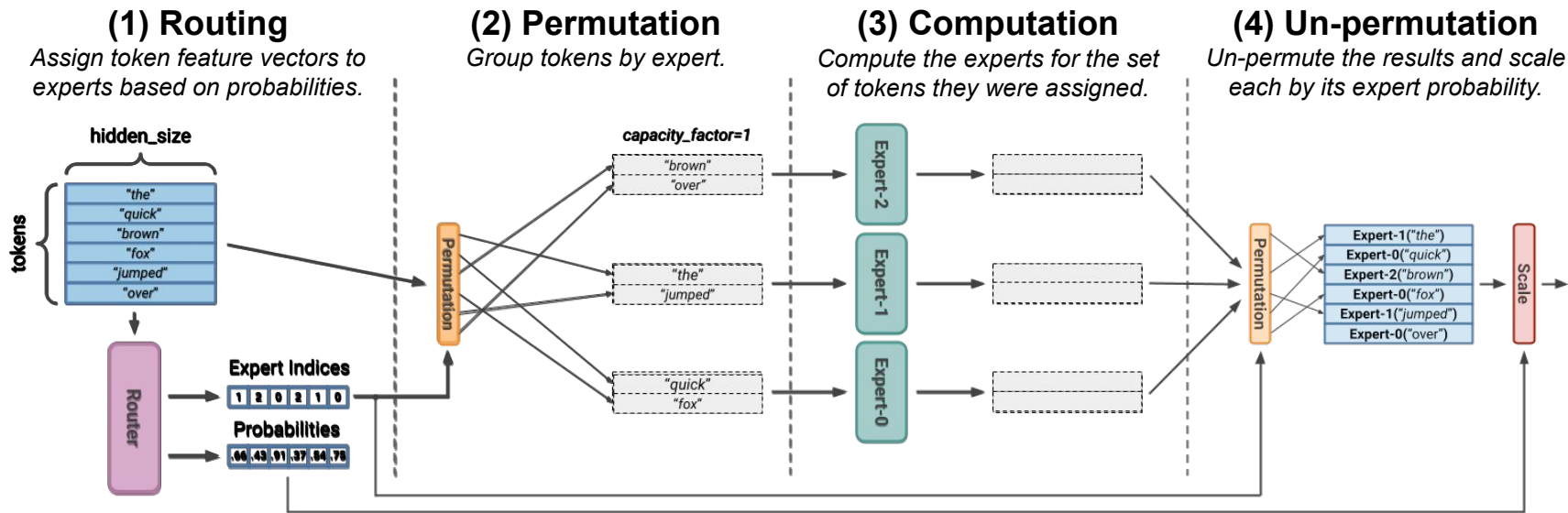


MegaBlocks: Efficient Sparse Training with Mixture-of-Experts

Trevor Gale^{*†}, Deepak Narayanan[‡], Cliff Young[†], Matei Zaharia^{*}

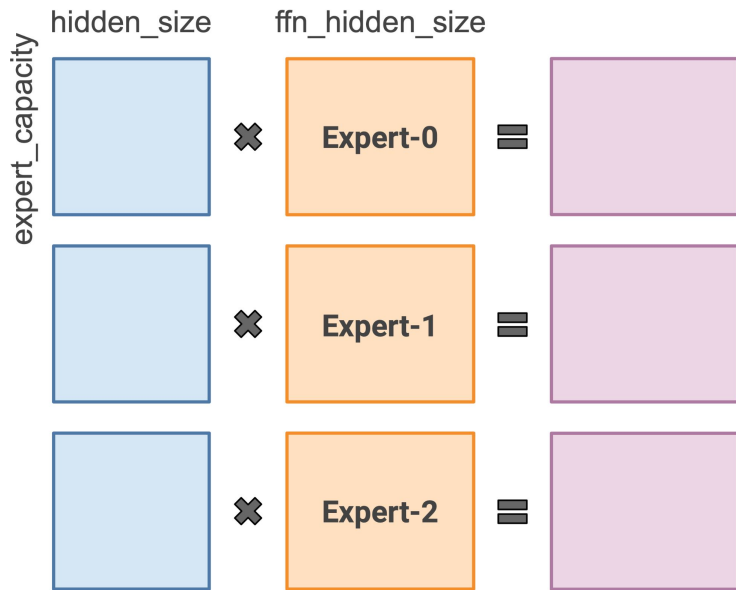
^{*}Stanford University, [†]Google DeepMind, [‡]Microsoft Research
UMD, March 2025

