Scorch

A library for Sparse machine learning

Bobby Yan[†], Alexander J. Root[†], Trevor Gale[†‡], Fredrik Kjolstad[†]
[†]Stanford University, [‡]Google Research

AHA Retreat 2023 | August 31, 2023

Sparse machine learning

The Future of Sparsity in Deep Neural Networks

by Trevor Gale on Dec 3, 2020 | Tags: Accelerators, deep learning, sparsity

Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT

Jul 20, 2021

+1 Like Discuss (13)

By Jeff Pool, Abhishek Sawarkar and Jay Rodge

DeepSparse

NEURAL MAGIC

An inference runtime offering GPU-class performance on CPUs and APIs to integrate ML into your application

DOCUMENTATION | SLACK | SUPPORT FORUMS | BUILD HTTPS://GITHUB.COM/BADGES/SHIELDS/ISSUES/8671 | RELEASE V1.5.3

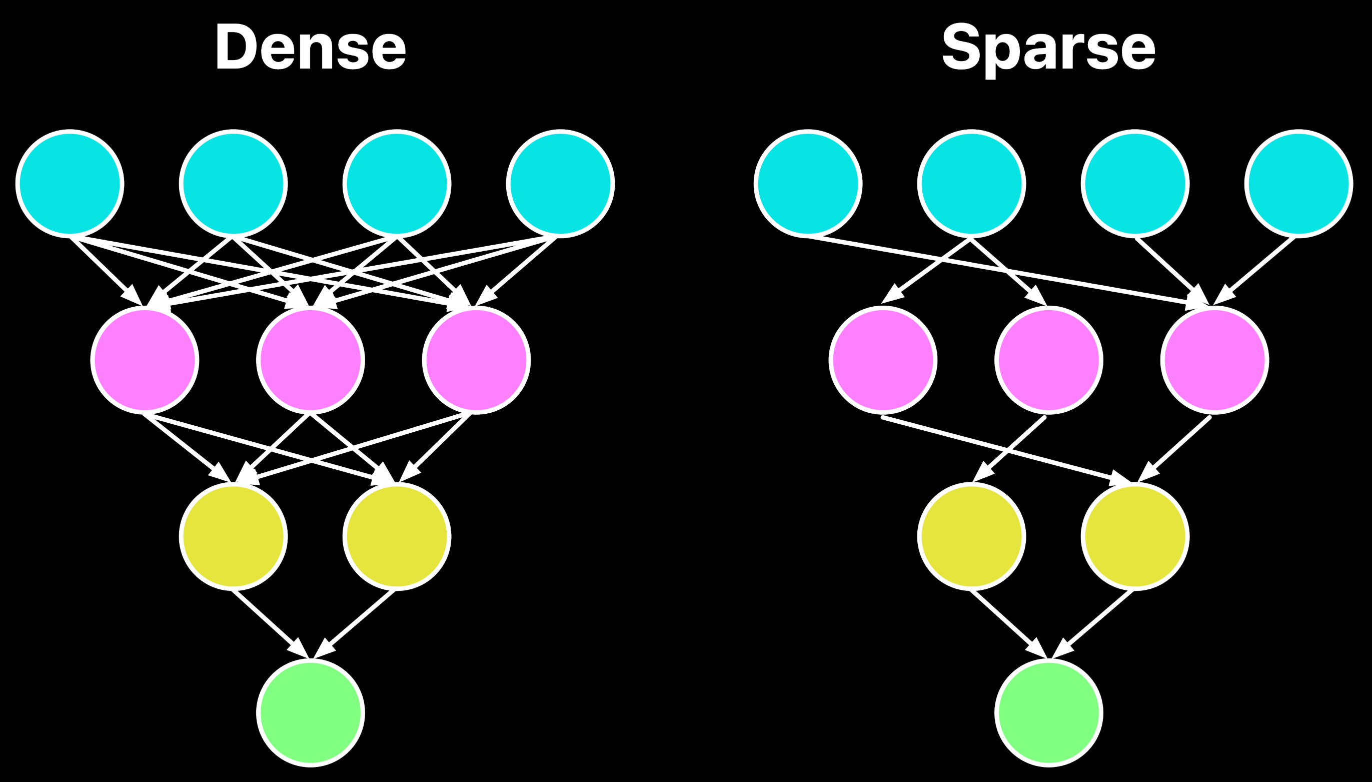
CONTRIBUTOR COVENANT V2.1 ADOPTED | YOUTUBE | MEDIUM | Follow

DeepSparse is a CPU inference runtime that takes advantage of sparsity within neural networks to execute inference quickly. Coupled with SparseML, an open-source optimization library, DeepSparse enables you to achieve GPU-class performance on commodity hardware.

