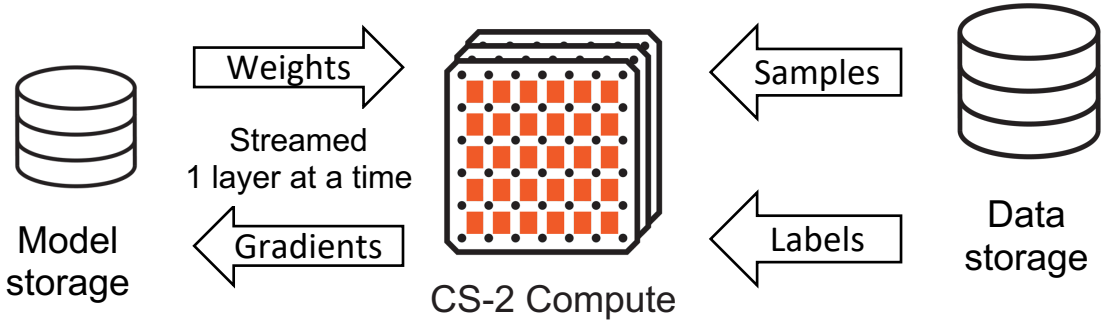
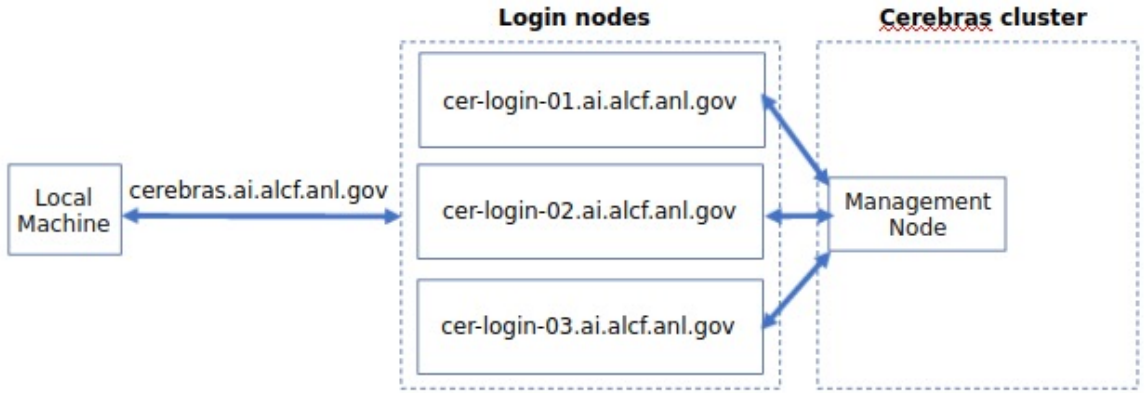
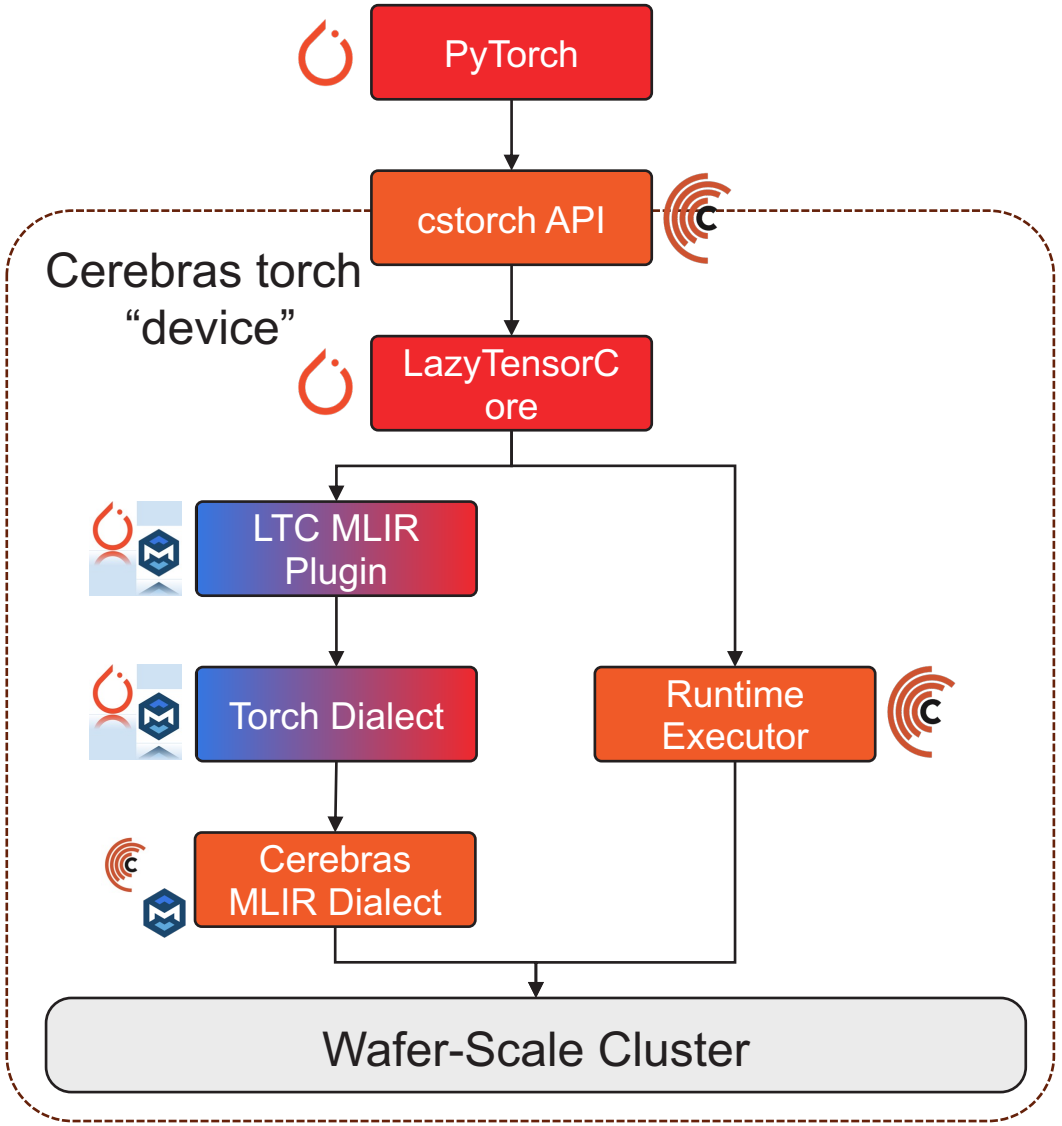
# Day 2: Agenda

# Recap of Day 1



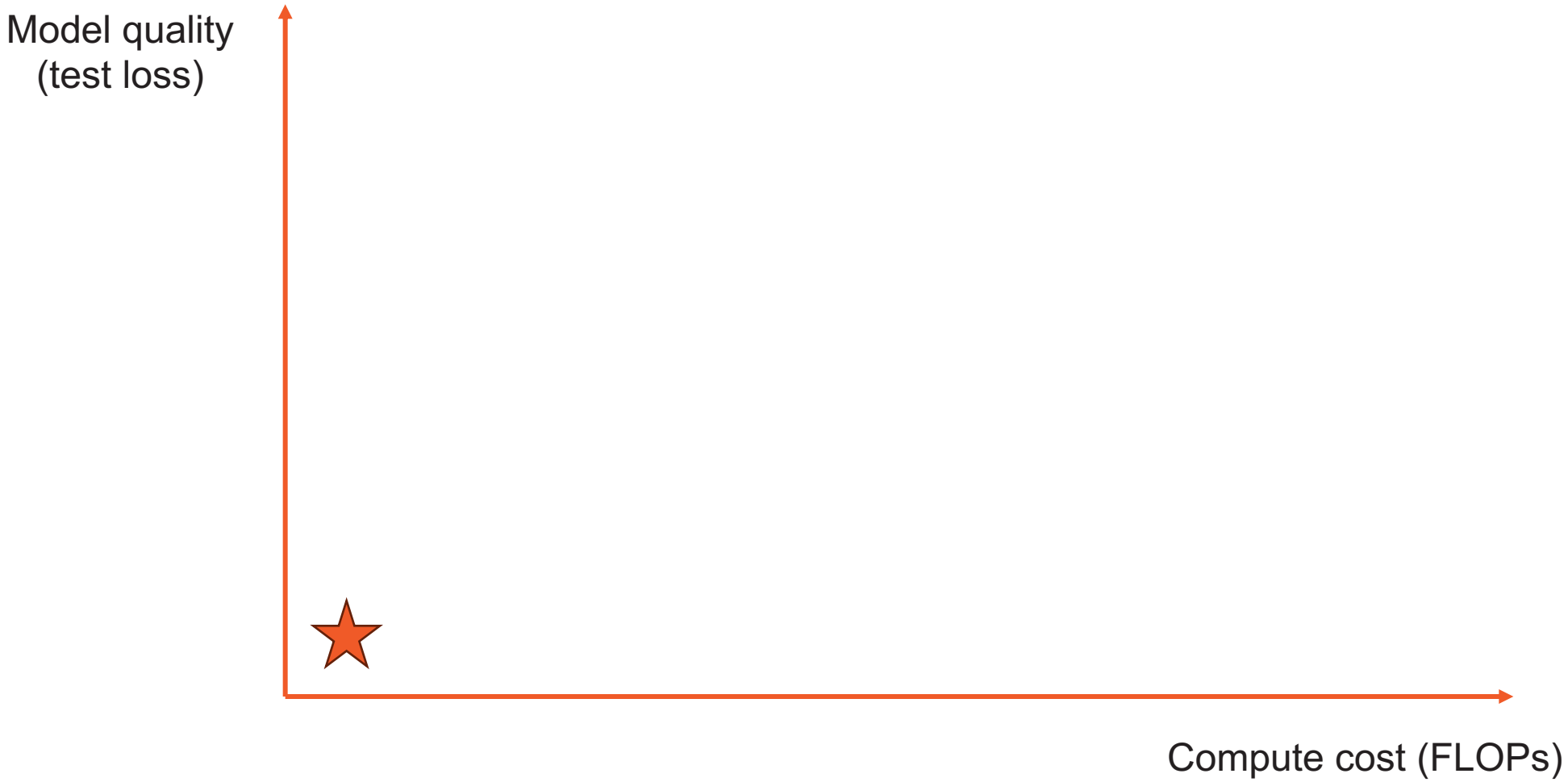



# Agenda

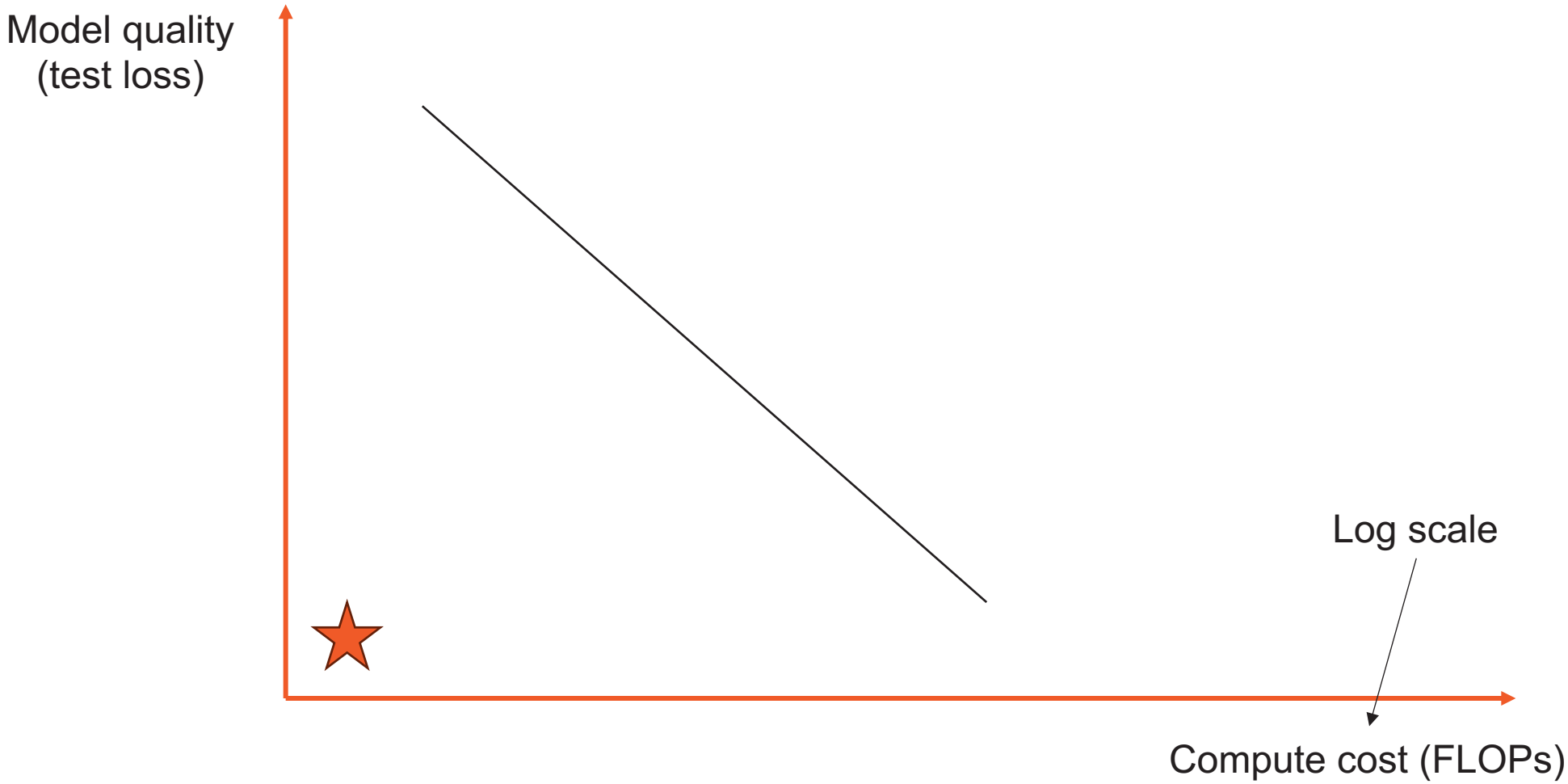
| Time | Topic |
|---|---|
| **Day 2: Wednesday 7 May 1:00pm-4:30pm CDT (11:00am-2:30pm PDT)** | |
| 1:00 - 1:45pm | Efficient training with Cerebras, scaling laws, how to train LLMs |
| 1:45 - 2:45pm | User training: hands-on LLM model Training |
| 2:45 - 2:50pm | Q&A |
| 2:50 - 3:05pm | Break |
| 3:05 - 4:05pm | HPC: CS for HPC: SDK, CSL and past examples |
| 4:05 - 4:25pm | Roadmap presentation |
| 4:25 – 4:30pm | Closing, final Q&A |

