Mixture of Experts

Sanjay Adhikesaven
Yuezhou Hu

Why do we need MoE?

- Larger models are stronger, but slower; small models are faster, but weaker.
- What if we have a sparsified large model?

In par

- Wei
- Tra
    -
    -
    -

A Simple and Effective Pruning Approach for Large Language Models

Mingjie Sun[1*] **Zhuang Liu**[2*] **Anna Bair**[1] **J. Zico Kolter**[1,3]
[1]Carnegie Mellon University [2]Meta AI Research [3]Bosch Center for AI

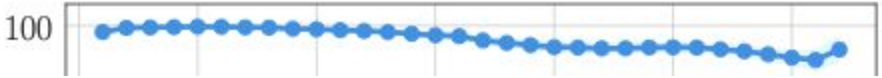
Abstract

As their size increases, Large Languages Models (LLMs) are natural candidates for network pruning methods: approaches that drop a subset of network weights while striving to preserve performance. Existing methods, however, require either retraining, which is rarely affordable for billion-scale LLMs, or solving a weight reconstruction problem reliant on second-order information, which may also be computationally expensive. In this paper, we introduce a novel, straightforward yet effective pruning method, termed Wanda (Pruning by **W**eights **and a**ctivations), designed to induce sparsity in pretrained LLMs. Motivated by the recent observation of emergent large magnitude features in LLMs, our approach prunes weights with the smallest magnitudes multiplied by the corresponding input activations, on a *per-output* basis. Notably, Wanda requires *no* retraining or weight update, and the pruned LLM can be used *as is*. We conduct a thorough evaluation of our method Wanda on LLaMA and LLaMA-2 across various language benchmarks. Wanda significantly outperforms the established baseline of magnitude pruning and performs competitively against recent method involving intensive weight update. Code is available at `https://github.com/locuslab/wanda`.

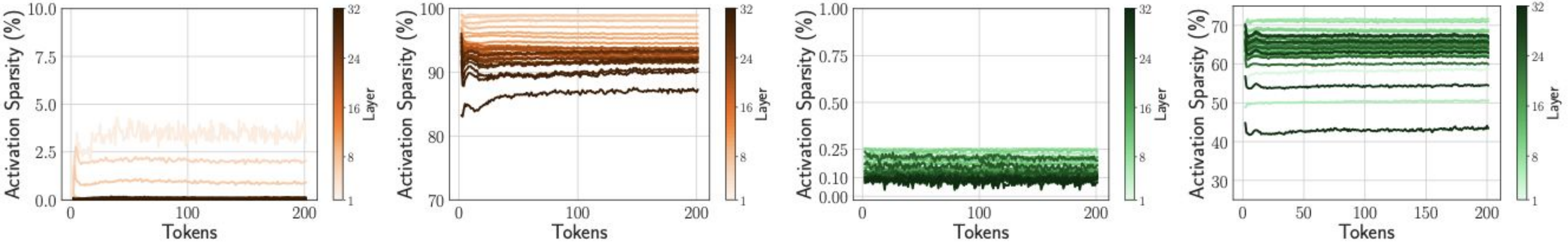
What i

- Spa ... ons

ReLU Strikes Back:
Exploiting Activation Sparsity in Large Language Models



Iman M



(a) Falcon 7B (GELU) (b) Falcon 7B (ReLU) (c) Llama 7B (SiLU) (d) Llama 7B (ReLU)

Figure 4: Activation sparsity of Falcon and Llama models improves significantly after *relufication*.

tion fu
compu
inference step, where efficiency is paramount. Exploring sparsity patterns in ReLU-based LLMs, we unveil the reutilization of activated neurons for generating new tokens and leveraging these insights, we propose practical strategies to substantially reduce LLM inference computation up to three times, using ReLU activations with minimal performance trade-offs.

In summary, we need...

- Activation sparsity
- Structured sparsity
- Or even better, channel level sparsity (sparsify on the hidden dimension)
- Then, we can combine the dense channels in an independent linear layer to be…

Congratulations! You've invented MoE

$$\text{MoE module}(x) = \sum_{i \in \text{Top}-k(r(x))} \text{softmax}\left(r(x)\right)_i E_i(x)$$

- MoE = sparse FFN layer: replace one FFN with many experts + router
- Capacity >> FLOPs: many total params, few active per token

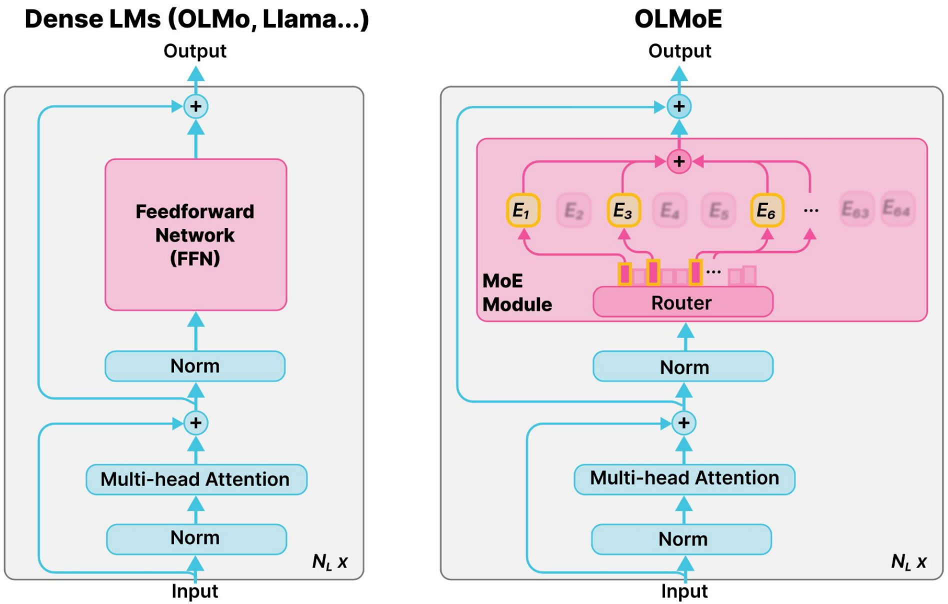
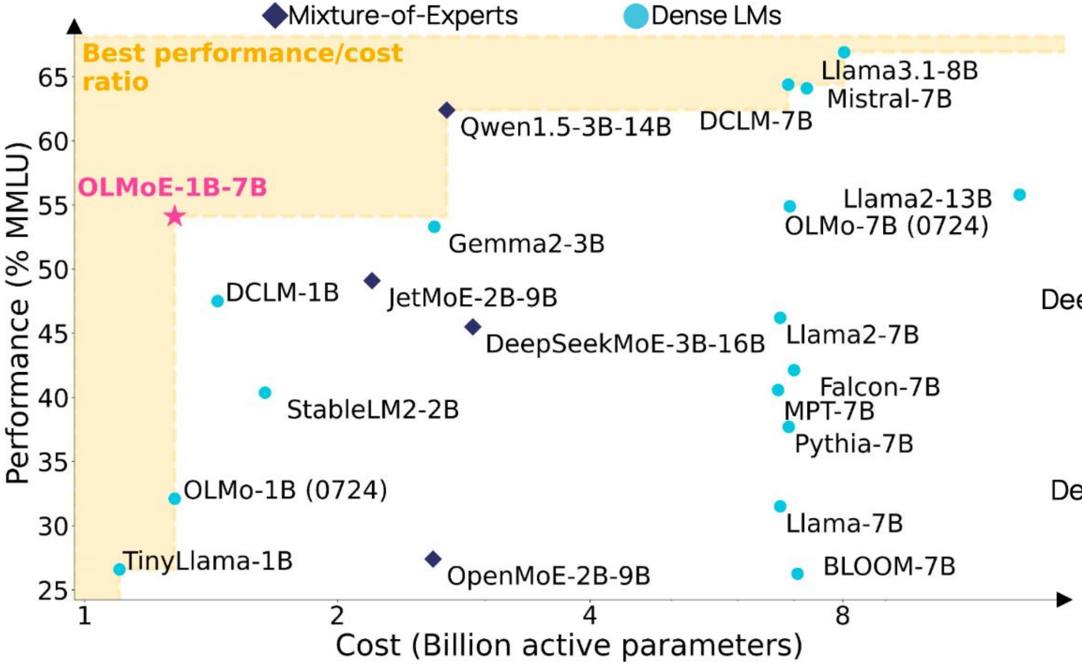


Figure 2: **Comparison of the architecture of dense LMs and MoE models like OLMoE.** The figure excludes some details, e.g., **OLMoE-1B-7B** also uses QK-Norm (§4.2.5).

OLMoE: Open Mixture-of-Experts Language Models



How open are open MoEs?

Name	Model	Data	Code	Logs	#ckpts
Grok-86B-314B	✔	✗	✗	✗	1
Mixtral-39B-141B	✔	✗	✗	✗	1
DBRX-36B-132B	!!	✗	✗	✗	1
Skywork-22B-146B	!!	✗	✗	✗	1
DeepSeekV2-21B-236B	!!	✗	✗	✗	1
Arctic-17B-480B	✔	!!	✗	✗	1
Qwen2-14B-57B	✔	✗	✗	✗	1
Mixtral-13B-47B	✔	✗	✗	✗	1
Jamba-12B-52B	✔	✗	✗	✗	1
DeepSeekMoE-3B-16B	!!	✗	✗	✗	1
Qwen1.5-3B-14B	!!	✗	✗	✗	1
OpenMoE-3B-9B	✔	✔	✔	✗	6
JetMoE-2B-9B	✔	!!	!!	✗	1
OLMoE-1B-7B	✔	✔	✔	✔	**244**

Figure 1: **Performance, cost, and degree of openness of open MoE and dense LMs.** Model names contain rounded parameter counts: `model-active-total` for MoEs and `model-total` for dense LMs. `#ckpts` is the number of intermediate checkpoints available. We highlight MMLU as a summary of overall performance; see §3 for more results. **OLMoE-1B-7B** performs best among models with similar active parameter counts and is the most open MoE.

