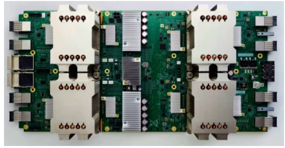
Efficient Mixture-of-Experts Models

Trevor Gale
Google DeepMind
April 3rd, 2025

Introduction

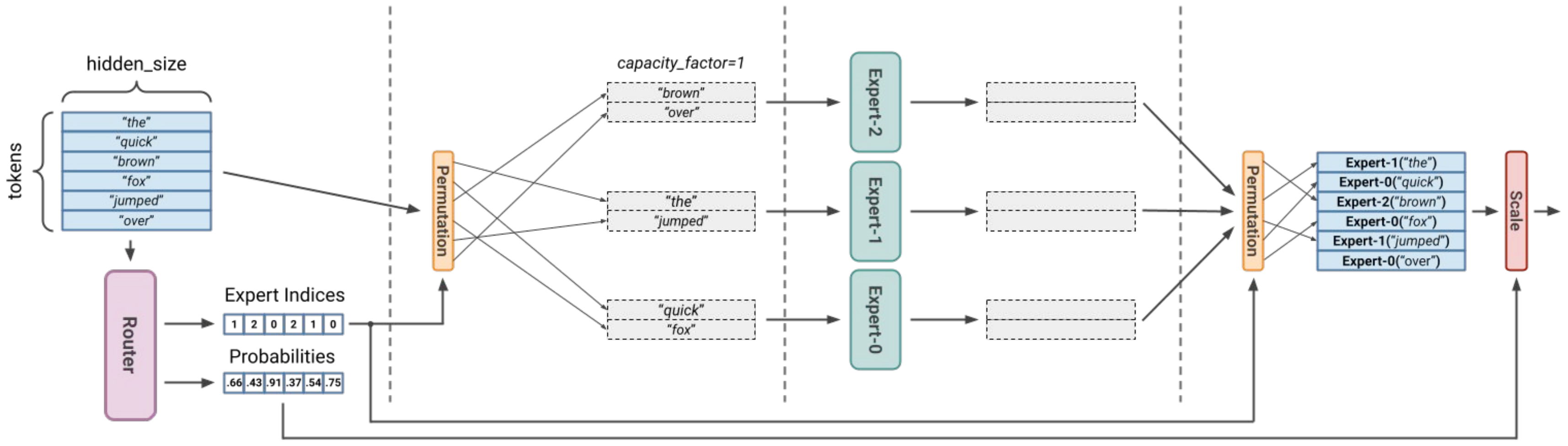
I'm a staff research scientist at Google DeepMind, where I've worked for the past 7 years.

My background is in programming and designing hardware accelerators. These days, I lead a team designing a new hardware accelerator. In the past, I've worked on Gemini pre-training and AI codesign.



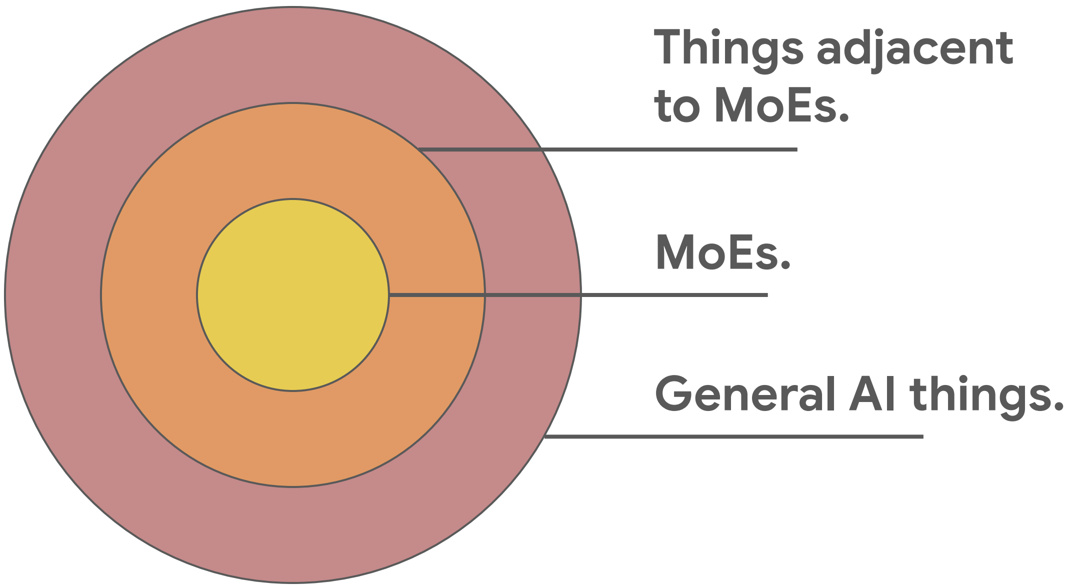
I did my PhD at Stanford, where I worked on sparse neural networks (in the same lab at Deepti!).

Today's Topic: Mixture-of-Experts Layers



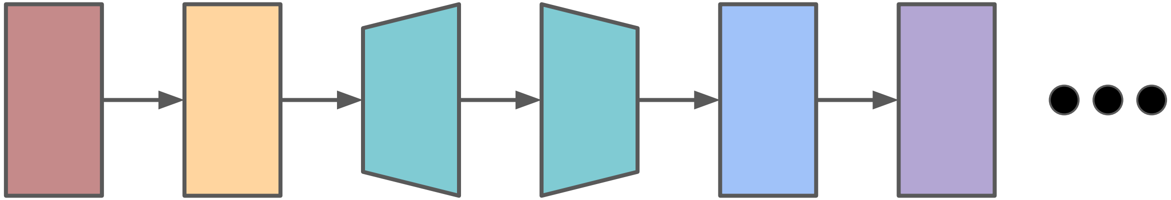
(Shazeer et al., 2017, Lepikhin et al., 2020, Fedus et al., 2021)

Goals for Today



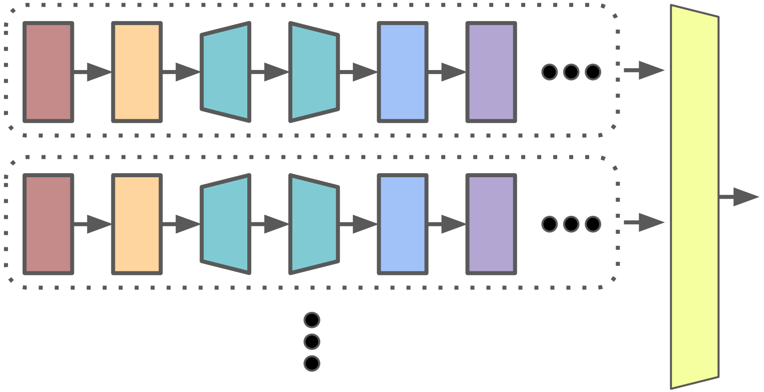
Please ask questions! The value of me being here in person is that we can interact.

Background: Deep Neural Networks



Data flows unconditionally through the layers. Layers compute some nonlinear transformation of the data.

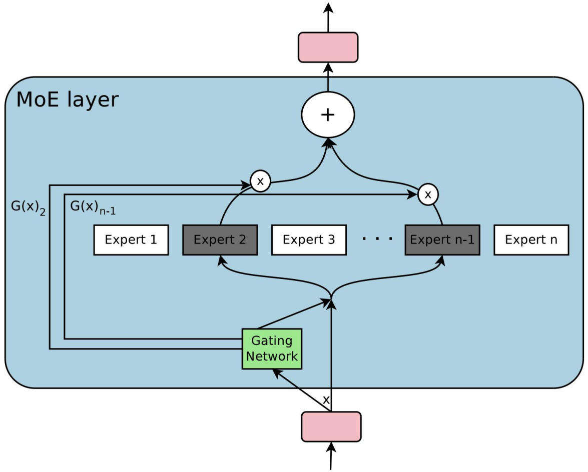
Background: Mixture-of-Experts



Have a bunch of “experts” (models, sub-models). Select between or combine their predictions. Related to ensembling.

(Shazeer et al., 2017)

Background: Sparsely-Gated Mixture-of-Experts

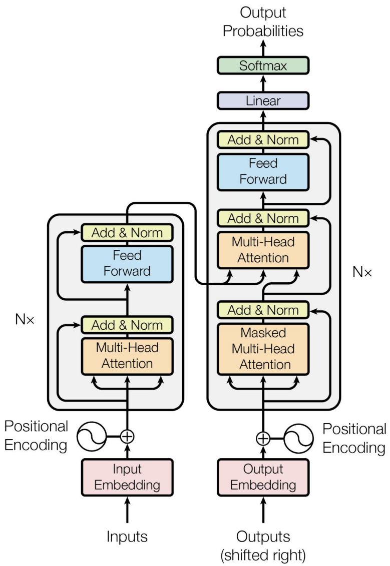


A type of layer containing many sub-layers where tokens are routed to a subset of these “experts”. What people mean when they say MoE today.

Predates Transformers! First evaluated on RNNs.

(Shazeer et al., 2017)

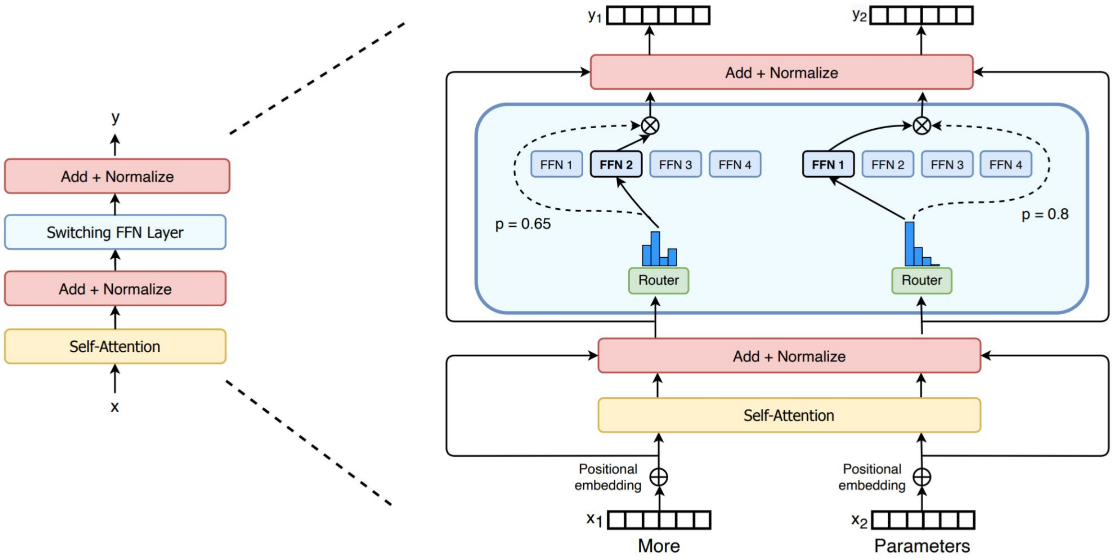
Background: Transformers



Attention to model sequences, rather than recurrence or convolution. Designed to use {GPU, TPU} efficiently!

(Vaswani et al., 2017)

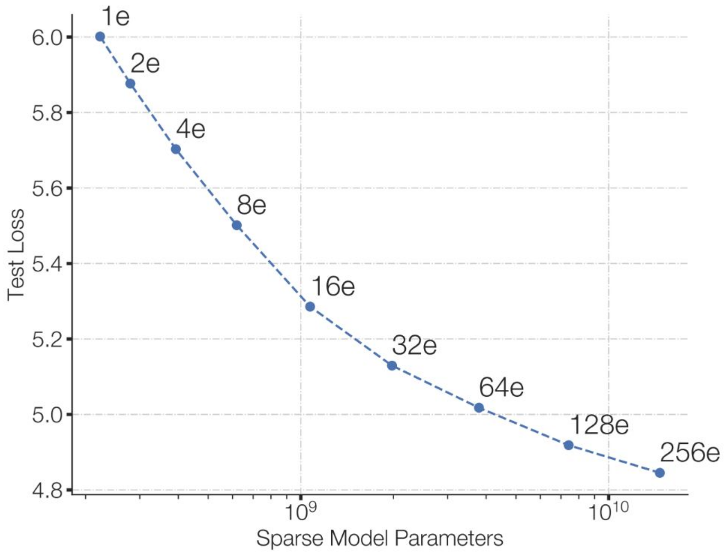
Background: Transformer MoEs

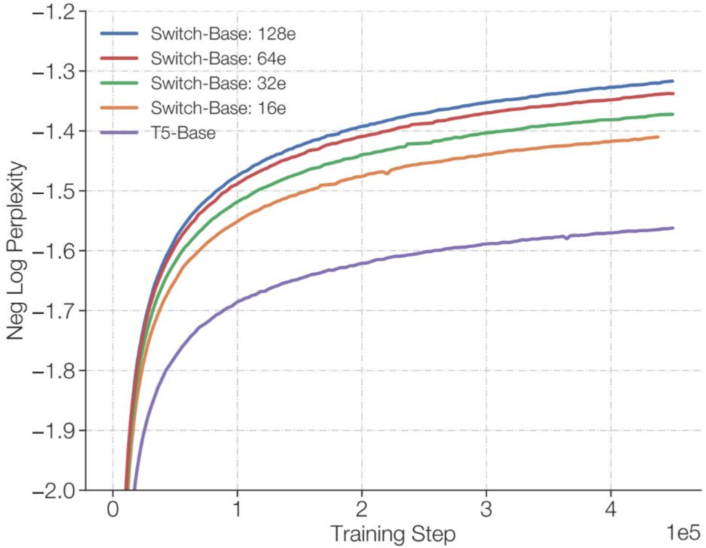


Make Transformer feed-forward layers SG-MoEs. What we all use today.

(Lepikhin et al., 2020, Fedus et al., 2021)

MoEs Decouple Compute & Parameter Count

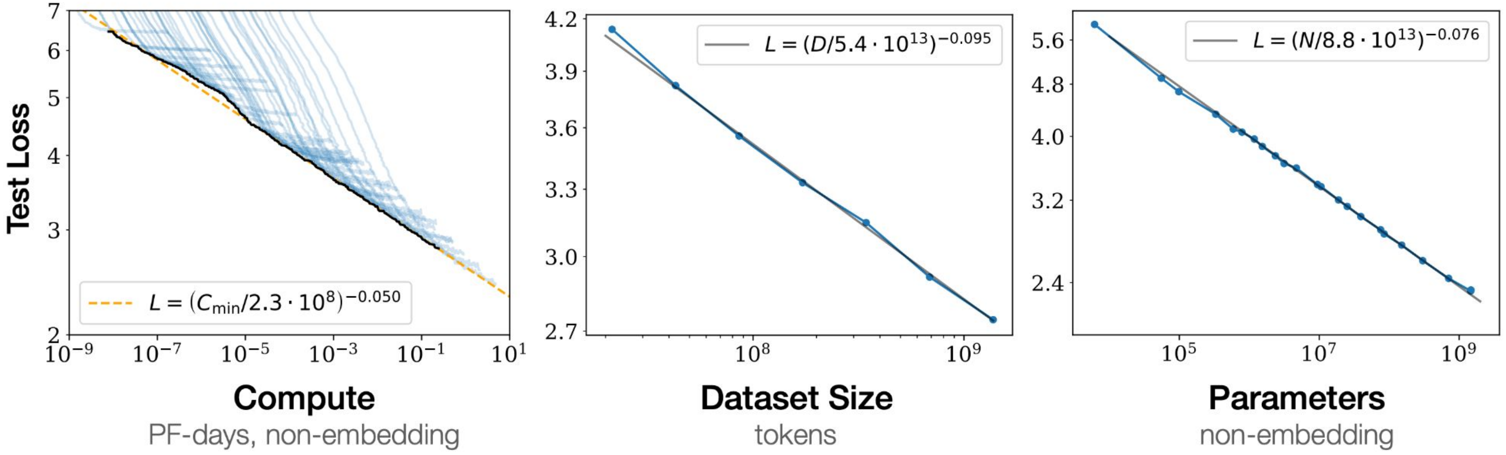




In sparse models, we can configure parameter count and compute per token **independently**.
This expanded design space yields **more efficient models**.