# Efficient training with Cerebras, scaling laws

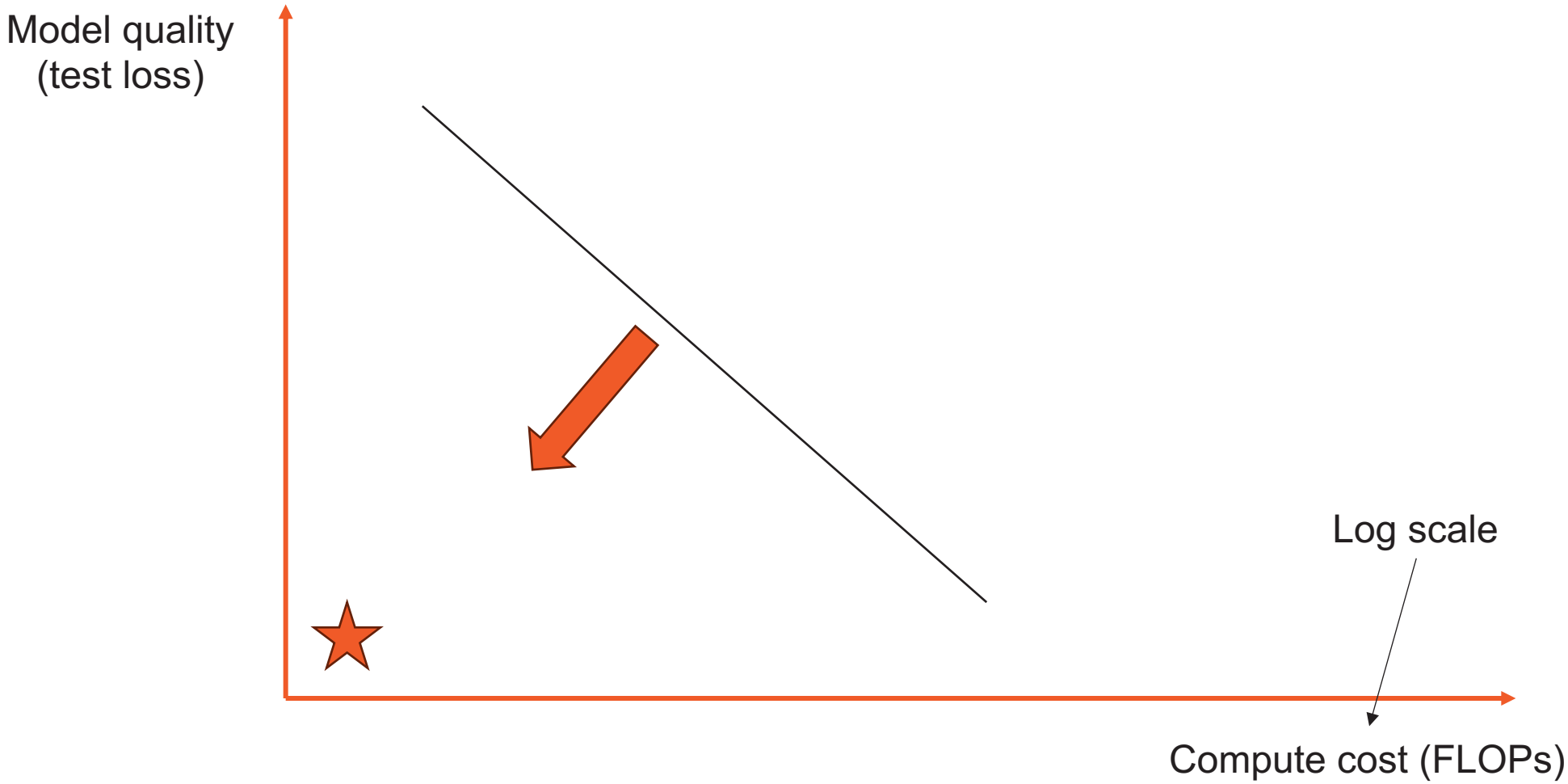
# The goal of computational efficiency



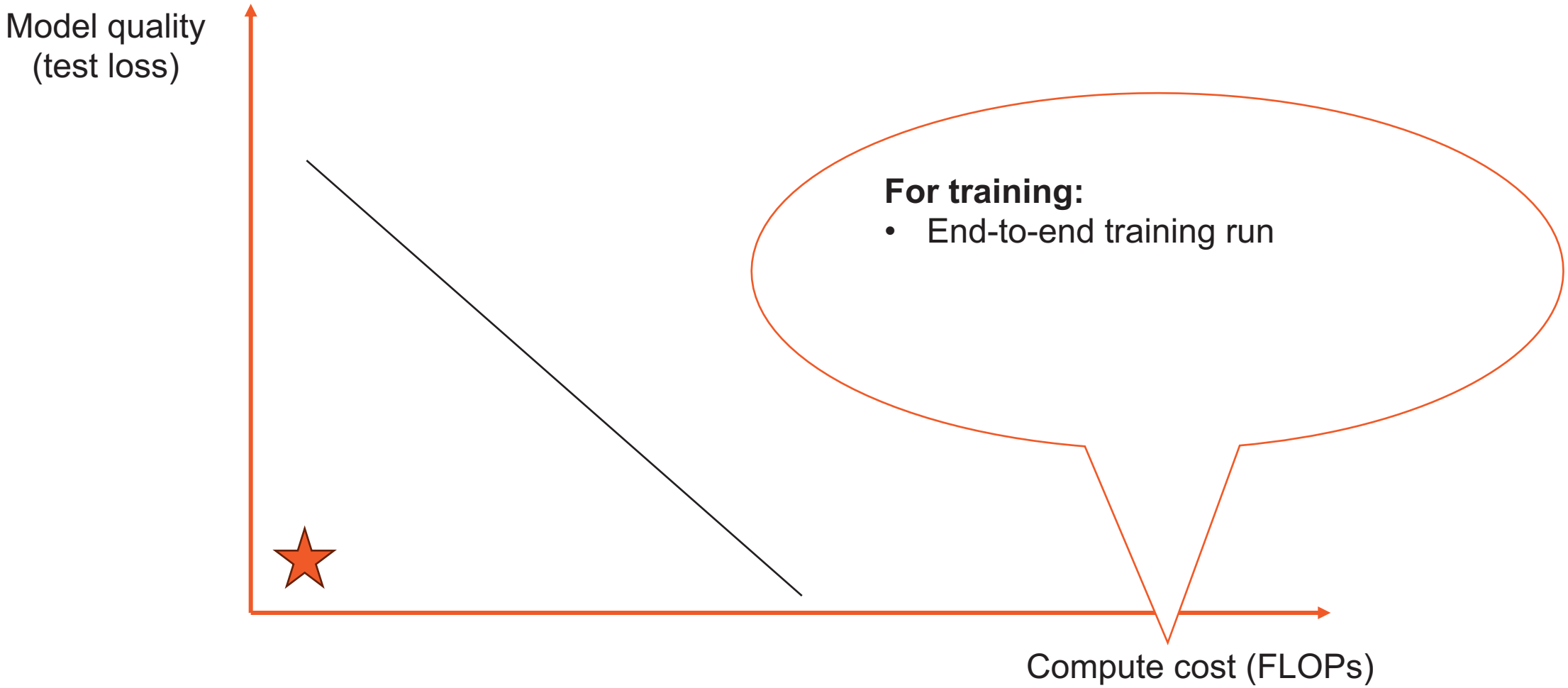
# The goal of computational efficiency



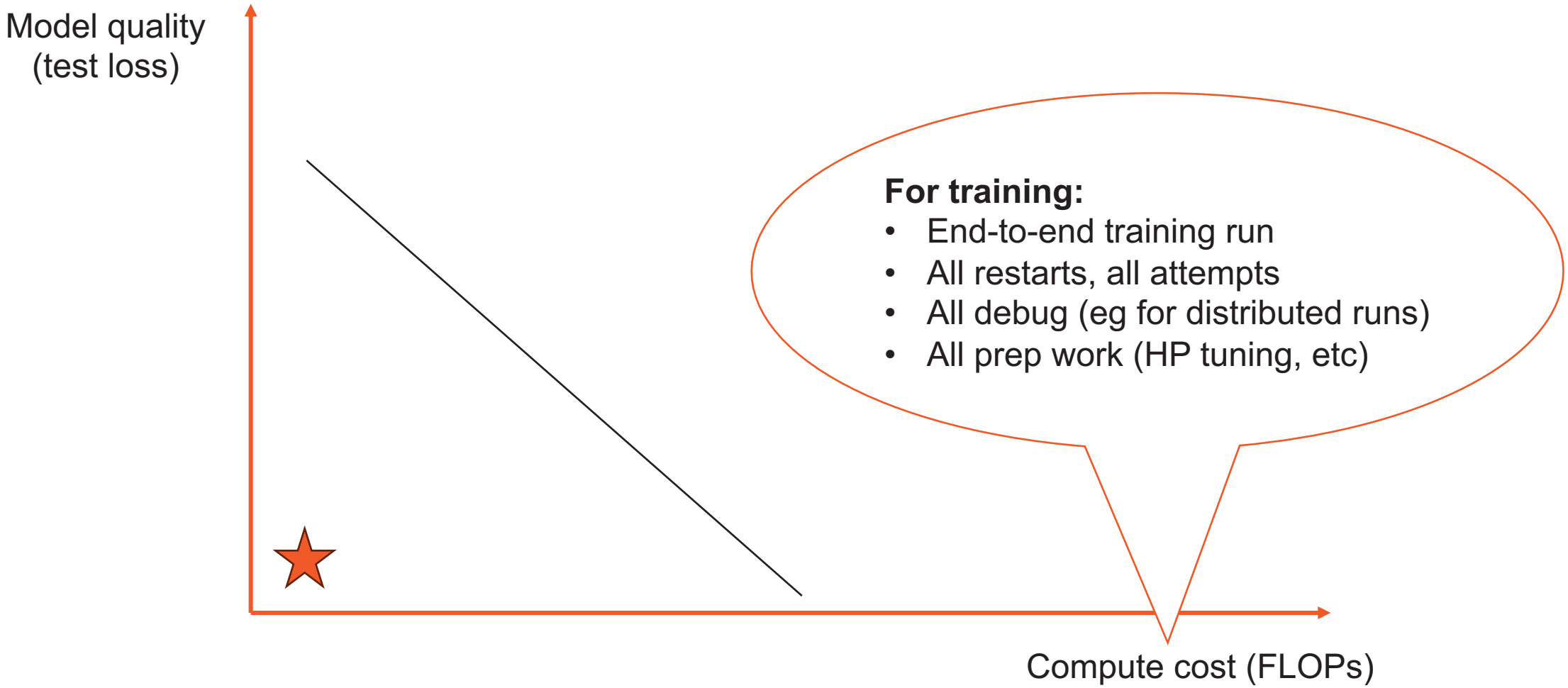
# The goal of computational efficiency



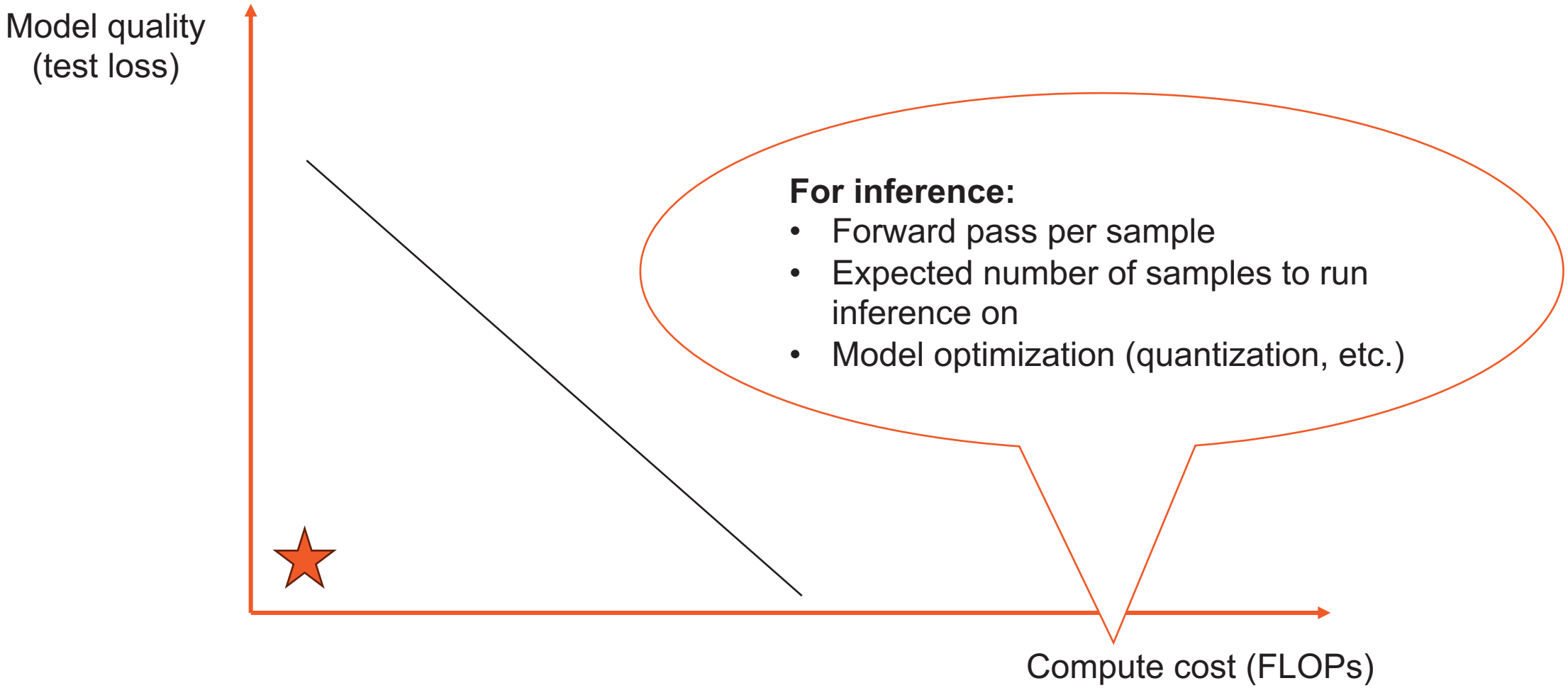
# The goal of computational efficiency



# The goal of computational efficiency



# The goal of computational efficiency



# A few approaches to higher efficiency

- Efficiency with **current models**
  - Careful data prep and cleaning
  - Meticulous selection of model features and training approaches via **small-scale experiments**
  - Precise planning and goal setting with **scaling laws**

- Efficient **new models**
  - A special sorcery: **sparsity** (requires specialized hardware)

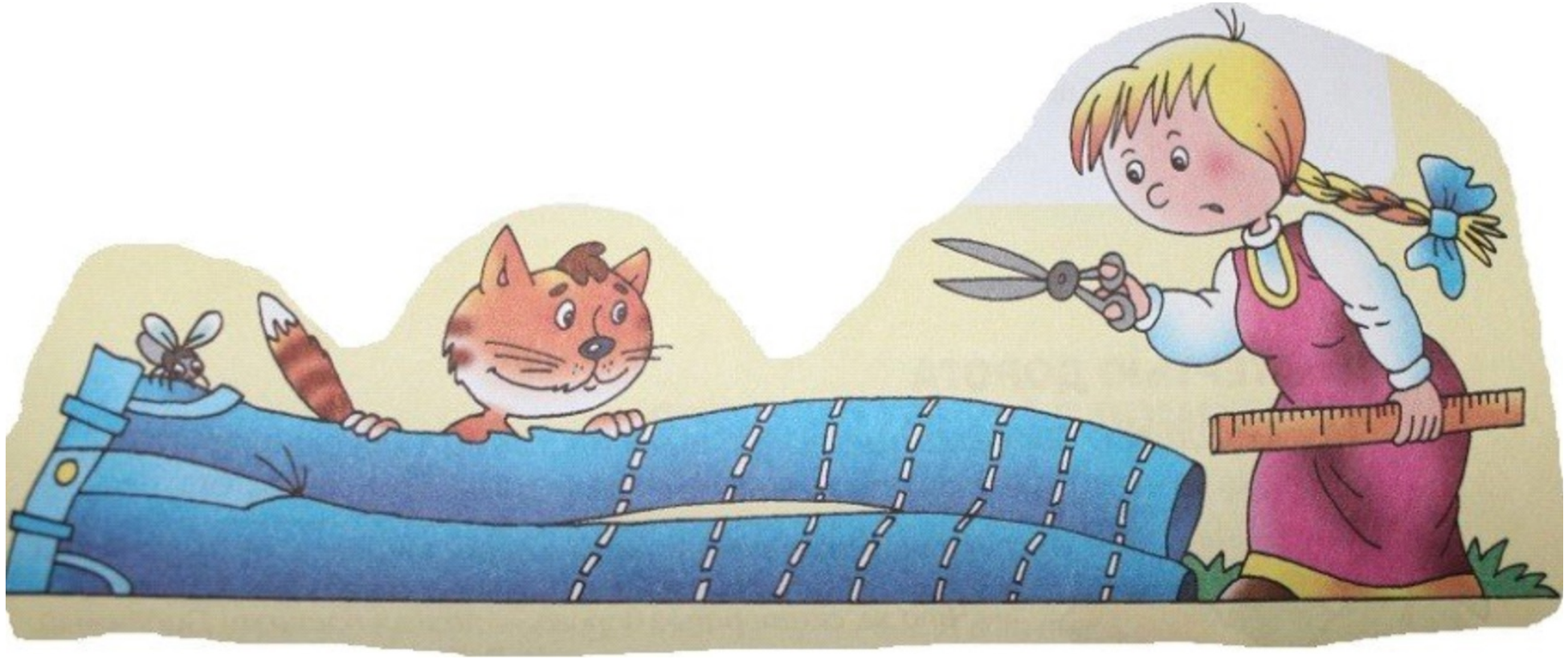
# A few approaches to higher efficiency

- Efficiency with **current models**
  - Careful data prep and cleaning
  - Meticulous selection of model features and training approaches via **small-scale experiments**
  - Precise planning and goal setting with **scaling laws**

- Efficient **new models**
  - A special sorcery: **sparsity** (requires specialized hardware)

# Семь раз отмерь, один отрежь

Measure seven times, cut once (= Better safe than sorry)



https://lubok.club/iluystracii/32713-sem-raz-otmer-odin-raz-otrezh-illjustracija-54-foto.html

# Семь раз отмерь, один отрежь

## Measure seven times, cut once (= Better safe than sorry)

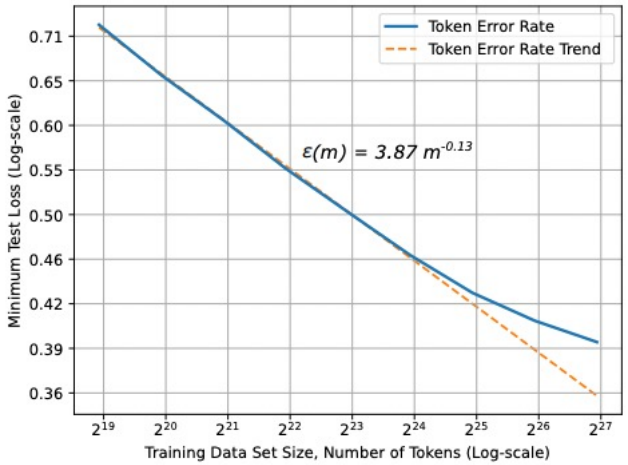
- Large GenAI model training runs are VERY long and expensive
  - Weeks to months on thousands of GPUs, millions of $$
- You want to be sure you are “on spot” and getting the best model you can
  - Errors are costly
- It’s better to spend a bit more time and resources on prep, but do the long training run right
  - Experiment as much as possible with small models, transfer learnings to a larger target model
- **Predict expected results**

# Predict expected results. How?

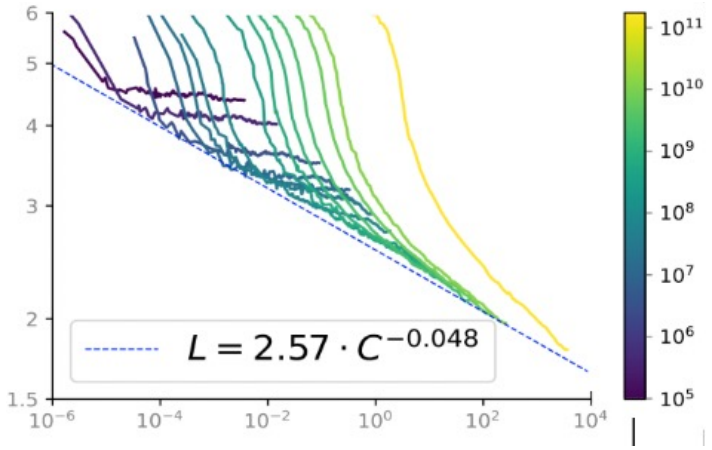
# Leverage scaling laws!