Key decisions in designing an MoE model

- Determining the number of activated and total parameters
- The design of the experts (e.g., granularity, whether or not to include shared experts)
- The choice of the routing algorithm
- Initializing from a dense model (sparse upcycling)
- Changing the training objective, such as including auxiliary load balancing and router z-losses

OLMoE's Choice

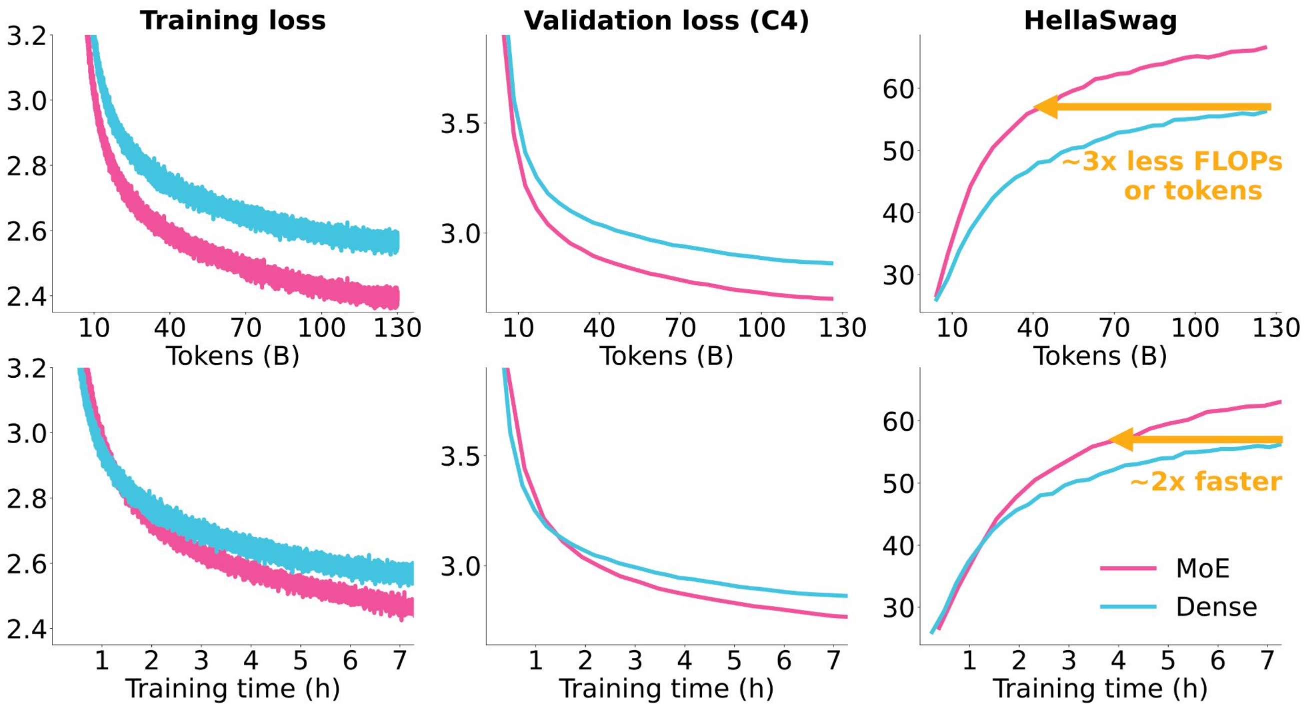
- 1.3B active parameters out of a total of 6.9B
- 8 activated experts out of 64 per layer
- Dropless token choice routing
  - For each input token, the learned router network determines 8 experts to process it
- Train OLMoE from scratch
- Two auxiliary losses
  - Load balancing loss
  - Router z-loss

$$\mathcal{L} = \mathcal{L}_{CE} + \alpha \mathcal{L}_{LB} + \beta \mathcal{L}_{RZ}$$

Mixture-of-Experts vs. Dense

- Compared to dense LM with equivalent active parameters:
  - OLMoE reaches the performance of the dense model with ~3×fewer tokens equivalent to ~3×less compute measured in FLOPs
  - But processes fewer tokens per second than the dense model (23,600 tokens per second per GPU for the MoE vs. 37,500 for dense)
  - Thus reaches only ~2×faster

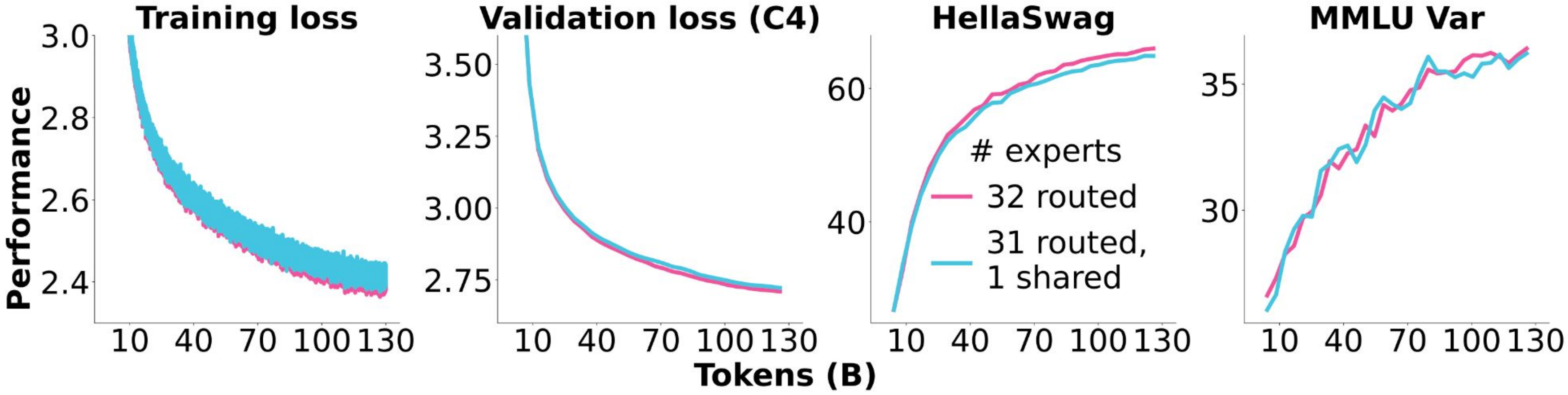
Mixture-of-Experts vs. Dense



Shared Experts

- Shared experts
  - Training with a shared/fixed expert that is always used in addition to the routed experts.
- Compare 1 shared expert + 1 routed expert vs. 2 routed experts
-
-
- Conclusion: Sharing an expert removes flexibility from the model; allowing for more expert combinations improves performance.
-

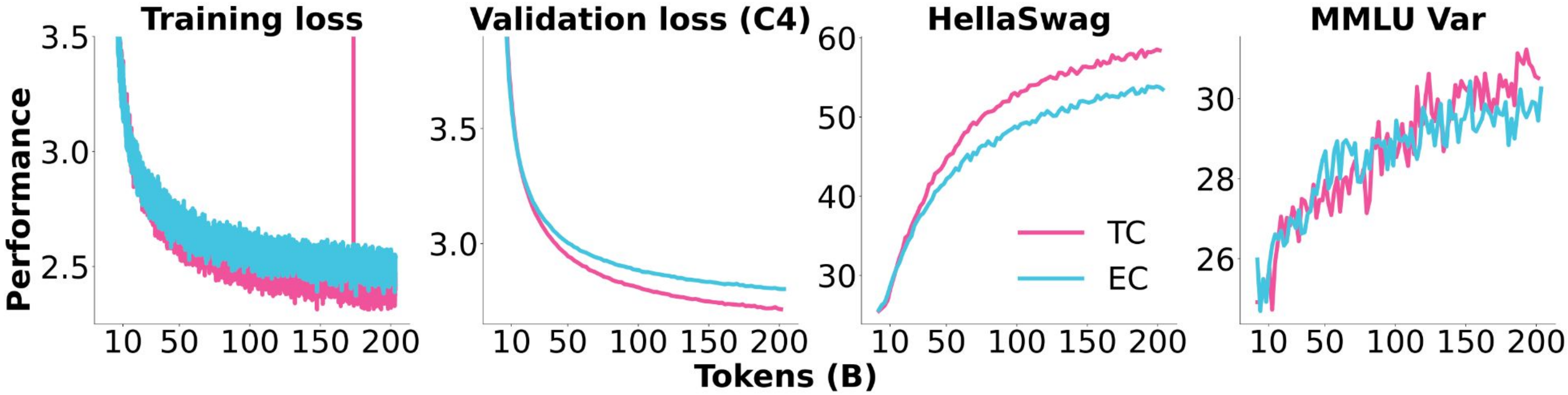
Shared Experts



Expert Choice vs. Token Choice

- For EC, each expert selects a fixed number of tokens from the incoming sequence.
  - Advantage: each expert processing the same number of tokens
  - not easily usable for autoregressive generation where a single token is processed at each step rather than the entire sequence in one
  - EC can lead to token dropping, where some tokens are not selected by any expert
  -
- For TC, each token selects a fixed number of experts.
  - This can lead to many tokens choosing the same expert, hurting training efficiency.
  - It is common to use TC with a load balancing loss to encourage equal distribution.
-