(Fedus et al., 2021)

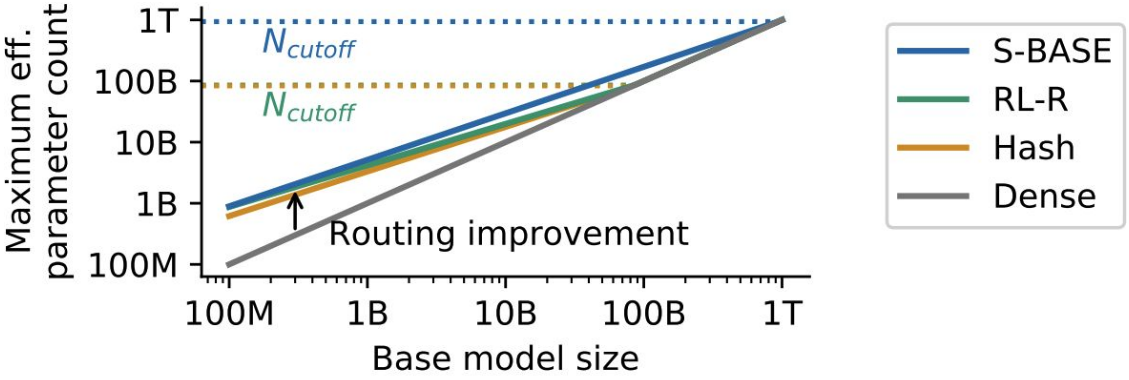
Background: Scaling Laws



Scaling laws formalize what we knew empirically: quality increases with {compute, data, model} scale. Reflection on the importance of this in AI in "Sutton's Bitter Lesson".

(Kaplan et al., 2020, Sutton, 2019)

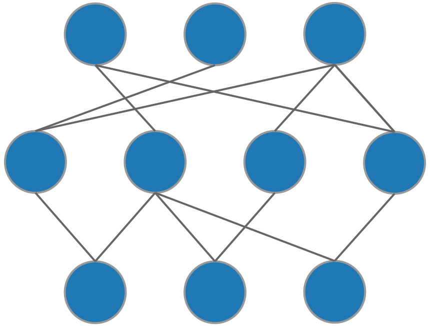
Scaling Laws For Mixture-of-Experts



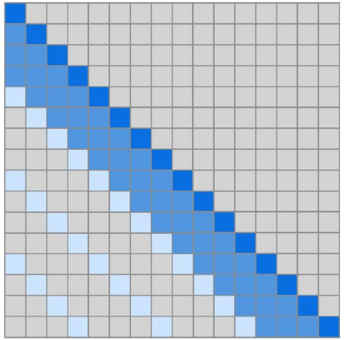
Formalize the efficiency wins of sparse architectures. Framed in terms of "effective parameter count".

(Clark et al., 2022)

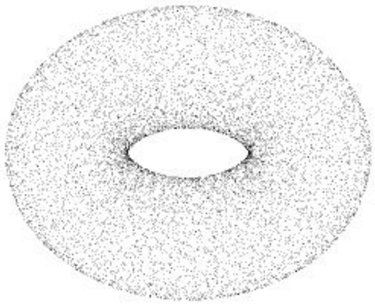
There Are Many Kinds of Sparsity



Weight Sparsity
Sources: Pruning, sparse training



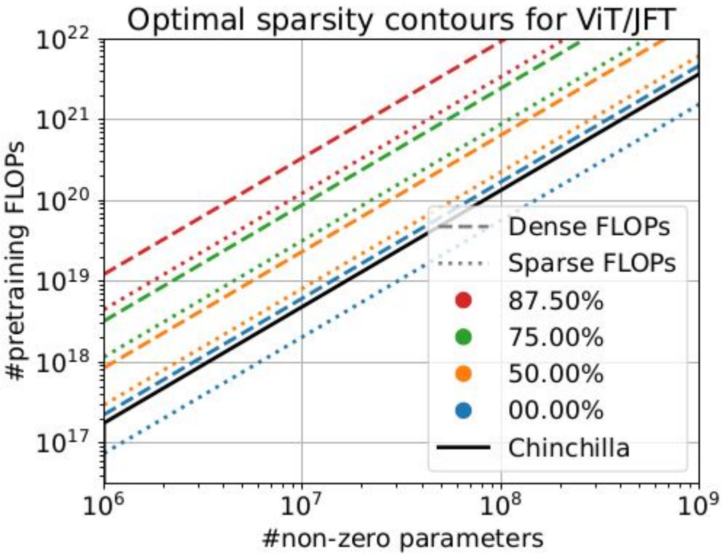
Activation Sparsity
Sources: ReLU, sparse attention, mixture-of-experts



Data Sparsity
Sources: Point clouds, graphs, etc.

All of these forms can be static or dynamic (e.g., changing based on the data)!

Other Kinds of Sparsity Show Similar Efficiency



For example, unstructured weight sparsity.

(Frantar et al., 2023)

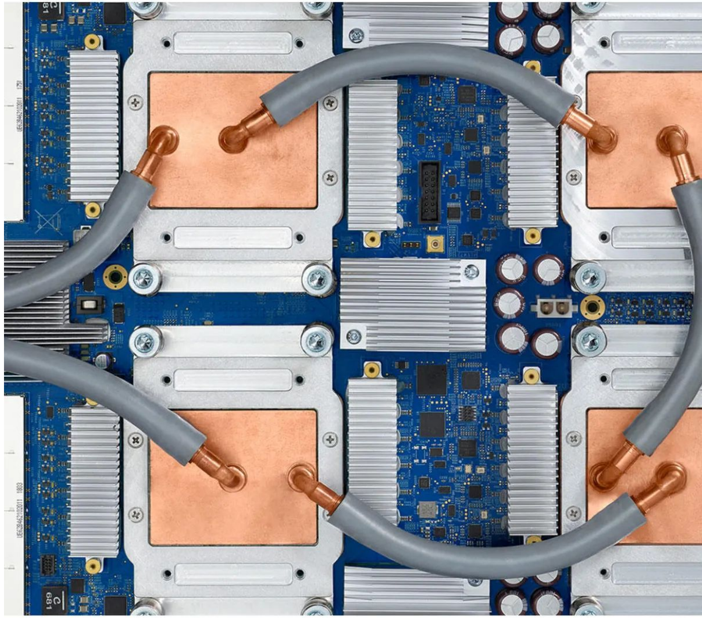
Why Are We Using Mixture-of-Experts Over Them?

MoEs are efficient on {GPU, TPU}.

Two key facts:

1. Scaling laws show quality scales with compute.
2. {GPU, TPU} maximize compute / watt, compute / mm^2 with today's semiconductor fabrication technology[1].

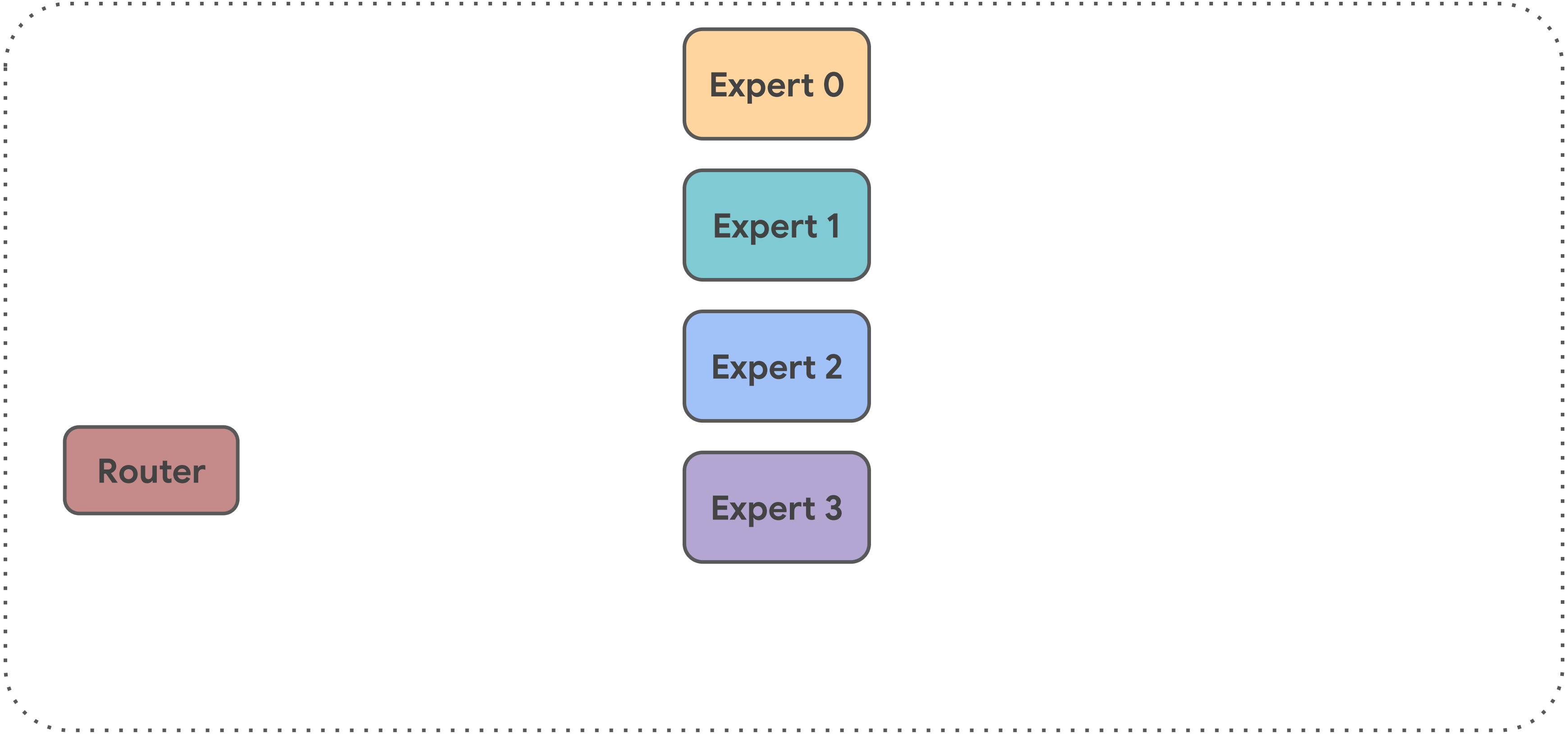
We want to use sparse methods that use our accelerators efficiently. **MoEs were designed for this.**



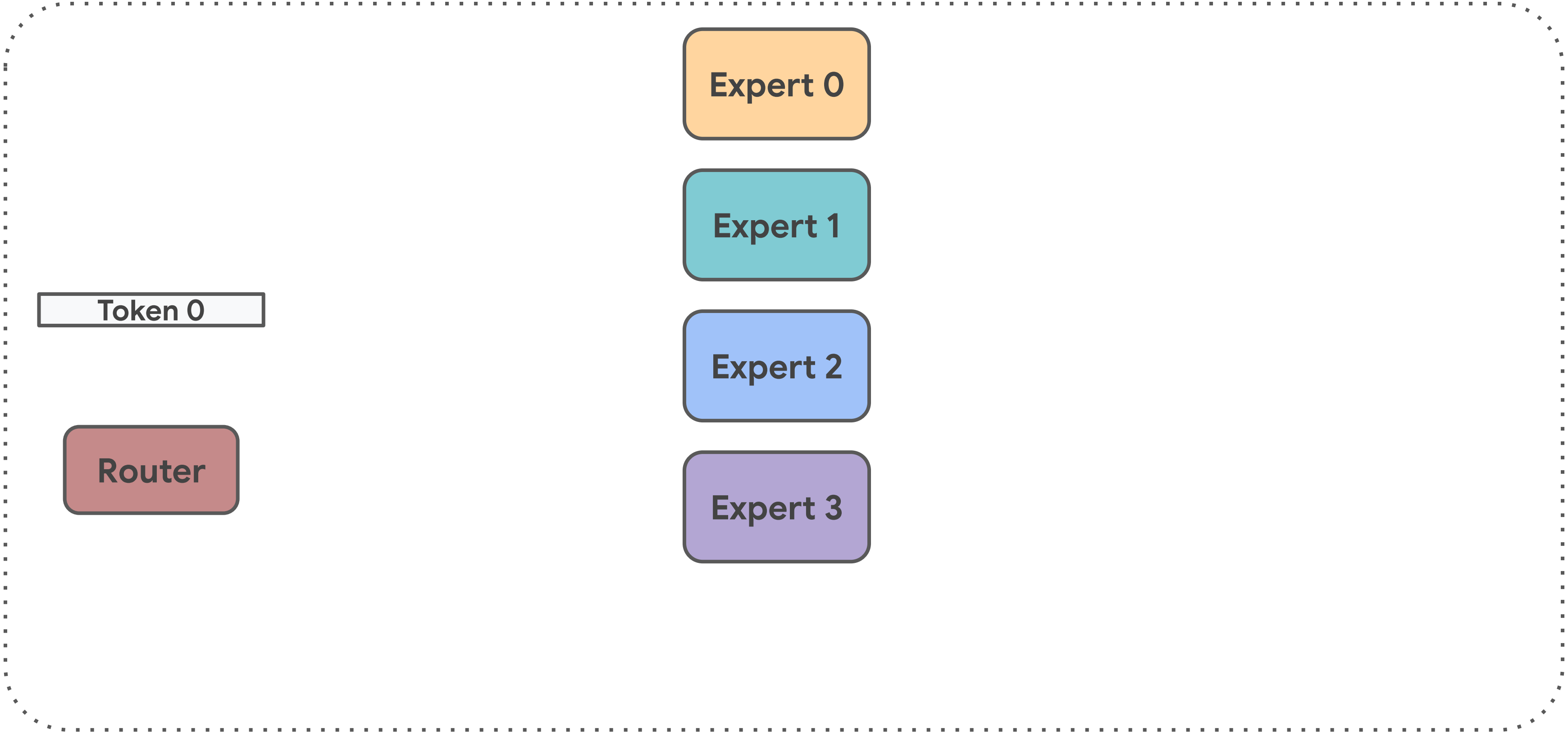
[1]Startups like MatX are betting on spending even more die area on compute.

