper
  - More experts and shared experts generally helpful

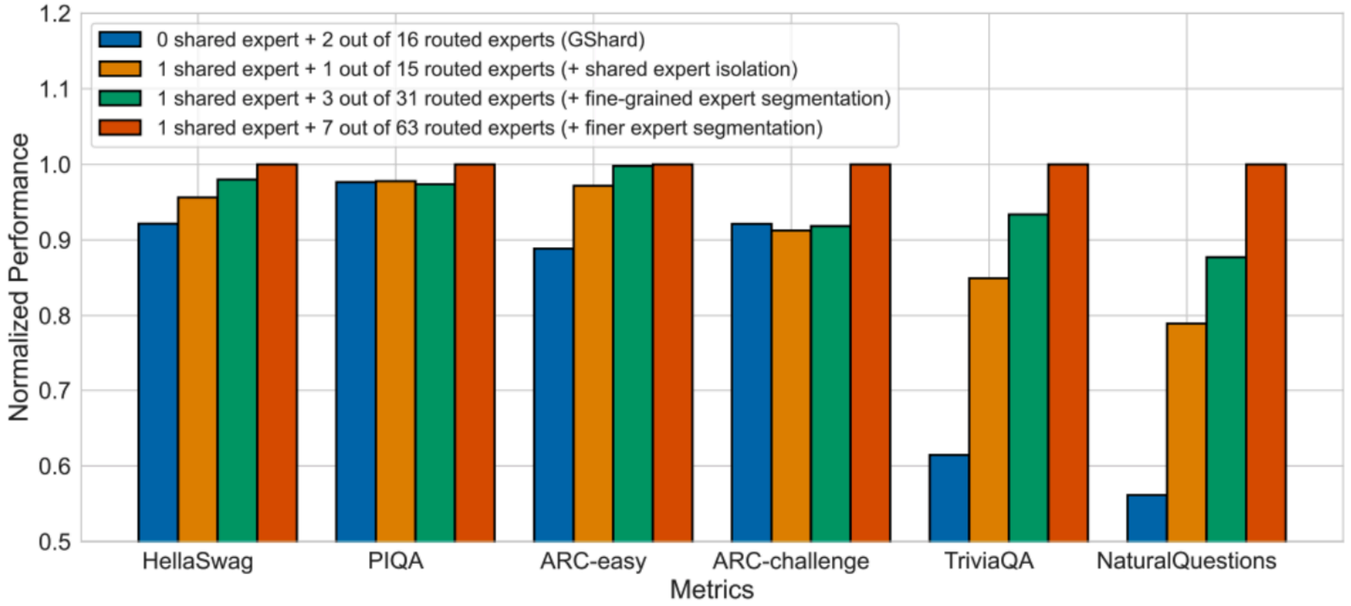
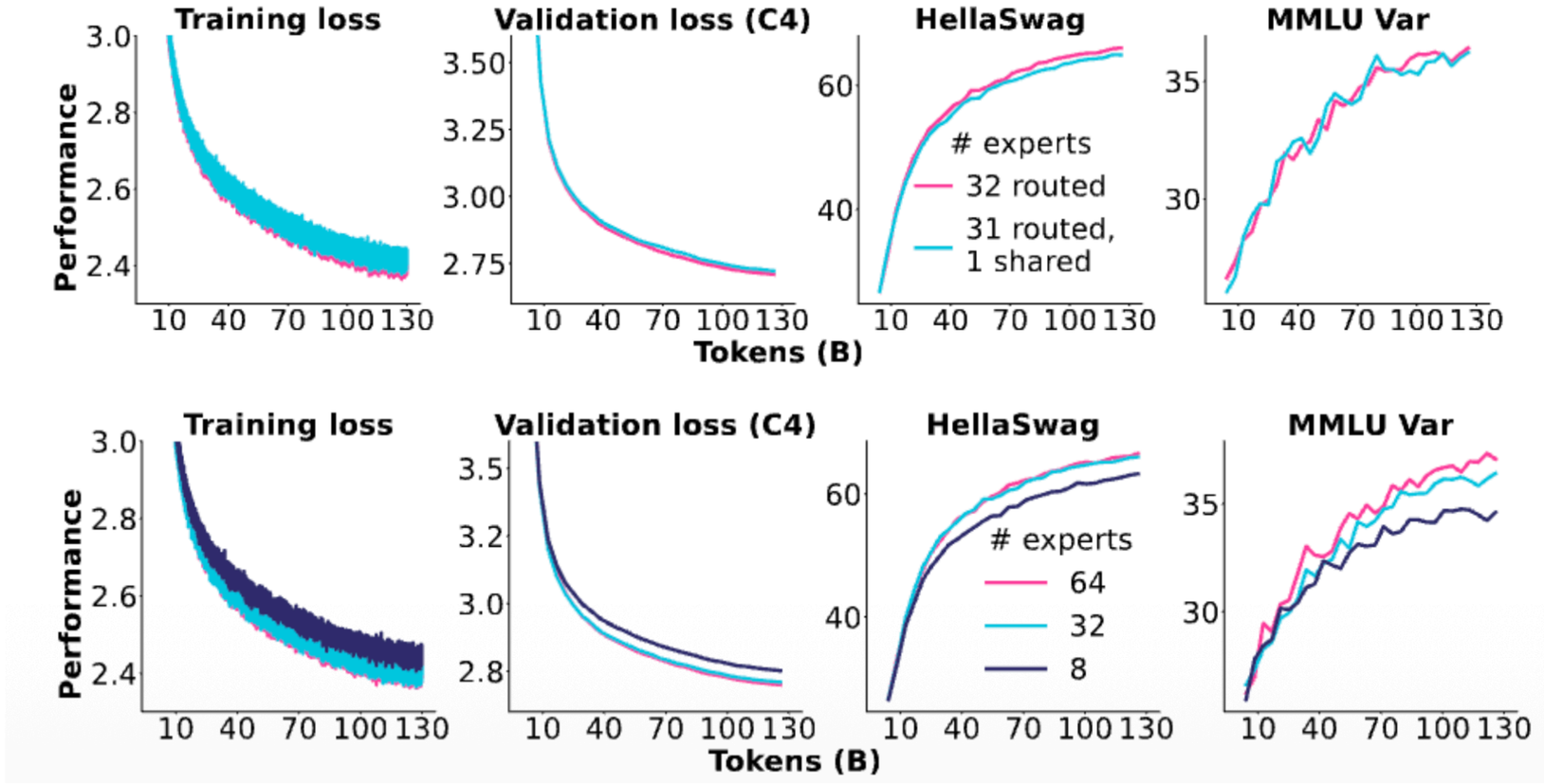