# What is a scaling law and why do we need one?

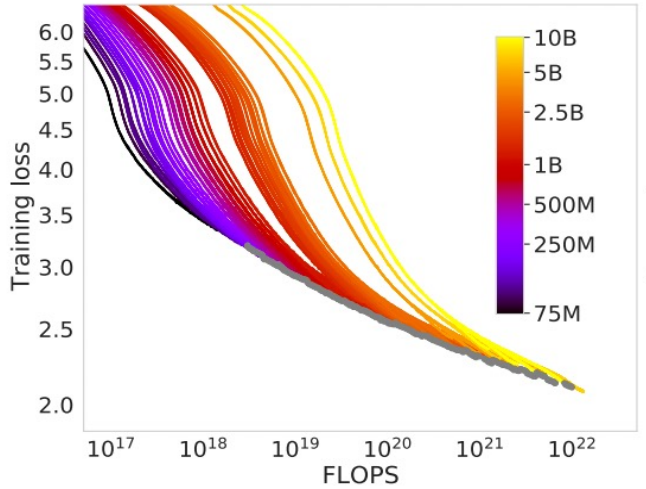
- Empirical scaling laws for the error (eg cross-entropy loss) as a function of training FLOPs
- Prior work:
  - “Deep Learning Scaling is Predictable, Empirically”, J. Hestness et al. (2017)
  - “Scaling Laws for Autoregressive Generative Modeling”, J. Kaplan et al. (2020)
  - “Training Compute-Optimal Large Language Models”, J. Hoffman et al. (2022)




J. Hestness, best-fit models for NMT




J. Kaplan, loss vs training PF-days




J. Hoffman, loss vs training FLOPs

Allows to predict model quality as a function of model size and dataset size

# Cerebras-GPT Compute-Optimal Scaling Law for Pile

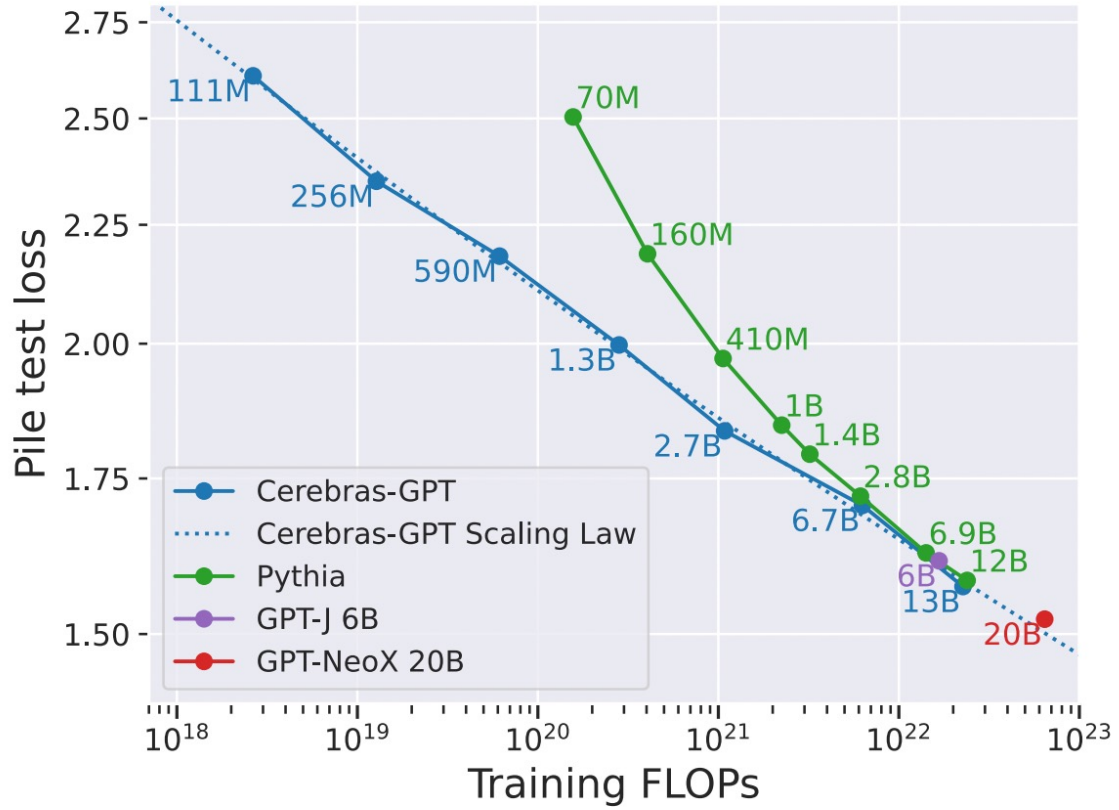



Figure 2: Pile test set loss given pre-training FLOPs for Cerebras-GPT, GPT-J, GPT-NeoX, and Pythia.

https://arxiv.org/abs/2304.03208

$$\mathcal{L}(f) = (f/5.984e22)^{-0.0737} + 0.5066$$

At 20 TPP (Tokens Per Parameter)

Can use scaling laws to

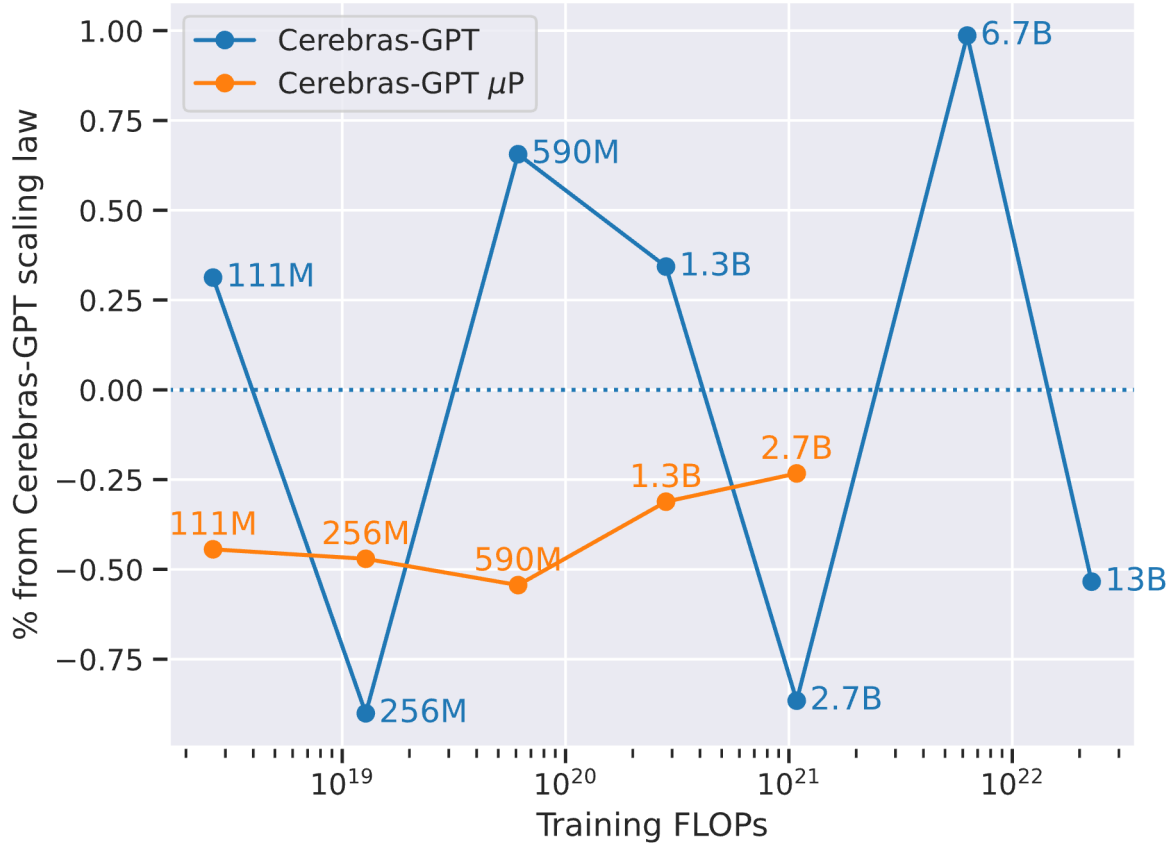
- Predict loss for given scale, set target
- Budget compute time
- Test whether training run is on-track

We can use scaling laws to predict model quality, assuming “good” hyperparameters.

How to find optimal hyperparameters for very large models?

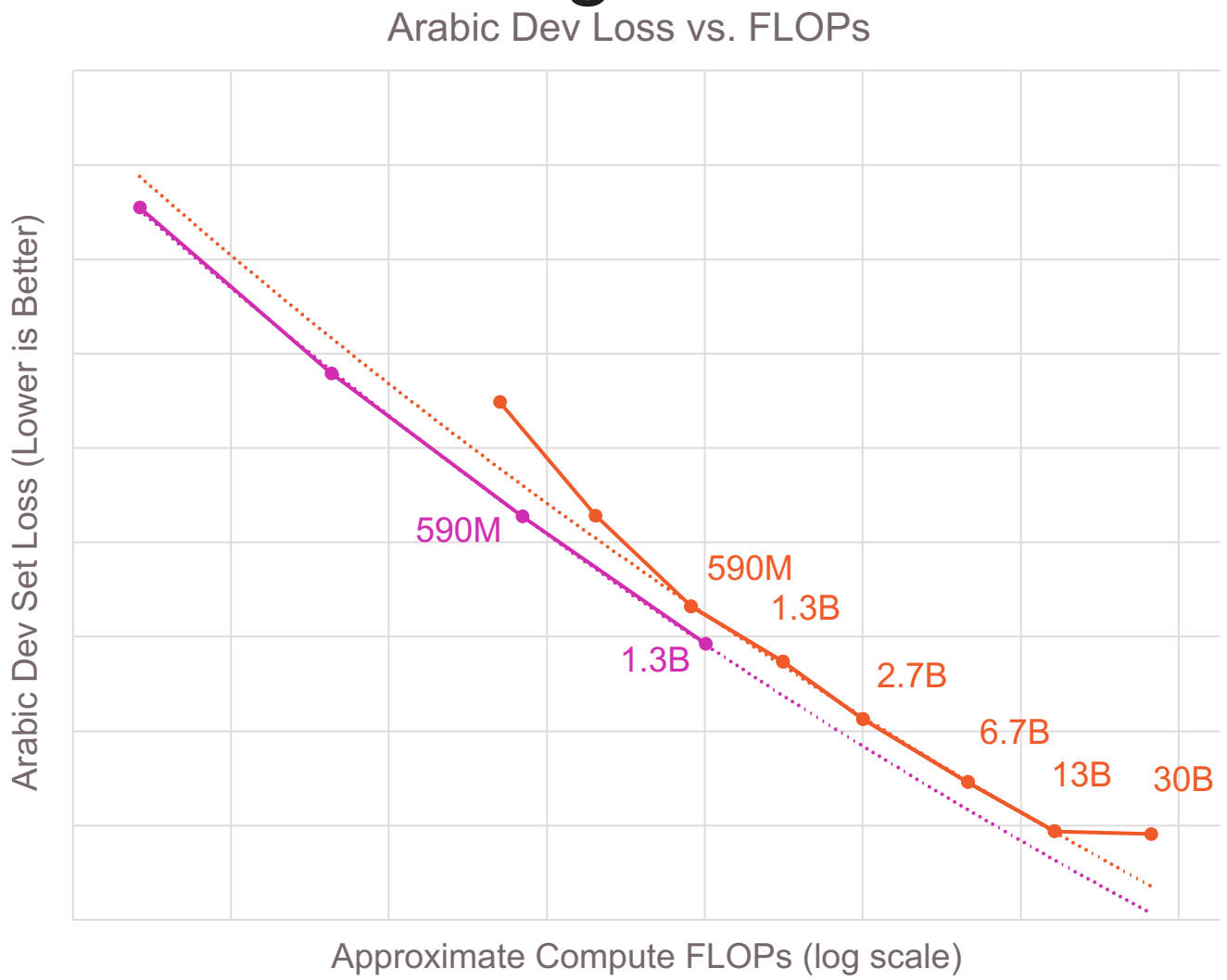
# Maximal Update Parameterization ($\mu$P) and $\mu$Transfer

- **Standard parameterization**
  - Weights are initialized from normal distributions
  - Does not account for dynamics at all scales
  - Different hyper-parameters for models of different sizes
- **Maximal Update Parameterization (µP)**
  - Control initialization, learning rate, activation magnitudes to be stable across model scale
  - µTransfer same hyper-parameters at all scales
- **Advantages of µP**
  - Tune LR hyper-parameters for smaller models, re-use for larger models
  - More stable training dynamics
  - More predictable scaling law
  - Better average downstream capabilities



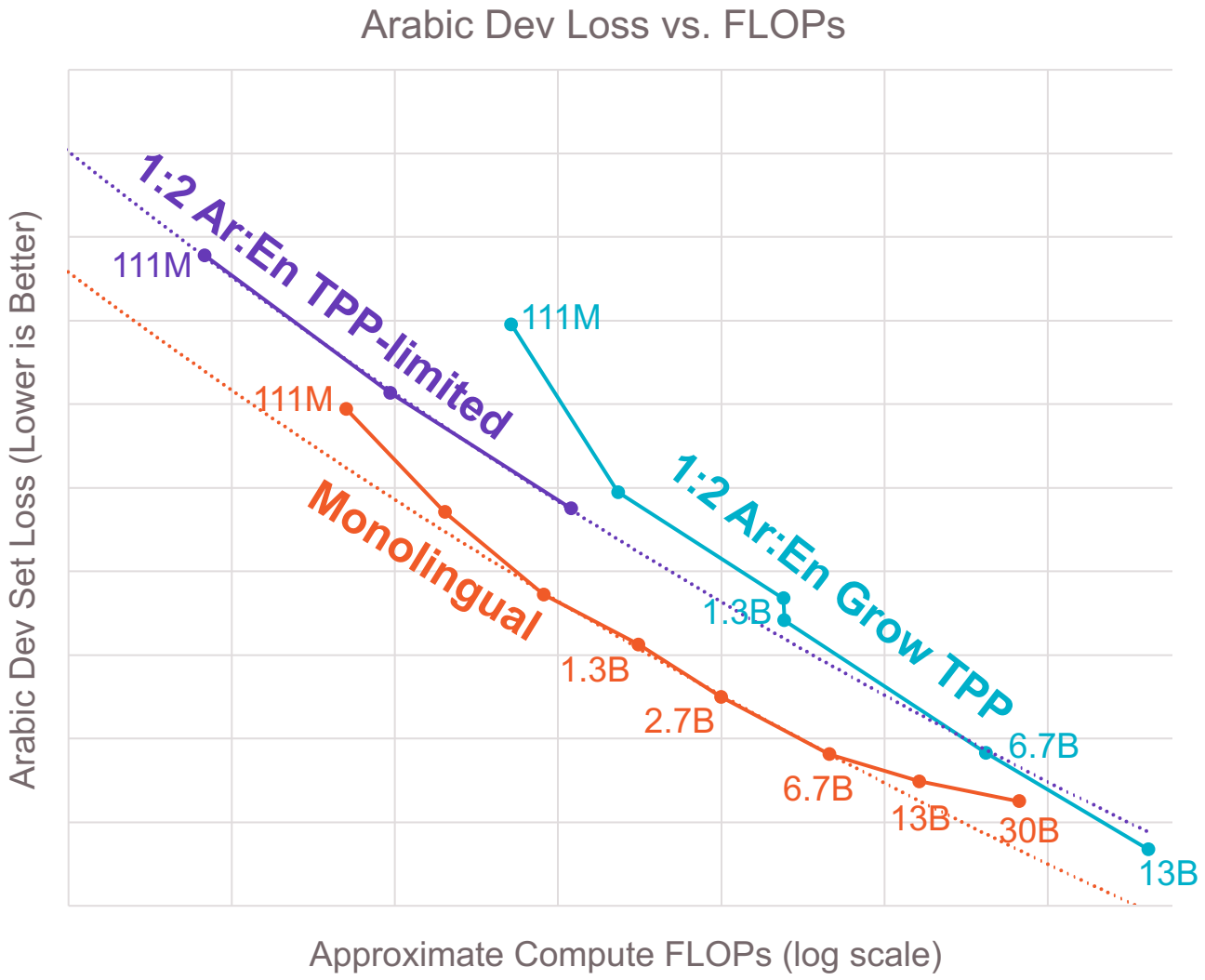
# Scaling laws in use: initial Arabic scaling laws

- Pink: restricted and fixed tokens/parameter, 20TPP
- Orange: full dataset for all runs, 55B tokens
- Similar power-law exponents
- Training on full dataset gives better loss for slightly suboptimal compute
  - 30B model only marginally better than 13B: suggests not enough data to continue scaling model size

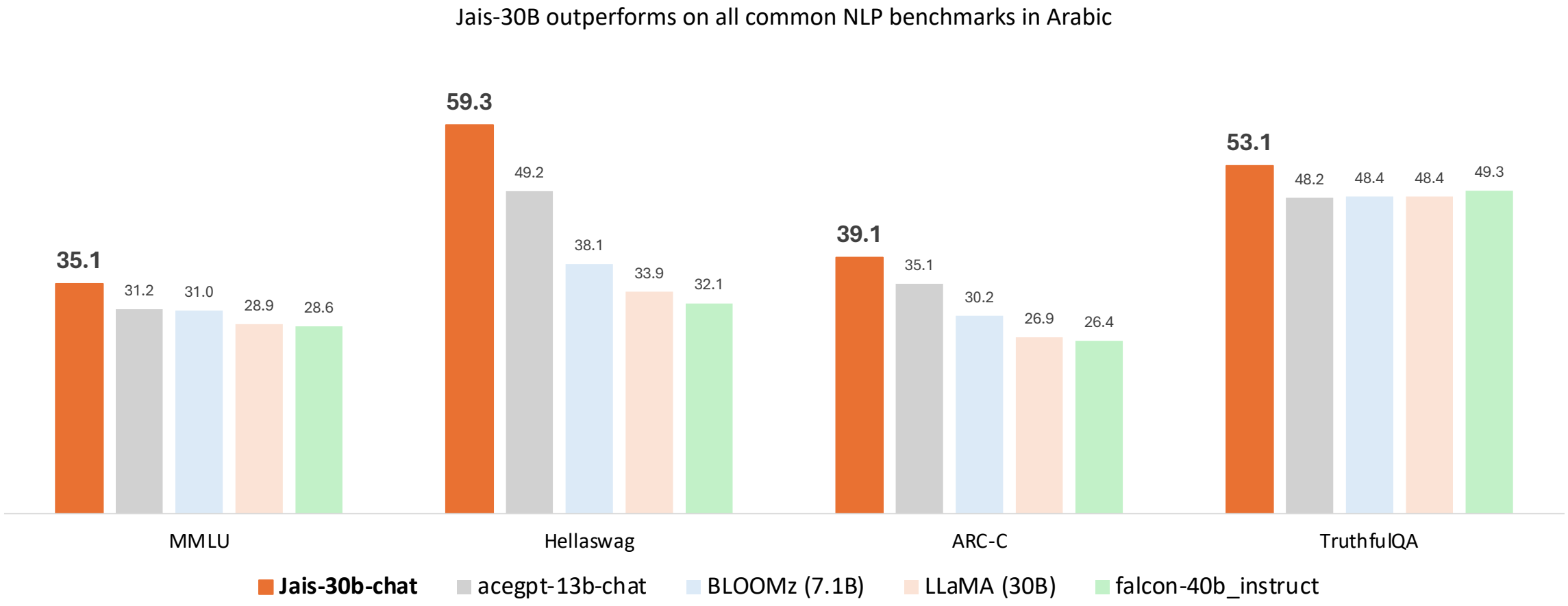


# Scaling laws in use: multilingual modeling for Jais

- Add more data: mix Arabic data with The Pile English corpus
  - Tested mix ratios 1:2, 1:1, 2:1 Arabic:English
- Trained 111M → 2.7B parameter models on different mixes
  - Multilingual: Costs extra compute
- Arabic modeling
  - Scaling laws allow us to inspect improvement over expected trend
  - To achieve similar loss to Arabic-only models, 1:2 Ar:En models need to increase compute ~3.7x
  - The “multilingual gap” is projected to shrink slowly with scale (dotted trend lines)
  - However, models improve faster when we grow dataset size (**Grow TPP**)