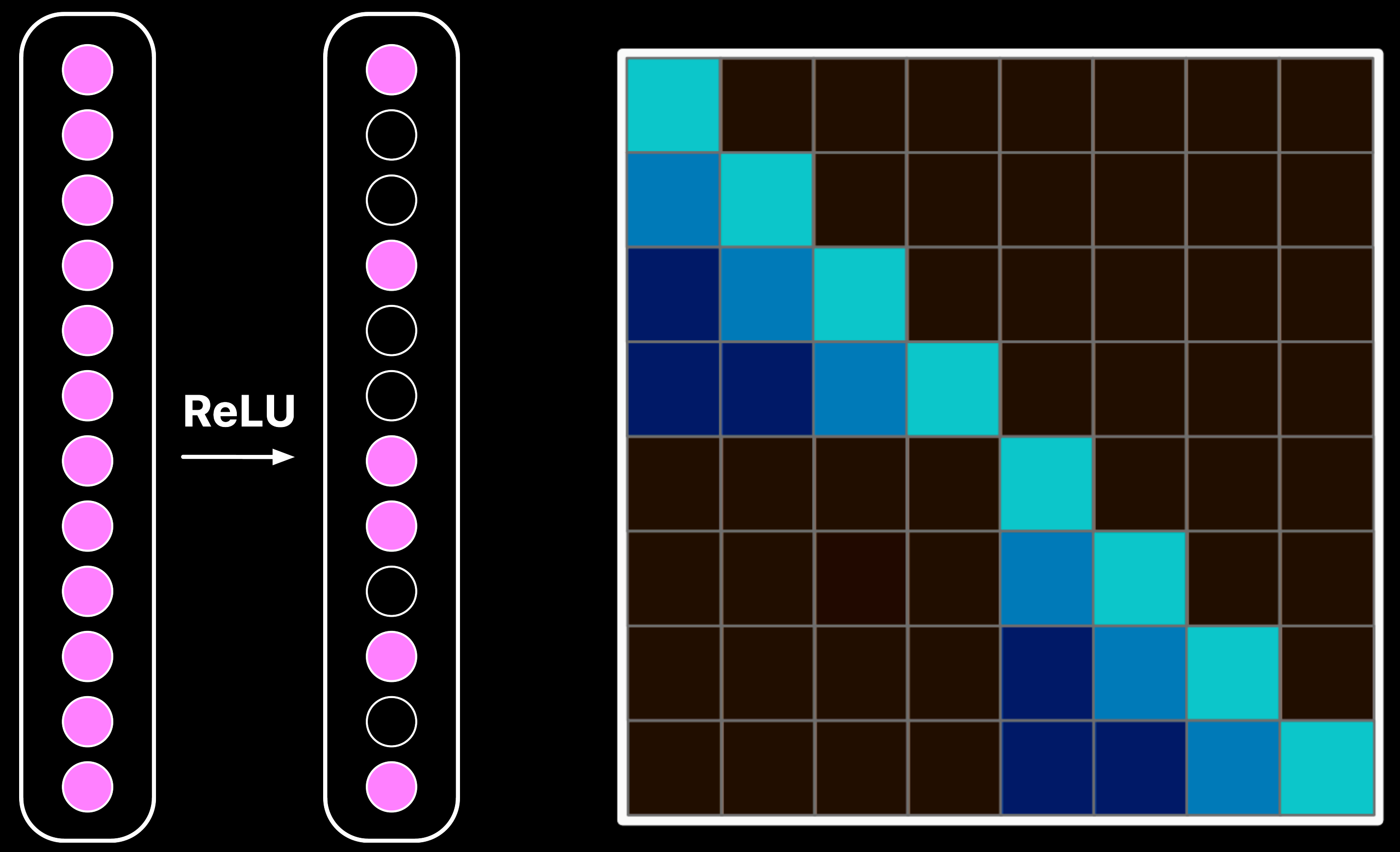
Two types of sparsity

Model Sparsity

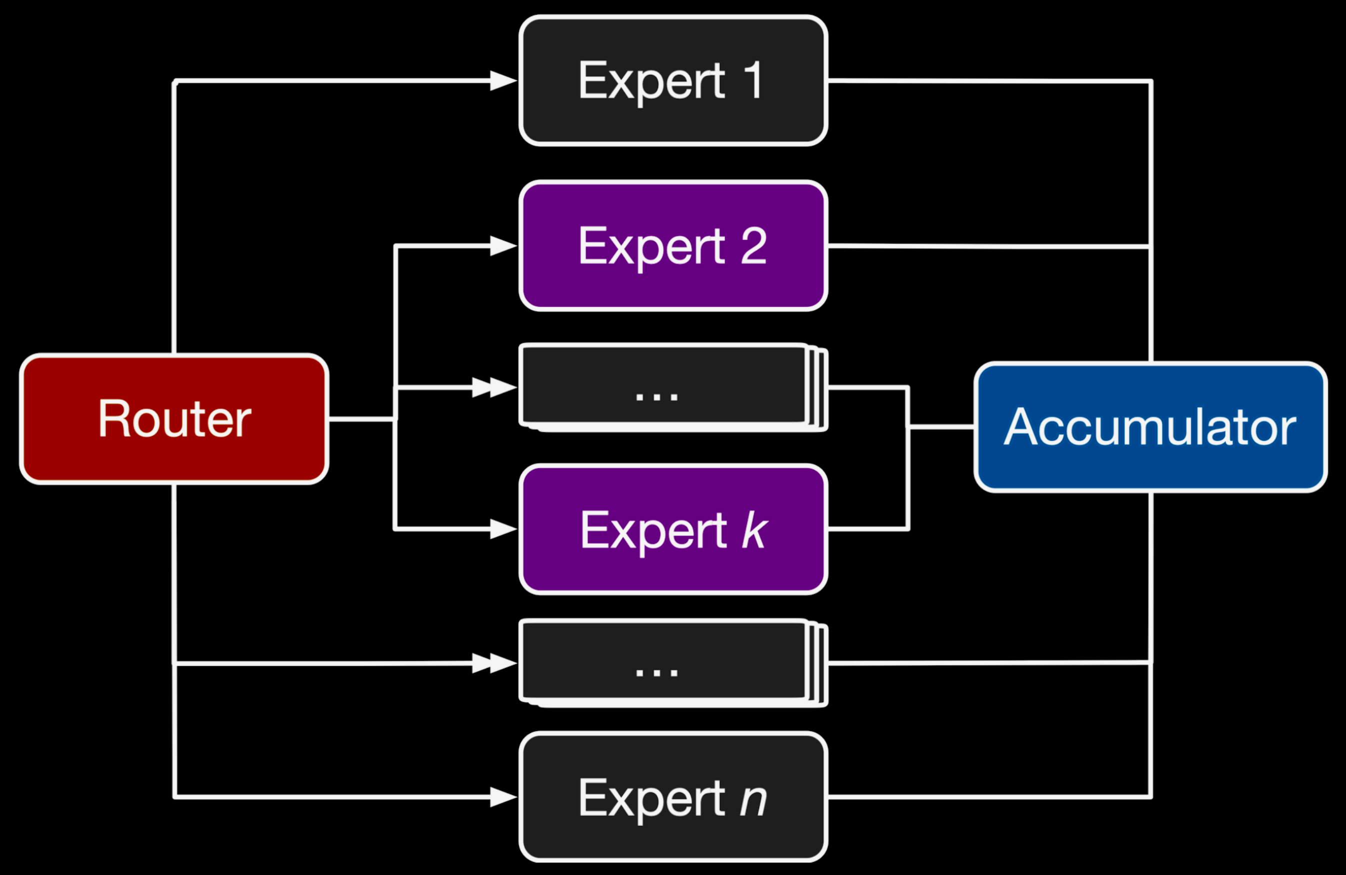
Model Sparsity



Weight Sparsity



Activation Sparsity



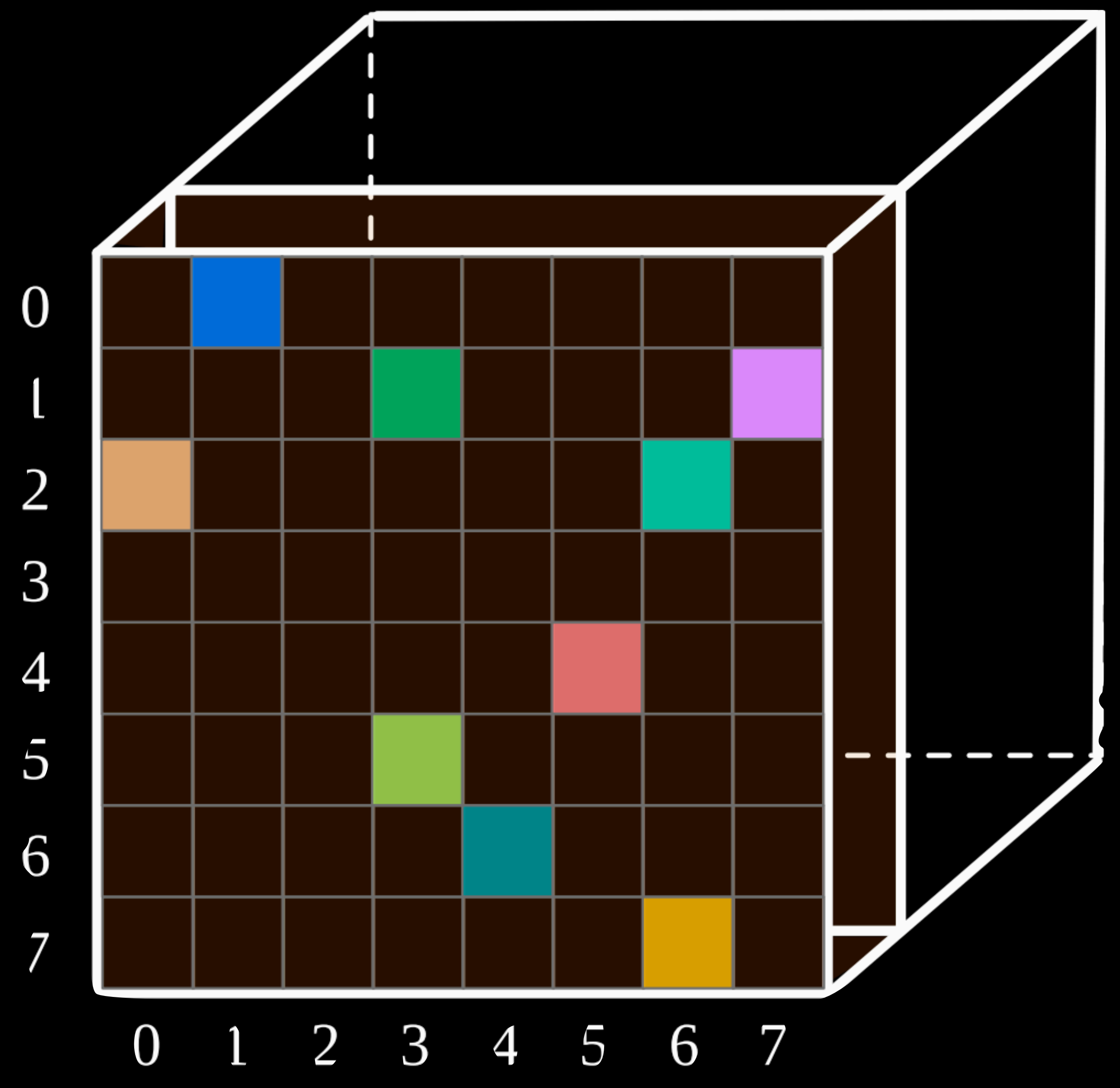
Structural Sparsity

Two types of sparsity

Model Sparsity