Mixture-of-Experts Layers



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

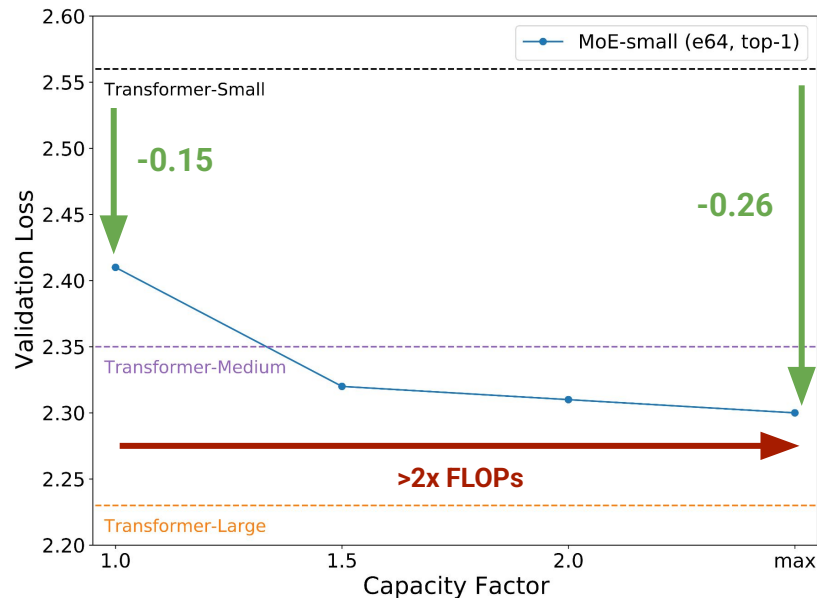
Batched Expert Computation



Batched Matrix Multiplication

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

Bad: Introduces quality-speed tradeoff.



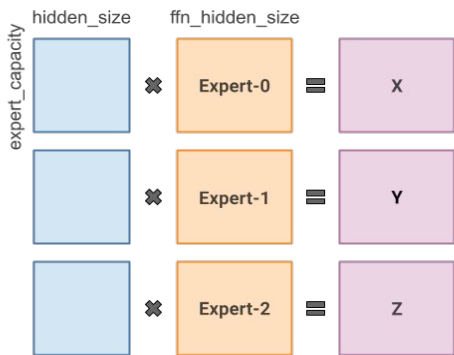
Token Dropping

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

MegaBlocks: Mixture-of-Experts with Structured Sparsity

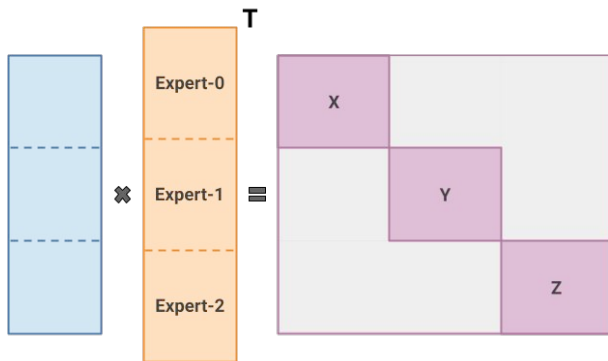
(Current) Batched Matmul

Parallel, but constrained expert computation.



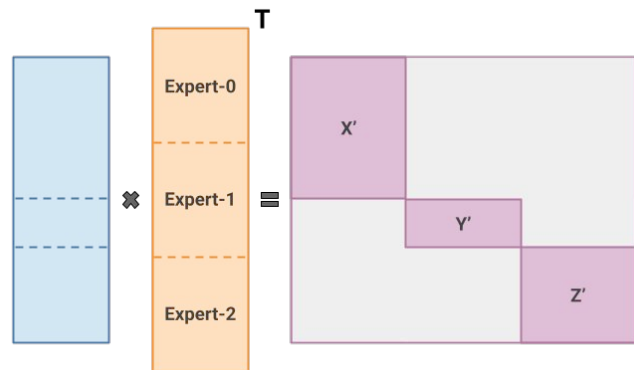
(Reformulation) Block Diagonal Matmul

Expert computation with block diagonal matrices.