Efficient Mixture-of-Expert Models

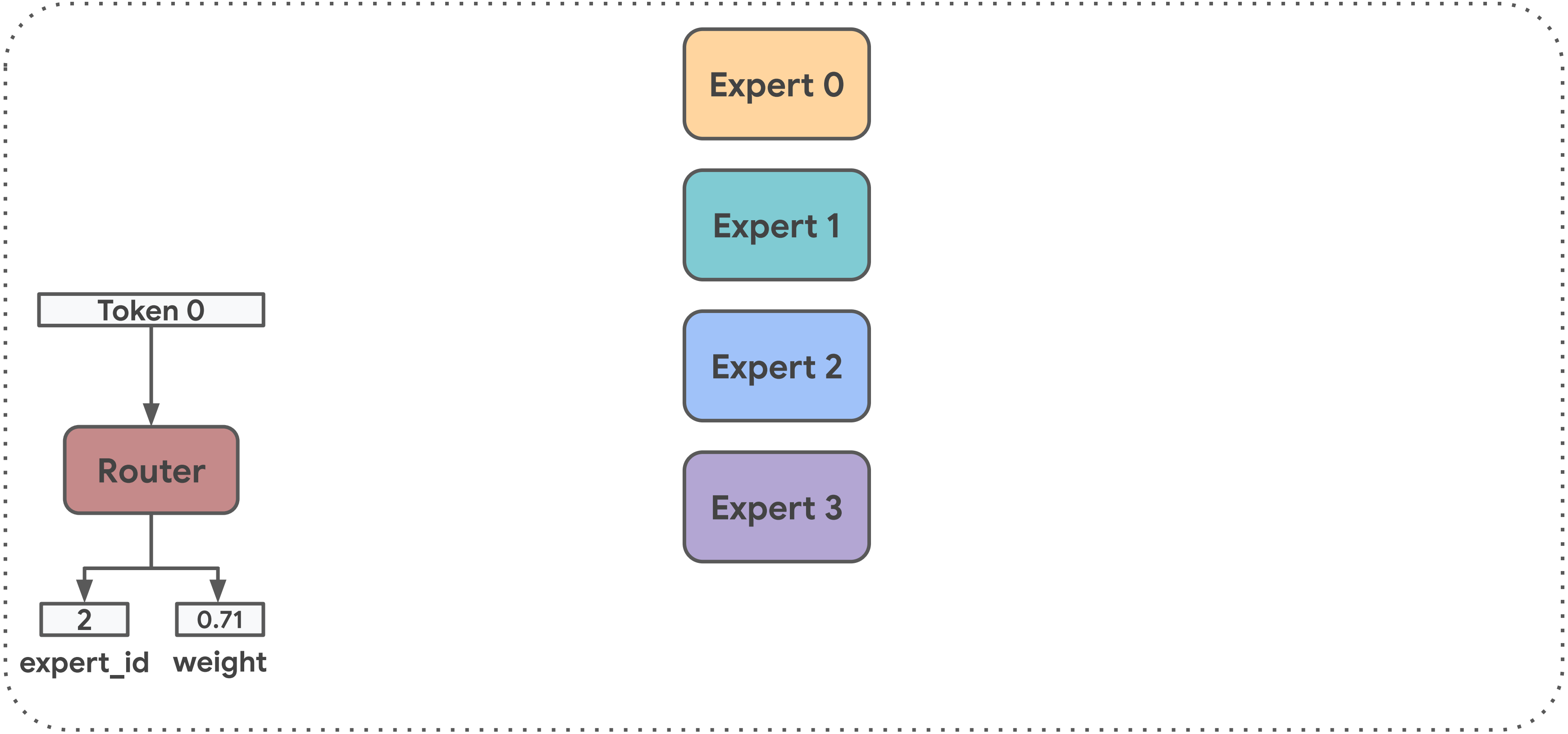
MoE Computation For One Token (1/5)



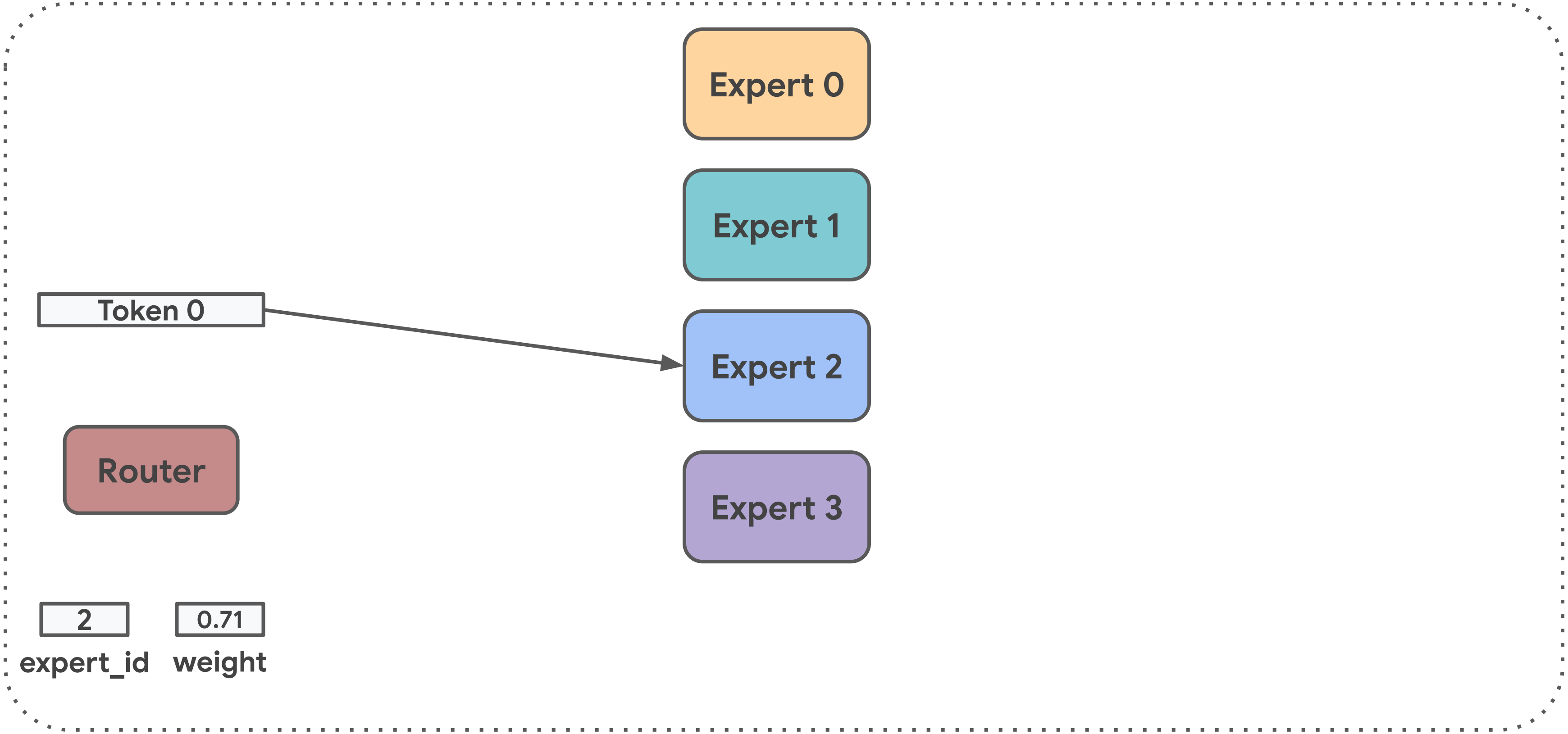
MoE Computation For One Token (2/5)



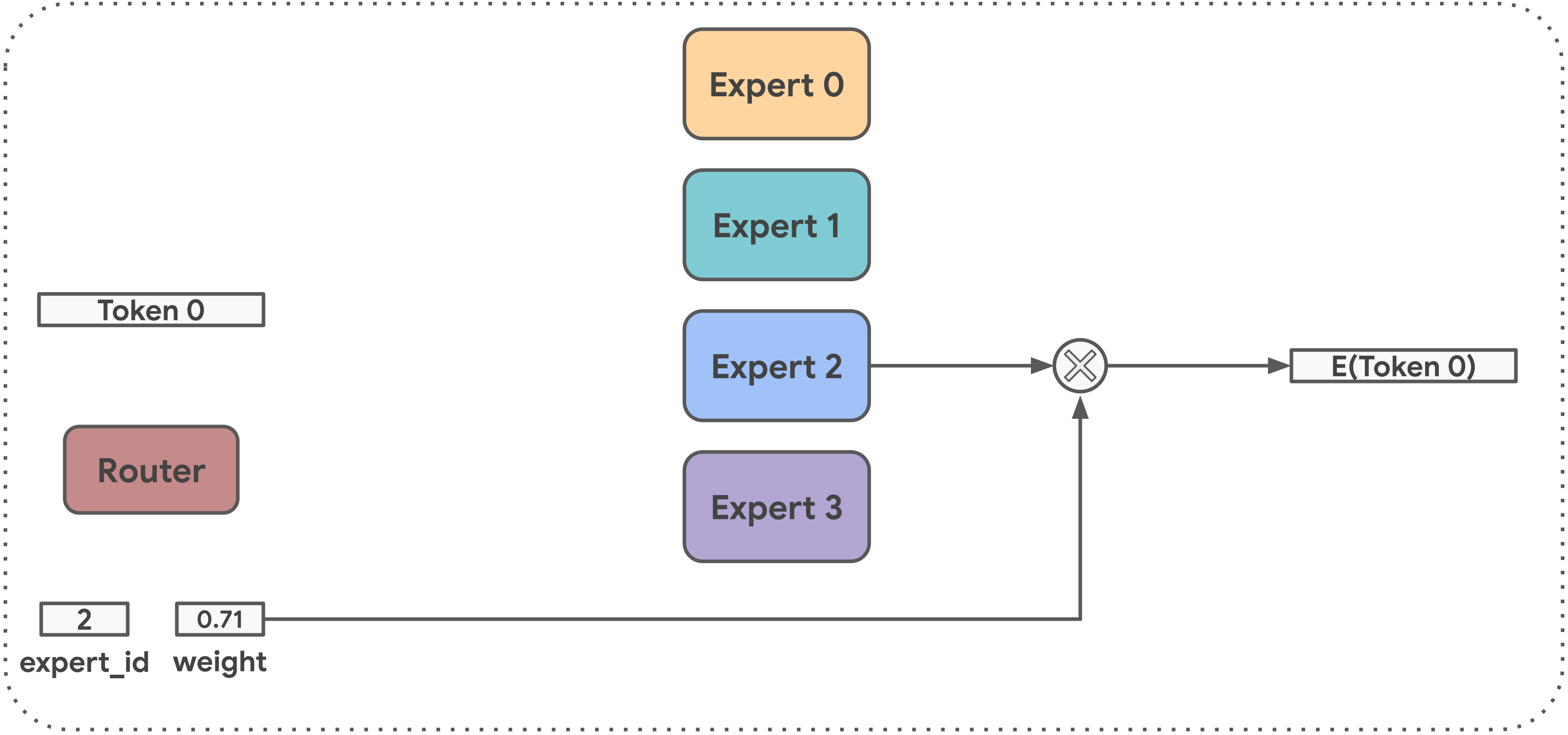
MoE Computation For One Token (3/5)



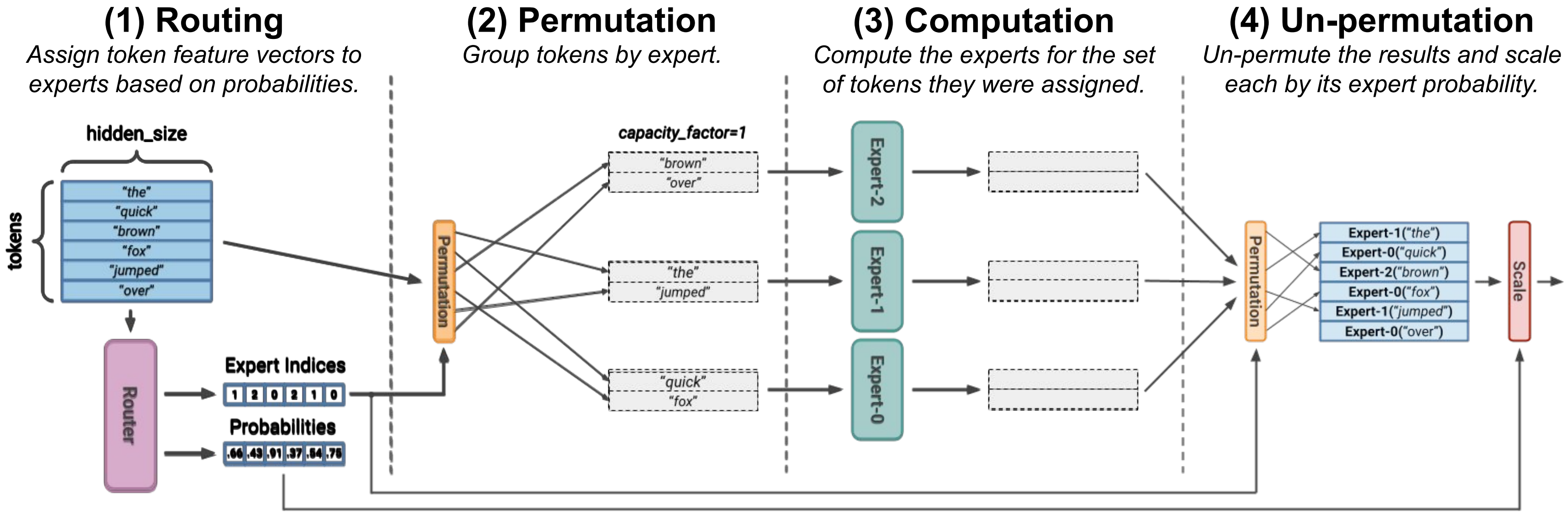
MoE Computation For One Token (4/5)



MoE Computation For One Token (5/5)



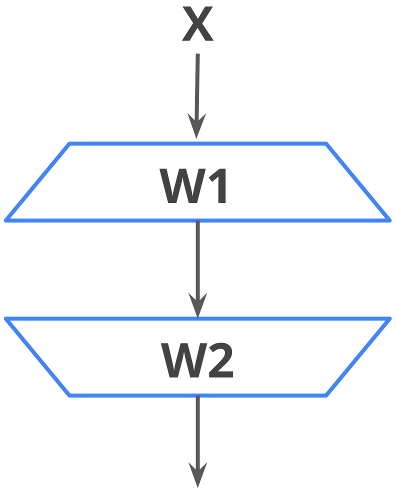
MoE Computation For Many Tokens



Q: What is the value of grouping tokens by expert?