# Jais-30B-v3 sets **new record** for open-source Arabic LLMs, finishes training on 1.3 Trillion tokens

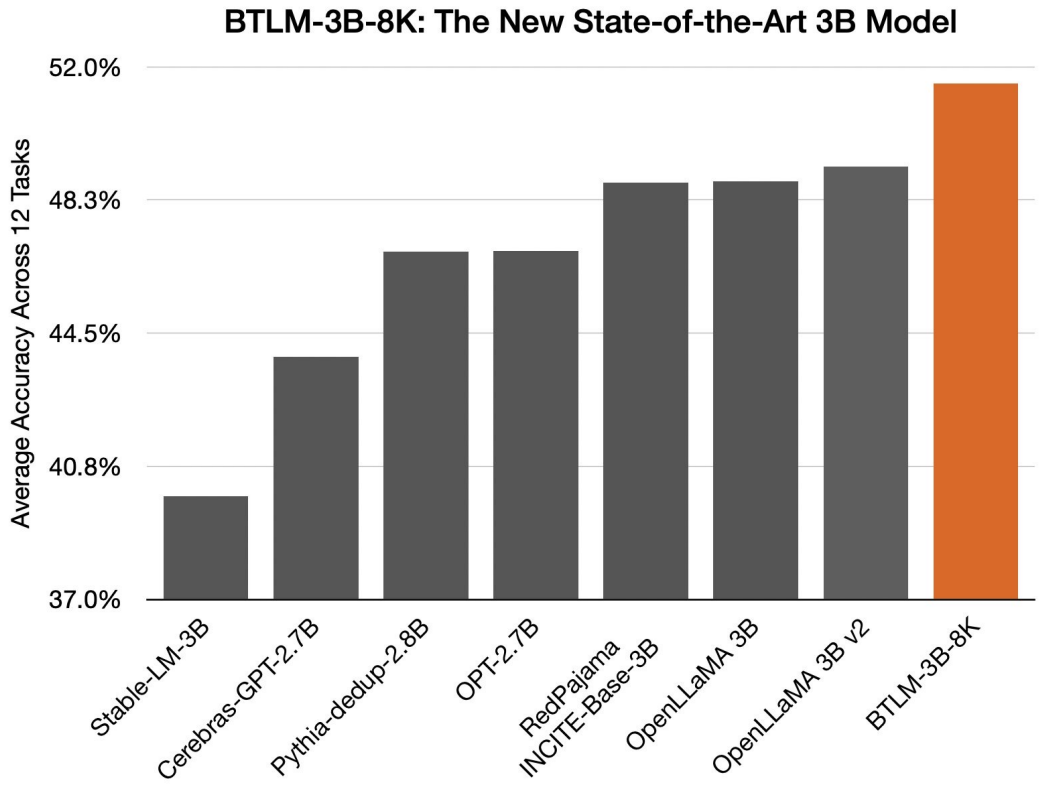



Note, results are displayed in order of the legend.

# BTLM-3B-8K: example of putting it all to work

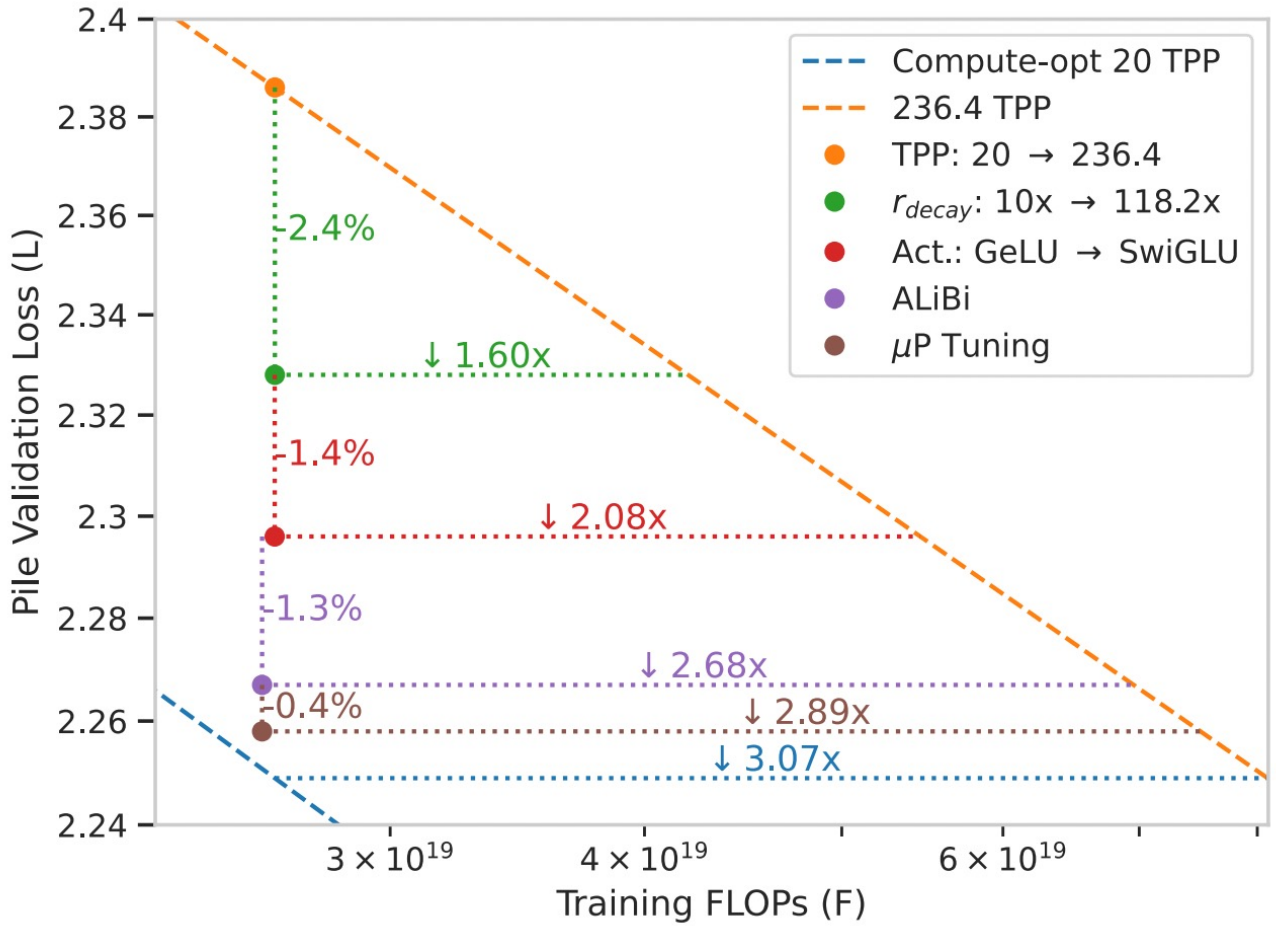
## Bittensor Language Model trained by Cerebras for OpenTensor

- The state-of-the-art 3B parameter open-source language model until release of Stable LM 3B
  - Beats many 7B parameter models
- Trained on SlimPajama, natively supports 8k sequence lengths
- Small parameter count makes it ideal for many edge use cases
- **The most popular 3B parameter model on HuggingFace with >1 million downloads**
- Recently released chat-optimized version adapted using IFT and DPO
- Apache 2.0 license for commercial use
- Created in partnership with OpenTensor



# Our secret behind high quality of BTLM?

# Many (cheap) ablations at 111M scale

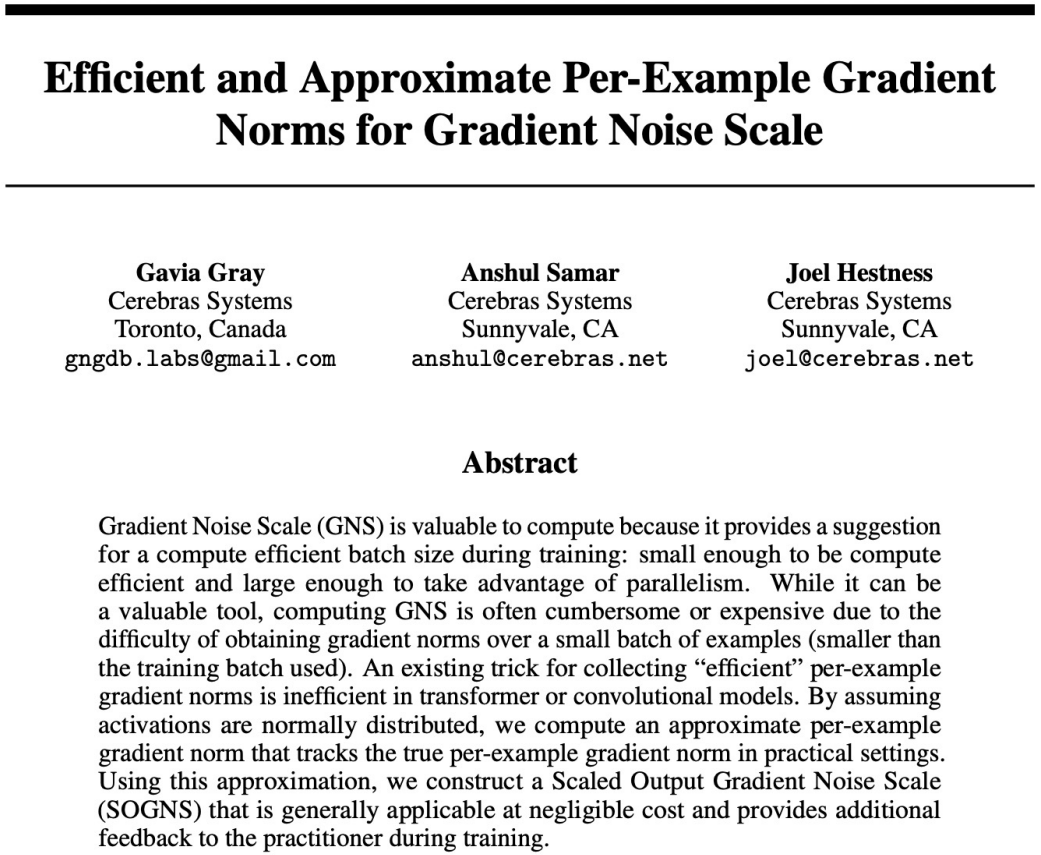



Loss improvements and changes in training FLOPs for each ablation starting from the Cerebras-GPT μP, 111M baseline.

| Variant | Loss | FLOPs |
|---|---|---|
| Baseline: Cerebras-GPT μP, 111M | 2.586 | 2.23e18 |
| TPP: 20 → 236 | 2.386 | 2.63e19 |
| $r_{\text{decay}}$ : 10× → 118× | 2.328 | 2.63e19 |
| Act.: GeLU → SwiGLU | 2.296 | 2.63e19 |
| ↳ RoPE | 2.259 | 2.60e19 |
| ↳ ALiBi | 2.267 | 2.60e19 |
| ↳ μP Tuning | 2.258 | 2.60e19 |

# One more advice: choose your batch size carefully

- Choose **compute efficient batch size**
- Batch size too small:
  - Gradient update is very noisy, poor approximation of the gradient over the entire dataset
  - Harder to parallelize
- Batch size too large:
  - Approximation is too close to true gradient over the entire dataset, updates from different batches are too similar to be useful
  - Easy to parallelize, but wasteful
- How to choose the right batch size?
  - Use Gradient Noise Scale (GNS)



**Efficient and Approximate Per-Example Gradient Norms for Gradient Noise Scale**


**Gavia Gray**
Cerebras Systems
Toronto, Canada
gngdb.labs@gmail.com

**Anshul Samar**
Cerebras Systems
Sunnyvale, CA
anshul@cerebras.net

**Joel Hestness**
Cerebras Systems
Sunnyvale, CA
joel@cerebras.net



**Abstract**

Gradient Noise Scale (GNS) is valuable to compute because it provides a suggestion for a compute efficient batch size during training: small enough to be compute efficient and large enough to take advantage of parallelism. While it can be a valuable tool, computing GNS is often cumbersome or expensive due to the difficulty of obtaining gradient norms over a small batch of examples (smaller than the training batch used). An existing trick for collecting "efficient" per-example gradient norms is inefficient in transformer or convolutional models. By assuming activations are normally distributed, we compute an approximate per-example gradient norm that tracks the true per-example gradient norm in practical settings. Using this approximation, we construct a Scaled Output Gradient Noise Scale (SOGNS) that is generally applicable at negligible cost and provides additional feedback to the practitioner during training.

# Practical implications for compute-efficient training

- Leveraging scaling laws allows predicting model quality as a function of available data and model size
- Constants in the power law are dataset-dependent
- When start working on a new model, **start with smaller models**, fit the power law
- If I spent $X on training, I might expect model quality Y, and my inference cost will be $Z
- Maximal Update Parameterization (μP) allows to find optimal hyperparameters via hyperparameter sweeps on small models and transfer these optimal hyper-parameters to more expensive large model runs. It also makes training more stable.

# A few approaches to higher efficiency

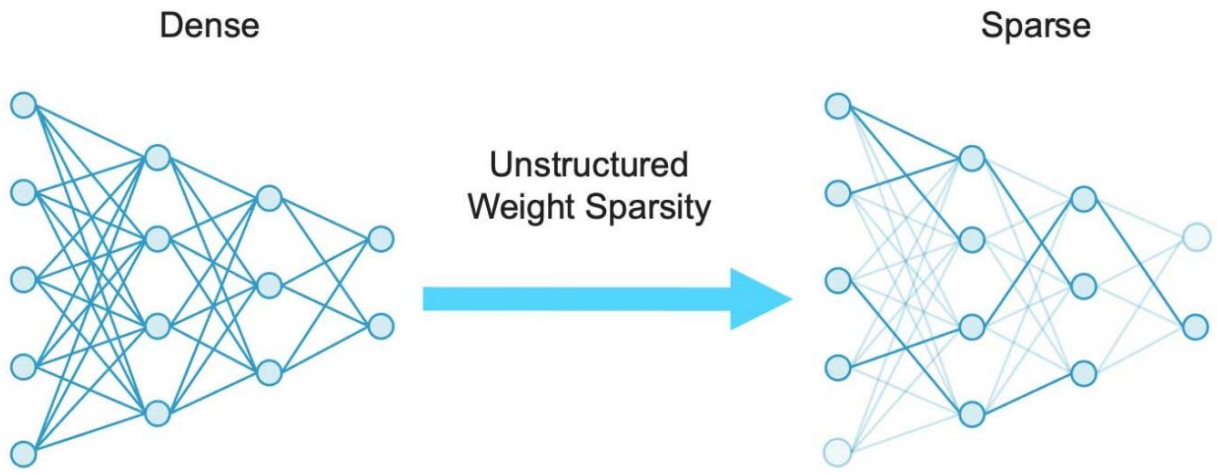
- Efficiency with **current models**
  - Careful data prep and cleaning
  - Meticulous selection of model features and training approaches via **small-scale experiments**
  - Precise planning and goal setting with **scaling laws**

- Efficient **new models**
  - A special sorcery: **sparsity** (requires specialized hardware)

# Neural Networks are Sparse

## Sparsity opportunities are everywhere

- Neural networks have native sparsity
  - e.g. ReLU or Dropout
- Neural networks can be made sparse
  - e.g. sparse weights
  - Models are over parameterized by design
  - Training is act of discovering important weights