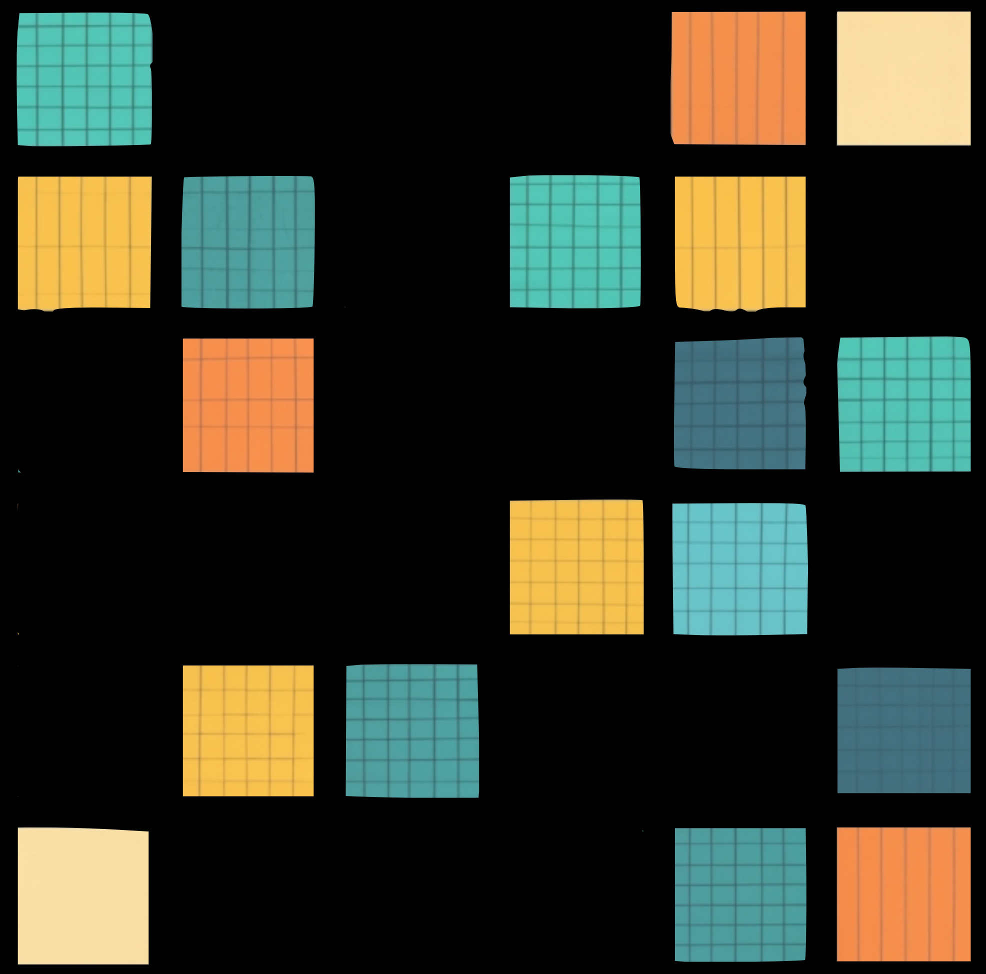
Data Sparsity

Data Sparsity



Graphs

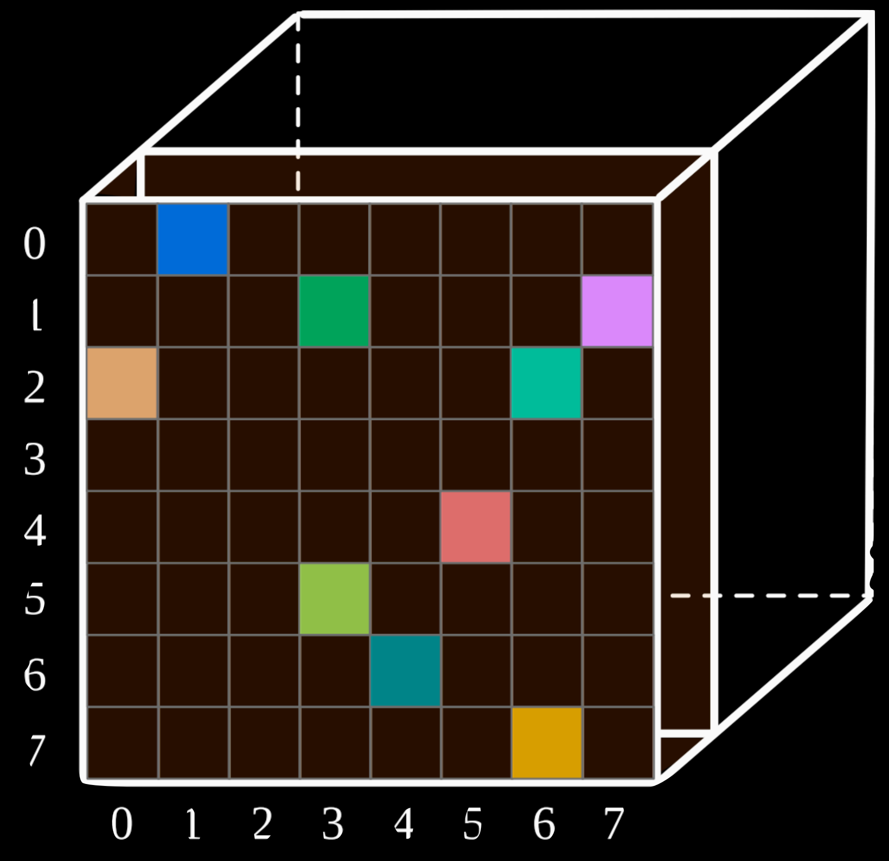
Point clouds

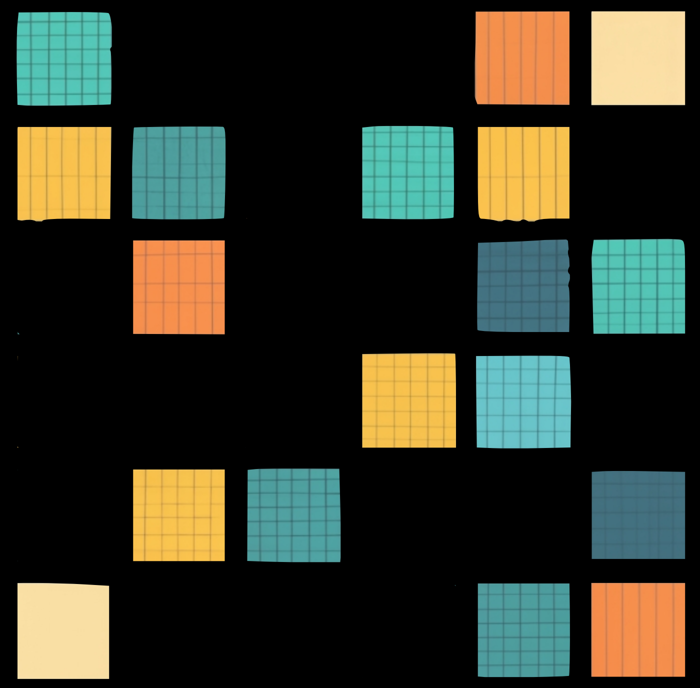


Recommender systems

Image source: https://icnweb.kr/2021/46155/

Data Sparsity

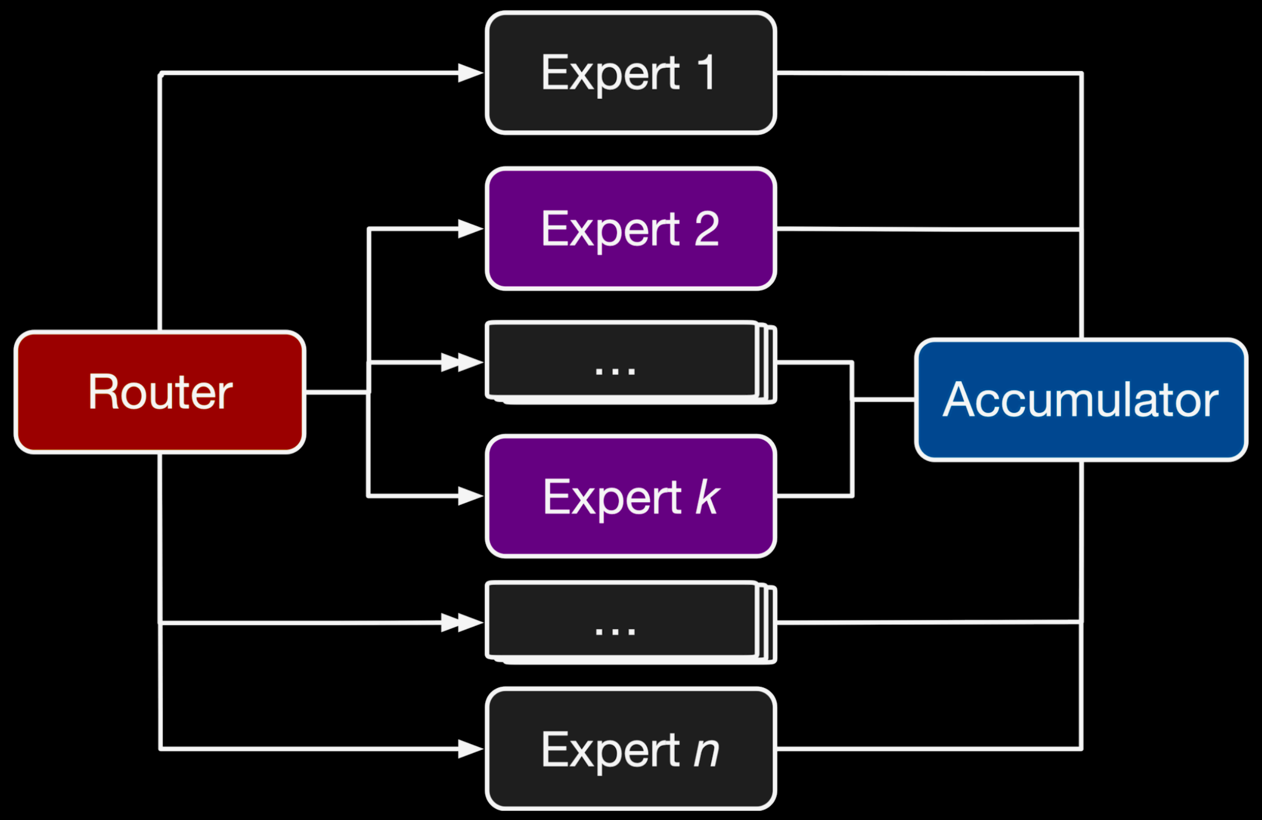