Arithmetic Intensity Analysis (1/3)

Dense Feed-Forward Network



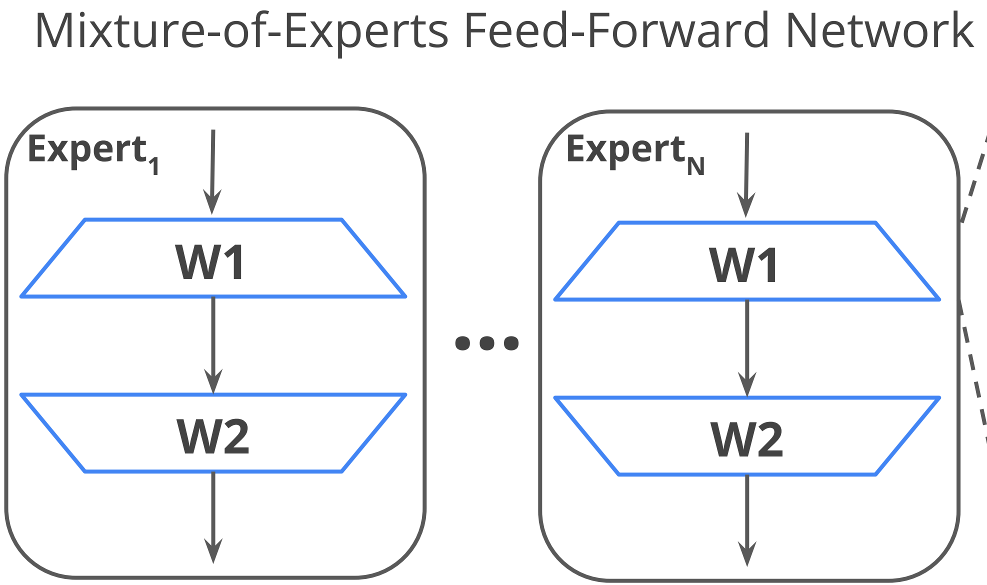
Matmul with shapes:

$$[tokens,\ d_{model}]\ @\ [d_{model},\ d_{ff}]\ =\ [tokens,\ d_{ff}]$$

$$arithmetic\_intensity\ =\ \frac{ops}{bytes}$$

$$\propto \frac{tokens * d_{model} * d_{ff}}{tokens * d_{model} + d_{model} * d_{ff} + tokens * d_{ff}}$$

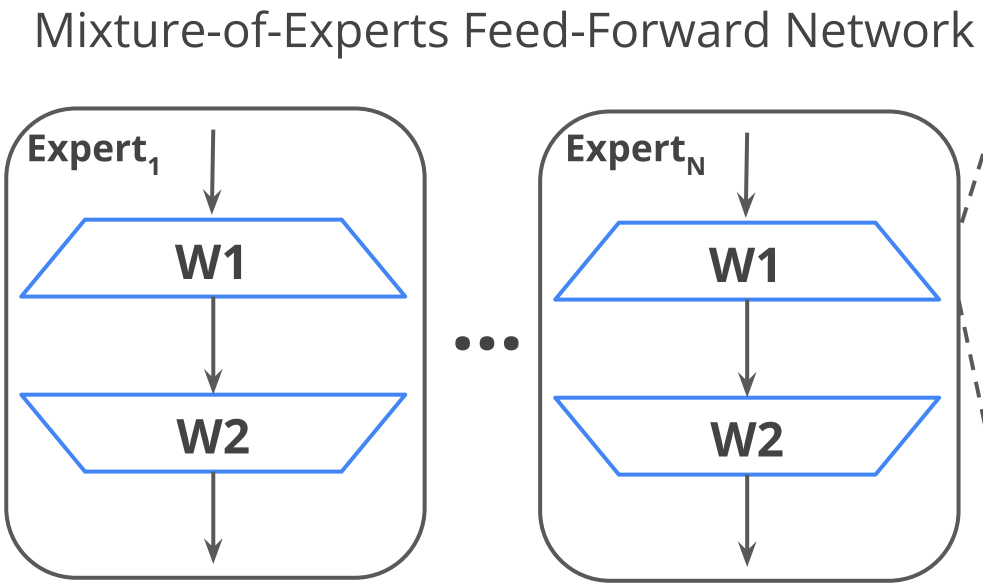
Arithmetic Intensity Analysis (2/3)



num_experts times more bytes for the same op count!

$$ai_{MoE} \approx \frac{1}{num\_experts} \times ai_{dense}$$

Arithmetic Intensity Analysis (3/3)

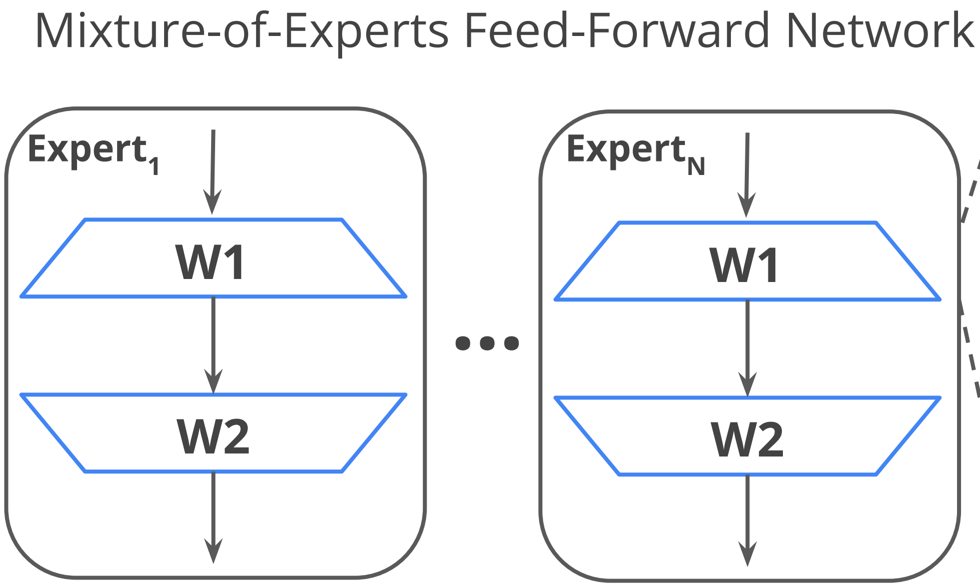


Each token sees *top_k* experts!

$$ai_{MoE} \approx \frac{top\_k}{num\_experts} \times ai_{dense}$$

This is the *sparsity* of the MoE
(the fraction of the weights that each token will touch)

Point 1: Token Grouping Maximizes Per-Expert Batch



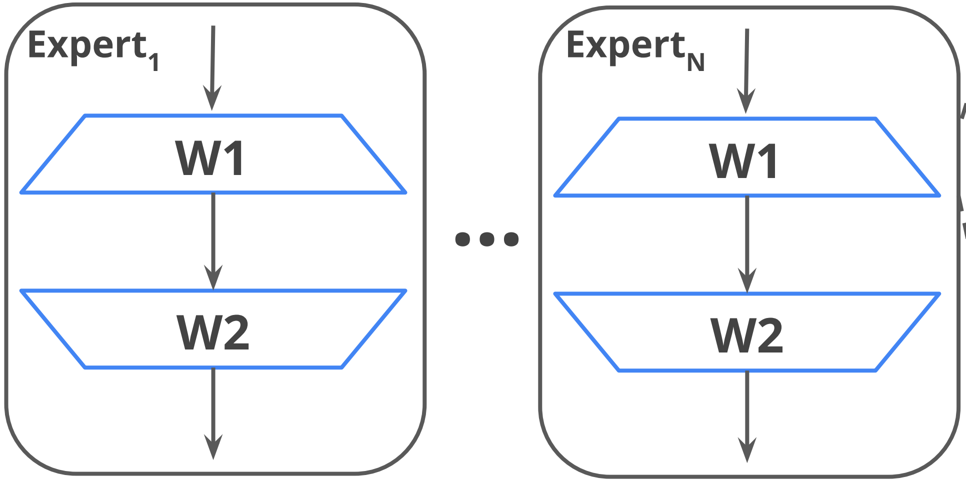
Maximizing expert batch size maximizes arithmetic intensity.

$$ai_{MoE} \approx \frac{top\_k}{num\_experts} \times ai_{dense}$$

Maximizing arithmetic intensity maximizes throughput.

Point 2: MoEs Have Lower Arithmetic Intensity Than Dense

Mixture-of-Experts Feed-Forward Network



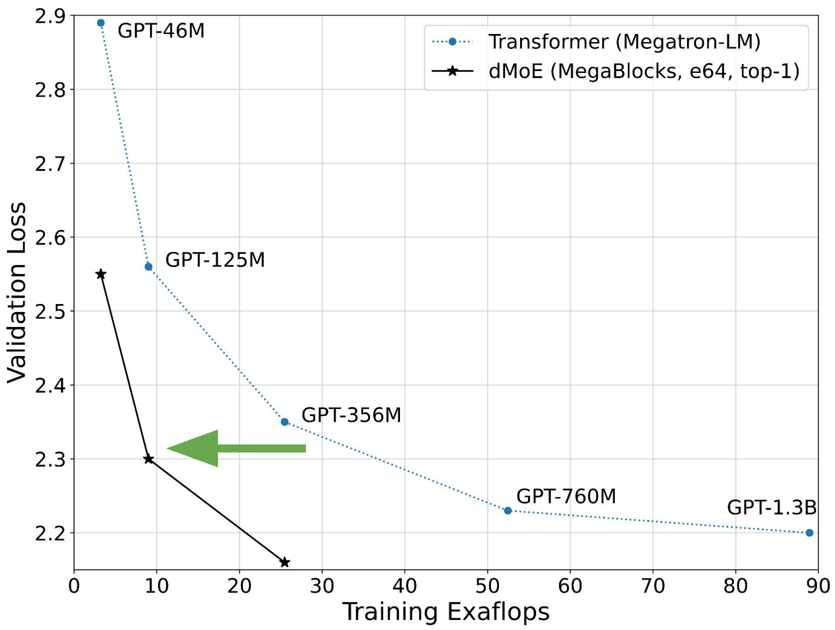
By a factor of the sparsity!

$$ai_{MoE} \approx \frac{top\_k}{num\_experts} \times ai_{dense}$$

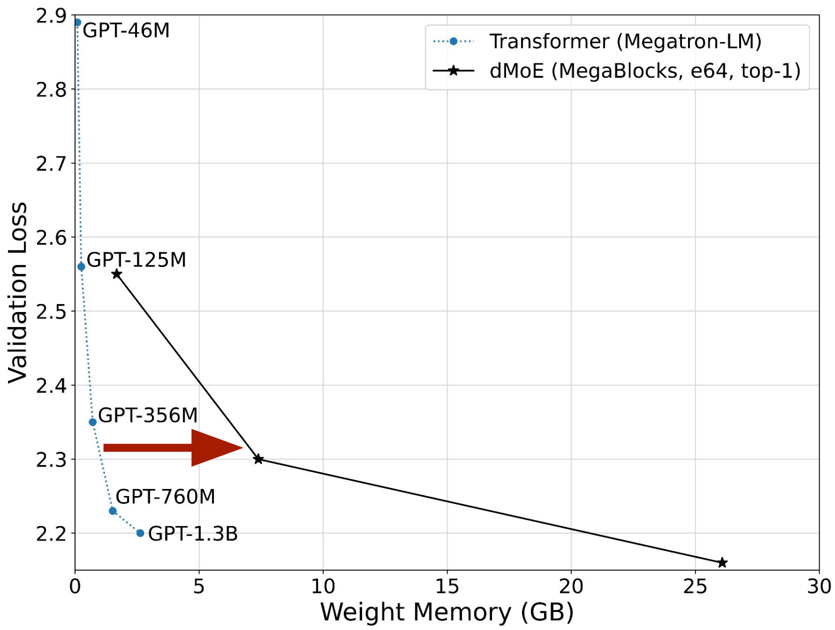
Trivial, but impractical solution:
Increase batch size 1/sparsity.

(Scaling Book)

MoEs Trade Compute for Storage



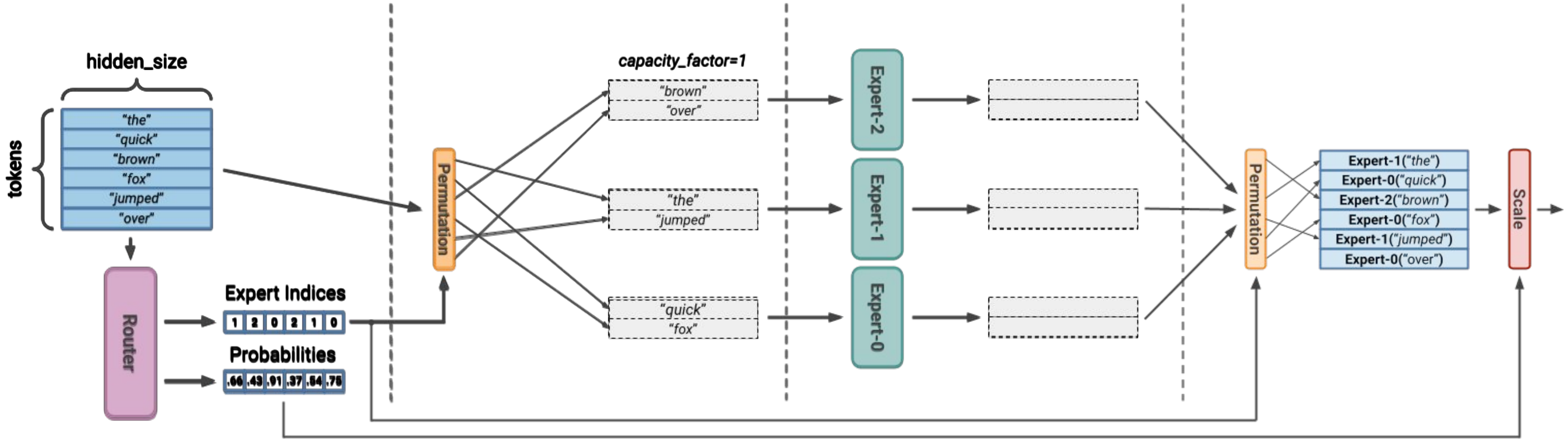
~3x Reduction in FLOPs.



~10x More Parameters.