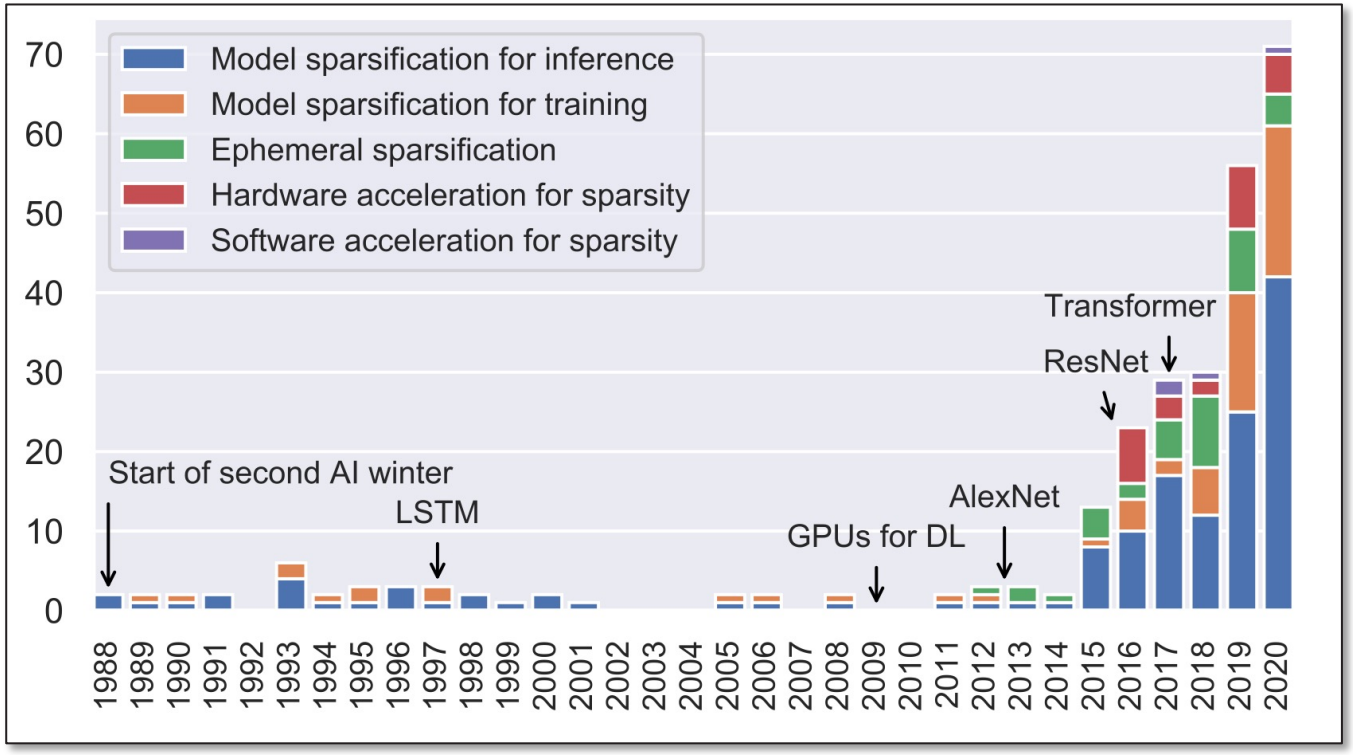

## Training dense is wasteful and inefficient

- But not all hardware can take advantage of all forms of sparsity

# Neural Networks Can be Made Sparse

**Extensive sparsity research community**

- Techniques show 10x+ opportunity
- Practical benefits include reducing compute/memory and improving accuracy
- Research has increased dramatically




Torsten Hoefler et al., Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks

**ML Community has invented various sparsity techniques**

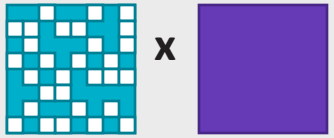
# Sparsity Acceleration is Memory Bound

## Memory bandwidth built for sparsity

- **Traditional hardware built for dense**
  - **High data reuse → caching → low mem bw**
- **Wafer-scale memory built for sparse**
  - **Low data reuse → ~~caching~~ → high mem bw**
  - **Enabled by orders of magnitude more mem bw**

## CS-3 accelerates all forms of sparsity

- **Static and dynamic sparsity**
- **Structured and unstructured sparsity**

| | Memory Bandwidth (Byte/FLOP) | |
|---|---|---|
| | Required | Available |
| Dense MatMul  | ~0.001 | H100<br>0.003 |
| **Sparse MatMul**  | **~1** | WSE-3<br>**2** |

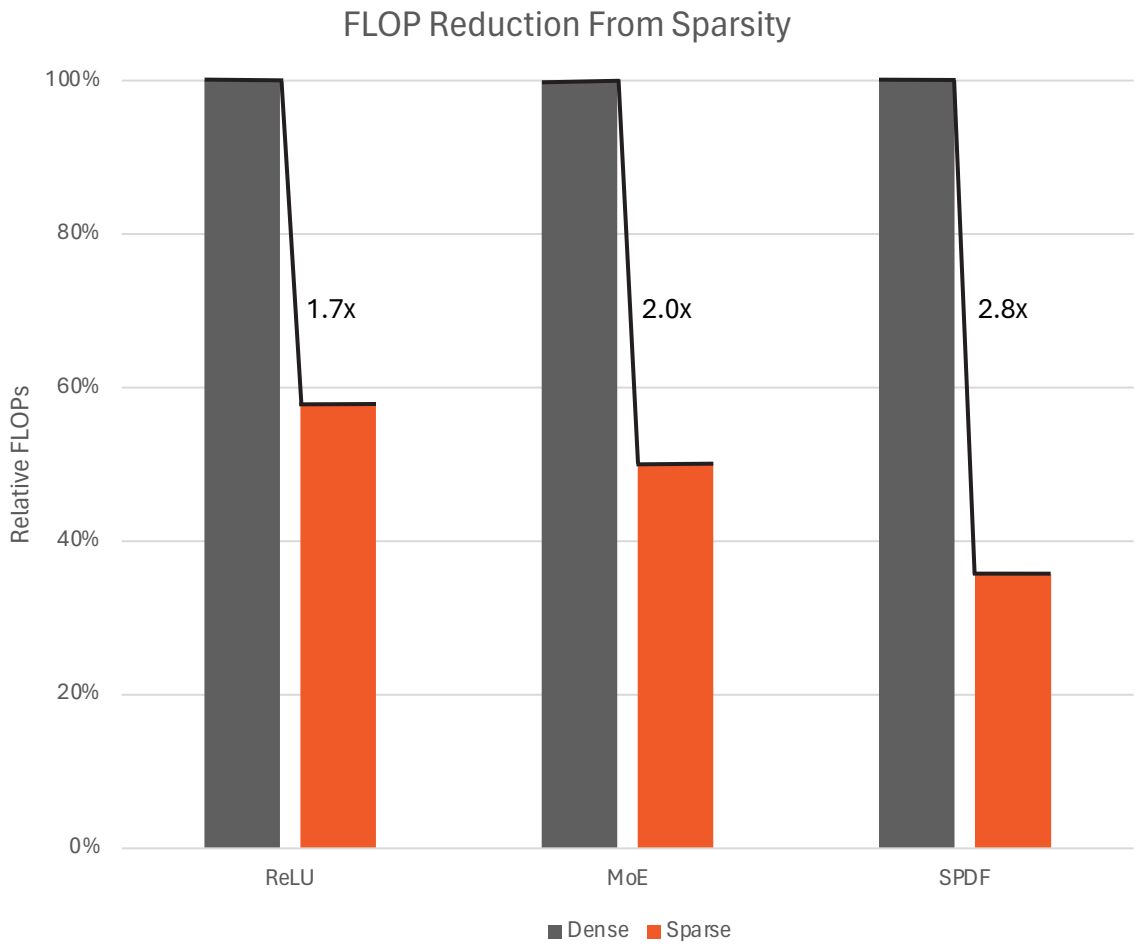
# Accelerating All Forms of Sparse Training

## Examples of sparse training opportunities

- Dynamic activation sparsity
  - e.g. Google: 95% sparse ReLU FFN in LLMs[1]
- Structured weight sparsity
  - e.g. Mistral: 75% sparse FFN MoE 8x7B[2]
- Unstructured weight sparsity
  - e.g. Cerebras: 75% sparse SPDF GPT[3]

## Solving unsustainable scaling for training

- Only HW to accelerate all forms of sparsity
- Even future sparse techniques




1 Li et al., The Lazy Neuron Phenomenon: On Emergence of Activation Sparsity in Transformers, 2023
2 Jiang et al., Mixtral of Experts, 2024
3 Thangarasa et al., SPDF: Sparse Pre-training and Dense Fine-tuning for Large Language Models, 2023

# To summarize…

**We care about computational efficiency of training and inference: improve model quality, decrease cost**

- Efficiency with **current models**
  - Run **many experiments at a small scale** – they are cheap! (Measure seven times, cut once)
  - Rely on **scaling laws** to reason about future large runs, before starting an expensive final run
  - Rely on **Maximal Update Parametrization** to transfer hyper-parameters tuned on small models
- A **special sorcery** for even higher computational efficiency: **sparsity**
  - Requires **specialized hardware** to translate theoretical speed-ups into practical, we at Cerebras are lucky to have it!

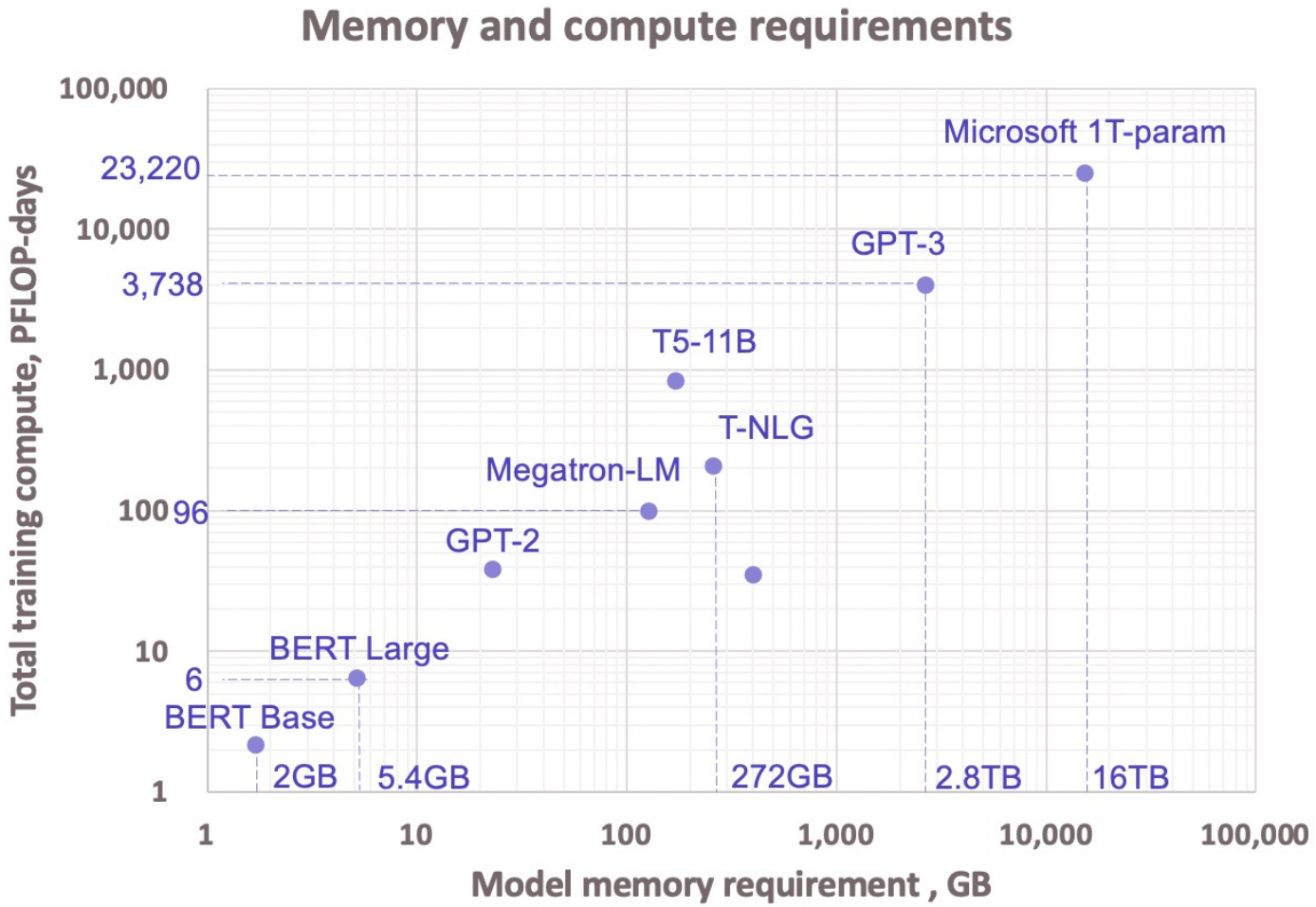
# Potential GenAI use cases for science

- Foundation LLMs for science: extract key insights and summarize content from scientific literature
  - Ingest all existing knowledge from publications, books, etc
  - Add other modalities, e.g. plots from papers
  - RAG with customized domain-specific embedding models
  - Q&A and search
- Genomic foundation models: personalized medicine, better understanding of diseases
  - Predict functional consequences of genetic variations
  - Predict functional elements such as promoters, enhancers, transcription factor binding sites
  - Better diagnostics, predict drug responses
- Molecular foundation models: protein engineering, material science
  - Predict drug-target interactions
  - Predict material properties
- Multimodal models for science
  - E.g. radiology scans and reports; satellite imagery and other climate-related imagery and climate reports

# Sparsity-Accelerated Training

## The Missing Piece

# Modern models need more and more compute




**Estimated time-to-train:**

- NVIDIA Megatron-LM:
  trained on **512 V100** (32 DGX-2H)
  for **about 10 days**
- OpenAI GPT-3:
  trained on **1024 V100** (64 DGX-2H)
  for **about 116 days**

**Model growth not sustainable**

# Accelerating beyond today's models with sparsity

To scale beyond today's state of the art, we need **more than only larger models**

Large neural networks are highly over-parameterized (e.g., pruning is common for inference)

**Sparsity opens up another dimension** of advancement beyond improving model architecture

**Cerebras is the only system capable of accelerating AI with any sparsity**

- **Faster training from sparse models**
  - Thangarasa et al., SPDF: Sparse Pre-training and Dense Fine-tuning for Large Language Models
  - GPT-3 1.3B pre-trained with up to 75% sparsity and **2.5x less training FLOPs** with same downstream accuracy and inference FLOPS as dense
- **Higher accuracy from larger sparse models**
  - Saxena et al., SIFT: Sparse Iso-FLOP Transformations for Maximizing Training Efficiency
  - ResNet 90% sparse **is 3.5% higher accuracy with 2x fewer FLOPs** than larger model
  - GPT 50% sparse is **0.4 better perplexity with 2.4x fewer FLOPs** than larger model
- **Faster inference from sparse models**
  - Iterative magnitude pruning on GPT
  - GPT-3 1.3B pruned to 84% sparsity and **3x less inference FLOPs** with same accuracy as dense

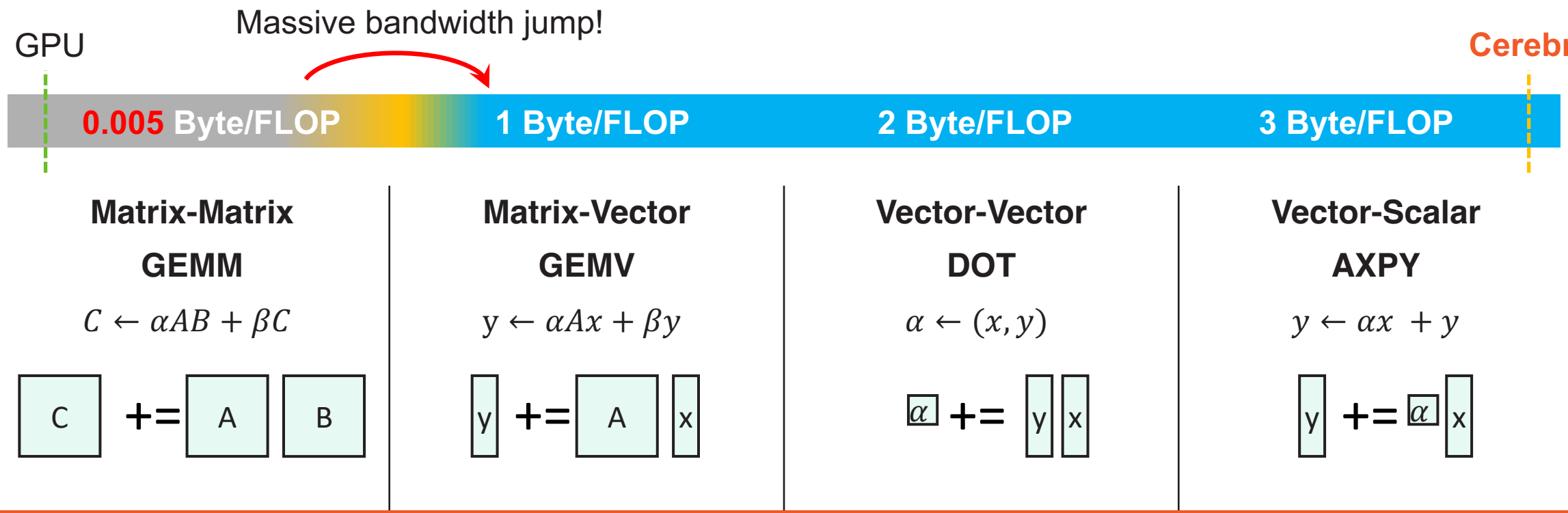
# Sparsity Demo

- Sparse weights remain sparse for the entire duration of training (“static sparsity”). To change sparsity levels, training will need to be re-started.