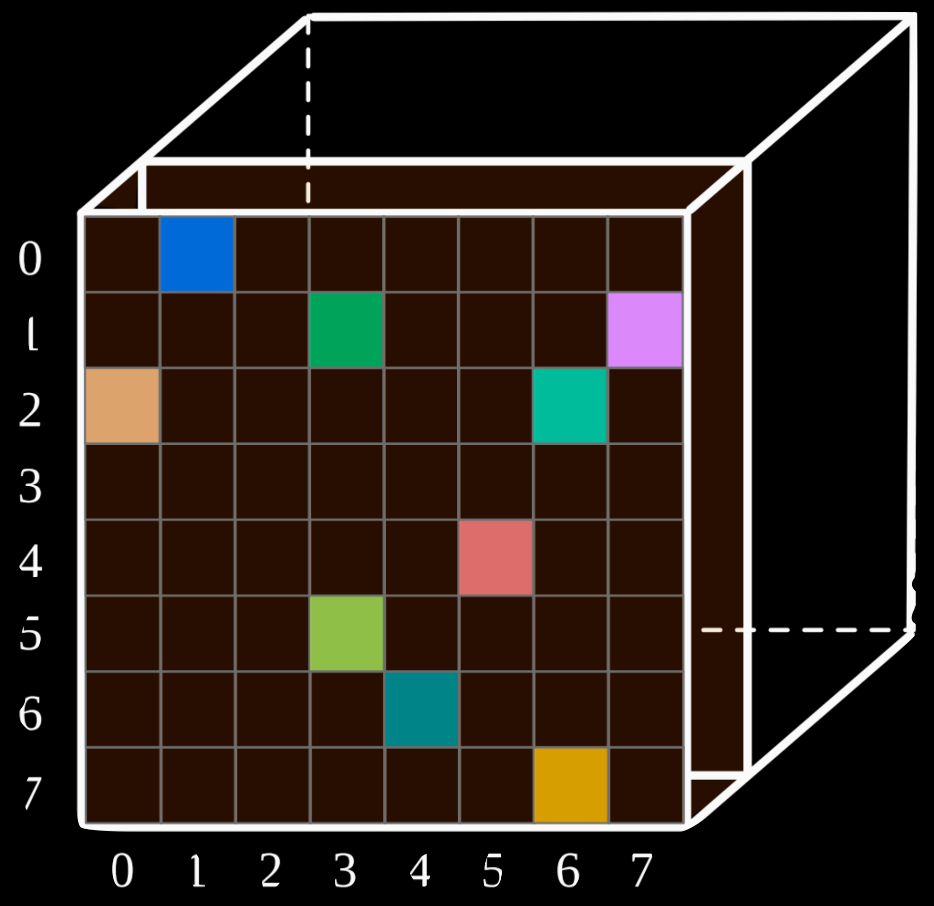
Graphs	Point clouds	Recommender systems
Bag of words	Time series	Speech data
Genomic data	Transaction data	Sensor data

Image source: https://icnweb.kr/2021/46155/

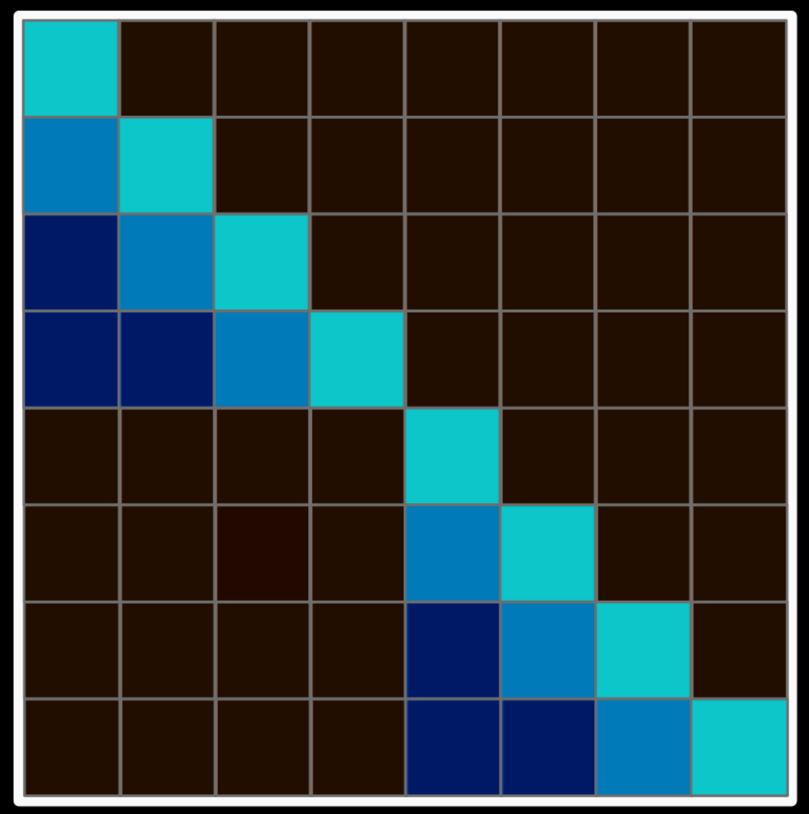
Sparsity comes from data and model design