The Tradeoff with MoEs

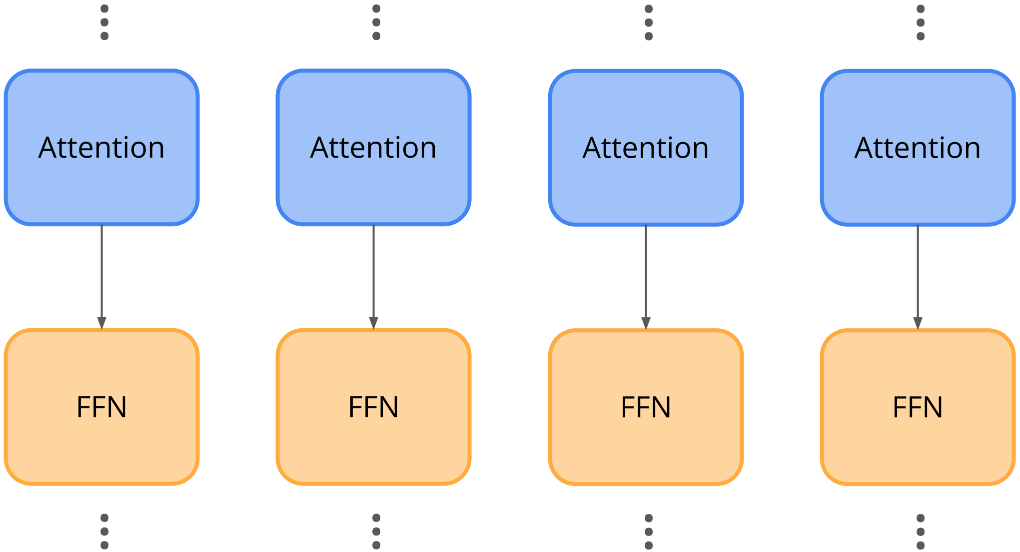
Compute efficient but parameter inefficient, low arithmetic intensity



Expert model parallelism helps address both downsides of MoEs!

The Value of Expert Model Parallelism (1 of 3)

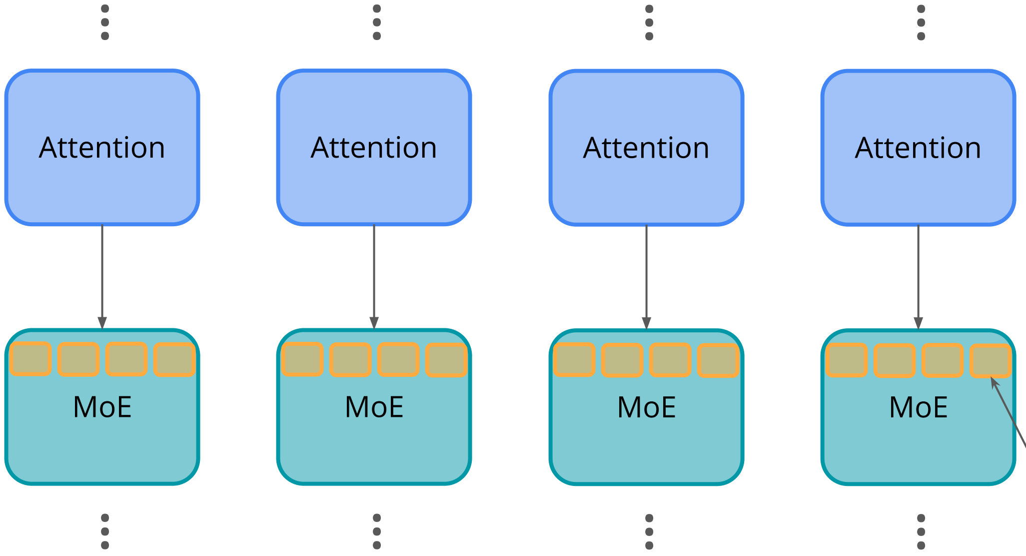
Scenario: data parallel training/serving a dense Transformer



- 4-way data parallelism (fwd)
- All parameters replicated

The Value of Expert Model Parallelism (2 of 3)

Scenario: data parallel training/serving a Transformer MoE



- 4-way data parallelism (fwd)
- All parameters replicated
- 4-experts / top-1 routing

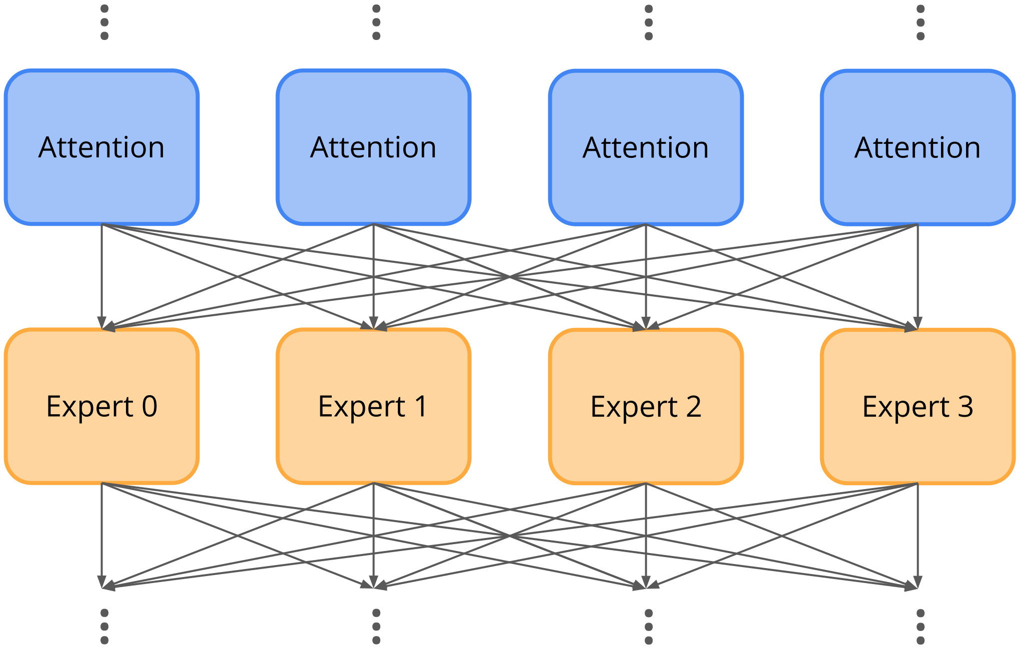
4x FFN weights per device

¼ batch per FFN compute

(not to scale - each is full FFN)

The Value of Expert Model Parallelism (3 of 3)

Scenario: expert parallel training/serving a Transformer MoE



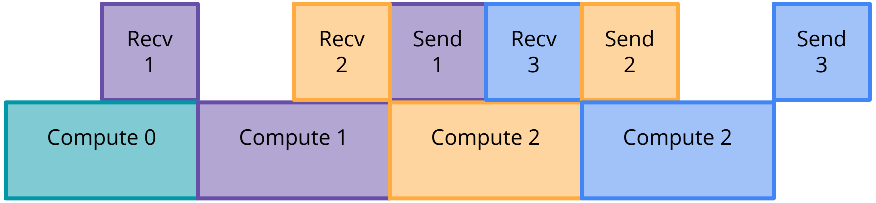
- 1 expert per device
- "all to all" communication to route tokens to experts (2x)

Same parameter count per device as dense scenario

Same arithmetic intensity as dense scenario

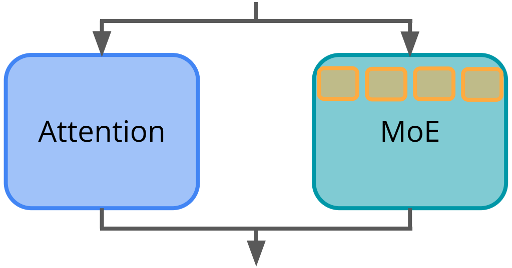
Expert Parallelism Challenge #1: All-to-All Cost

Token routing to remote devices can be (somewhat) expensive.



Pipeline It!

Perform a2a in chunks. Overlap with compute on those chunks[1].



Overlap It!

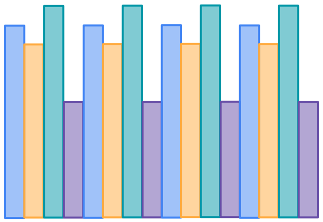
Parallel Transformer block[2] exposes independent operations to hide a2a.

[1] Tutel: Adaptive Mixture-of-Experts at Scale, Hwang et al., 2022

[2] GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model, Wang et al., 2021

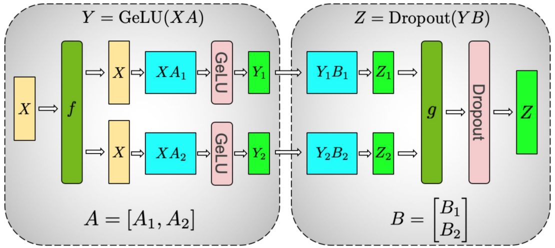
Expert Parallelism Challenge #2: Load Imbalance

Experts can receive different numbers of tokens!



Load Balancing Loss / Jitter

Levers for controlling imbalance at train time.
Balance between model quality and step time.
Or, newer aux-loss-free load balancing[1]!



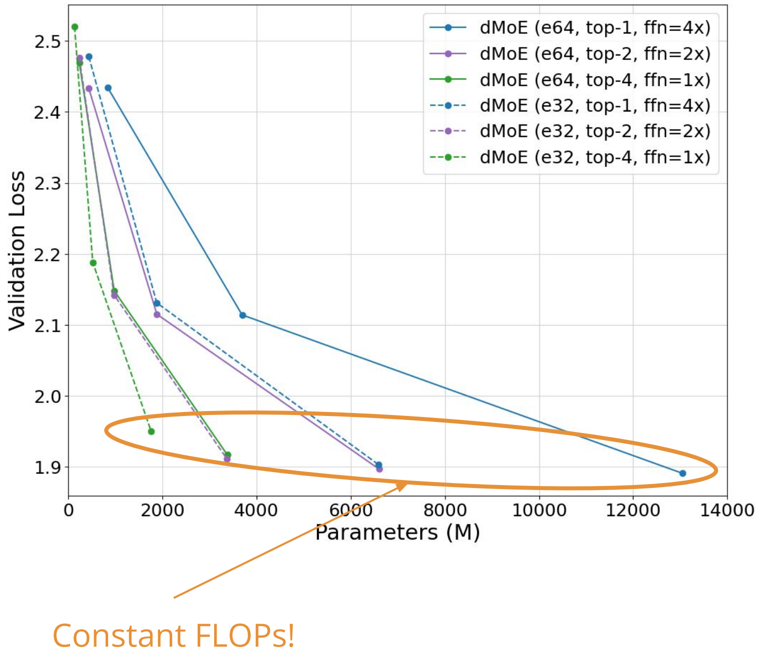
Alternative Sharding

Mixing expert sharding with tensor parallelism[2]
can help reduce load imbalance across devices.

[1] Auxiliary-Loss-Free Load Balancing Strategy for Mixture-of-Experts, Wang et al., 2024
[2] Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism, Shoeybi et al, 2020

Aside: Model Design Matters a Lot!

Architectural parameters like *d_ff*, *num_experts*, and *top_k* affect parameter efficiency and arithmetic intensity!

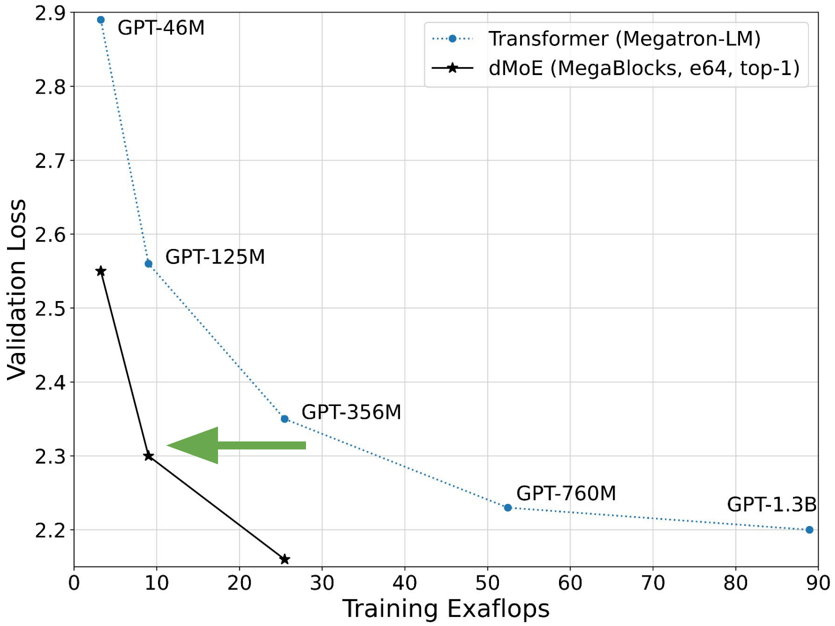


Scaling laws work for MoEs:

[1] Unified Scaling Laws for Routed Language Models, Clark et al., 2022

[2] Scaling Laws for Fine-Grained Mixture of Experts, Krajewski et al., 2024

Not all FLOPs are Created Equal



MoEs are compute efficient!

At training time we realize high % of this theoretical win.

Batch size limitations for serving can change this calculus dramatically.

The "U" of MoE Serving Efficiency (1 of 4)

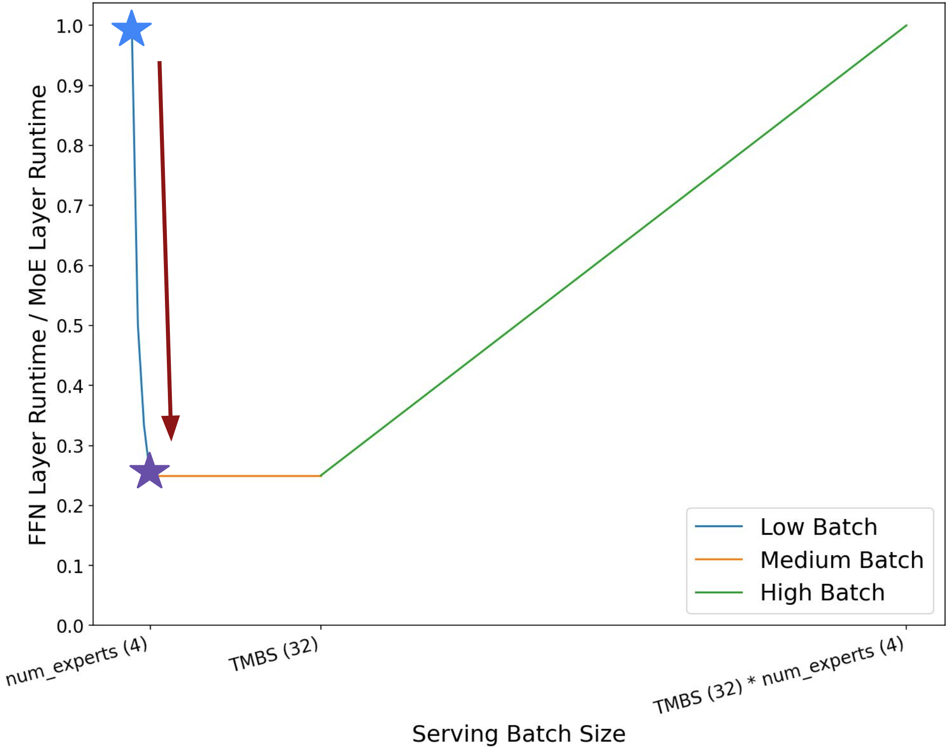
Three Regions of Efficiency.