- Sparsity config parameters:
  - sparsity: the desired sparsity level between 0 and 1.
  - init_method: the type of sparsification (random or topk).
    - Random, weights are sparsified randomly
    - Topk, the weights with the lowest magnitude are sparsified
  - param_name_patterns: optional parameter to specify which layers to sparsify. Any regex provided here will be matched to layer names and if it appears in the layer name, that layer will be sparsified

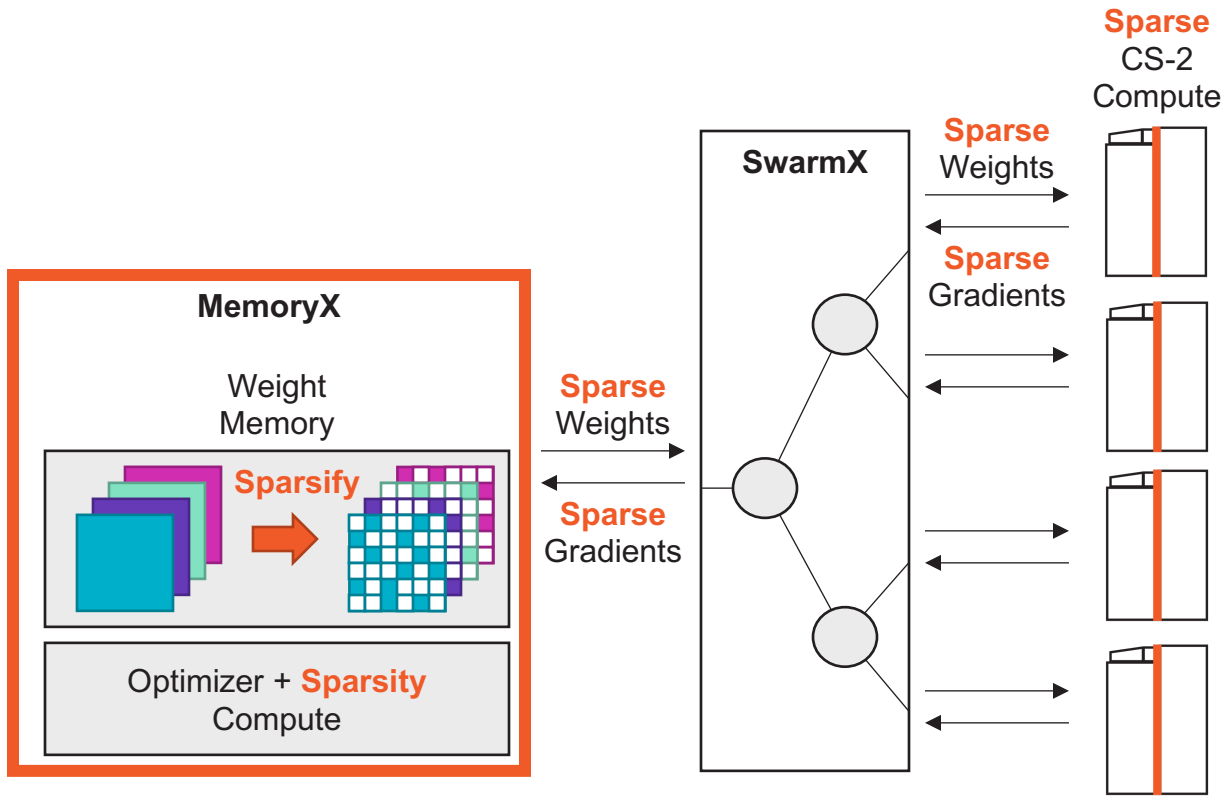
# Memory Designed for Unstructured Sparsity

## Full Performance on All BLAS Levels




**Sparse GEMM is one AXPY per non-zero weight**

# Streaming Sparse Weights




Weight sparsity induced in MemoryX

- **Sparse weights** streamed to all CS-2s
- **Sparse gradients** reduced on the way back
- **Sparse weight updates** on sparse matrix

No change to the weight streaming model

**Same flow supports dense and sparse**

# Demo: LLM Variations

**LLM Families, Scale-Out, Sparsity Acceleration, and Long Sequence Lengths**

# Q&A Session

# Cerebras SDK for HPC Research and Applications

**Leighton Wilson**

leighton.wilson@cerebras.net

**May 2024**

# Agenda

- Architecture and Programming Model
- Cerebras SDK Overview
- HPC Research and Applications
- Local Access and Next Steps

# Architecture and Programming Model

# Cerebras Wafer-Scale Engine (WSE-2)

The (2nd) Largest Chip in the World

**850,000** cores optimized for sparse linear algebra

**46,225 mm²** silicon

**2.6 trillion** transistors

**40 Gigabytes** of on-chip memory

**20 PByte/s** memory bandwidth

**220 Pbit/s** fabric bandwidth

**7nm** process technology

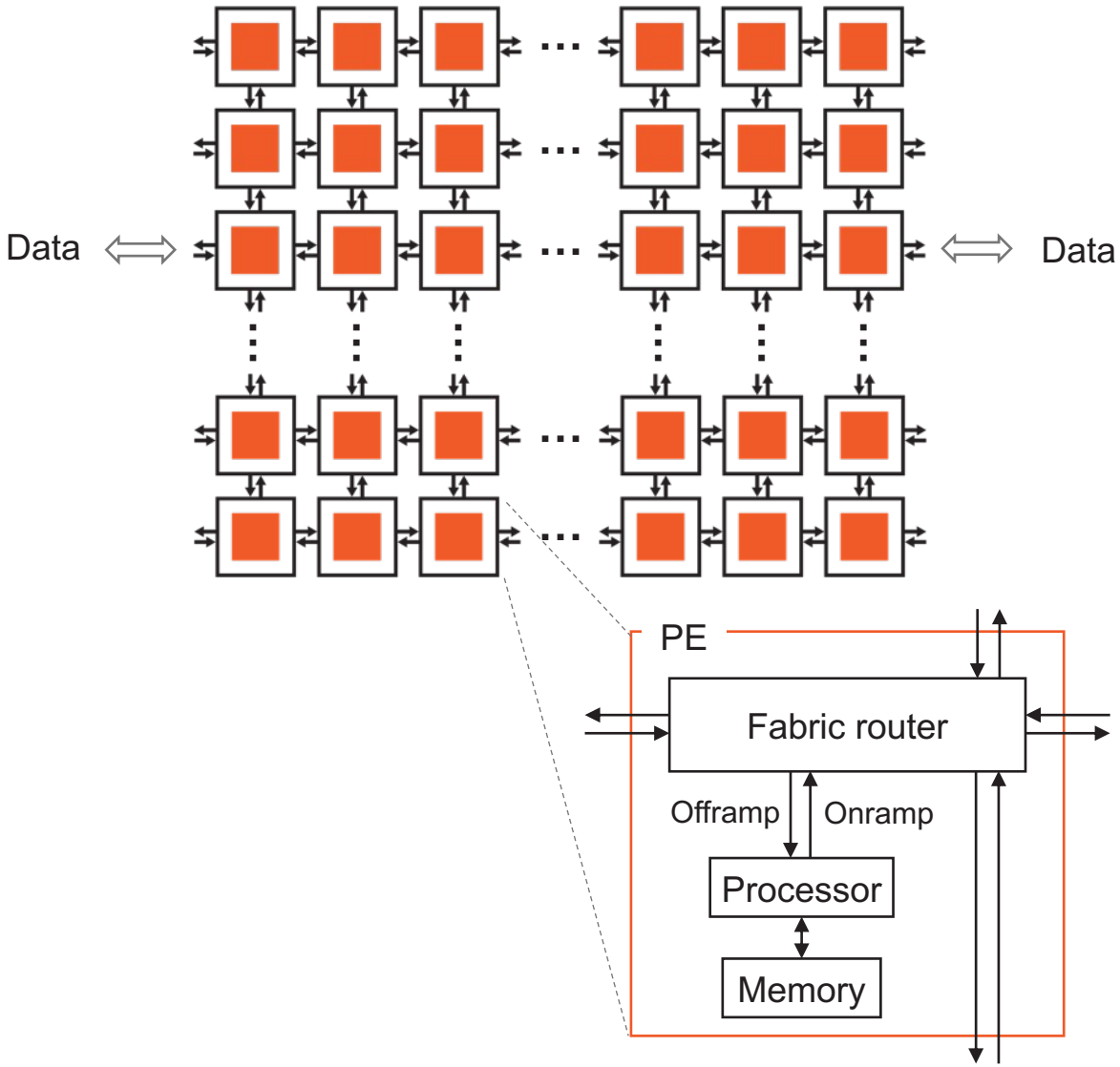
**Cluster-scale acceleration on a single chip**

# Cerebras CS-2 System

**The world's (2nd) most powerful AI and HPC accelerator**

- Powered by WSE
- Install, deploy easily into a standard rack
- Programmable via our SDK or PyTorch

# CS-2 Architecture Basics




The CS-2 appears as a logical 2D array of individually programmable Processing Elements

## Flexible compute

- 850,000 general purpose CPUs
- 16- and 32-bit native FP and integer data types
- **Dataflow programming**: Tasks are activated or triggered by the arrival of data packets

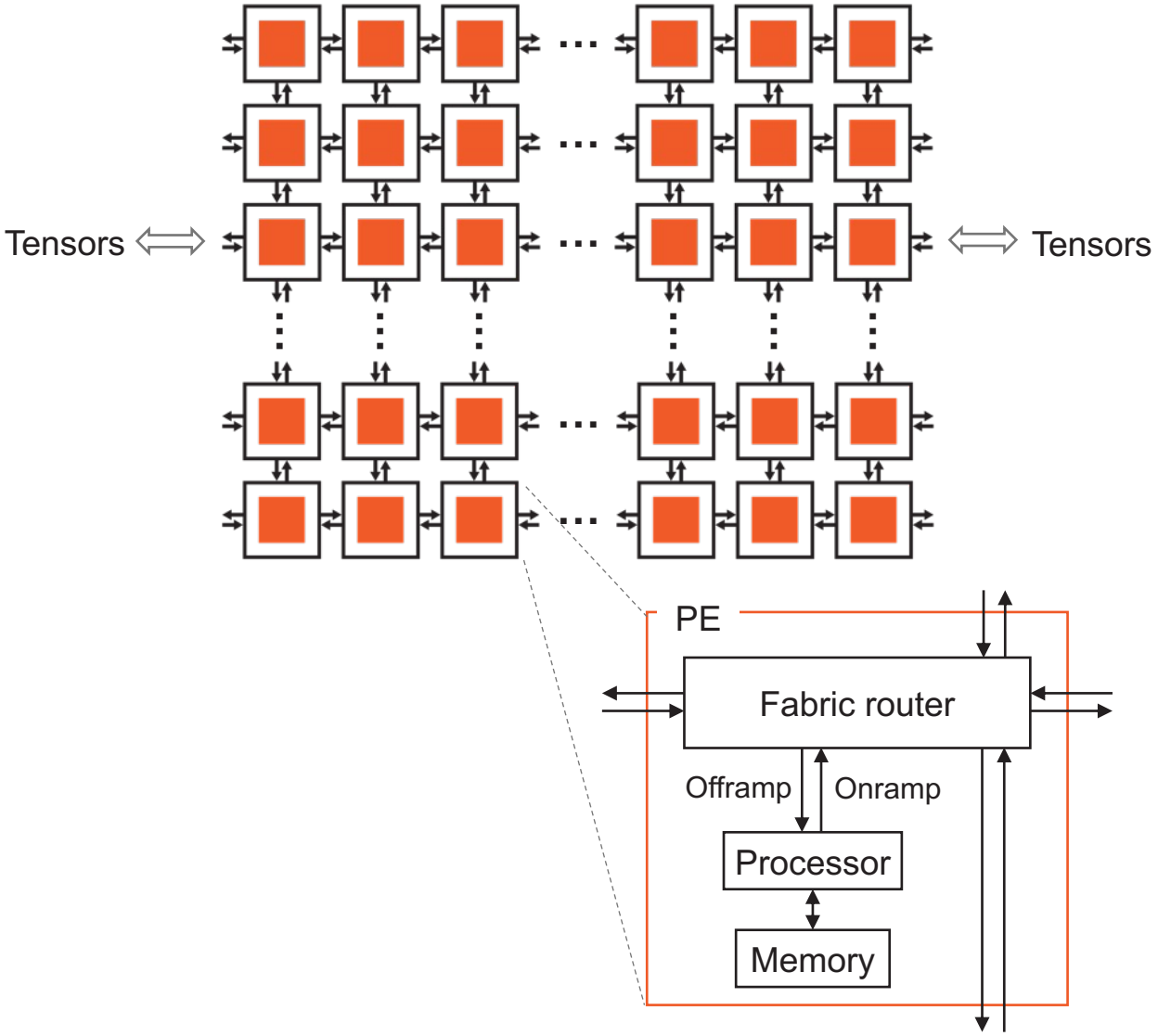
## Flexible communication

- Programmable router
- Static or dynamic routes (**colors**)
- Data packets (**wavelets**) passed between PEs
- 1 cycle for PE-to-PE communication

## Fast memory

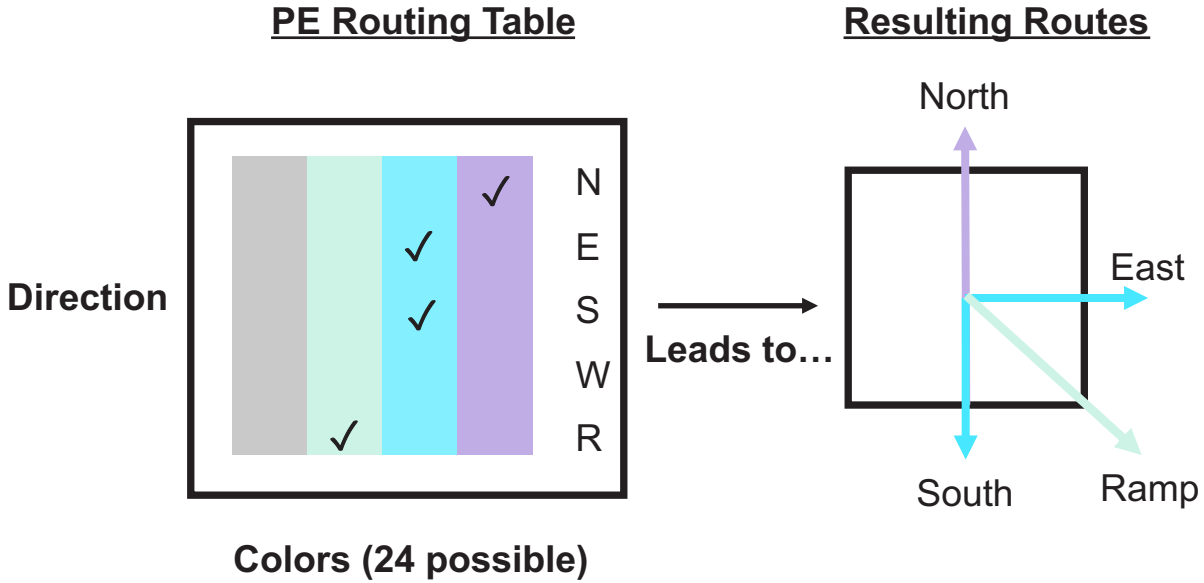
- 40GB on-chip SRAM
- Data and instructions
- 1 cycle read/write

# Flexible Compute




- **Dataflow Execution Model**
  - Tasks may be triggered by **wavelets** or activated
  - Each color activates a distinct task
- Independent programs specified for regions of PEs
  - Programs specify computation for the processor and communication via **colors**
  - Parametrized programs allow execution of different control flow on different PEs
- Asynchronous operations performed by launching *microthreads*
- Control flow is straightforward to reason about
  - Tasks are non-preemptive
  - Instruction to activate another task enable state-machine behavior

# Flexible Communication

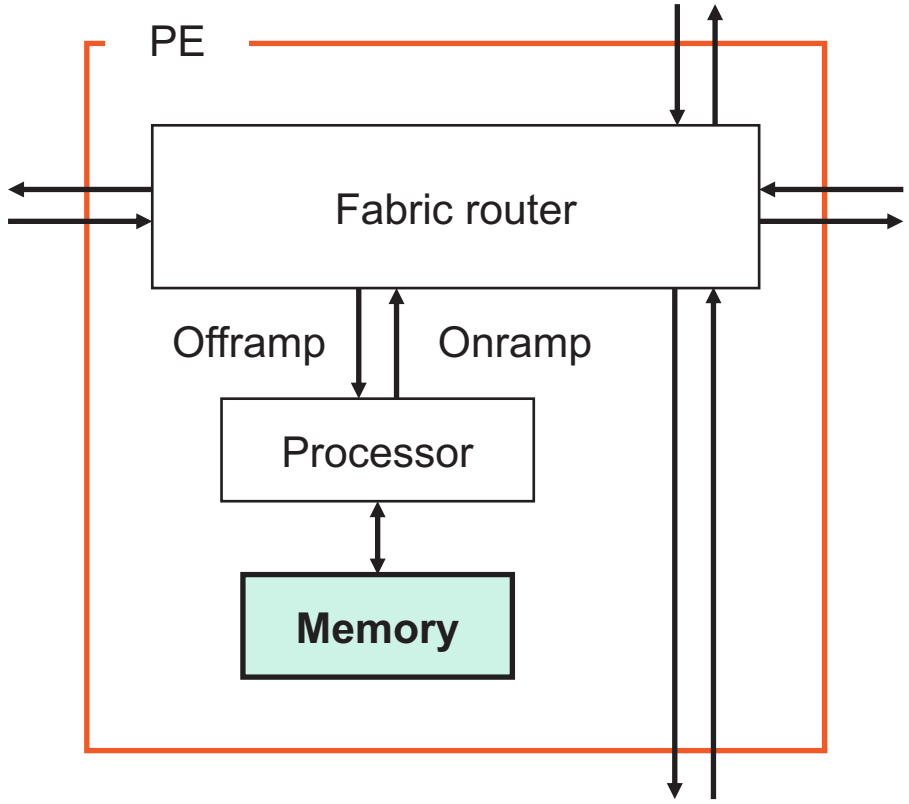



Router-to-router communication: **1 cycle**

Router-to-processor communication: **7 cycles**

- PEs communicate to adjacent PEs and their processor through their **routers**
- The **router** is a 24-entry table on each PE associating colors with directions
  - Table entries mapped to PE memory
  - Up to 24 routes (i.e. **colors**) may be specified at compile-time for each PE
- Complex communication patterns
  - Dynamic updating of routes at runtime
  - Multiple routing table entries per color enable *multicast*: broadcasting data in multiple directions at once each cycle
- Input/ output queues in each PE alleviate back pressure at routers during runtime
- Programmer feeds tensors into the fabric from outside world, specified in host program

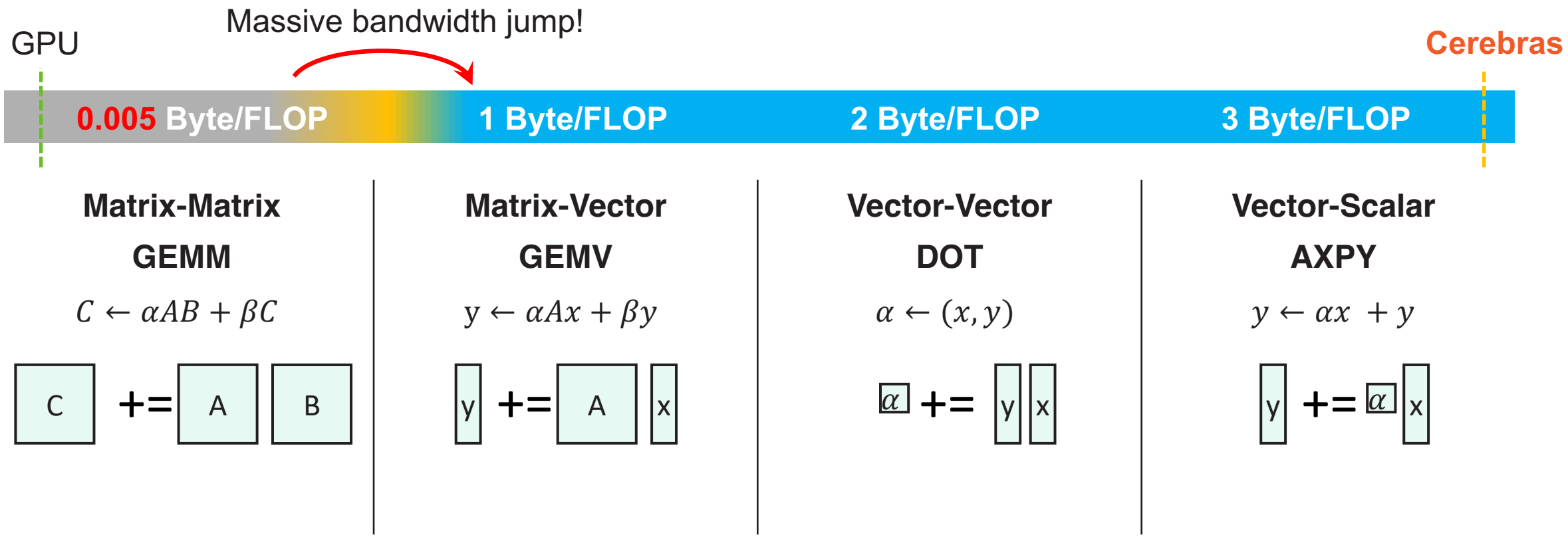
# Fast Memory




PE local memory read-write: **1 cycle**

- 40GB of on-chip SRAM
  - Uniformly distributed on wafer
  - 48kB per PE
- Programmer can read/write memory for regions of PEs at once from host
- Local PE memory is not directly addressable by other PEs, but is directly addressable by host program
- SIMD possible for vector instructions

# Memory performance at all BLAS levels



# Cerebras SDK

# Cerebras SDK

A general-purpose parallel-computing platform and API allowing software developers to write custom programs (“kernels”) for Cerebras systems.