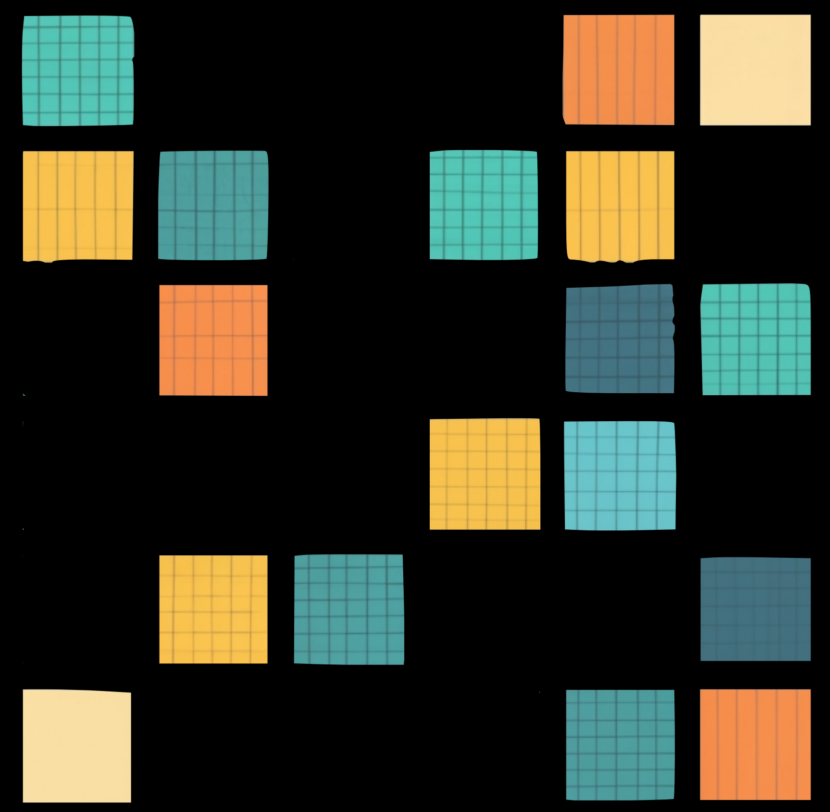
Mixture of experts



Graph neural networks

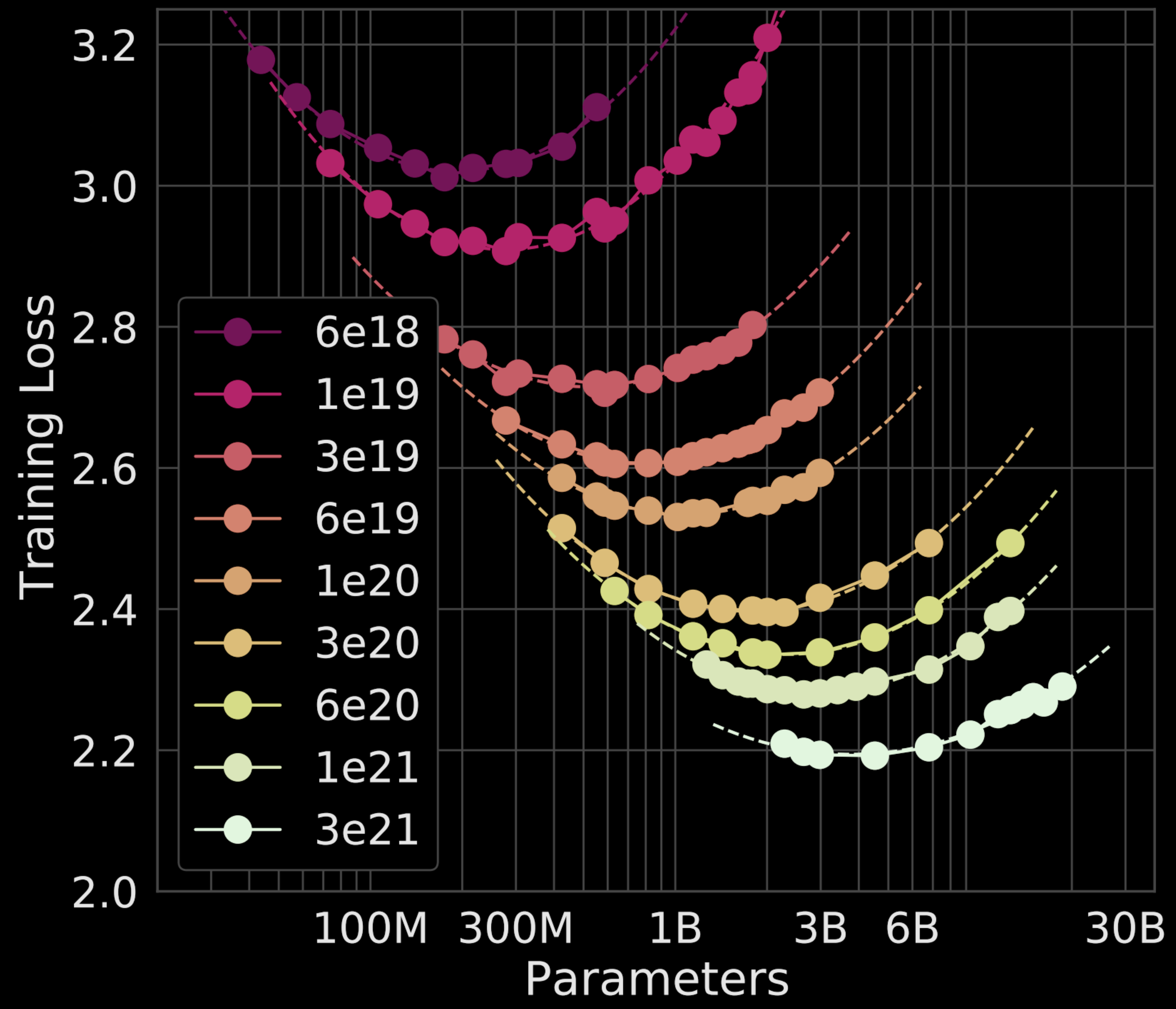


Sparse transformers

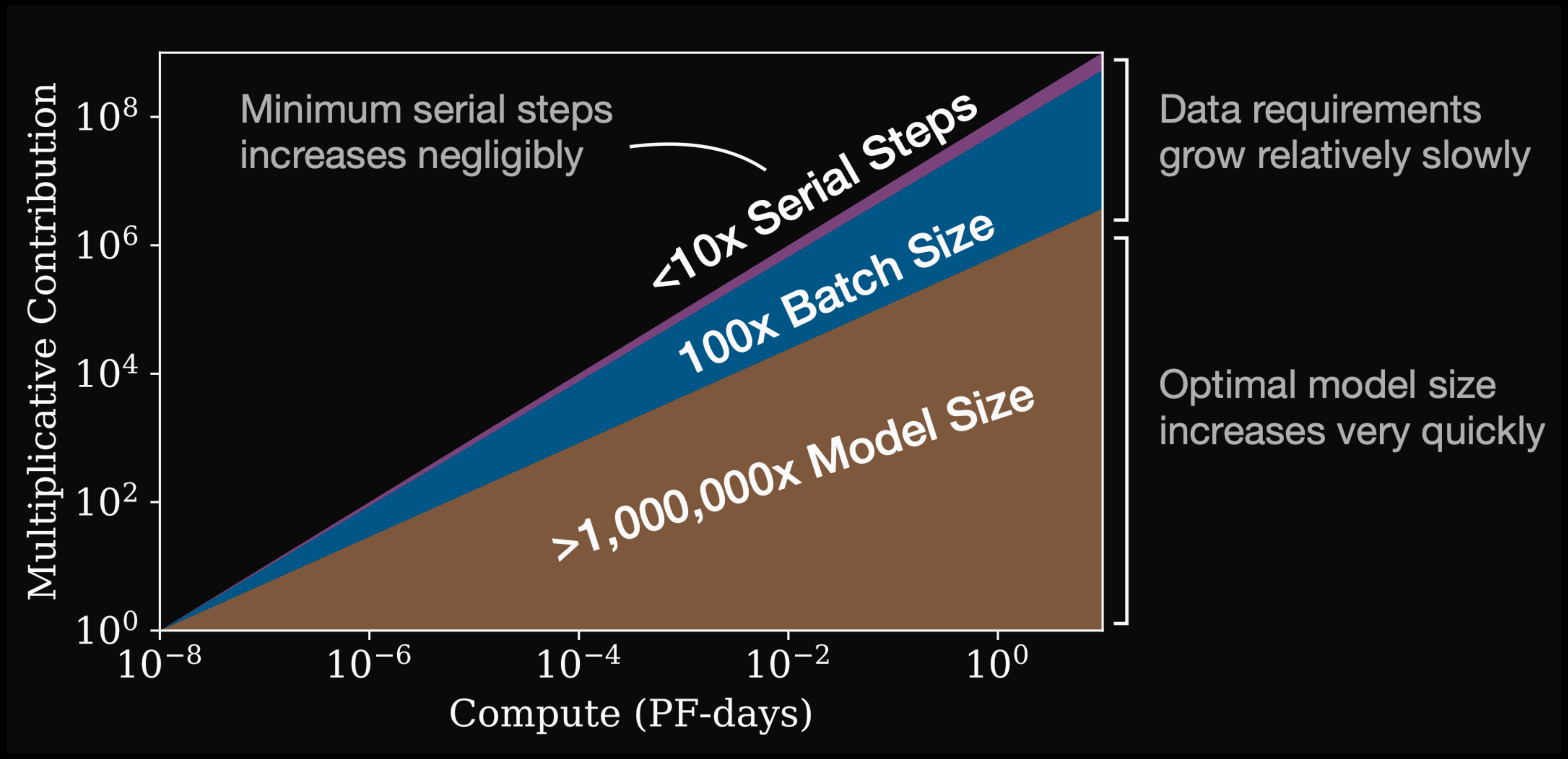


Recommender systems

Bigger is better, so we need sparsity



Training Compute-Optimal Large Language Models, 2022



Scaling Laws for Neural Language Models, 2020

Stanford | Computer Science

Image sources: https://arxiv.org/pdf/2203.15556.pdf, https://arxiv.org/pdf/2001.08361.pdf

We should care about Sparse ML.

Software for Sparse ML

Software for Dense ML

Dense programming model is unified

PyTorch TensorFlow 2 JAX

Common abstraction

Similar APIs

Similar feature sets

Sparse programming model is fragmented

torch.sparse

tf.sparse

jax.sparse

PyG

DGL

LightFM

TorchRec

TACO

MLIR Sparse

Sparse programming model is fragmented

torch.sparse

tf.sparse

jax.sparse

PyG

DGL

LightFM

TorchRec

Differing abstractions

Isolated optimizations

Duplicated efforts

Barriers to adoptions

Sparse programming model is fragmented

torch.sparse

tf.sparse

jax.sparse

PyG

DGL

LightFM

TorchRec

TACO

MLIR Sparse

Sparse programming model can be unified

torch.sparse

tf.sparse

jax.sparse

PyG

DGL

Scorch

LightFM

TorchRec

TACO

MLIR Sparse

Tensor algebra is all you need.