Region #1: Low Batch Serving

Perf limiter: memory bw loading weights.

Batch=1: MoE/dense touch the same number of weights! ★

Batch=num_experts: MoE touches num_experts times as many weights[1]. ★



TMBS = "Throughput Maximizing Batch Size", i.e., batch size where dense ~saturates math units.

[1] Assuming uniform distribution of tokens to experts.

The “U” of MoE Serving Efficiency (2 of 4)

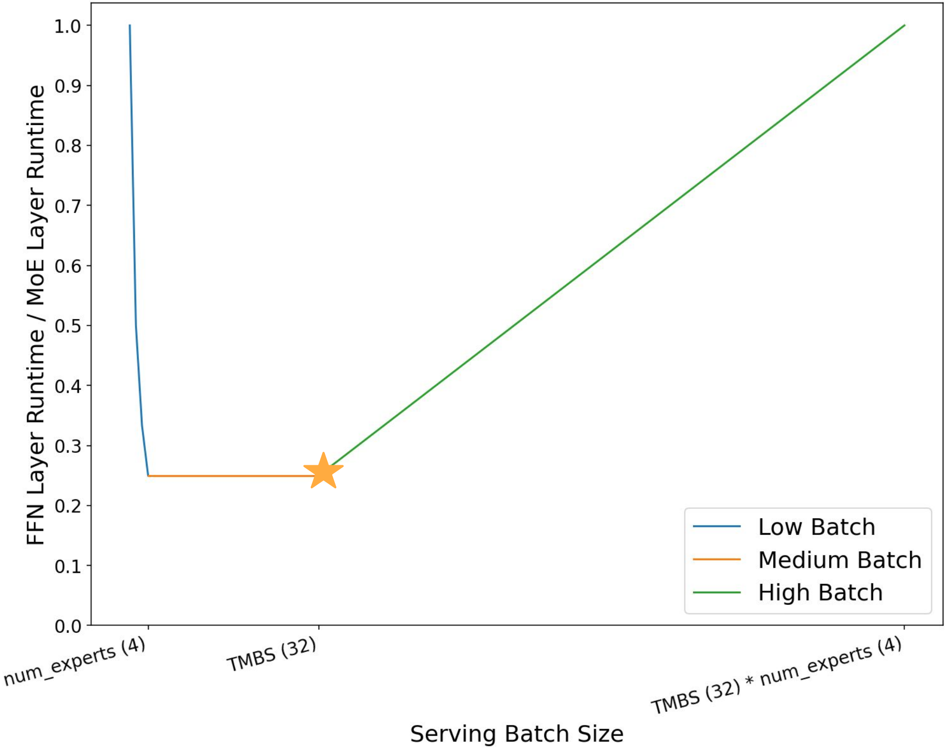
Region #2: Medium Batch Serving

Dense/MoE runtime ~constant as compute utilization ramps up.

Batch=TMBS: Dense FFN ~saturates compute.

Further increases in batch incur proportionate runtime increase for dense.

MoE is not yet saturating compute due to lower arithmetic intensity (1/num_experts)



TMBS = “Throughput Maximizing Batch Size”, i.e., batch size where dense ~saturates compute.

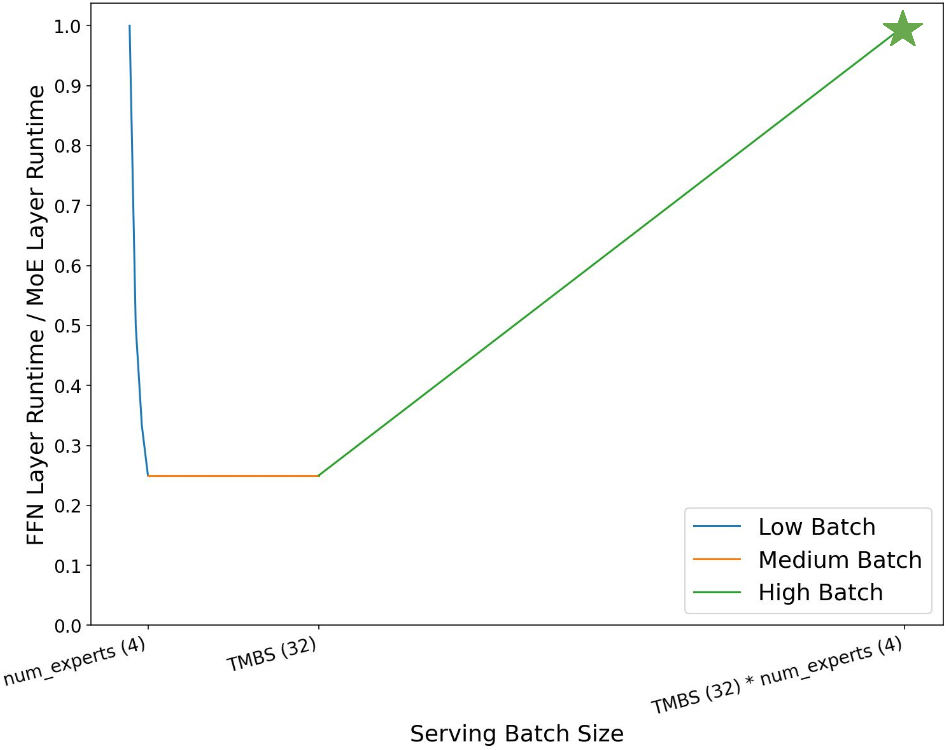
The “U” of MoE Serving Efficiency (3 of 4)

Region #3: High Batch Serving

Dense/MoE runtime ramps up as MoE approaches compute saturation.

Batch=TMBS*num_experts: MoE ~saturates compute.

Once batch is sufficient, both dense/MoE saturate compute and have ~equal runtime.



TMBS = “Throughput Maximizing Batch Size”, i.e., batch size where dense ~saturates compute.

The “U” of MoE Serving Efficiency (4 of 4)

Commentary:

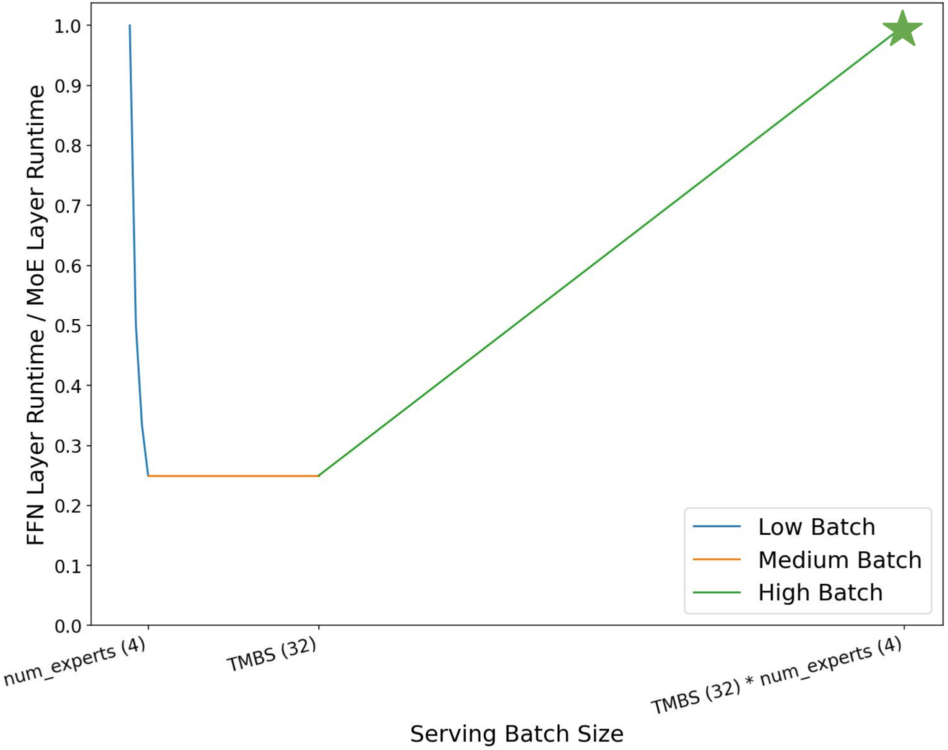
This is ~a roofline model for dense/MoE perf. There are factors not modeled.

The “U” is shallower if you compare e2e runtime.

MoE is higher quality - you might be happy in the medium batch regime.

num_experts, *top_k*, *d_ff*, sharding affect the depth/width of this “U”.

Latency constraints limit num_experts. ★



TMBS = “Throughput Maximizing Batch Size”, i.e., batch size where dense ~saturates compute.

Broader Topics in AI

Commoditization of Frontier Models

Unique Model Capabilities

“Please put a photorealistic dragon flying in the top left corner of the image.”

Commoditized Intelligence?

For some applications, I expect the quality we have today is good enough.

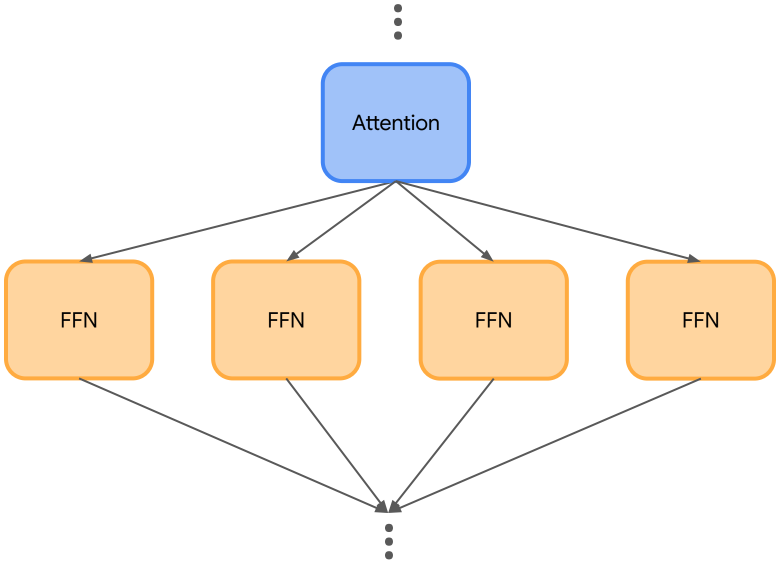
(1) We will probably push the cost of these to 0 in the next couple years

(2) CPUs could become relevant for these applications!

https://github.com/google/gemma.cpp

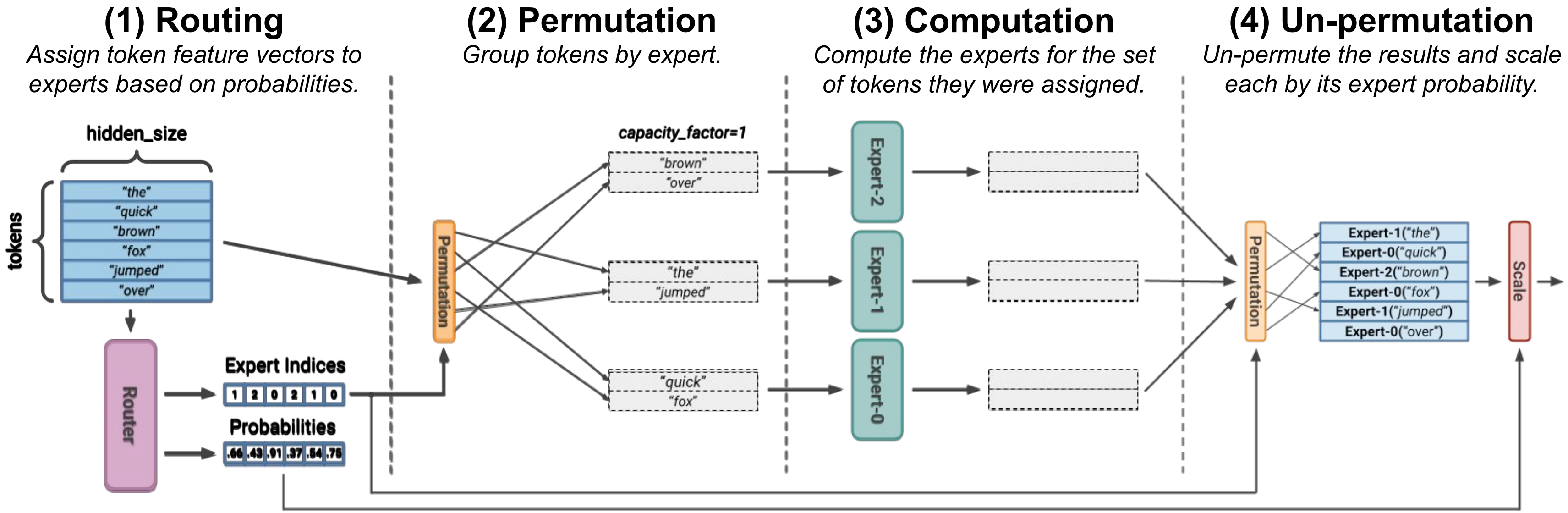
The End

MoEs are just the beginning for sparsity, adaptivity and dynamic computation!



MegaBlocks: Efficient Sparse Training with Mixture-of-Experts

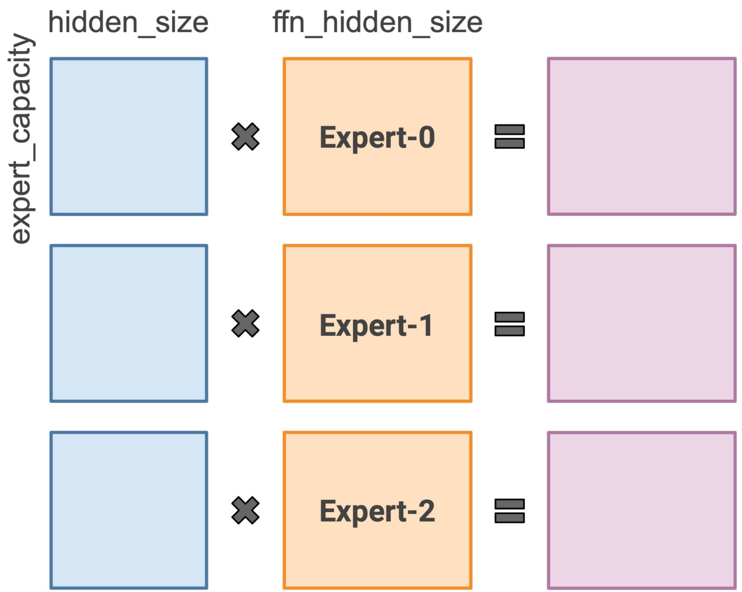
Mixture-of-Experts Layers



As expert count increases, individual expert computation gets smaller. **Computing the experts in parallel is key to good performance!**

(Shazeer et al., 2017, Lepikhin et al., 2020)

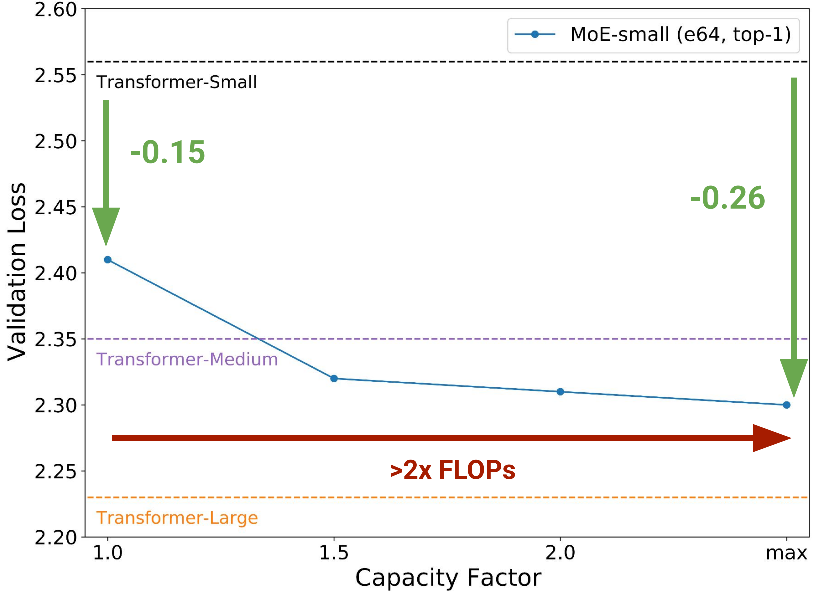
Batched Expert Computation



Batched Matrix Multiplication

Experts must have same number of tokens! Set via `capacity_factor` hyperparameter.