**Language**

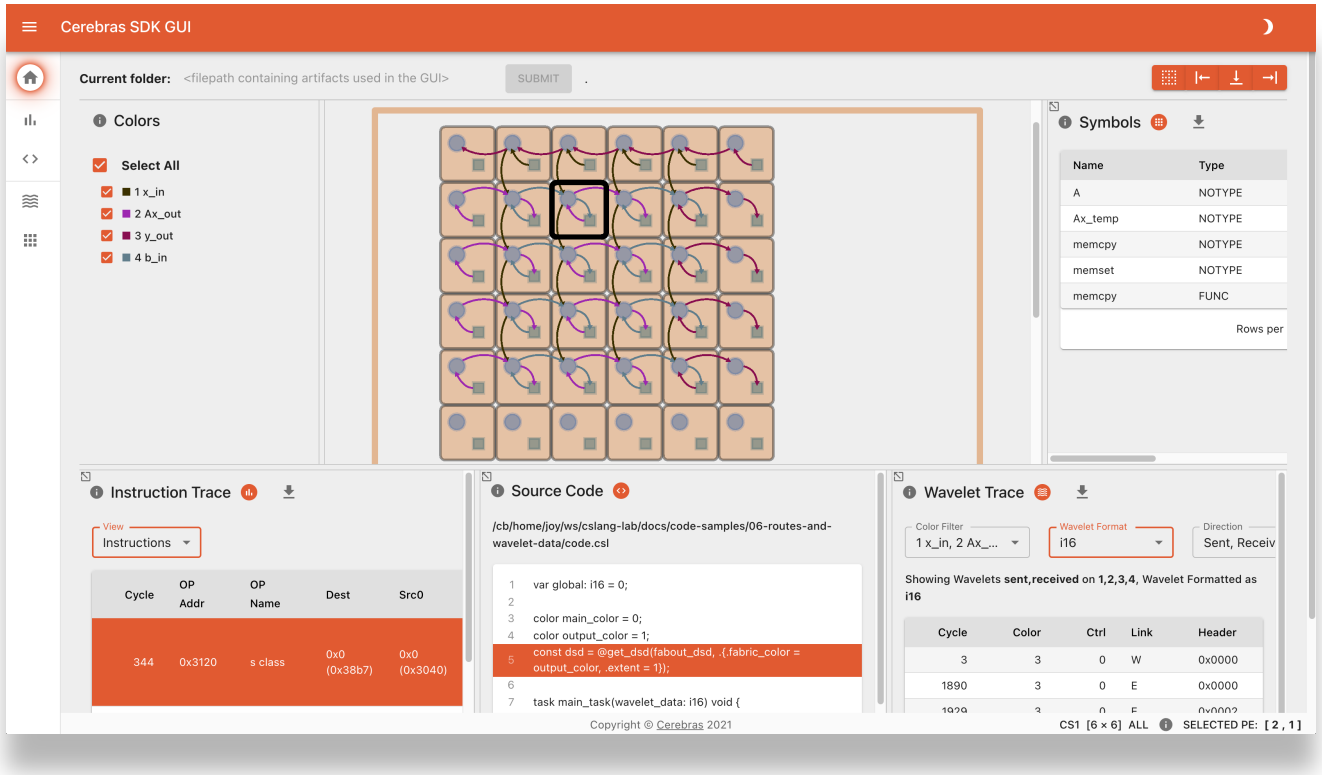
- CSL: Cerebras Software Language
- Host APIs with Python

**Libraries**

- Optimized primitives

**Tools**

- Simulator
- Debugger
- Performance profiler
- Visualization



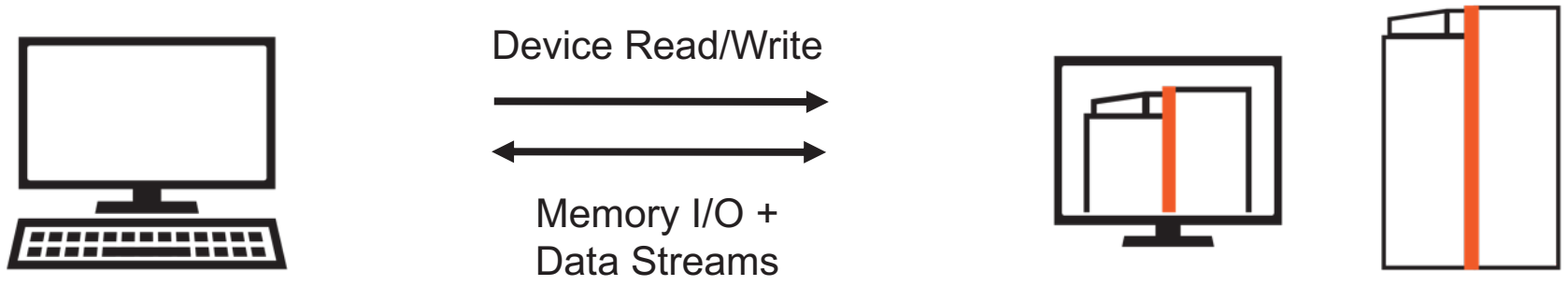
# From a Programmer's Perspective

## Host CPU(s): Python

- Loads program onto simulator or CS-2 system
- Streams in/out data from one or more workers
- Reads/writes device memory

## Device: CSL

- Target software simulator or CS-2
- CSL programs run on groups of cores on the WSE, specified by programmer
- Executes dataflow programs



# CSL: Language Basics

- Types
- Functions
- Control structures
- Structs/Unions/Enums
- Comptime

Straight from C
(via Zig)

- Builtins
- Module system
- Params
- Tasks
- Data Structure Descriptors
- Layout specification

CSL specific

**Used for writing device kernel code**

**Familiar to C/C++/HPC programmers**

# Familiar Features

## Types

- Syntax similar to other modern languages – Go, Swift, Scala, Rust
- Float (`f16`, `f32`), signed (`i16`, `i32`), unsigned (`u16`, `u32`), boolean (`bool`)

```
var x : i16;
const y = 42;
var arr : [16, 4]f32;
var ptr : *i16;
```

## Functions

- Zig-style syntax
- Pass by value or reference and inlining automatically handled

```
fn factorial(x : i32) i32 {
    if (x <= 2) return x;
    return x * factorial(x - 1);
}
```

## Control Structures

- Traditional control flow: **if**, **for**, **while**, with zig and C style syntax

```
if (x < 10) {
    y += 5;
} else {
    y += 10;
}
```

conditionals

```
var x: u16 = 100;
while(x > 99) {
    ...
}
```

**while** loop

```
var idx: u16 = 0;
while (idx < 5) : (idx += 1) {
    ...
}
```

**while** loop with iterator

```
const xs = [10]i16 { 0, 1, 2, 4 };
for (xs) |x,idx| {
    ...
}
```

range **for** loop
(also provides C-style **for**)

# Quality of Life Features

## Comptime

- From Zig, block of code where all evaluation occurs at compile time
- Useful for frontloading computation to avoid runtime overhead

```
comptime {
  const f23 = factorial(23);
  ...
}
```

## Params

- Like `#define`, but strongly typed
- Have to be “bound” completely during compilation

```
param M : i16;
param N : i16;
param is_left_edge : bool;
```

## Modules

- Any CSL source code file is a “Module,” importable into other modules
- Imported modules acts as an *instance* of a unique struct type
- Multiple imports of the same module allowed

m1.csl

```
var x = 0;
fn incr() void {
    x = x + 1;
}
```

p1.csl

```
const v1 = @import_module("m1.csl");
const v2 = @import_module("m1.csl");

v1.incr();
v2.incr(); v2.incr();

// v1.x == 1; v2.x == 2;
```

# Performance Features

## Builtins

- Similar to function calls with @ in front of function name
- Language extensions without special syntax
- Used for invoking special compiler functionality

```
// Initialize a tensor of four rows
// and five columns with all zeros.
  var matrix = @zeros([4,5]f16);
```

## Tasks

- Core building blocks of CSL
- Special functions used to implement dataflow programs
- Data tasks are triggered by incoming wavelets on a specific color
- Local tasks are triggered with calls to `@activate`

```
color recvColor;
var globalValue: u16 = 0;

task recvTask(data: u16) void {
  globalValue = data;
}

comptime {
  @bind_data_task(recvTask, recvColor);
  @set_local_color_config(recvColor,
    .{ .rx = .{ WEST }, .tx = .{ RAMP } });
}
```

# Performance Features

## Data Structure Descriptors (DSDs)

- Provide a mechanism to consider an array, and an access pattern, as a complete unit
- Operations using DSDs run for multiple cycles to complete an instruction on all data referenced by the DSD
- Performance *and* ease of use: lifts level of program to talking about whole structures, while lowering cost of computing indexing into hardware

```
const dstDsd = @get_dsd(mem1d_dsd, .{ .tensor_access = |i|{5} -> dst[i] });
const src0Dsd = @get_dsd(mem1d_dsd, .{ .tensor_access = |i|{5} -> src0[i] });
const src1Dsd = @get_dsd(mem1d_dsd, .{ .tensor_access = |i|{5} -> src1[i] });

const fabDsd = @get_dsd(fabout_dsd, .{.fabric_color = output_color, .extent = 1});

task main_task() void {
  @faddh(dstDsd, src0Dsd, src1Dsd);
  @fmovh(fabDsd, dstDsd);
}
```

DSDs are a ***unifying concept*** that provides for complex memory reads and writes and fabric reads and writes

# SDK Example Programs Available

**Repository:** github.com/Cerebras/csl-examples

- Introductory Tutorials
- GEMV
- GEMM
- Cholesky Decomposition
- 1D and 2D FFT
- 7-Point Stencil SpMV
- Power Method
- Conjugate Gradient
- Preconditioned Conjugate Gradient
- Finite Difference Stencil Computations
- Mandelbrot Set Generator
- Shift-Add Multiplication
- Hypersparse SpMV
- Histogram Computation

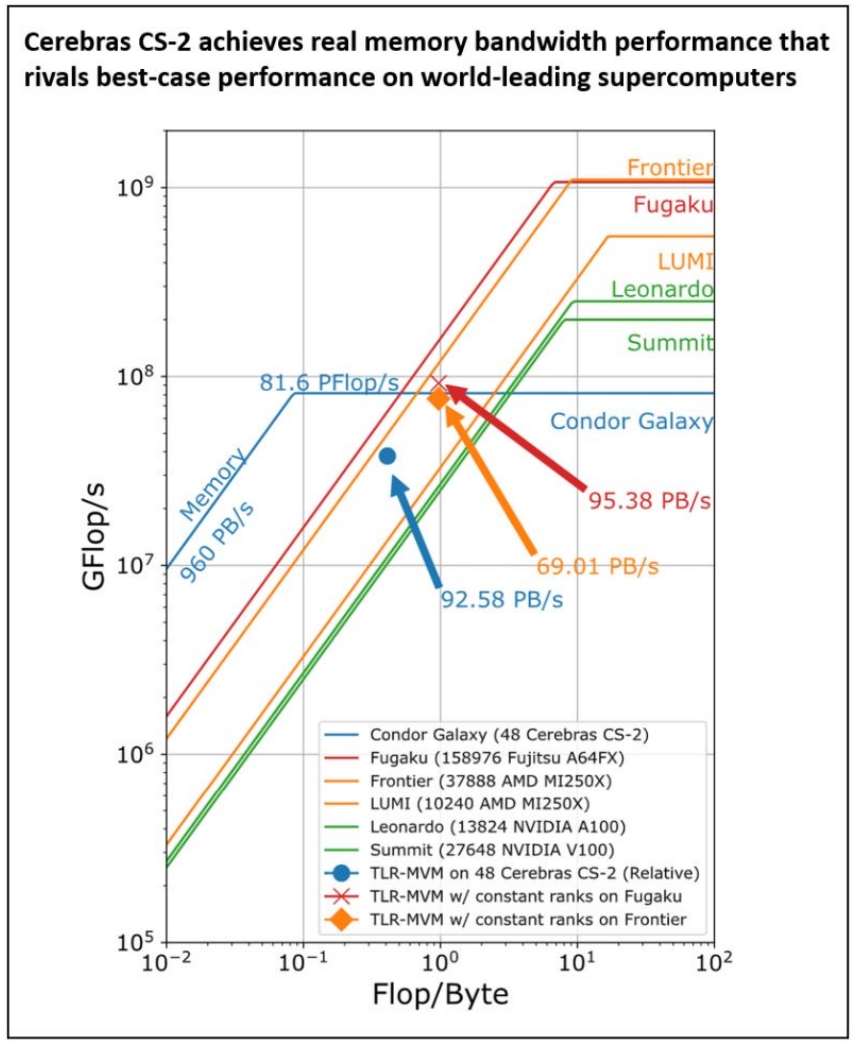
# Some Research and Applications

# Cerebras and KAUST use CS-2 to achieve performance comparable to world's largest supercomputers

- Researchers redesigned a Tile Low-Rank Matrix-Vector Multiplication (TLR-MVM) algorithm for Cerebras CS-2, taking advantage of the ultra high memory bandwidth
- Provided researchers with CG-1 AI supercomputer to run this simulation
- Achieved sustained memory bandwidth of **92.58 PB/s** across 48 CS-2 systems – higher than Frontier (#1 TOP500), comparable to Fugaku (#4 TOP500)

***2023 Gordon Bell Prize finalist***

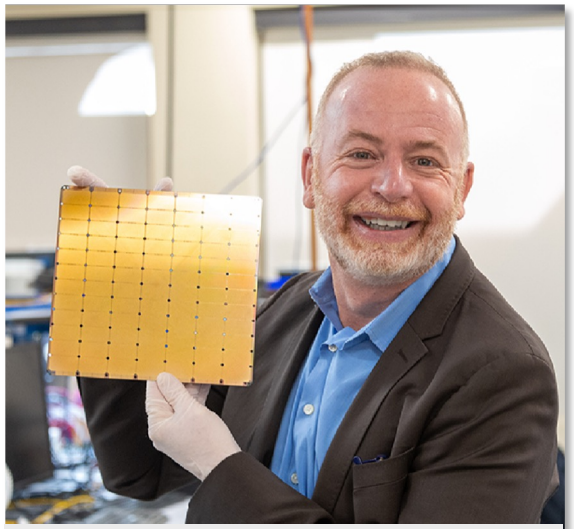
**Paper:** https://dl.acm.org/doi/10.1145/3581784.3627042



# TotalEnergies achieves 228x speedup vs. A100 on seismic imaging algorithm

*"As can be seen, when the largest problem is solved, a speedup of 228x is achieved... **Moreover…it is unlikely that such a performance gap can be closed**… given the strong scalability issues encountered by this kind of algorithm when using a large number of multi-GPU nodes in HPC clusters."*

**Speedup of 228x achieved with Cerebras**



**Diego Klahr VP**
**VP of Engineering at TotalEnergies**

**Paper:** https://arxiv.org/abs/2204.03775

Massively scalable stencil algorithm


Mathias Jacquelin[§]
*Cerebras Systems Inc.*
Sunnyvale, California, USA
mathias.jacquelin@cerebras.net

Mauricio Araya-Polo[§] and Jie Meng
*TotalEnergies EP Research & Technology US, LLC.*
Houston, Texas, USA
mauricio.araya@totalenergies.com





***Abstract*—Stencil computations lie at the heart of many scientific and industrial applications. Unfortunately, stencil algorithms perform poorly on machines with cache based memory hierarchy, due to low reuse of memory accesses. This work shows that for stencil computation a novel algorithm that leverages a localized communication strategy effectively exploits the Cerebras WSE-2, which has no cache hierarchy. This study focuses on a 25-point stencil finite-difference method for the 3D wave equation, a kernel frequently used in earth modeling as numerical simulation. In essence, the algorithm trades memory accesses for data communication and takes advantage of the fast communication fabric provided by the architecture. The algorithm —historically memory bound— becomes compute bound. This allows the implementation to achieve near perfect weak scaling, reaching up to 503 TFLOPs on WSE-2, a figure that only full clusters can eventually yield.**



***Index Terms*—Stencil computation, high performance computing, energy, wafer-scale, distributed memory, multi-processor architecture and micro-architecture**


| Traditional architecture | WSE |
|---|---|
| L1 | Memory |
| L2 & L3 | ∅ |
| DRAM | ∅ |
| Off-node interconnect | Fabric & routers |

TABLE I: Equivalences between traditional architectures and the WSE

## I. Introduction

Stencil computations are central to many scientific problems and industrial applications, from weather forecast ( [32]) to earthquake modeling ( [19]). The memory access pattern of this kind of algorithm, in which all values in memory are accessed but used in only very few arithmetic operations, is particularly unfriendly to hierarchical memory systems of traditional architectures. Optimizing these memory operations is the main focus of performance improvement research on the topic.