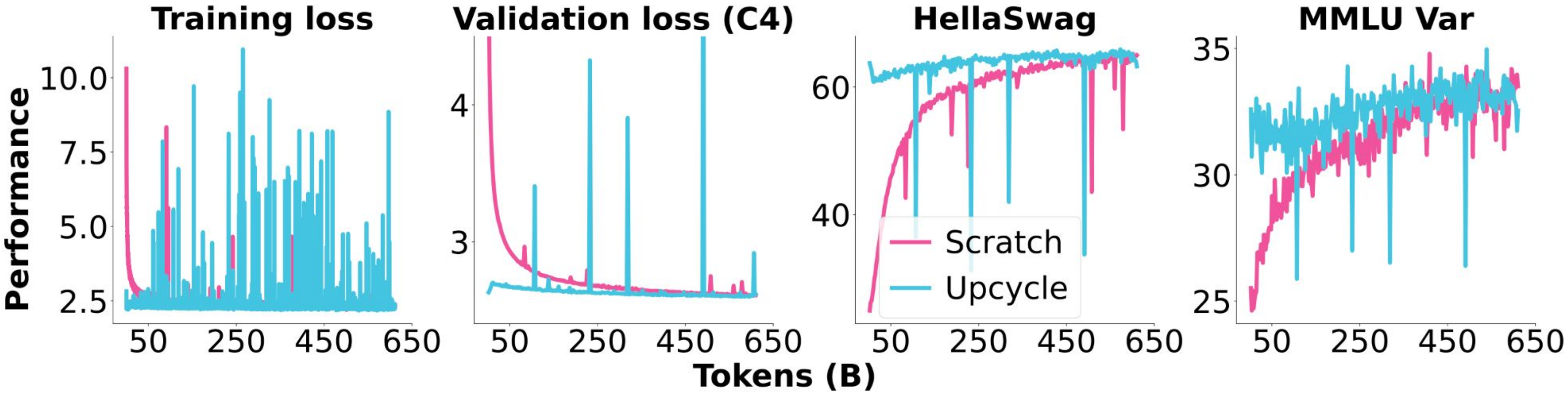
Expert Choice vs. Token Choice



Sparse Upcycling

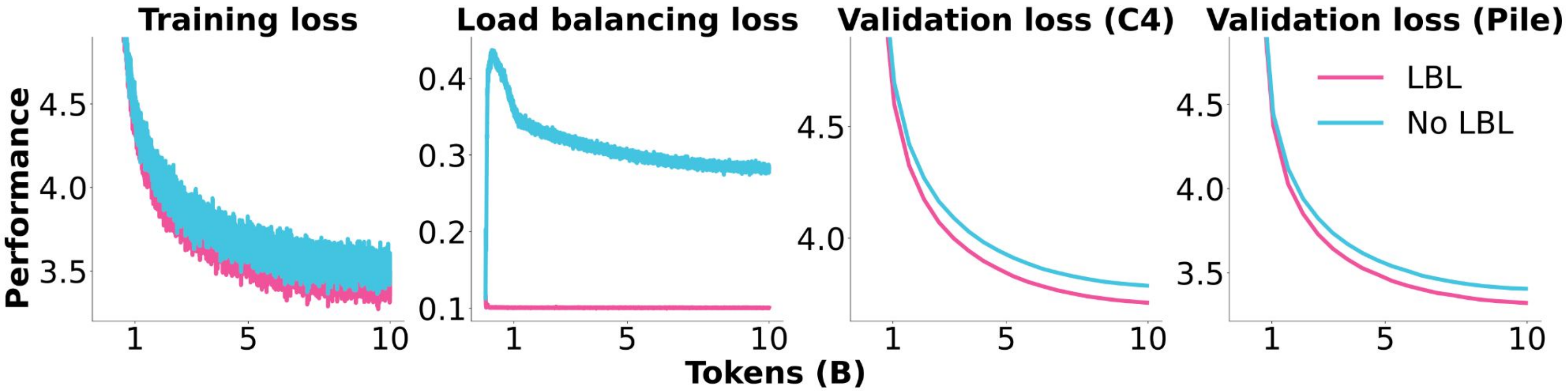
- Turning a dense model into a Mixture-of-Experts model via sparse upcycling:
  - (1) The dense MLP is cloned for each desired expert to constitute MoE layers.
  - (2) A newly initialized router is added in front of each MoE layer.
  - (3) Pretraining continues with the new model so that the cloned MLPs can gradually specialize in different things and the router can be learned.
  - With 120% tokens can train-from-scratch match sparse upcycling
- It only requires 25% of the compute budget of the original dense model to catch up as opposed to the 120% reported
  - Dense model has already been significantly overtrained; its parameters are likely already in a very optimal range for a dense model, which may limit the amount of additional exploration possible after upcycling.
- OLMoE do not use upcycling

Sparse Upcycling



Load Balancing Loss

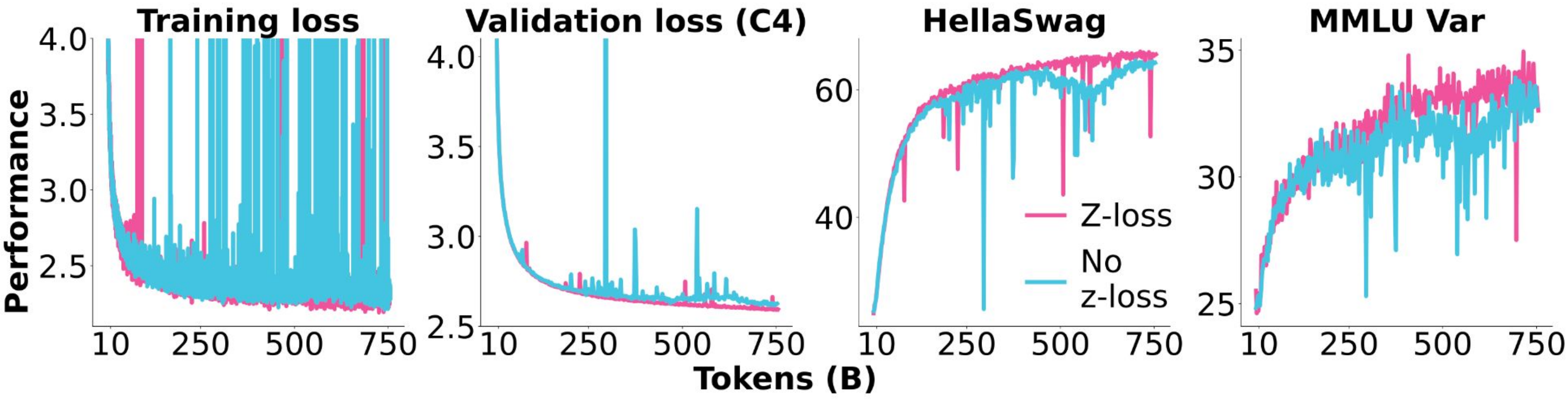
$$\mathcal{L}_{LB} = N_E \cdot \sum_{i=1}^{N_E} f_i \cdot P_i$$



Router Z-loss

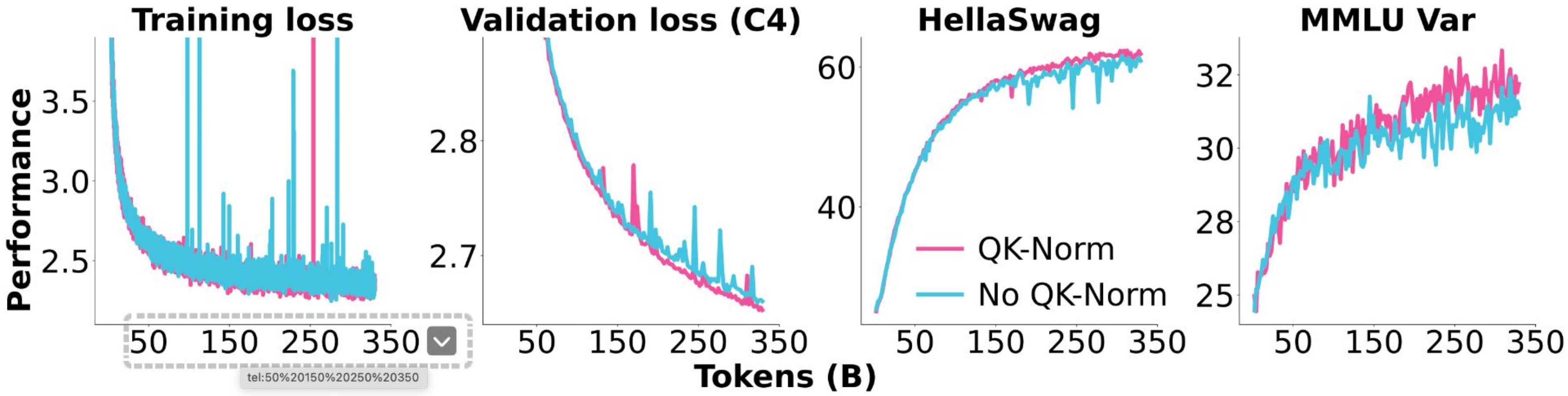
$$\mathcal{L}_{RZ}(x) = \frac{1}{B} \cdot \sum_{i=1}^{B} \left(\log \sum_{j=1}^{N_E} \exp(x_j^{(i)}) \right)^2$$