(Generalization) Load Imbalanced Routing

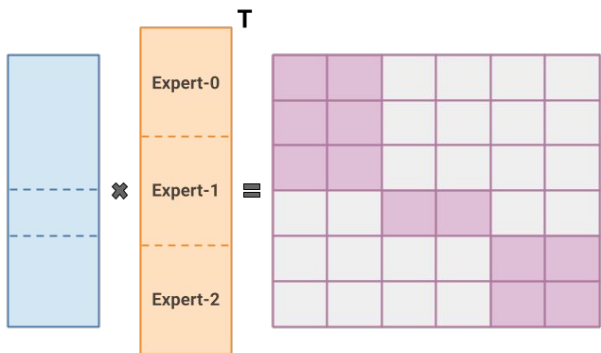
Use variable block sizes to enable load imbalanced routing.



MegaBlocks: Mixture-of-Experts as Structured Sparsity

(This Paper) Load Imbalanced Routing

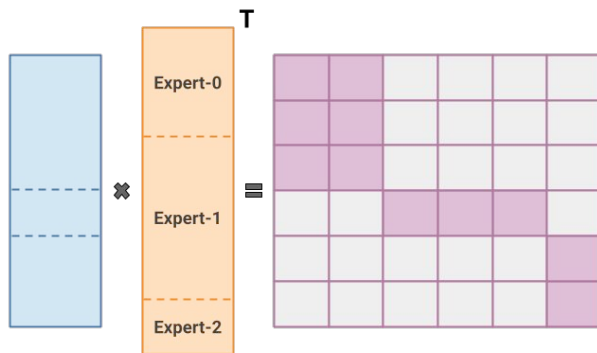
Realized via block-sparsity.



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

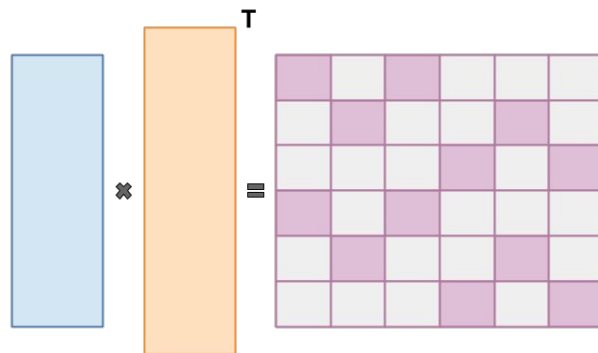
(Future) Adaptive Computation

With variable sized experts.



(Future) Dynamic Activation Sparsity

Maximum flexibility with block-sparsity.

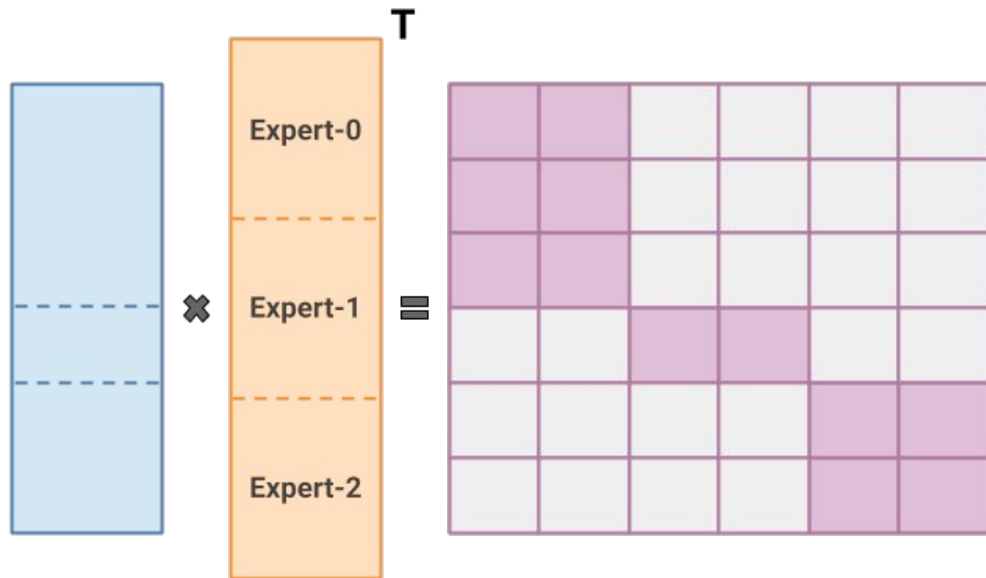


This is the first step towards our goal to improve quality / flop by generalizing MoEs.

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

Droptless-MoEs With Block-Sparsity

Droptless-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.