Figure 3 | Ablation studies for DeepSeekMoE. The performance is normalized by the best performance for clarity in presentation. All compared models have the same number of parameters and activated parameters. We can find that fine-grained expert segmentation and shared expert isolation both contribute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

Fine-Grained Routed + Shared Experts

- Ablation from the OLMoE paper
  - Gains from fine-grained experts, but not from shared experts



Expert Setting for Recent MoEs

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-MoE	64	7	1	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

How to Train MoEs?

- Major challenge
  - We need sparsity for training-time efficiency
  - But sparse and discrete gating decisions are not differentiable

  - Under vanilla NTP loss, routing becomes a "rich-get-richer" process
    - Expert Collapse: Early-performing experts receive more gradients, while others remain under-optimized (so not used forever)

- Current solutions
  - Add heuristic load balancing losses

Heuristic Load Balancing Losses [Fedus+ 2022]

- Our goal: the router assigns experts evenly

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

N = # experts

α: balancing hyperparameter

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

- fraction of tokens routed to expert i
- non-differentiable (argmax)

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

- fraction of the router prob. (softmax) for expert i
- differentiable

Heuristic Load Balancing Losses [Muennighoff+ 2024]

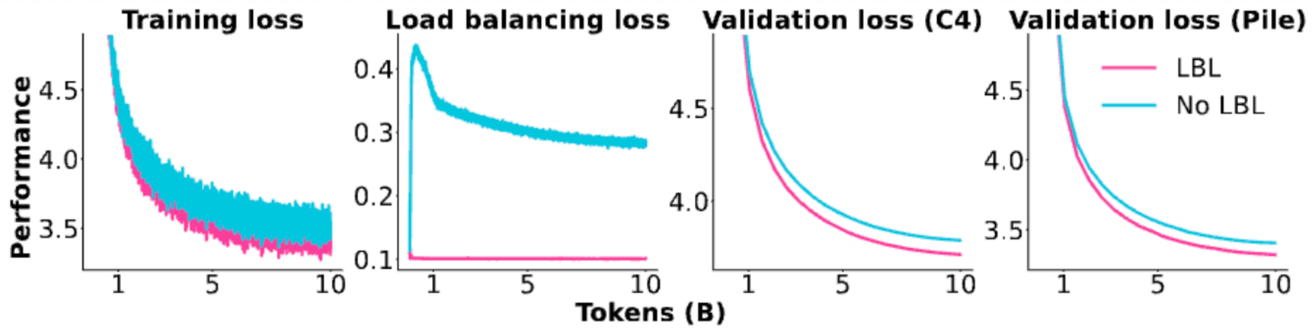


Figure 9: **Impact of applying a load balancing loss (LBL).** The training loss plot excludes the load balancing loss for both models. More results, logs, and configurations: https://wandb.ai/ai2-llm/olmoe/reports/Plot-LBL-vs-No-LBL--Vmlldzo4OTkyNDg4