- Improve both the stability and quality of MoE models
- Penalizes large logits coming into the gating network



QK-Norm

- Layer normalization after the query and key projections ("QK-Norm")
- Prevent the subsequent attention operation from leading to very large logits that may lead to numeric overflows and destabilize the



Analysis to OLMoE

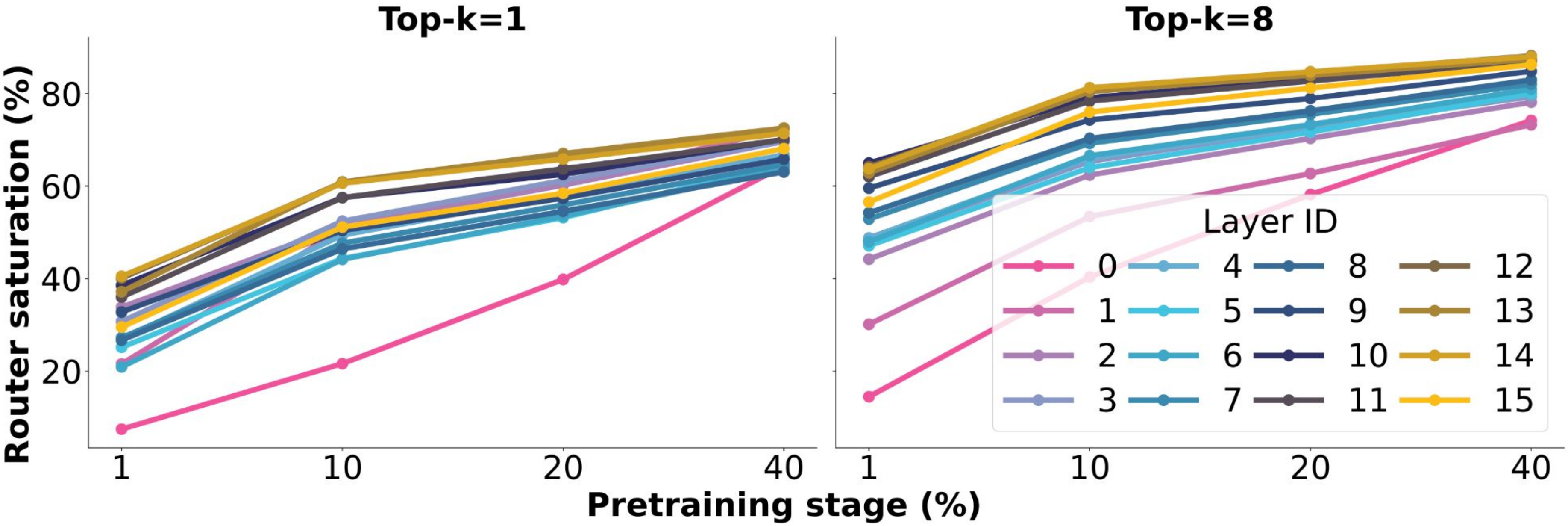
Router Saturation

$$\text{Router Saturation}(t) = \frac{1}{N}\sum_{i=1}^{N}\frac{|\mathcal{E}_i^{(t)} \cap \mathcal{E}_i^{(T)}|}{k},$$

- After 1% of pretraining (5000 steps or 20B tokens), up to~60% of routing to the top-8 activated experts has already saturated
- At 40% of pretraining, saturation reaches up to~80%
- However, which top-1 expert has the highest routing probability saturates slower
- We find that routing in later layers saturates earlier during pretra… …ly than …

- k: The number of top-k experts activated per input token. While we train with $k = 8$ (§2), we also analyze $k = 1$ by only looking at the expert with the highest routing probability.
- $\mathcal{E}_i^{(t)}$: The set of k experts activated for the ith token at the tth checkpoint.
- $\mathcal{E}_i^{(T)}$: The set of k experts activated for the ith token at the final checkpoint T.
- $|\mathcal{E}_i^{(t)} \cap \mathcal{E}_i^{(T)}|$: The number of common experts activated for the ith token between the tth and final checkpoints.

Router Saturation



Expert Co-activation

$$\text{Expert co-activation}(E_i, E_j) = \frac{N_{E_i,E_j}}{N_{E_i}}, \tag{6}$$

where:

- E_i: The first expert.
- E_j: The second expert.
- N_{E_i,E_j}: The number of times experts E_i and E_j are activated together.

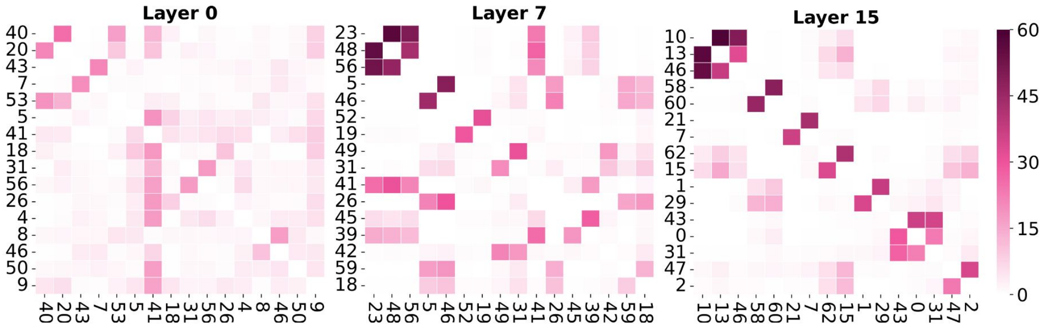


Figure 21: **Co-activation among experts of OLMoE-1B-7B on a random 0.5% of the C4 validation data.** We display the 32 experts with the highest maximum co-activation score via their expert IDs on the x- and y-axis.

Domain Specialization

$$\text{Domain specialization}(E_i, D) = \frac{N_{E_i,D}^{(k)}}{N_D}, \tag{7}$$

where:

- E_i: The ith expert in the model.
- D: The domain from which the data originates.
- k: The number of experts considered (e.g., $k = 8$ means considering the top 8 experts with the highest routing probabilities).
- $N_{E_i,D}^{(k)}$: The number of tokens from domain D for which E_i is among the top-k selected experts.
- N_D: The total number of tokens from domain D processed by the MoE.

Domain Specializ

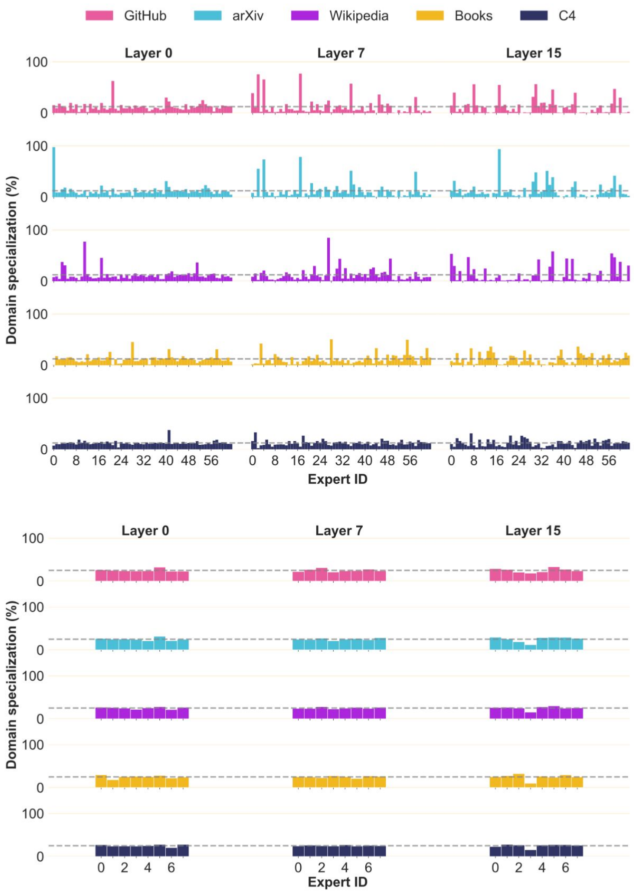


Figure 22: **Domain specialization of OLMoE-1B-7B (top) vs. Mixtral-8x7B (bottom).** We visualize how often tokens from different domains get routed to the 64 (**OLMoE**) or 8 (Mixtral) experts at the end of pretraining. We consider tokens routed to any of the $k = 8$ (**OLMoE**) or $k = 2$ (Mixtral) active experts (Equation 7). Horizontal gray lines correspond to random chance or uniform routing (8/64=12.5% per expert for **OLMoE-1B-7B** with 8 active out of 64 total experts per layer and 2/8=25% for Mixtral with 2 active out of 8 total experts per layer). See Figure 34 for $k = 1$ results.

Vocabulary Specialization

$$\text{Vocabulary specialization}(E_i, x) = \frac{N_{x,E_i}^{(k)}}{N_x}, \tag{8}$$

where:

- E_i: The ith expert in the model.
- x: The token ID being analyzed.
- k: The number of experts considered (e.g., $k = 8$ means considering the top 8 experts with the highest routing probabilities).
- N_{x,E_i}: The number of times input data is routed to E_i for x.
- N_x: The total number of times input data is routed across all experts for x.