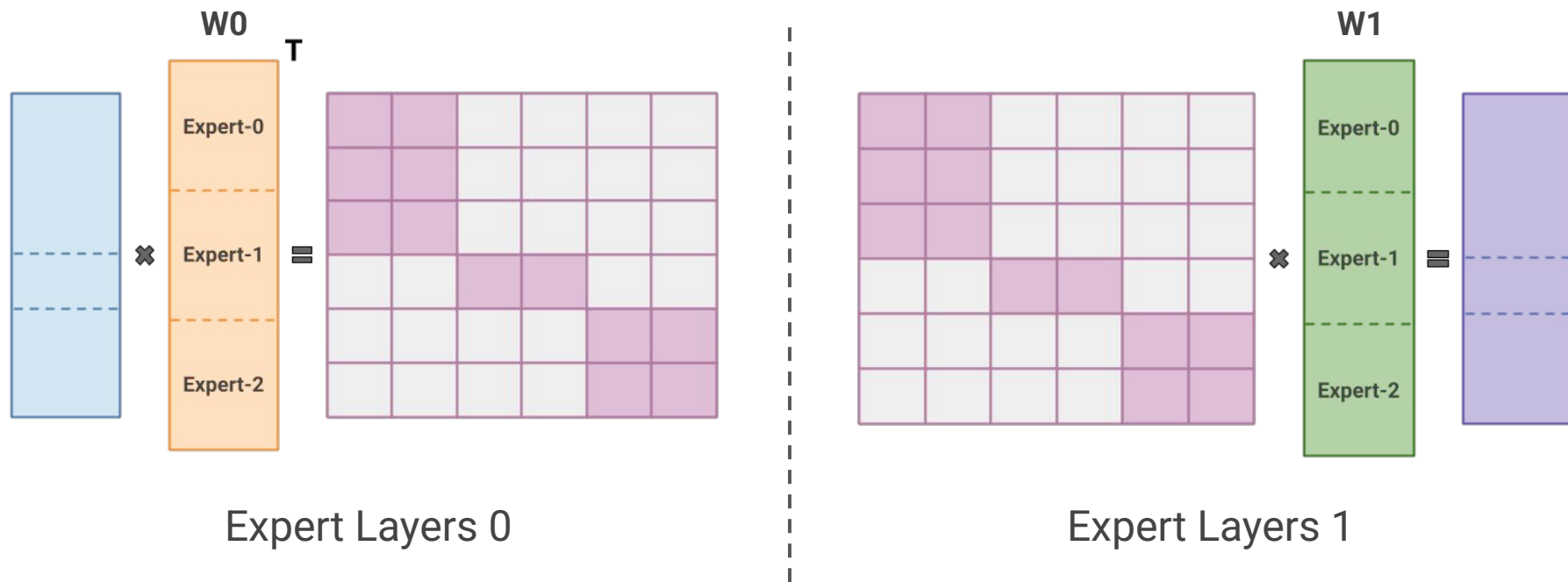
5 : Un-permute and scale by router weights.