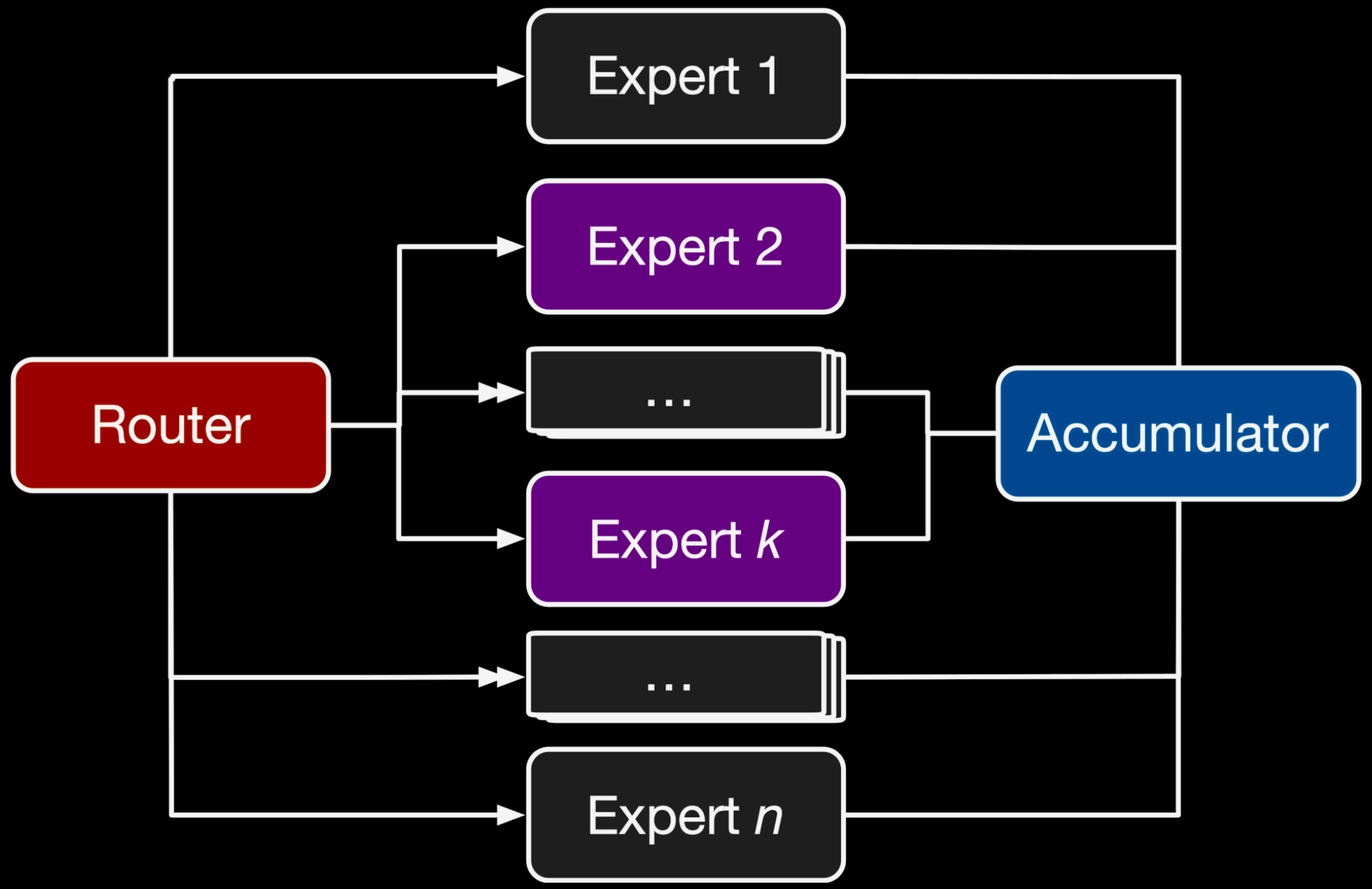
Sparse learning should be easy.

Making it happen

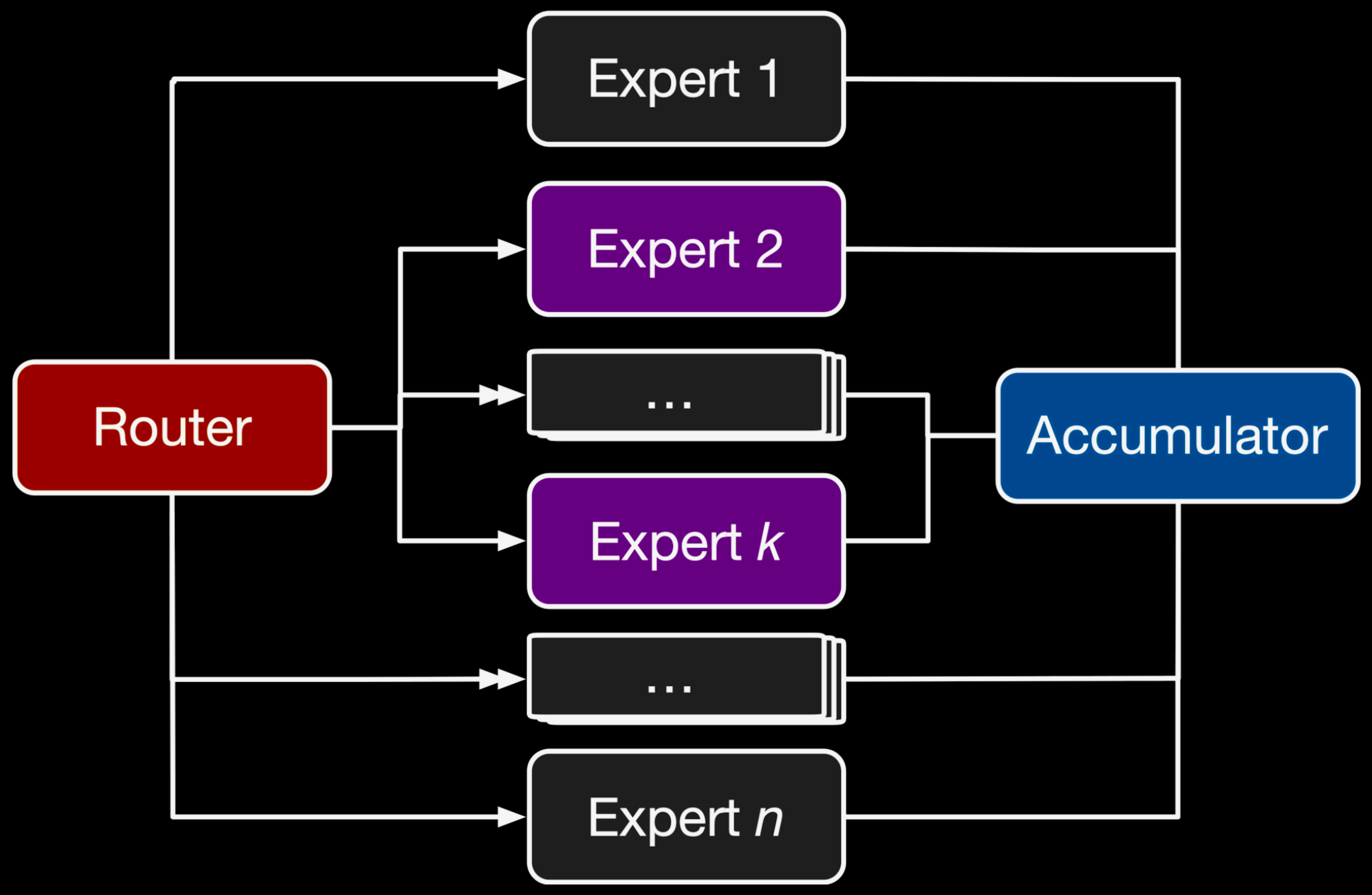
```
import scorch as torch
```

Do everything you're doing with dense tensors.
Now with sparse tensors, too.
Can it be any easier?

Sparse activation with dense weights?



Sparse activation with dense weights?




```python
import scorch as torch
from scorch.nn import functional as F

# Inputs, (B, D_in)
x = torch.randn(B, D_in)
# Expert embeddings, (N_experts, D_in, D_out)
E = torch.randn(N_experts, D_in, D_out)
# Sparse gating function, (B, N_experts)
gates = torch.rand(B, N_experts)
# Select one expert per input
gates = F.one_hot(gates.argmax(1), N_experts)
# Dispatch inputs to experts, (B, N_experts, D_in)
x_dispatch = torch.rearrange(x, "bd->bnd", n=N_experts)
# Apply experts, (B, N_experts, D_out)
y_experts = torch_einsum("bnd,ndh->bnh", x_dispatch, E)
# Combine expert outputs, (B, D_out)
y = torch.einsum("bnd,bn->bd", y_experts, gates)
```

Coarse-grained, high-level **structural sparsity** in model architecture.

Sparse formulation.