Vocabulary Specialization

- Vocabulary specialization is higher in later layers, similar to how later layers saturate earlier
- Earlier layers there is more uncertainty about which token the model will predict

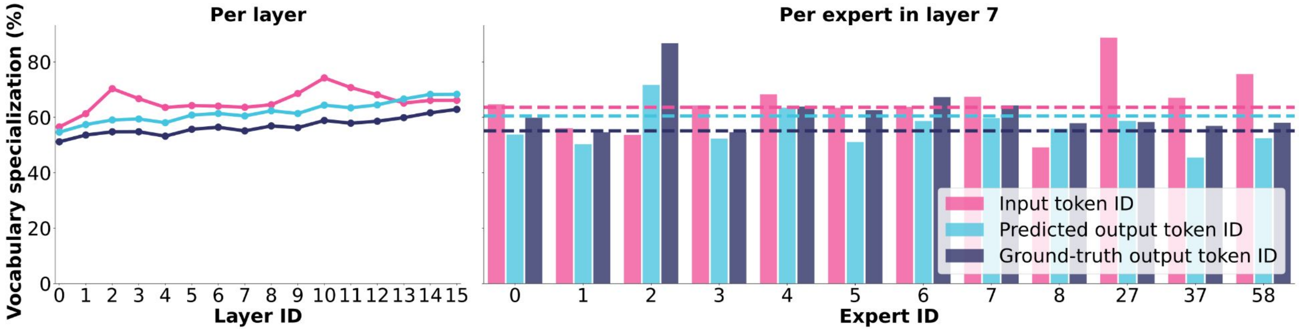


Figure 23: **Vocabulary specialization of OLMoE-1B-7B across layers and experts.** To compute vocabulary specialization per layer (left) we average the specialization of each expert in that layer. Dashed lines (right) correspond to the average of layer 7 as depicted left. We display the first 32 experts out of 64. This plot is for $k = 1$ (Equation 8) and we provide $k = 8$ and a comparison with Mixtral-8x7B in Appendix G.

DeepSeek-V3 Technical Report

DeepSeek-V3 Overview

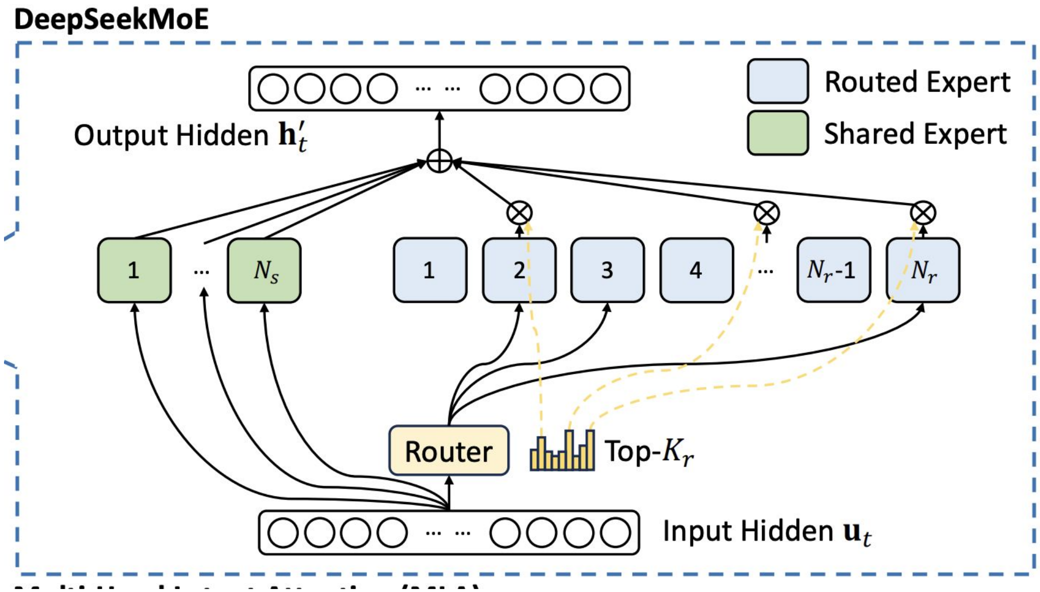
MoE is great for high capacity with low compute per token, but some issues include load balancing as well as communication between nodes

Prior Work: OLMoE includes a load-balancing loss – this can hurt quality at times if router optimizes for balancing over routing

DeepSeek-V3: loss-free balancing + node-limited routing – total of 671B params with 37B active per token

DeepSeek-V3 Architecture

- DeepSeek MoE block replaces FFN layers for all layers except first 3
- Has shared FFN path + routed experts (256 total, 8 activated)
- Experts are sharded across devices
- Motivates why balancing + comms b/w devices matter



Auxiliary-Loss-Free Load Balancing

Benchmark (Metric)	# Shots	Small MoE Aux-Loss-Based	Small MoE Aux-Loss-Free	Large MoE Aux-Loss-Based	Large MoE Aux-Loss-Free
# Activated Params	-	2.4B	2.4B	20.9B	20.9B
# Total Params	-	15.7B	15.7B	228.7B	228.7B
# Training Tokens	-	1.33T	1.33T	578B	578B
Pile-test (BPB)	-	0.727	**0.724**	0.656	**0.652**
BBH (EM)	3-shot	37.3	**39.3**	66.7	**67.9**
MMLU (EM)	5-shot	51.0	**51.8**	**68.3**	67.2
DROP (F1)	1-shot	38.1	**39.0**	**67.1**	**67.1**
TriviaQA (EM)	5-shot	**58.3**	**58.5**	66.7	**67.7**
NaturalQuestions (EM)	5-shot	**23.2**	**23.4**	27.1	**28.1**
HumanEval (Pass@1)	0-shot	22.0	**22.6**	40.2	**46.3**
MBPP (Pass@1)	3-shot	**36.6**	35.8	59.2	**61.2**
GSM8K (EM)	8-shot	27.1	**29.6**	70.7	**74.5**
MATH (EM)	4-shot	**10.9**	**11.1**	37.2	**39.6**

- Adds learnable, per-expert bias used only in top-k gating, updated during training via token counts
- Avoids coupling LM loss with a balancing loss – easier to tune
- Results show that this strategy achieves consistently better results

Expert Specialization Pattern

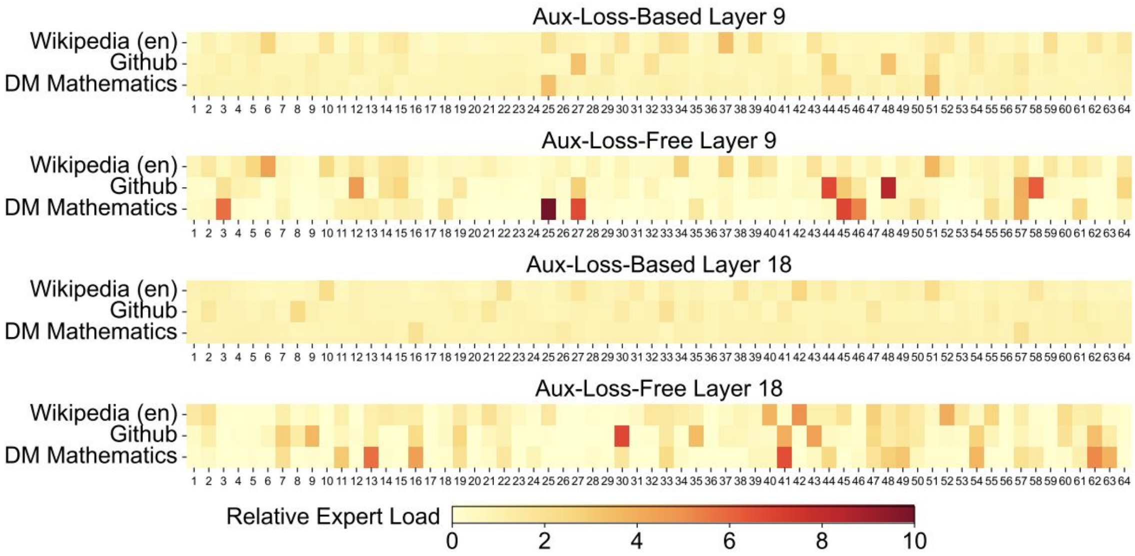


Figure 9 | Expert load of auxiliary-loss-free and auxiliary-loss-based models on three domains in the Pile test set. The auxiliary-loss-free model shows greater expert specialization patterns than the auxiliary-loss-based one. The relative expert load denotes the ratio between the actual expert load and the theoretically balanced expert load. Due to space constraints, we only present the results of two layers as an example, with the results of all layers provided in Appendix C.

Node-Limited Routing

- Problem: each token's activations get sent to experts scattered across many GPUs → higher latency
- Solution: Each token's experts are limited to at most M nodes
  - Selected via the sum of the highest k/M experts per node
  - Keeps only the top M nodes, then picks the top-k experts from those
- DeepSeek V3 limits this to M = 4, limits communication costs during training
- Leads to slightly less flexible routing

Branch-Train-MiX:
Mixing Expert LLMs into a MoE LLM

Why BTX?