Bad: Introduces quality-speed tradeoff.



Token Dropping

Tokens will be skipped if too many are assigned to an expert.

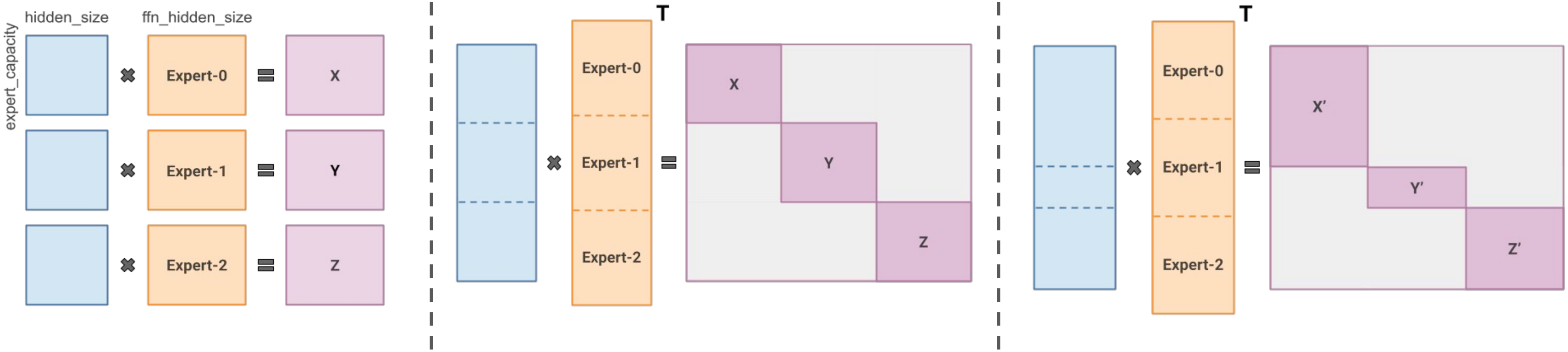
expert_capacity = capacity_factor * num_tokens / num_experts

MegaBlocks: Mixture-of-Experts with Structured Sparsity

(Current) Batched Matmul
Parallel, but constrained expert computation.

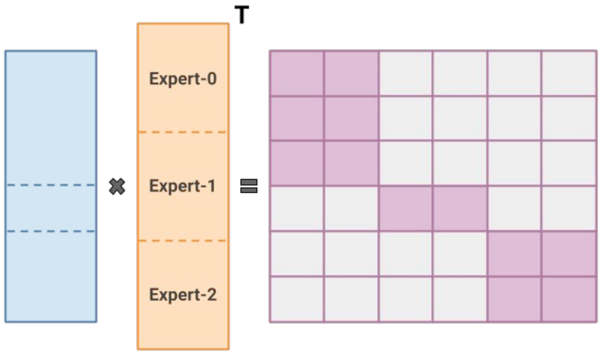
(Reformulation) Block Diagonal Matmul
Expert computation with block diagonal matrices.

(Generalization) Load Imbalanced Routing
Use variable block sizes to enable load imbalanced routing.



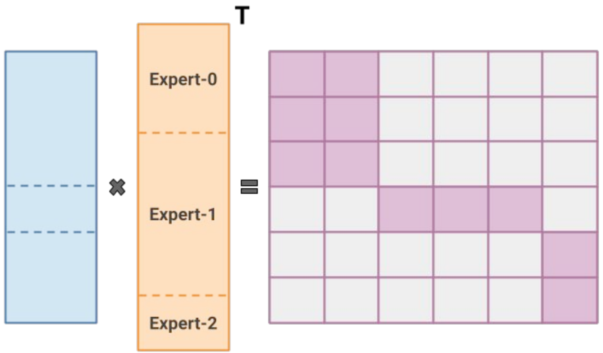
MegaBlocks: Mixture-of-Experts **as** Structured Sparsity

(This Paper) Load Imbalanced Routing
Realized via block-sparsity.

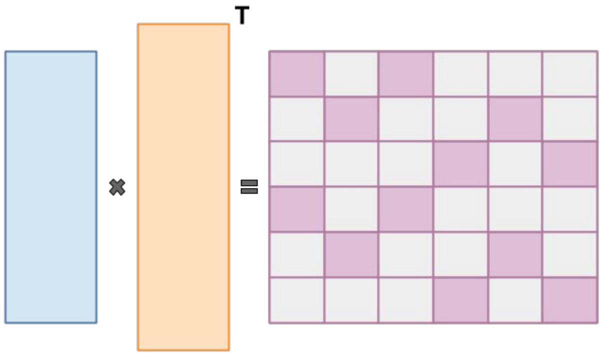


1.2x - **1.4x** faster than state-of-the-art MoEs.

(Future) Adaptive Computation
With variable sized experts.



(Future) Dynamic Activation Sparsity
Maximum flexibility with block-sparsity.

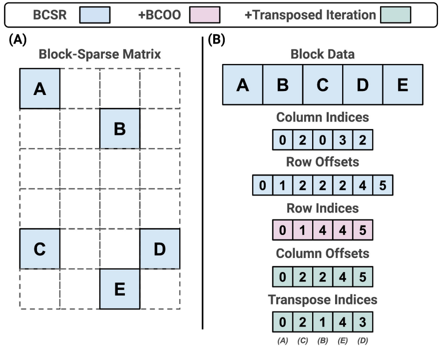


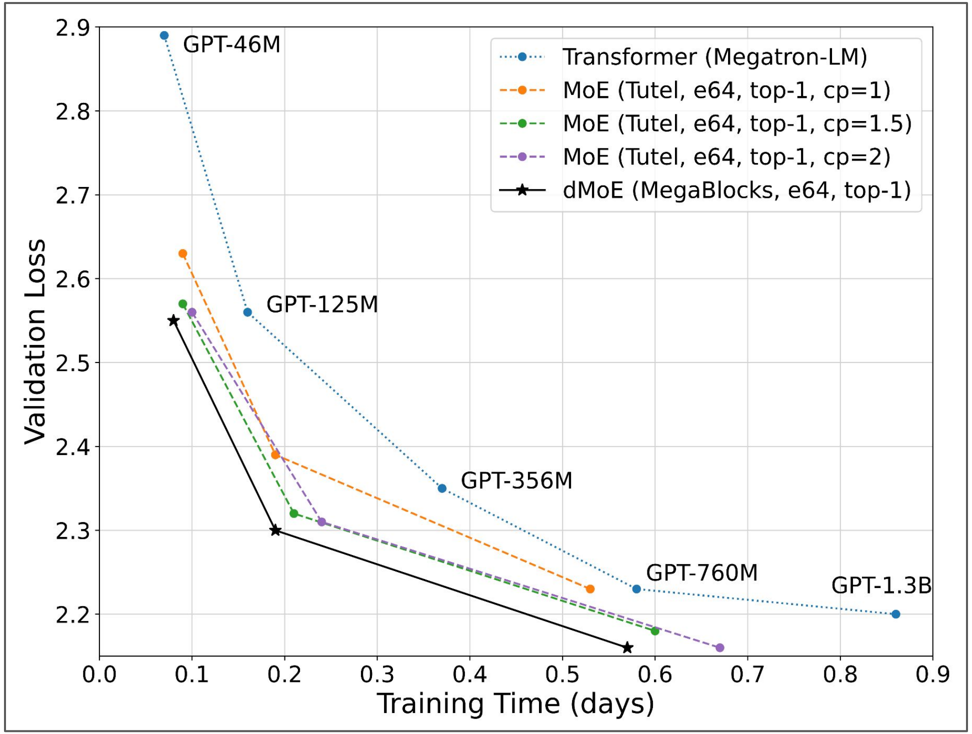
This is the first step towards our goal to improve quality / flop by generalizing MoEs.

MegaBlocks: Efficient Sparse Training with Mixture-of-Experts

1.2x - 1.4x faster than Tutel MoEs. **1.8x - 2.4x** faster than Megatron-LM Transformers.

Enabled by efficient sparse implementation!





More in our paper:

MLSys [MegaBlocks: Efficient Sparse Training with Mixture-of-Experts, MLSys'23]

Trevor Gale, Deepak Narayanan, Cliff Young, Matei Zaharia

Impact & Adoption

Models Using MegaBlocks

Collaborated with Databricks to train **DBRX** with MegaBlocks.

March 2024: MegaBlocks becomes an official Databricks project => github.com/databricks/megablocks.

Mixtral 8x7B released with MegaBlocks reference implementation.

JetMoE trained with MegaBlocks.

Libraries Using MegaBlocks

github.com/microsoft/tutel

github.com/huggingface/nanotron

EleutherAI

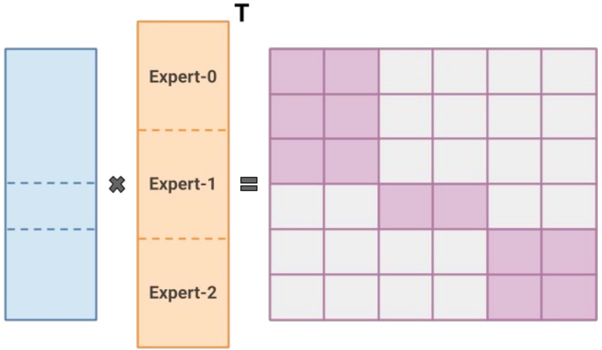
github.com/EleutherAI/gpt-neox

MegaBlocks on TPU (!)

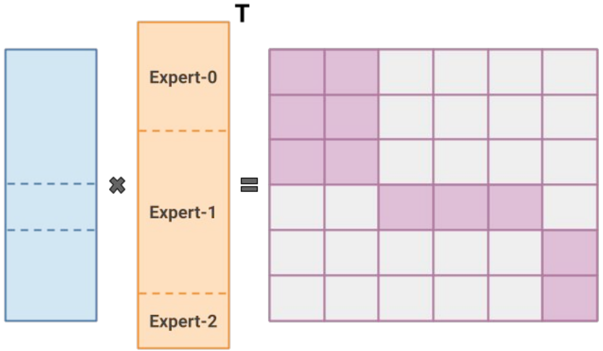
github.com/google/jax => ops, written in Pallas
github.com/google/maxtext => dMoE in JAX on TPU
github.com/pytorch/xla => dMoE in PyTorch on TPU

MegaBlocks Was Built to Enable New Forms of Sparsity

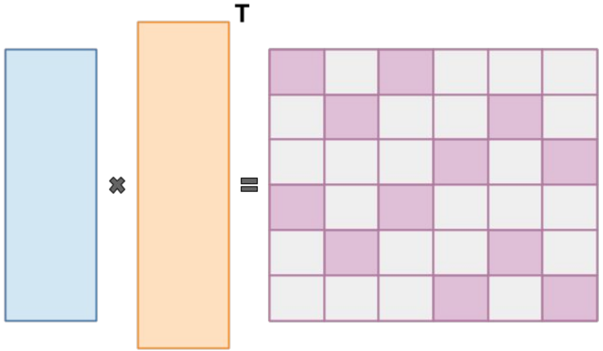
(This Paper) Load Imbalanced Routing
Realized via block-sparsity.



(Future) Adaptive Computation
With variable sized experts.



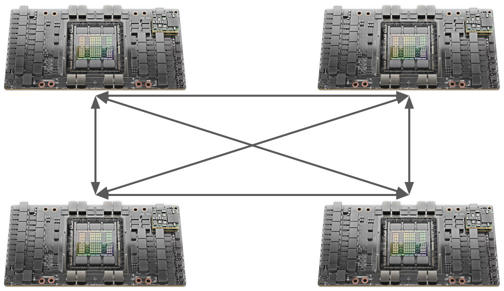
(Future) Dynamic Activation Sparsity
Maximum flexibility with block-sparsity.



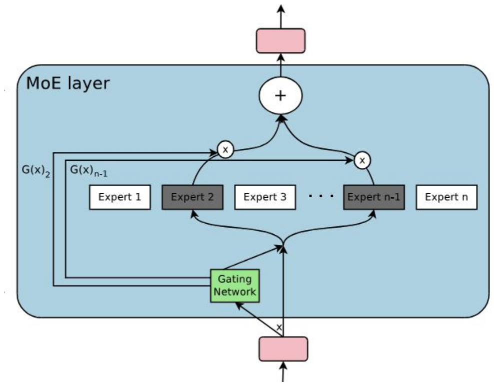
Lots of demand for this!

Long term goal was to get here!

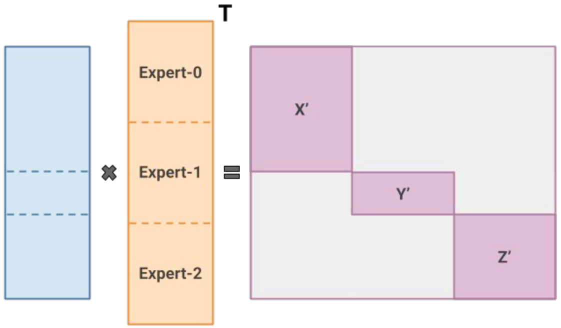
MegaBlocks Is More Than Just Sparse Compute



Dropless Expert Model Parallelism
Also other sharding, like FSDP.



Memory Optimizations
Manual buffer reuse in backward pass, activation function rematerialization.