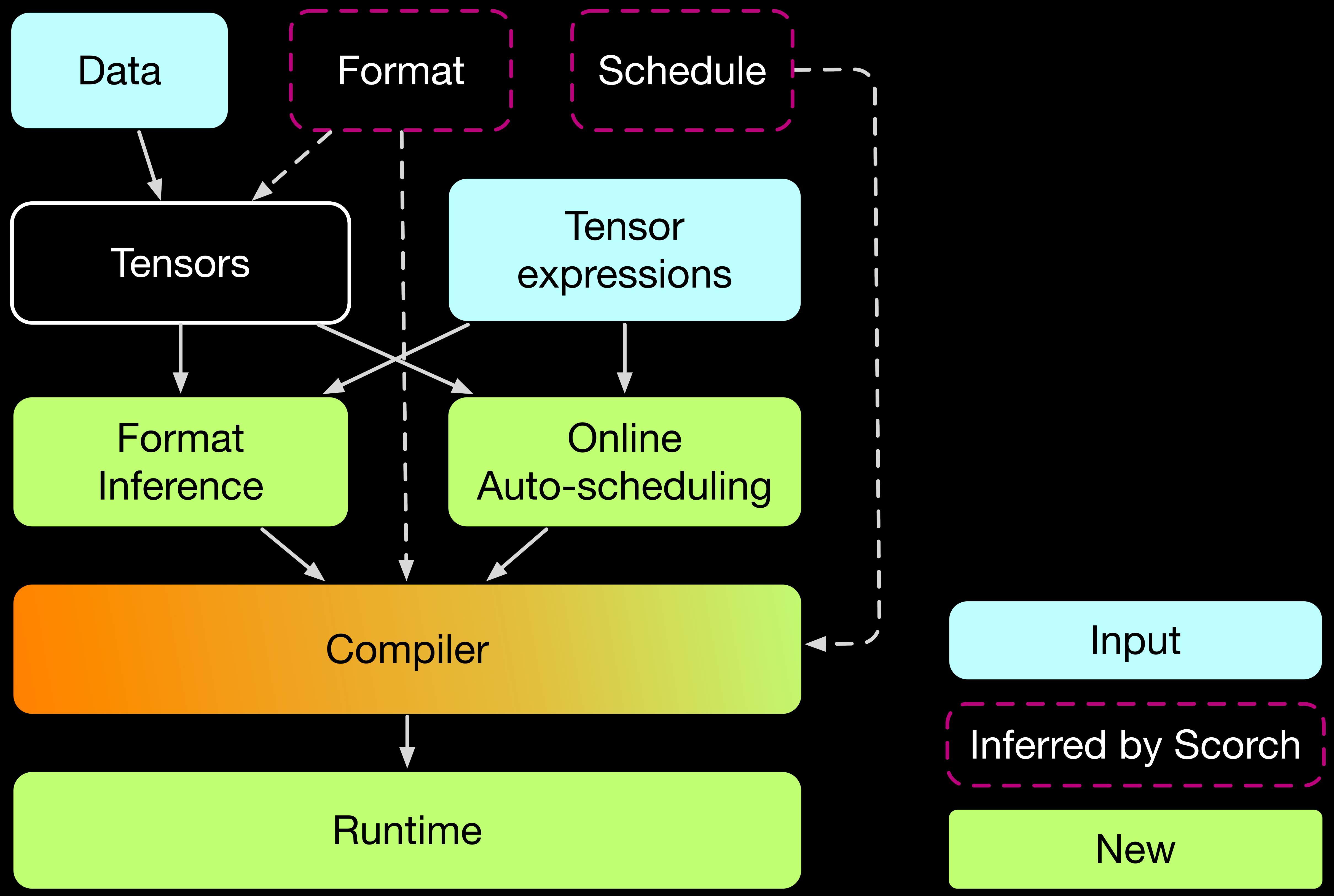
Example: Mixture of experts

```python
import torch
from torch.nn import functional as F

# Inputs, (B, D_in)
x = torch.randn(B, D_in)
# Expert embeddings, (N_experts, D_in, D_out)
E = torch.randn(N_experts, D_in, D_out)
# Sparse gating function, (B, N_experts)
gates = torch.rand(B, N_experts)
# Select one expert per input
gates = F.one_hot(gates.argmax(1), N_experts)
# Dispatch inputs to experts, (B, N_experts, D_in)
x_dispatch = torch.rearrange(x, "bd->bnd", n=N_experts)
# Apply experts, (B, N_experts, D_out)
y_experts = torch_einsum("bnd,ndh->bnh", x_dispatch, E)
# Combine expert outputs, (B, D_out)
y = torch.einsum("bnd,bn->bd", y_experts, gates)
```

Example: Mixture of experts

```python
import scorch as torch
from scorch.nn import functional as F

# Inputs, (B, D_in)
x = torch.randn(B, D_in)
# Expert embeddings, (N_experts, D_in, D_out)
E = torch.randn(N_experts, D_in, D_out)
# Sparse gating function, (B, N_experts)
gates = torch.rand(B, N_experts)
# Select one expert per input
gates = F.one_hot(gates.argmax(1), N_experts).to_sparse()
# Dispatch inputs to experts, (B, N_experts, D_in)
x_dispatch = torch.rearrange(x, "bd->bnd", n=N_experts)
# Apply experts, (B, N_experts, D_out)
y_experts = torch_einsum("bnd,ndh->bnh", x_dispatch, E)
# Combine expert outputs, (B, D_out)
y = torch.einsum("bnd,bn->bd", y_experts, gates)
```

Overview



Data
Format
Schedule
Tensors
Tensor expressions
Format Inference
Online Auto-scheduling
Compiler
Runtime
Input
Inferred by Scorch
New