Problem: training LLMs to perform well across multiple specialized domains in a synchronized manner is costly and hard

Prior Work: BTM (Branch-Train-Merge): branches seed LLM into domain models, trains dense models in parallel, then averages weights into one dense model

BTX Goal: keep BTM's parallel dense expert training, but preserve FFN specialization and add a router

Overview of BTX Method

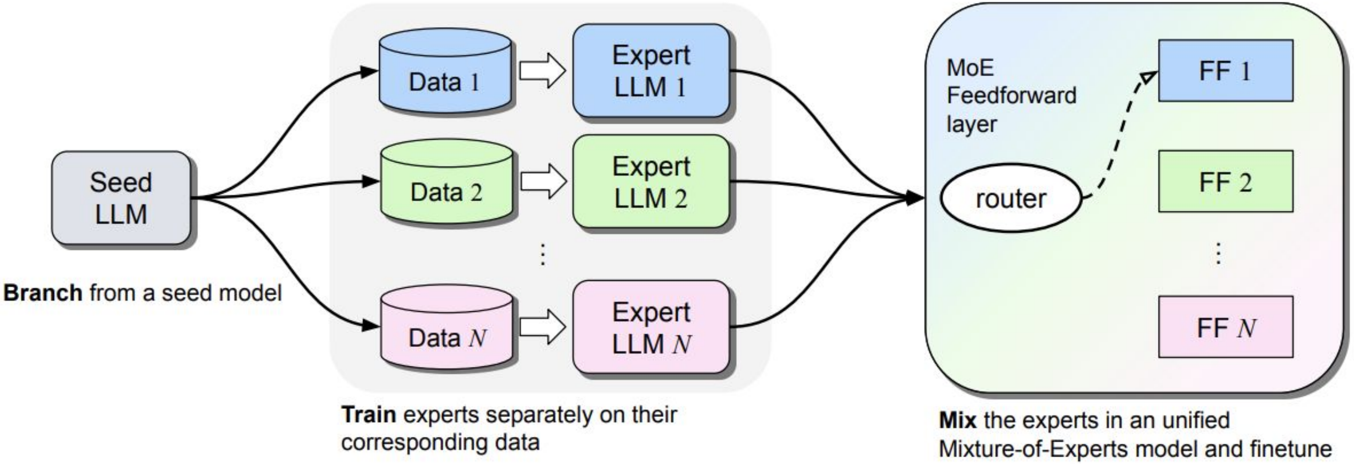


Figure 1 The Branch-Train-MiX (BTX) method has three steps: **1) branch** from a pretrained seed LLM by making multiple copies of it; **2) train** those copies separately on different subsets of data to obtain expert LLMs; **3) mix** those expert LLMs by combining them into a single LLM using mixture-of-experts feedforward (FF) layers, and finetuning the overall unified model.

BTX Choices vs BTM

Component	BTX Choice	BTM Choice
FFN Blocks (per expert)	Kept as experts (routed)	Averaged
Self-attention/embeddings /layer norms	Averaged	Averaged
New Params Introduced	Router	None
Inference Path	Top-k MoE	Single dense model

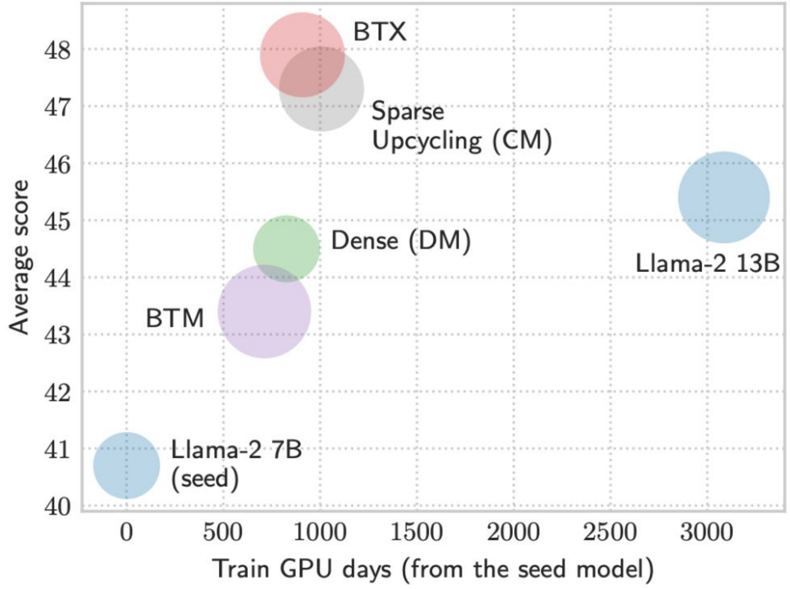
BTX preserves domain FFNs as experts, so tokens choose specialized paths

Router Learning after MiX

- Freeze the shared backbone and train a small router layer which outputs scores from all experts in that layer
- Uses the same LM loss on a mixture of all domains
- Applies load balancing via a standard auxiliary load-balancing loss

Compute-Performance Tradeoff

- At the same compute (X axis), BTX beats other approaches (dense, BTM, sparse-upcycling)
- Preserves domain FFN diversity and keeps BTM's parallel training throughput



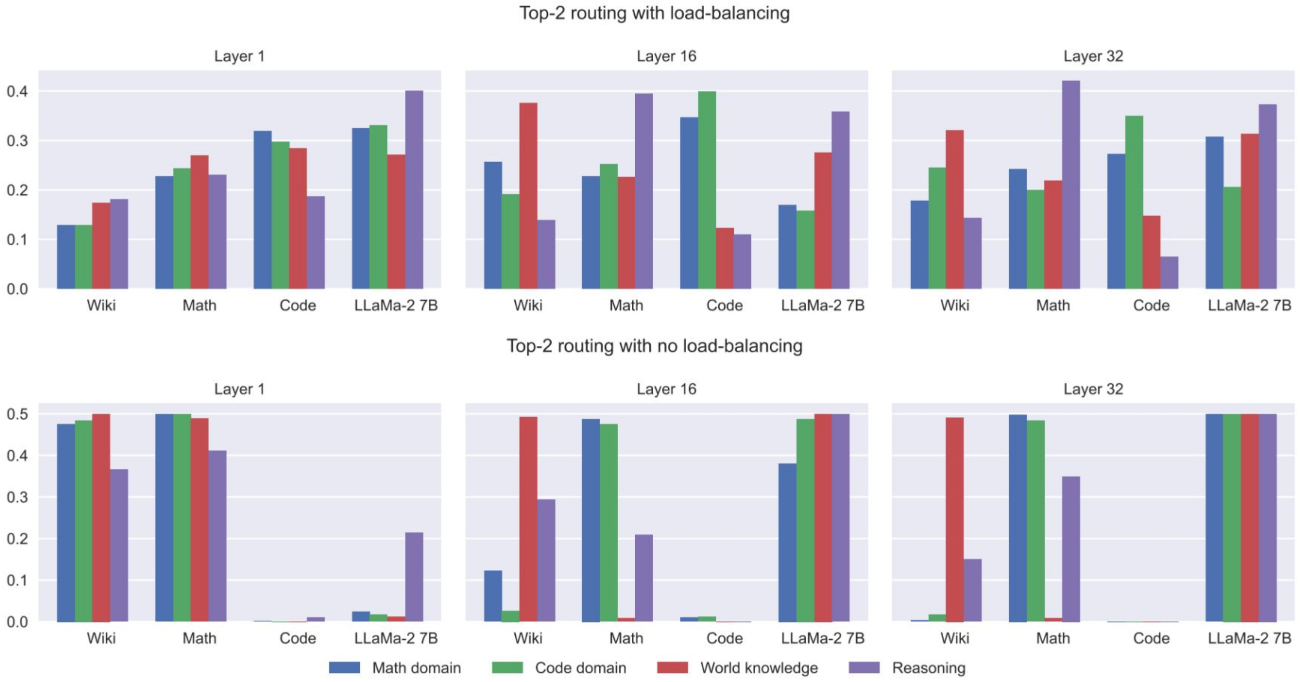
Overall Performance

	Math	Code	Knowledge	Reasoning	MMLU	Average
Specialized LLMs						
CodeLlama 7B	8.1	36.3	22.2	56.6	38.6	37.9
Llemma 7B	28.0	33.5	17.2	38.8	33.5	32.1
Generalist LLMs						
Llama-2 7B	8.6	16.8	37.4	63.3	46.1	40.7
Llama-2 13B	16.3	24.5	40.0	**66.1**	52.8	45.4
Dense (DM)	18.3	25.8	39.6	63.3	49.8	44.5
Sparse upcycling (DM), Top-2	**28.1**	34.7	34.0	62.3	51.1	46.3
BTM, Top-1	21.3	36.4	26.5	61.0	44.3	43.1
BTM, Top-2	21.5	**36.6**	26.9	61.2	44.3	43.4
BTX, Sample Top-1	26.4	31.5	40.1	63.7	**53.2**	47.3
BTX, Top-2	27.4	34.0	**41.0**	63.5	52.5	**47.9**

Table 2 Aggregated performance of BTX compared against various baselines, including both generalist and specialized pretrained models, tested on various capabilities aggregated across popular benchmarks. Dense, sparse upcycling, BTM and BTX are trained on exactly the same amount and mixture of data with the exception that BTM does not have the finetuning stage.

Routing Performance Ablation w/ Load Balancing

- With LB: loads even out, previously “dead” experts such as Code revive
- Without LB: Math expert dominates, others are underused



MoE Tradeoffs

Pros

- High capacity with lower total FLOPs due to sparse expert activation
- Individual experts can learn specialized domains, thus more efficient capacity usage

Cons

- Requires good load balancing across experts
- Communication overhead since experts are sharded across GPUs
- Requires designing the experts, picking a routing method, etc

Critics

Critique of the Branch-Train-Mix (BTX) Paradigm for MoE Models

Donghyun Lee, Téa Wright

10/14

Hand-selected experts

- **Lack of cross-domain learning**
  - We want some specialization but BTX is too siloed!
  - Experts don't share representations across domains
- If we hand pick the experts, are we **missing out on patterns that the model could otherwise discover**?