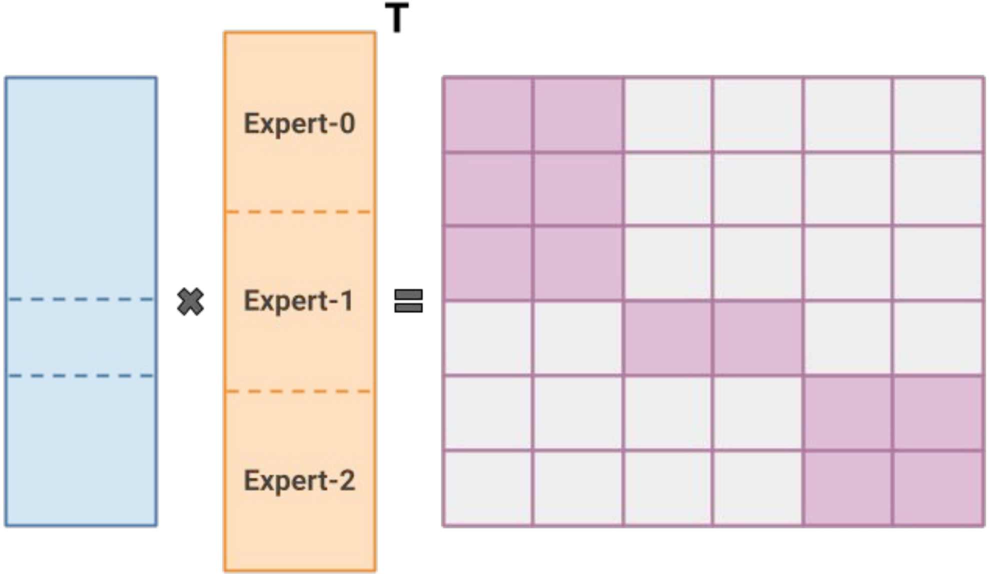
Grouped/Ragged Matmul
Block-sparse, but specific to dropless MoE computation. Easier to maintain and port to new architectures (H100, TPU)

Middle Diagram Source: [“Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer”]

Backup

Roadmap

~~0. Introduction~~
1. MoEs with Block Sparsity
2. Block-Sparse Kernels for MoEs
3. End-to-End Results with dMoEs



MoEs With Block Sparsity

Dropless-MoEs With Block-Sparsity

Dropless-MoE (dMoE) Computation:

1 : **Assign tokens to experts.**

2 : **Construct the sparse matrix from router outputs.**

3 : **Group the tokens by expert assignment.**

4 : **Use block-sparse products to compute expert layers.**

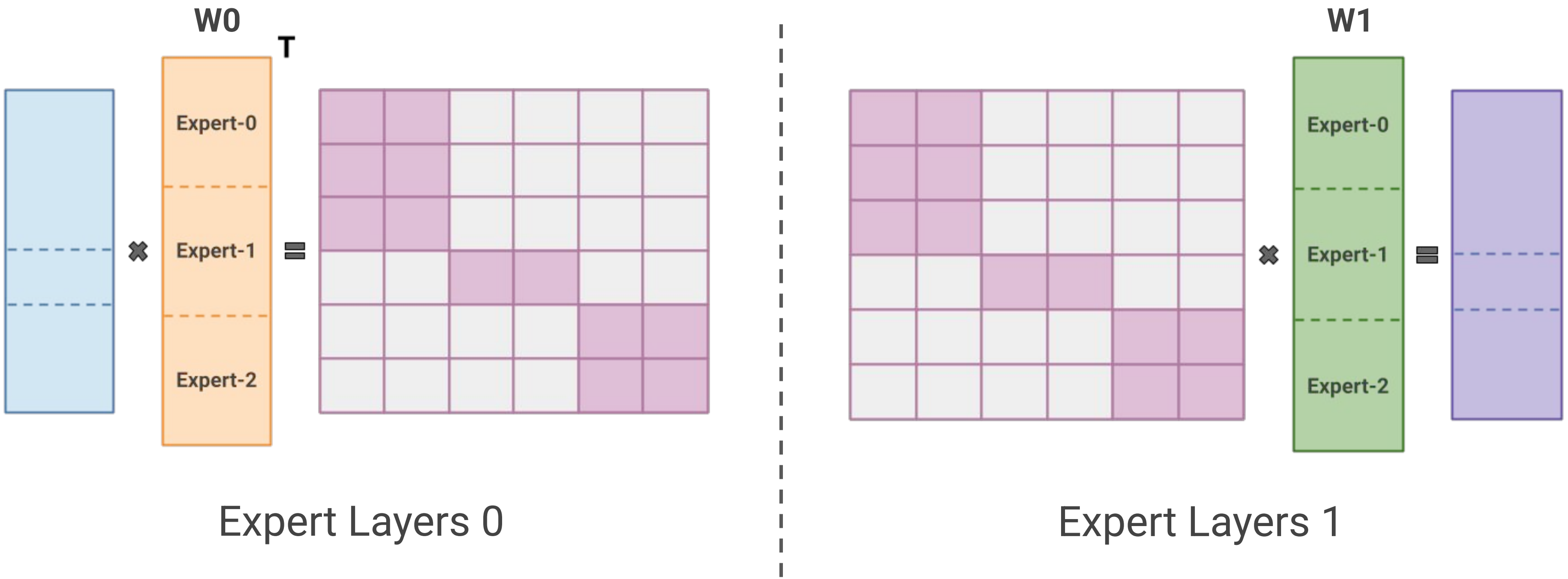
5 : **Un-permute and scale by router weights.**

Pseudocode for dMoE

```
def dmoe_forward(x):
  indices, weights = router(x)

  topology = make_topology(indices)

  x = padded_gather(x, indices)

  x = sdd(x, w1, topology)
  x = dsd(x, w2)

  x = padded_scatter(x, indices)

  return x * weights
```

(Changes for dMoE highlighted in blue)

Multi-Layer Expert Computation



The sparse matrix topology is determined by the router and re-used across the layers of the experts.

Block-Sparse Kernels for MoEs

Block-Sparse Kernels for MoEs

	For high throughput.	Fwd + bwd passes.	No token dropping.	Changes every use.
Library	**Large Blocks**	**Transposition**	**Load Imbalance**	**Fast Construction**
cuSPARSE	✓	✗	✗	✗
Triton Blocksparse	✓	✓	✓	✗

cuSPARSE not an option: no transposes + ELLPACK format.

Blocksparse does expensive preprocessing: **5-10x** slower than dense if not amortized.

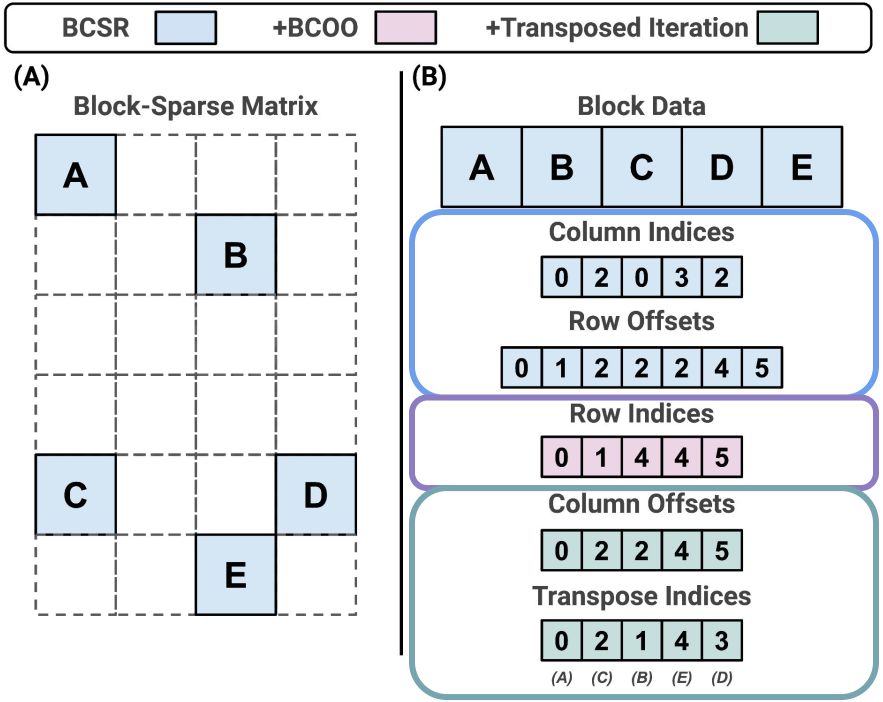
Our Solution: Hybrid Block-Sparse Format

Many "views" of the sparse matrix:

Blocked-CSR: sparse inputs

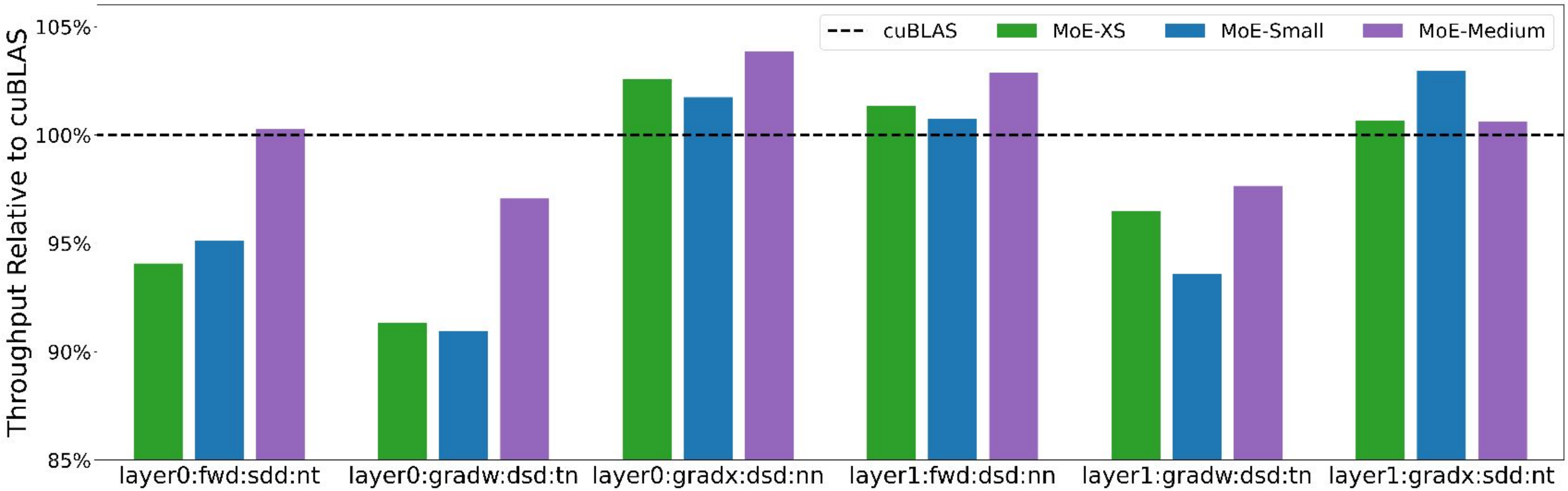
Blocked-COO: sparse outputs

Transpose Index: transposed sparse inputs



Metadata is cheap to compute and store:
<0.1% storage overhead for 128x128 blocks

MegaBlocks Block-Sparse Kernels



98.6% of cuBLAS performance.

A100-SXM4-80GB, CUDA 11.5, block_size=128x128

End-to-End Results

Evaluation Details

MegaBlocks is built on Megatron-LM + PyTorch.

Models:
Transformers-MoEs with 64-experts and top-1 routing.

Baselines:
MoE: Tutel (+ Megatron-LM)
Dense: Megatron-LM

Training:
10B tokens from The Pile on 8x A100 GPUs. Data parallelism for Transformers, 8-way expert model parallelism for MoE layers.

capacity_factor={1, 1.5, 2.0} for MoE baselines.

Transformer	hidden_size	num_layers	Weights (M)	GFLOPs
XS	512	6	46	316
Small	768	12	125	879
Medium	1024	24	356	2487
Large	1536	24	760	5122
XL	2048	24	1316	8684

Table 1: Baseline Transformer Models.

MoE	num_experts	top_k	Weights (M)	GFLOPs
XS	64	1	839	316
Small	64	1	3,693	879
Medium	64	1	13,041	2487

Table 2: Transformer-MoE Models.

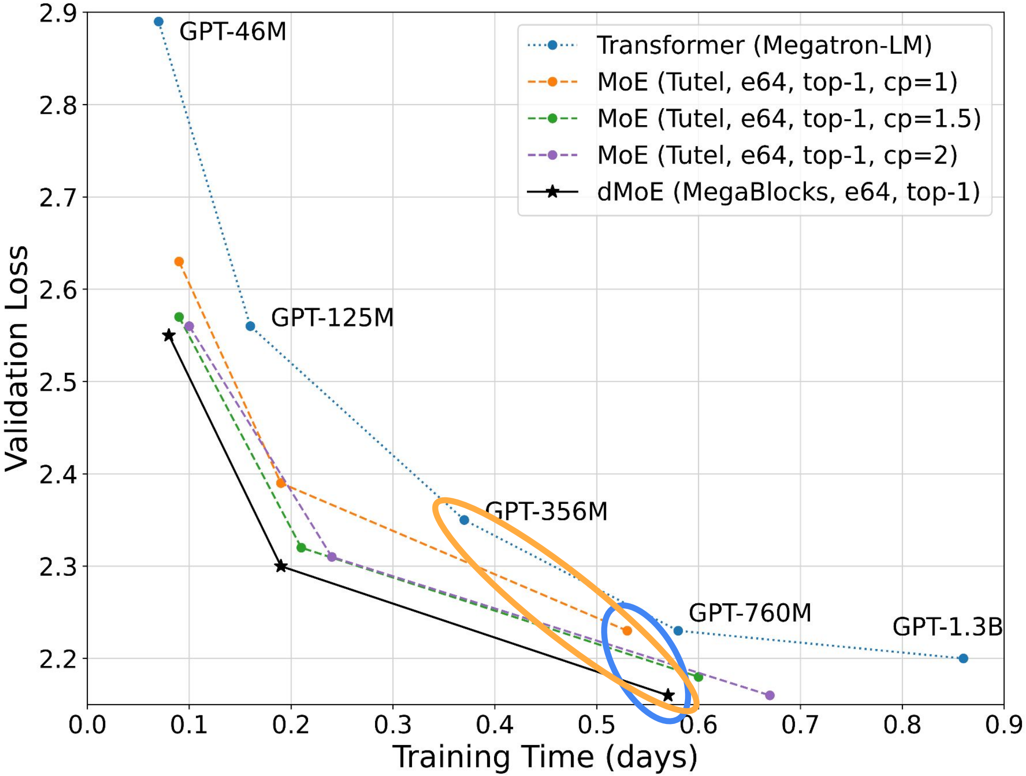
Training Transformer Language Models

Compared to best performing configuration with the same quality:

1.2x - 1.4x faster than Tutel MoEs.
1.8x - 2.4x faster than Megatron-LM Transformers.

MegaBlocks cp=1 speed and cp=inf quality.

- Some slowdown with smaller batch sizes from padding to 128.
- Some slowdown from using smaller batch than dense (memory usage).



MegaBlocks Retrospective