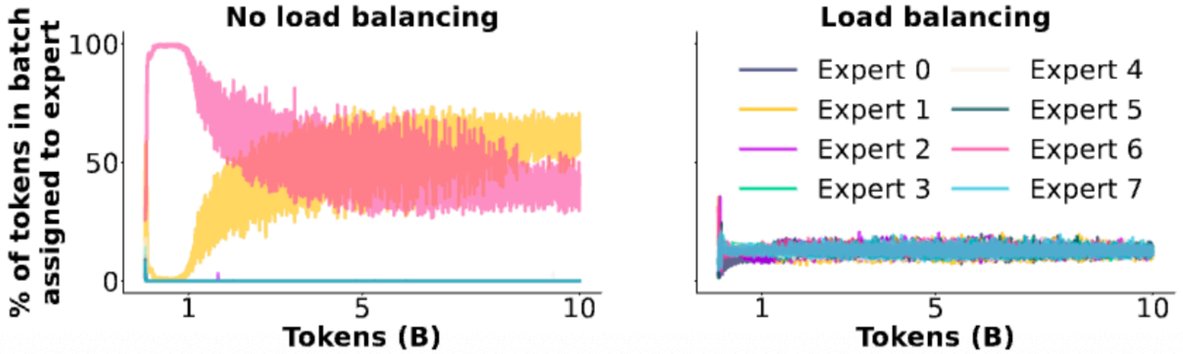


Figure 10: **Expert assignment during training when using or not using a load balancing loss for the first MoE layer.** More results, logs, and configurations: https://wandb.ai/ai2-llm/olmoe/reports/Plot-LBL-vs-No-LBL--Vmlldzo4OTkyNDg4

Issues of MoE: Instability [Zoph+ 2022]

Stabilizing Sparse Models

1. Many methods stabilize sparse models, but at the expense of worse quality.
2. The router z-loss stabilizes models without quality degradation.
3. Transformer modifications with more multiplicative components (GEGLU, RMS normalization) worsen stability, but boost quality.

Issues of MoE: Instability [Muennighoff+ 2024]

- z-loss in MoE helps a lot

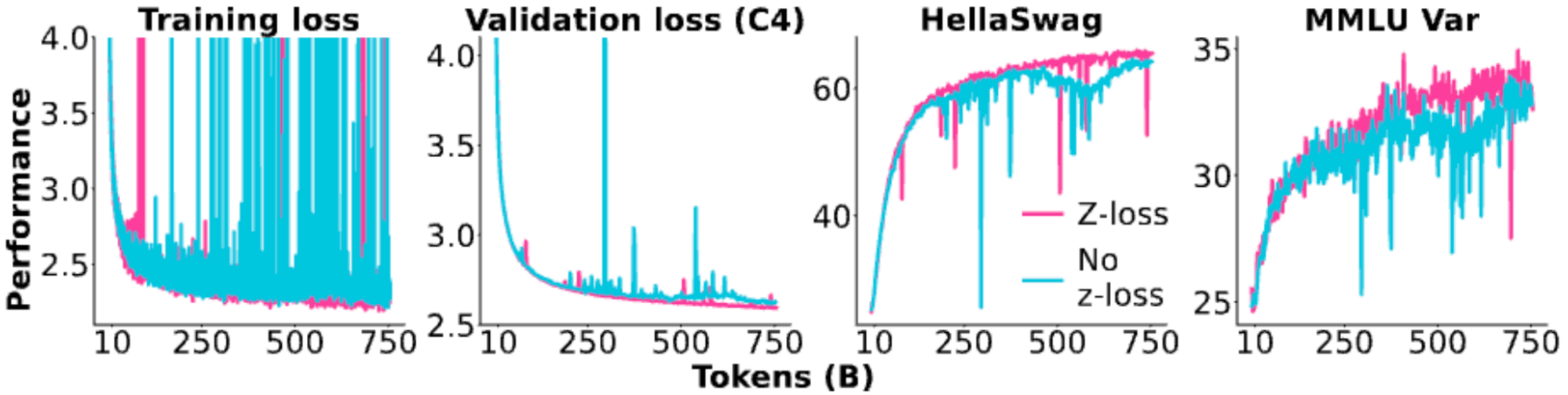


Figure 11: **Router z-loss.** We compare adding router z-loss with a loss weight of 0.001 versus no additional z-loss. More results, logs, and configurations: https://wandb.ai/ai2-llm/olmoe/reports/Plot-Zloss-vs-none--Vmlldzo4NDM4NjUz