Expert ID	Input token IDs	Predicted output token IDs
27	© (100%) ŀ (100%) ³ (100%) í (100%) ĭ (100%) ृ (100%) ঁ (100%) क (100%) Ш (100%) ИН (100%) а (100%)	l′ (100%) § (100%) © (100%) ĵ (100%) ij (100%) O (100%) の (100%) ₹ (100%) क (100%) ृ (100%) ल (100%)
58	(“ (100%) (” (100%) ‘ (94%) ’ (92%) “ (92%) ((92%) ” (90%) ’ (89%) “ (88%) $ (87%) [(87%) £ (86%)	such (100%) 486 (100%) see (95%) which (91%) driving (91%) UK (90%) who (88%) including (88%) normal (88%)
7	Him (100%) inde (100%) Jesus (98%) God (90%) pray (81%) Holy (80%) Quran (80%) God (77%) Lord (76%) glory (75%) Spirit (66%) Christ (65%)	rella (100%) Him (94%) sin (90%) prince (80%) glory (72%) Jesus (69%) Lord (68%) Christ (65%) Spirit (55%) Holy (53%) God (50%) Prayer (50%)
37	Sunday (100%) Tuesday (100%) Thursday (100%) Olympic (100%) Christmas (100%) rugby (100%) Championship (100%) weekends (100%)	days (91%) anniversary (90%) month (88%) week (84%) mpi (83%) semester (81%) mand (80%) Olympics (78%) cent (76%) season (76%) perm (75%)
43	Armenian (100%) ijan (100%) enia (96%) Iraq (95%) Iranian (92%) Iran (92%) Saudi (90%) northern (90%) Lebanon (90%) Singapore (88%) Turkey (88%) Asia (87%) Egypt (86%) western (86%)	enia (90%) invasion (80%) Arabia (76%) irregular (66%) regions (64%) border (63%) Kong (61%) ians (61%) bases (60%) Republic (59%) Ireland (58%) Korea (58%) War (55%) Carolina (52%)

Inflexible resource allocation

- BTX fixes capacity per domain, while traditional MoEs flexibly allocate it

Conventional MoE:

learns to *route tokens dynamically*

- **Harder/easier** domains naturally get **more/less** expert parameters
- **adaptive parameter allocation**

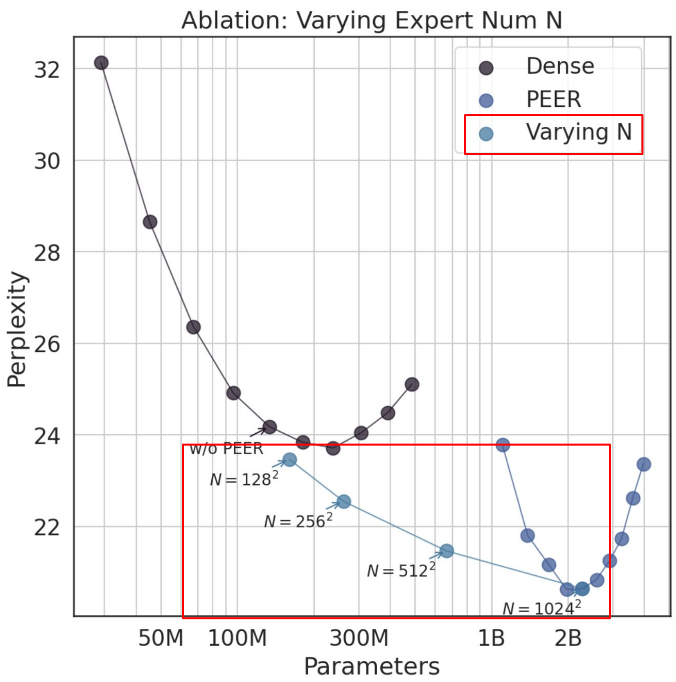
BTX:

each domain is tied to a *single, fixed expert*

- # of parameters per domain is **predefined** and **cannot grow** with task complexity

Scalability concerns

- BTX: Only a handful of total experts (4-8)
- Conventional MoEs can scale to thousands of experts for maximal parameter capacity across domains (e.g., *Mixture of a Million Experts**, *Outrageously Large Neural Networks***)
- How does explicitly assigning experts for thousands of domains scale in practice?
- How does averaging thousands of non-MoE weights impact performance?



*: He, Xu Owen. "Mixture of a million experts." arXiv preprint arXiv:2407.04153 (2024).
**: Shazeer, Noam, et al. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer." arXiv preprint arXiv:1701.06538 (2017).

BTX vs Conventional MoEs pretrained from scratch?

- We don't just want to upcycle a base model.
  We want the **best MoE for our use case**
- **Current baseline:** Upcycling **base** models
- **Missing baseline**: Finetuning a **pretrained MoE** (that already learned emergent expert specialization from scratch)
- **Challenge:** Fair comparison
- **How to test**: BTX on Qwen3 vs. Pruning/upcycling Qwen3MoE?

Proponents

OLMoE: Open Mixture-of-Experts Language Models

DeepSeek-V3 Technical Report

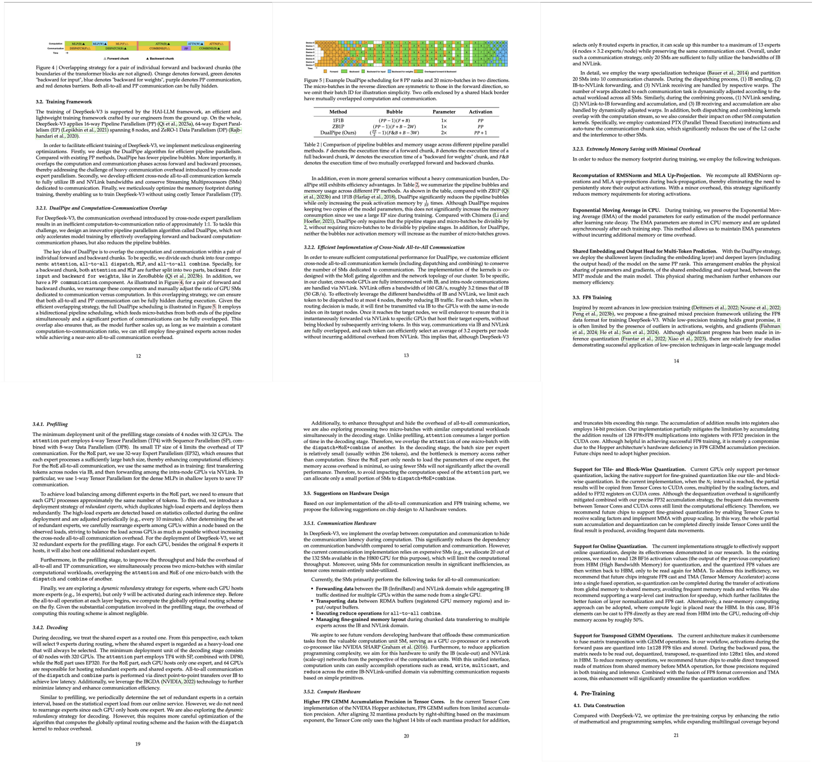
Branch-Train-MiX: Mixing Expert LLMs into a Mixture-of-Experts LLM