Pseudocode for dMoE

```
1 def dmoe_forward(x):  
2     indices, weights = router(x)  
3  
4     topology = make_topology(indices)  
5  
6     x = padded_gather(x, indices)  
7  
8     x = sdd(x, w1, topology)  
9     x = dsd(x, w2)  
10  
11    x = padded_scatter(x, indices)  
12  
13    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

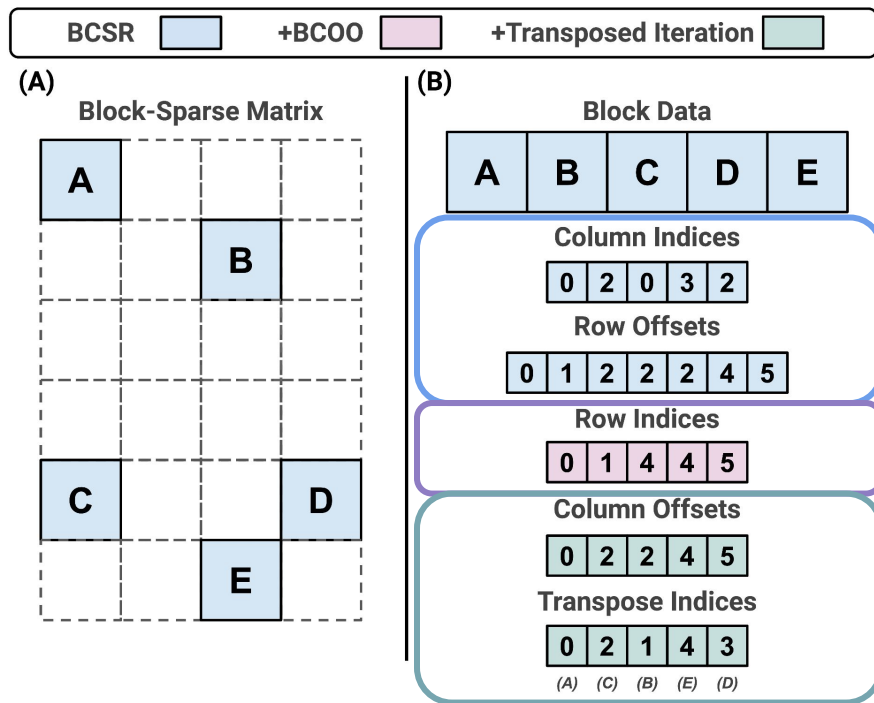
Many “views” of the sparse matrix:

Blocked-CSR: sparse inputs

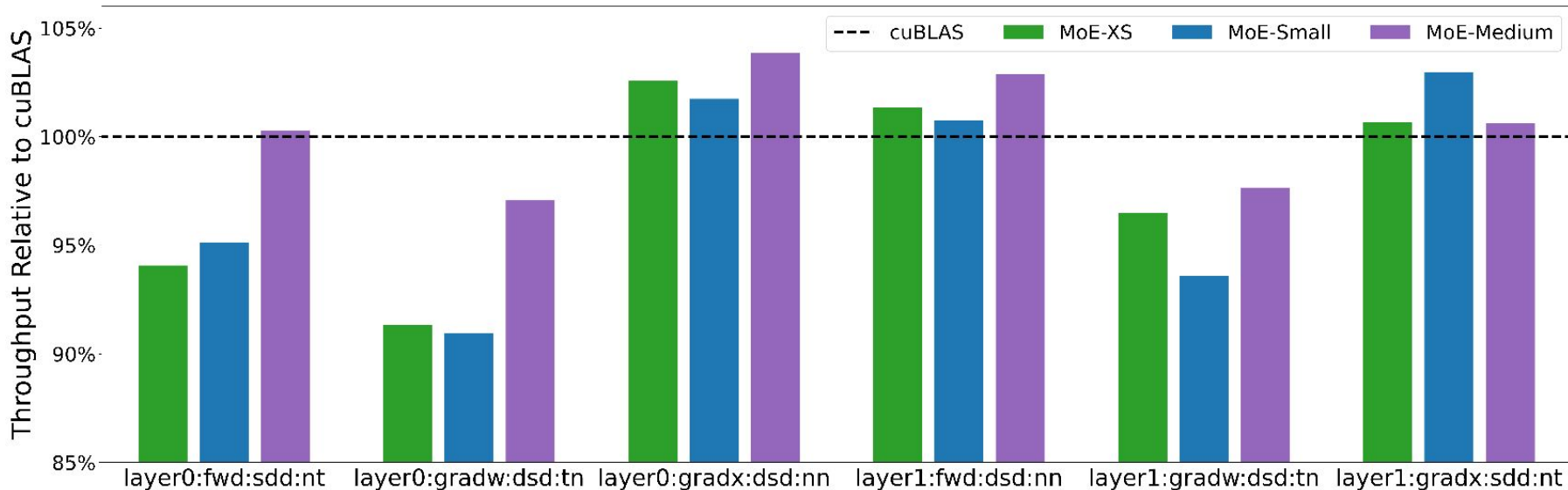
Blocked-COO: sparse outputs

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.

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

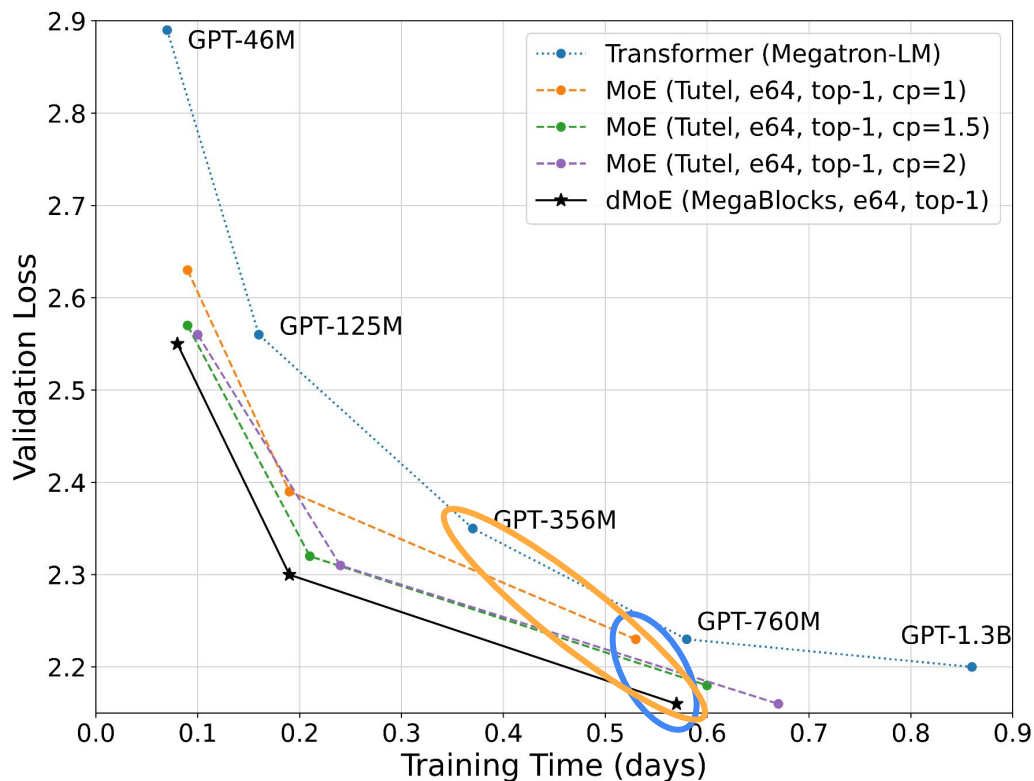
Compared to best performing configuration with the same quality:

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).



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 (!)

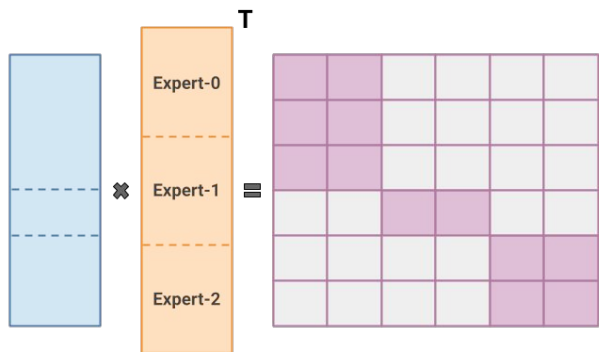


github.com/google/jax => ops, written in Pallas
github.com/google/maxtext => dMoE in JAX on TPU
github.com/pytorch/xla => dMoE in PyTorch on TPU

Questions?

(This Paper) Load Imbalanced Routing

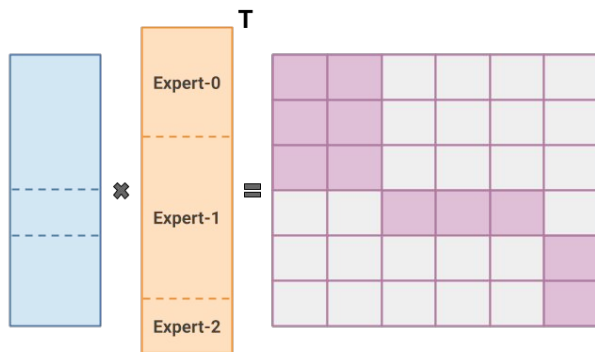
Realized via block-sparsity.



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

(Future) Adaptive Computation

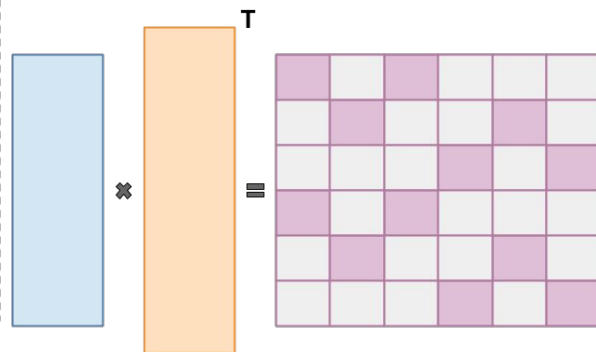
With variable sized experts.



???

(Future) Dynamic Activation Sparsity

Maximum flexibility with block-sparsity.



???



github.com/stanford-futuredata/megablocks



@tgale96