Issues of MoE: Overfitting [Zoph+ 2022]

- Sparse MoEs can overfit on smaller fine-tuning dataset

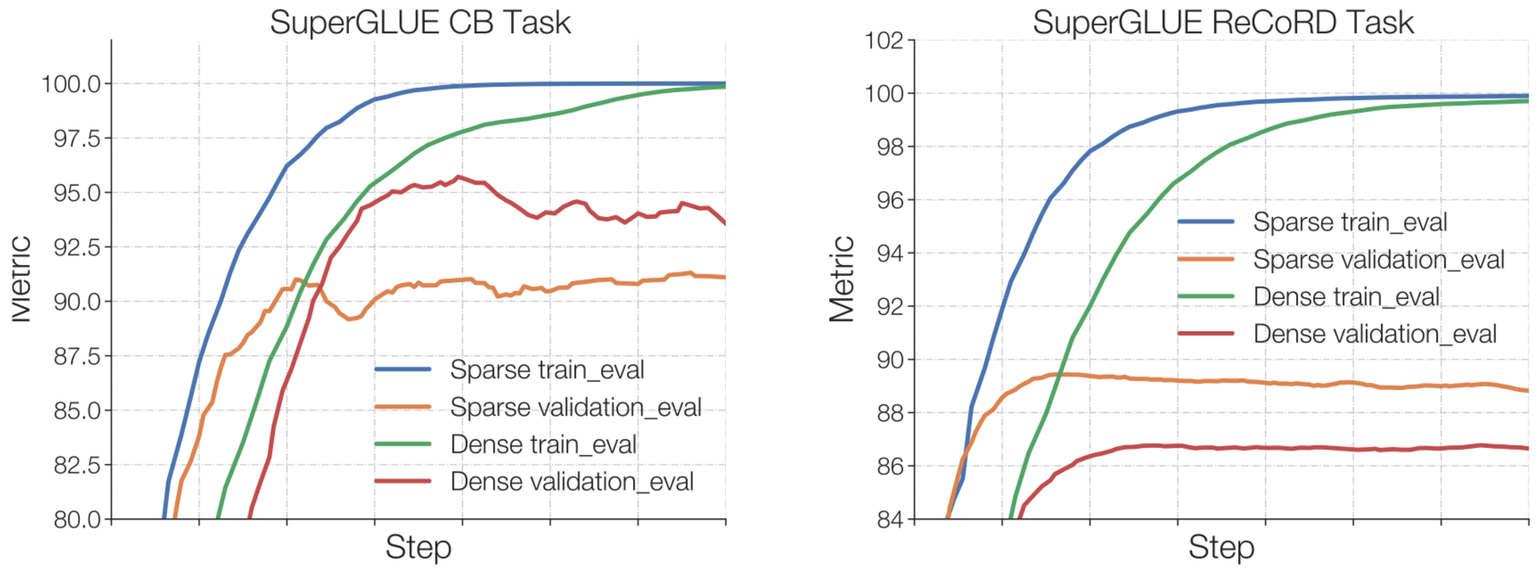


Figure 3: **Sparse models are prone to overfit.** We plot train and validation curves for our ST-MoE-L and a dense-L models fine-tuned on the CB task (250 train sequences) and ReCoRD (138k train sequences). In both cases, the sparse model learns more quickly on the train partition (blue exceeds green line). However, for the smaller CB task, the dense model outperforms the sparse model on the held-out validation set (red vs. orange). In contrast, on the larger ReCoRD task, the sparse model outperforms the dense model by several percentage points.