Optimization in Scorch

Format inference

Auto-scheduling

Multi-dimensional sparse workspaces

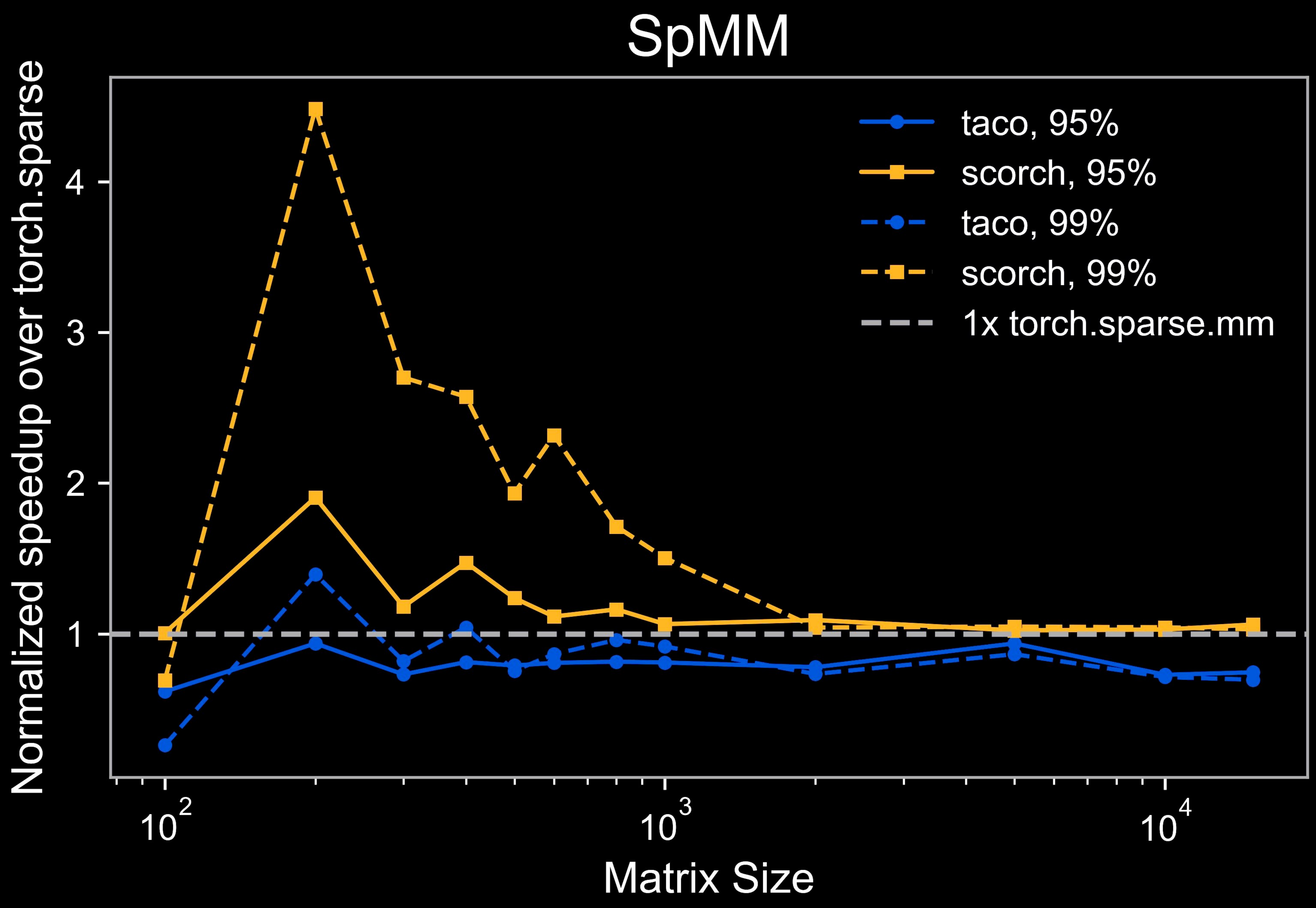
Dynamic dispatch

Compiler architecture

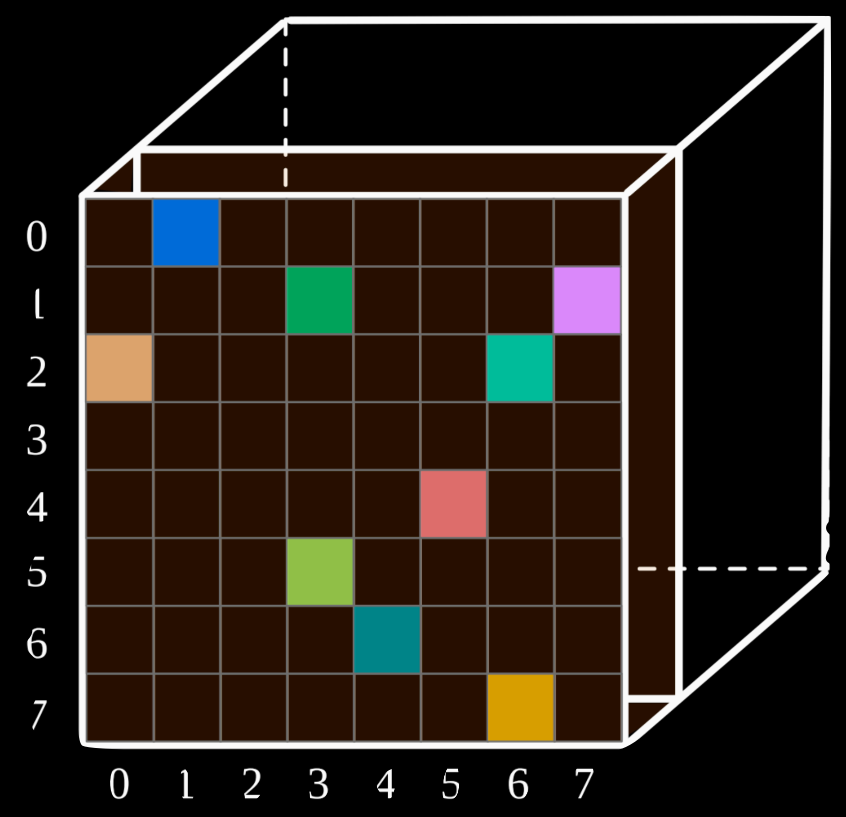
Simplified compilation model over TACO

- Format abstraction
- N-dimensional sparse workspaces
- Data structure selection
- Code generation

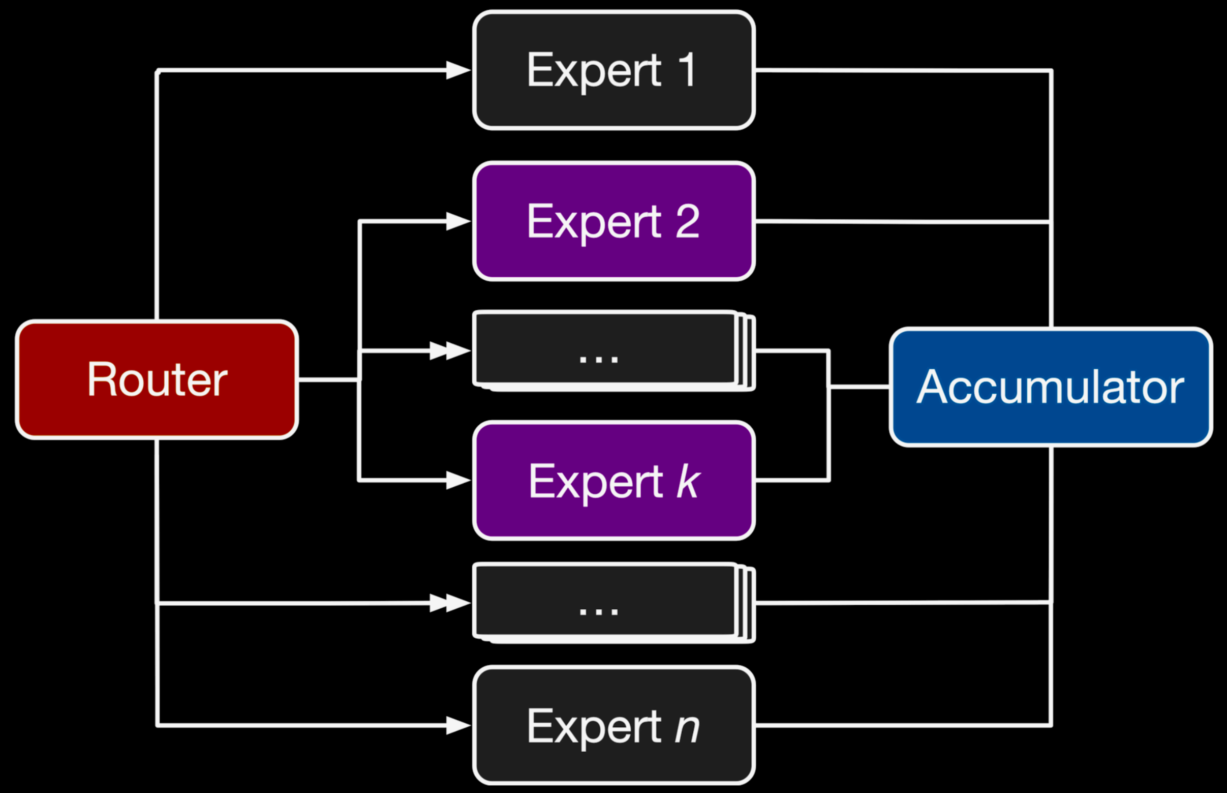
Performance



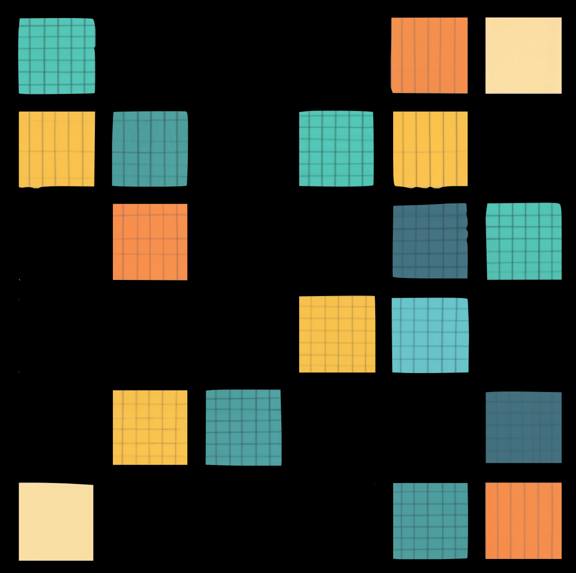
End-to-end applications



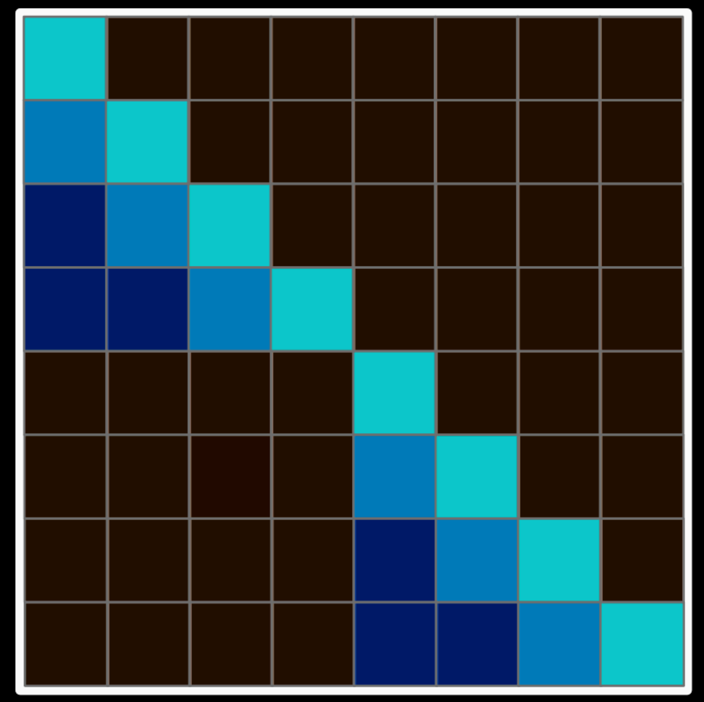
Graph neural networks



Mixture of experts



Recommender systems



Sparse transformers

Ongoing & Future work

- Sparse shaping operations
- Structured sparsity
  - Specialized tensor formats
  - Block formats and tensors
- Kernel fusion
- Broader hardware support