FlexOlmo: Open Language Models for Flexible Data Use

Ryan Wang, Kaiwen Hu

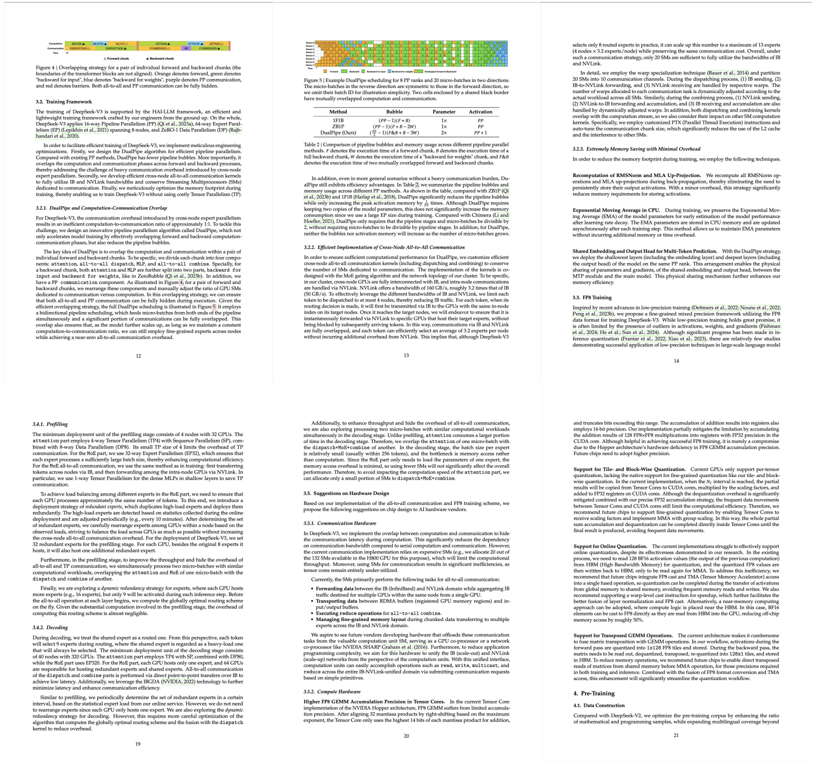
10/14

BTX is easier to train



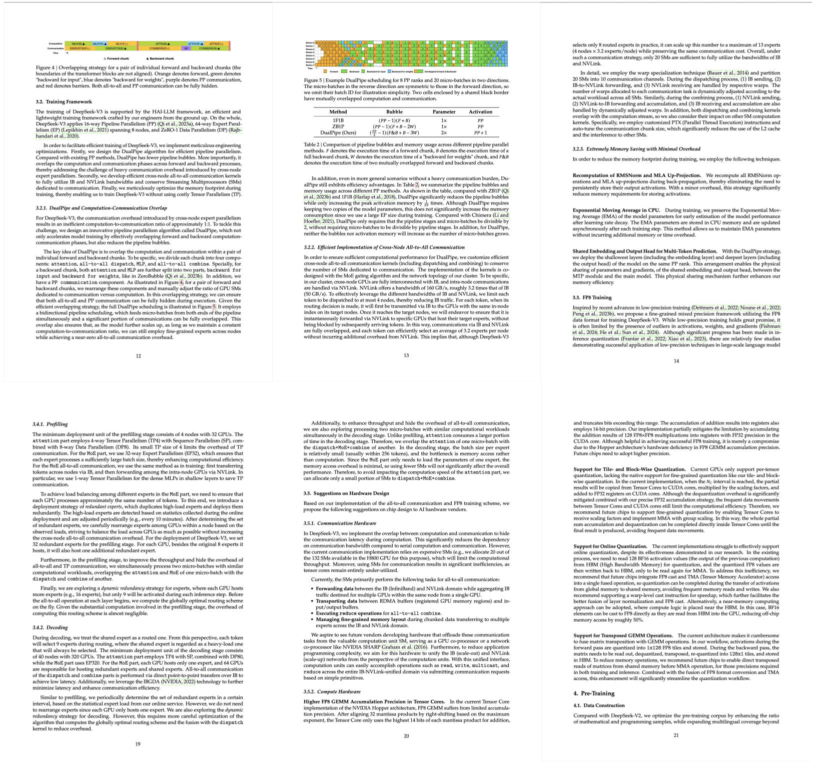
BTX is easier to train

- Parallel training of experts improves efficiency
- More robust since the failure of a single expert training will not affect the whole



BTX is easier to train

- Parallel training of experts improves efficiency
- More robust since the failure of a single expert training will not affect the whole
- Still uses load balancing loss during joint training



BTX paradigm is flexible (easier to deploy and adapt)

Scenario: Amazon customer service

Would you want to use…

BTX paradigm is flexible (easier to deploy and adapt)

Scenario: Amazon customer service

Would you want to use…

Deepseek-V3 (671B model)

BTX paradigm is flexible (easier to deploy and adapt)

Scenario: Amazon customer service

Would you want to use…

Deepseek-V3 (671B model)

BTX model initialized with math + conversational + legal experts

BTX paradigm is flexible (easier to deploy and adapt)

Scenario: Amazon customer service

Would you want to use…

Deepseek-V3 (671B model)

BTX model initialized with math + conversational + legal experts

BTX paradigm is also more natural for continual learning

BTX paradigm opens new possibilities

FlexOlmo: Open Language Models for Flexible Data Use

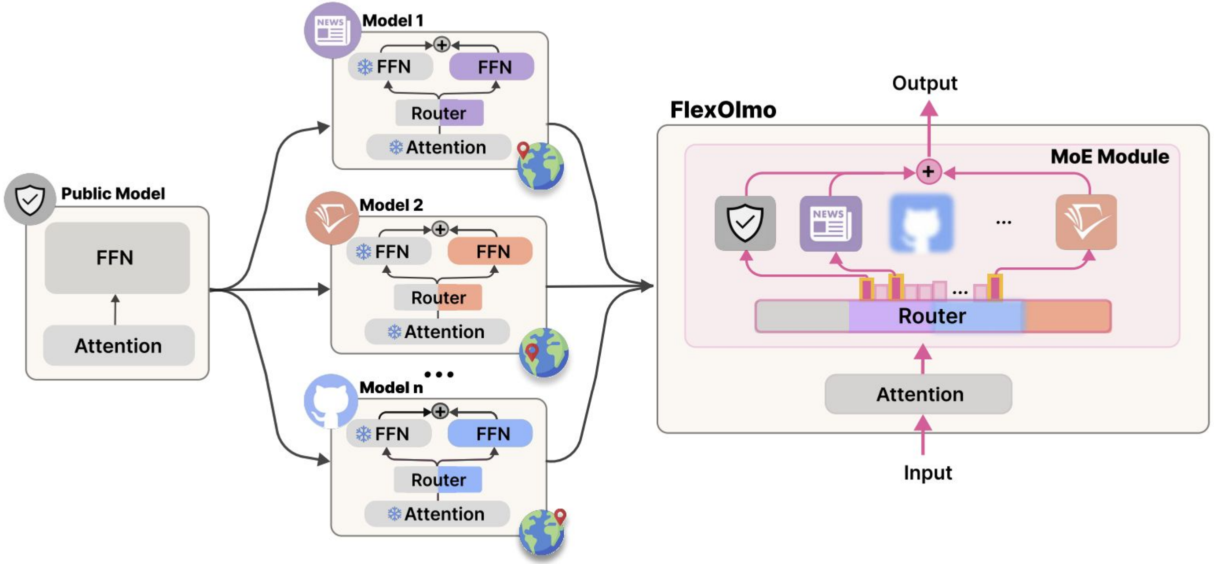


Figure 1: **An overview of FLEXOLMO.** Data owners can contribute without sharing the data by training their own expert modules (FFNs and router embeddings) with a shared public model as an anchor point. At inference, these modules are integrated into a MoE model via a novel router embedding concatenation. This design enables flexible inclusion or exclusion of experts and strict opt-out guarantees, e.g., Github data can be excluded at no cost (blurred) during inference.

BTX paradigm opens new possibilities

FlexOlmo: Open Language Models for Flexible Data Use

Shared public model

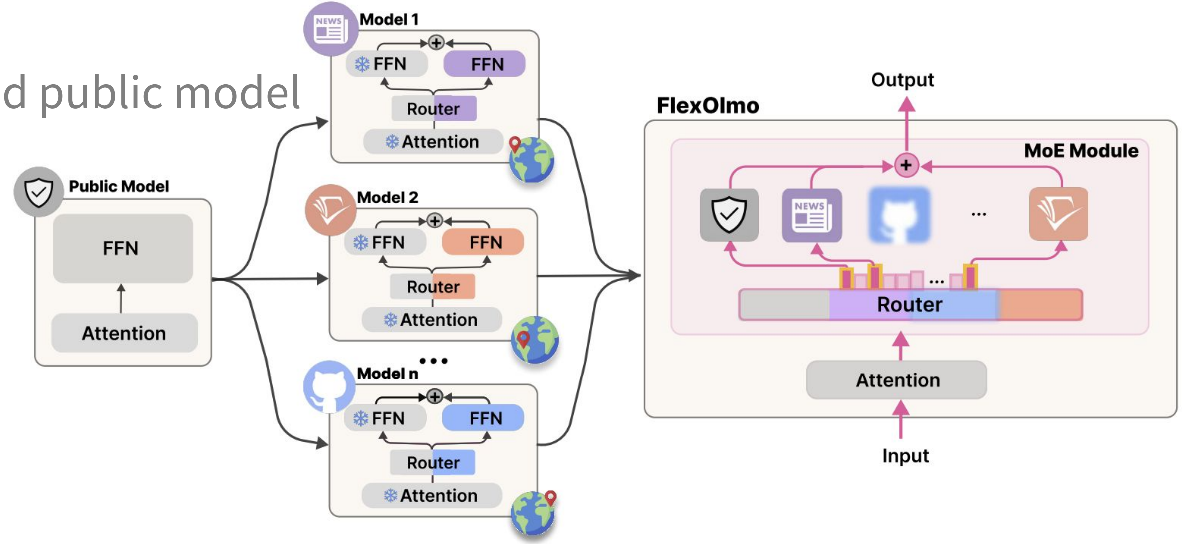


Figure 1: **An overview of FlexOlmo.** Data owners can contribute without sharing the data by training their own expert modules (FFNs and router embeddings) with a shared public model as an anchor point. At inference, these modules are integrated into a MoE model via a novel router embedding concatenation. This design enables flexible inclusion or exclusion of experts and strict opt-out guarantees, e.g., Github data can be excluded at no cost (blurred) during inference.

Merging router embeddings

$$\mathbf{W}_r = \begin{bmatrix} - & \mathbf{r}_{\text{pub}} & - \\ - & \mathbf{r}_1 & - \\ & \vdots & \\ - & \mathbf{r}_n & - \end{bmatrix}, \text{ where } \mathbf{r}_i = \frac{1}{|S_i|} \sum_{d_k \in S_i} \mathbf{E}(d_k) \in \mathbb{R}^h, S_i \subset D_i.$$

FlexOlmo vs BTX

- Shared public model
    - Prevents the expert models from deviating too much from the public model
- Separate routers during private training
    - No need for continual training for routers
    - No need for a load balancing loss
- Focuses on data privacy

Concerns from the critics

- Inflexible resource allocation
    - Domain does not necessarily refer to a subject
    - For harder domains, take more data sources → more experts
    - Scale up the expert number by selecting multiple data sources
- Fair comparison
    - A reasonable future direction!
    - Need to control model size and pretraining data to ensure fairness