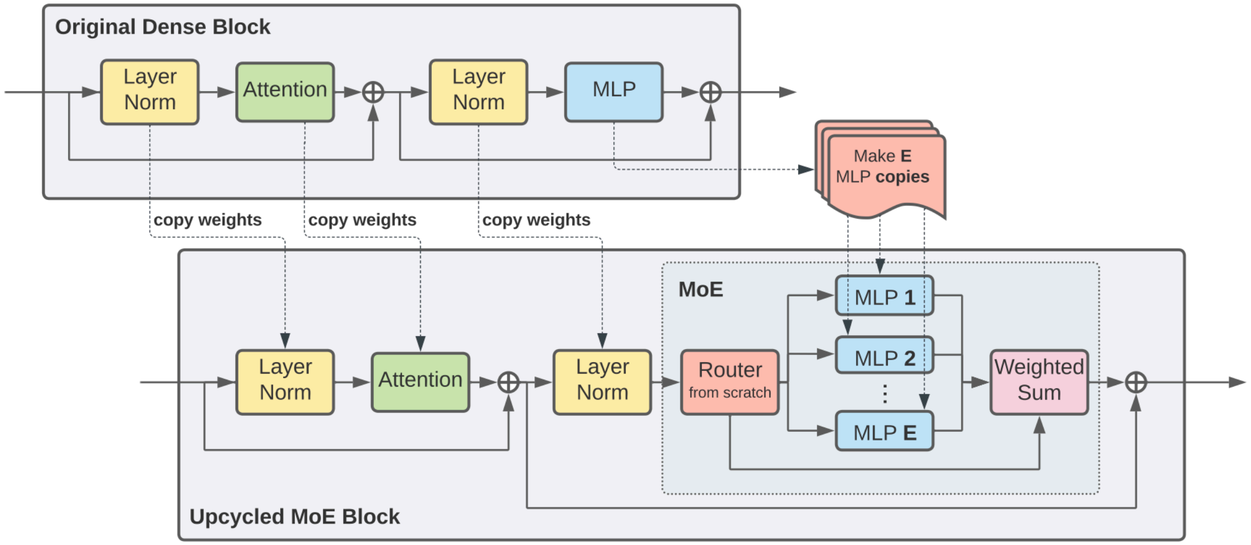
Issues of MoE: Overfitting [Zoph+ 2022]

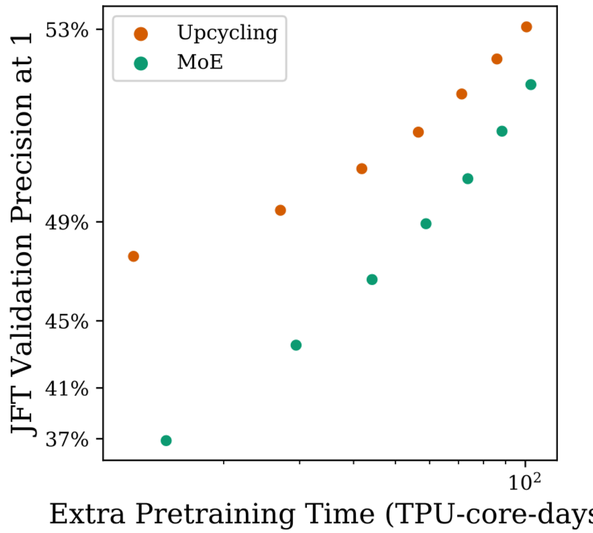
- Sparse MoEs can overfit on smaller fine-tuning dataset

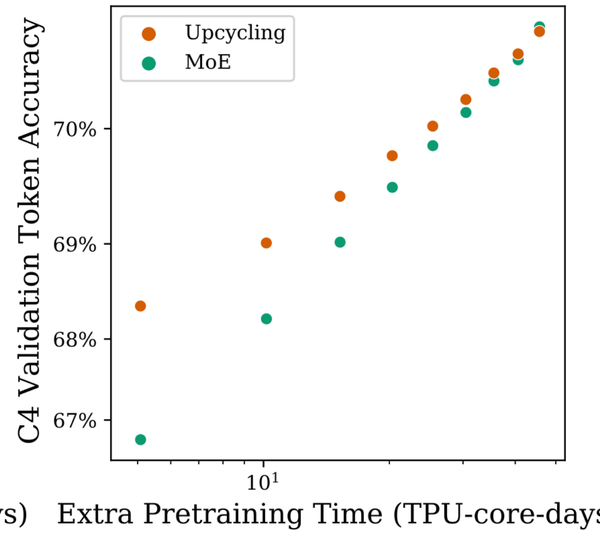
- Solution?
  - Fine-tune non-MoE MLP layers only
  - Regularization (expert dropout)
  - Use lots of dataset

Sparse Upcycling [Komatsuzaki+ 2022]

- Can we train MoE models from dense checkpoints? - Yes!







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
  - Although there are some issues of MoE