Subsurface characterization is another area where stencils are widely used. The objective is to identify major structures in the subsurface that can either hold hydrocarbon or be used for $CO_2$ sequestration. One step towards that end is called seismic modeling, where artificial perturbations of the subsurface are modeled solving the wave equation for given initial and boundary conditions. Solving seismic modeling efficiently is crucial for subsurface characterization, since many perturbation sources need to be modeled as the subsurface model iteratively improves. The numerical simulations required by seismic algorithms for field data are extremely demanding, falling naturally in the HPC category and requiring practical evaluation of technologies and advanced hardware architectures to speed up computations.

Advances in hardware architectures have motivated algorithmic changes and optimizations to stencil applications for at least 20 years ( [23]). Unfortunately, the hierarchical memory systems of most current architectures is not well-suited to stencil applications, therefore limiting performance. This applies to multi-core machines, clusters of multi-cores, and accelerator-based platforms such as GPGPUs, FPGAs, etc. ( [2], [5]). Alternatively, non-hierarchical architectures were explored in this context, such as the IBM Cell BE ( [3]), yielding high computational efficiency but with limited impact.

A key element for large scale simulations is the potential of deploying substantial number of processing units connected by an efficient fabric. The Cell BE lacked the former and it had limited connectivity. Another example of non-hierarchical memory system is the Connection Machine ( [12]), which excelled on scaling but at the cost of a very complex connectivity. In this work, a novel stencil algorithm based on localized communications that does not depend on memory hierarchy optimizations is introduced. This algorithm can take advantage of architectures such as the WSE from Cerebras ( [4]) and potentially Anton 3-like systems ( [28]). These are examples of architectures addressing both limitations described above.

Another angle to be considered is the availability of hardware-based solutions in the market. Literature review yields no generally available hardware architecture addressing the specific bottlenecks of stencil applications. Only a few custom designs examples are available ( [10], [14]).

In this work, an implementation of such seismic modeling method on a novel architecture is presented. The proposed mapping requires a complete redesign of the basic stencil algorithm. The contribution of this work is multi-fold:

[§]Equal contribution.

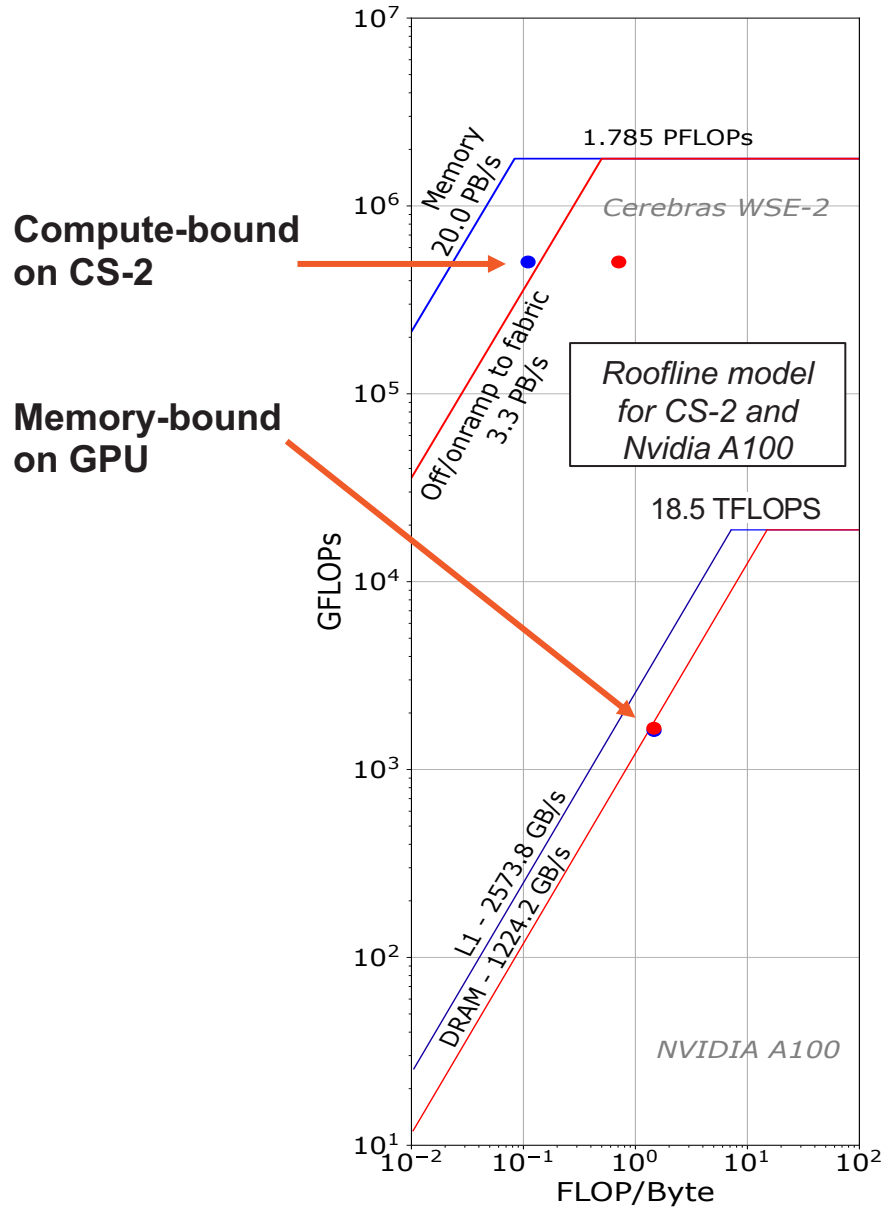
# TotalEnergies seismic research overview

Common computational approaches to solving seismic imaging problems, such as stencil methods, are typically memory-bound.

Additionally, strong scaling is typically limited by fabric bandwidth between compute nodes.

**Total has addressed these challenges with Cerebras:**

- Implemented 25-point stencil for the 3D wave equation with source perturbation, achieved **228x speedup over A100**. Presented at SC22.
- Implemented finite volume flux computation for single phase flow, achieved **204x speedup over A100**. Presented at SC23.
- Additionally developed proprietary RTM (Reverse Time Migration) code for internal use.




**Papers:** https://arxiv.org/abs/2204.03775 and https://arxiv.org/abs/2304.11274

# ANL uses CS-2 to accelerate Monte Carto particle transport kernel by **130x** over A100

*"The WSE is found to run **130 times faster** than a highly optimized CUDA version of the kernel run on an NVIDIA A100 GPU – significantly outpacing the expected performance increase given the relative number of transistors each architecture has"*

Last week, PHYSOR publication demonstrates **180x** over A100.

**Paper:** https://arxiv.org/abs/2311.01739

## Efficient Algorithms for Monte Carlo Particle Transport on AI Accelerator Hardware

John Tramm[a,*], Bryce Allen[a,b], Kazutomo Yoshii[a], Andrew Siegel[a], Leighton Wilson[c]

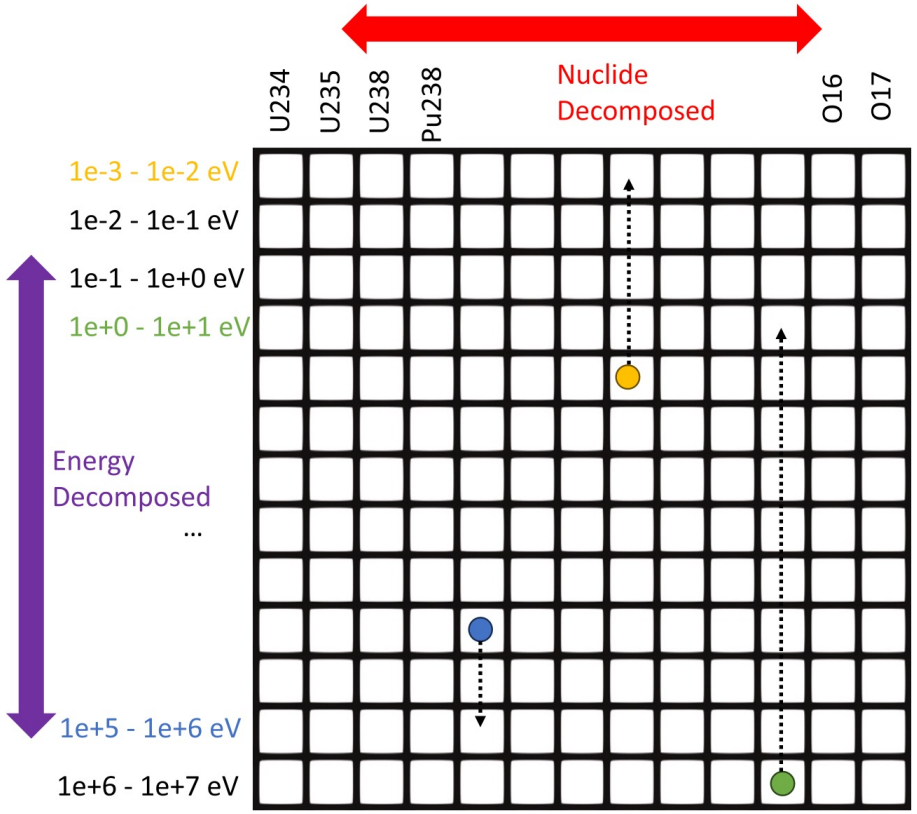
[a] *Argonne National Laboratory, 9700 S Cass Ave., Lemont, 60439, IL, USA*
[b] *University of Chicago, 5801 S. Ellis Ave., Chicago, 60637, IL, USA*
[c] *Cerebras Systems Inc., 1237 E Arques Ave, Sunnyvale, 94085, CA, USA*

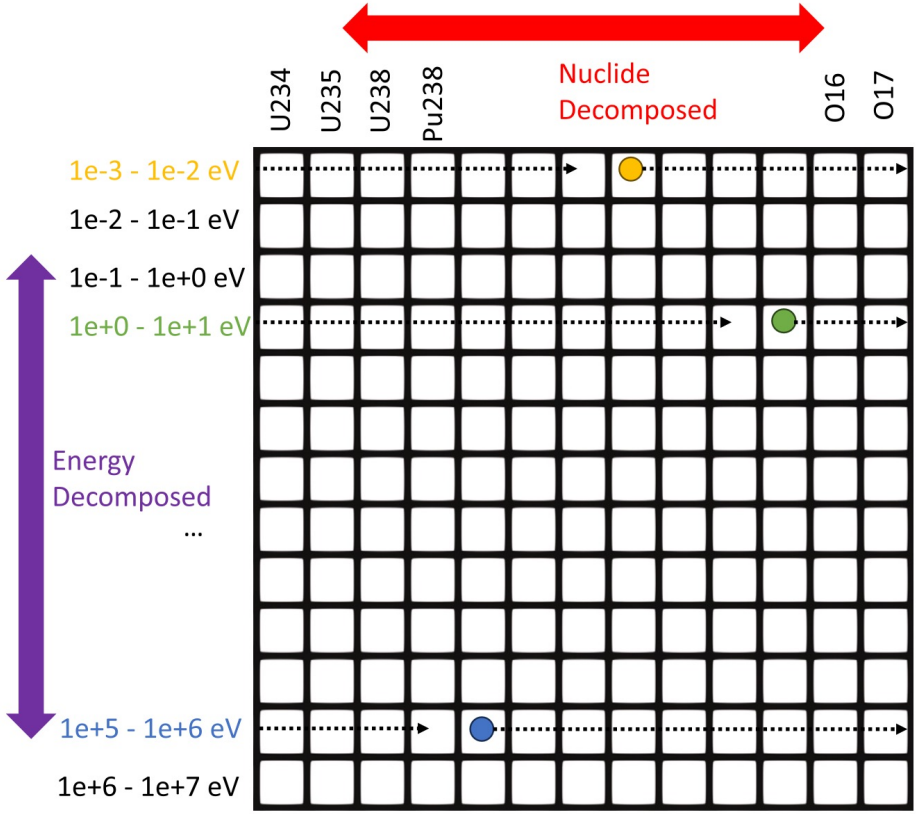
**Abstract**

The recent trend in computing towards deep learning has resulted in the development of a variety of highly innovative AI accelerator architectures. One such architecture, the Cerebras Wafer-Scale Engine 2 (WSE2), features 40 GB of on-chip SRAM making it an attractive platform for latency- or bandwidth-bound HPC simulation workloads. In this study, we examine the feasibility of performing continuous energy Monte Carlo (MC) particle transport by porting a key kernel from the MC transport algorithm to Cerebras' CSL programming model. We then optimize the kernel and experiment with several novel algorithms for decomposing data structures across the WSE2's 2D network grid of approximately 750,000 user-programmable distributed memory compute cores and for flowing particles (tasks) through the WSE2's network for processing. New algorithms for minimizing communication costs and for handling load balancing are developed and tested. The WSE2 is found to run 130 times faster than a highly optimized CUDA version of the kernel run on an NVIDIA A100 GPU — significantly outpacing the expected performance increase given the relative number of transistors each architecture has.

# ANL uses CS-2 to accelerate Monte Carto particle transport kernel

$$\Sigma_r(e) = \sum_{n}^{material} \sigma_{r,n}(e)\rho_n$$




J. Tramm et al., Efficient algorithms for Monte Carlo particle transport on AI accelerator hardware, *Commun. Comput. Phys.* (2024).
J. Tramm et al., Monte Carlo with single-cycle latency, *PHYSOR* (2024).

# CS-2 Accelerates molecular dynamics for metallic alloys
# ***179x faster than Frontier***

*"Measured performance and power efficiency of WSE, GPU, and CPU systems on 800,000-atom simulations. WSE used FP32 precision while GPU and CPU used FP64 precision. (a) A **single WSE wafer results in 179x and 55x speedup compared to Frontier** and **CPU** based simulations; (b) WSE provides one to two orders of magnitude improvement in power efficiency over both CPU and GPU systems; (c) Relative power efficiency and speedup of WSE compared to CPU and GPU systems."*

**Paper:** Manuscript submitted to SC24

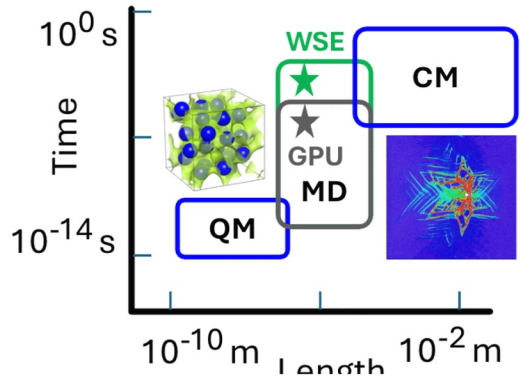


## Fast Molecular Dynamics on a Wafer-Scale System

Kylee Santos*, Stan Moore†, Tomas Oppelstrup‡, Amirali Sharifian*, Ilya Sharapov*, Aidan Thompson†,
Delyan Z Kalchev*, Danny Perez§, Scott Pakin§, Edgar A. Leon‡, James H Laros III†,
Michael James*, and Sivasankaran Rajamanickam†
*Cerebras Systems, Sunnyvale, CA
†Sandia National Laboratories, Albuquerque, NM
‡Lawrence Livermore National Laboratory, Livermore, CA
§Los Alamos National Laboratory, Los Alamos, NM

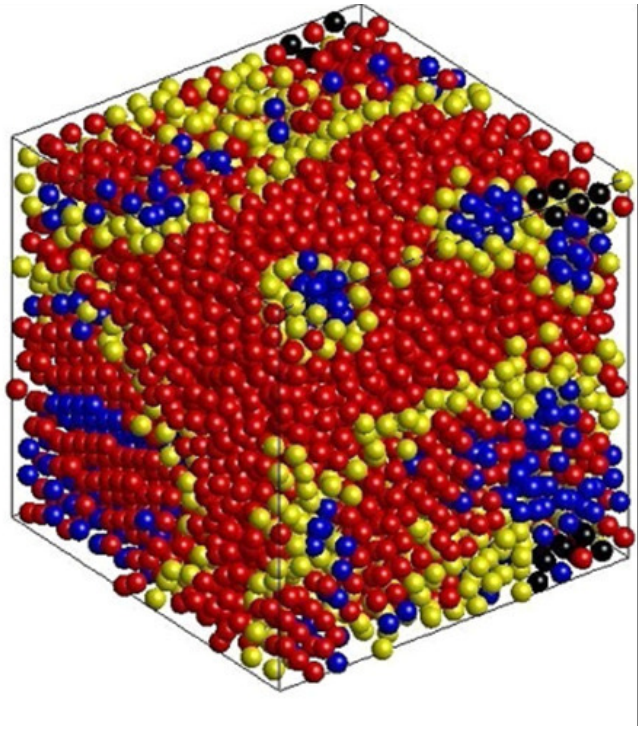
***Abstract*—Molecular dynamics (MD) simulations have transformed our understanding of atomic systems, driving breakthroughs in material science, computational chemistry and several other fields like biophysics and drug design. Using the Cerebras Wafer-Scale Engine, we demonstrate an improvement in MD iteration rate that enables a transformative capability for long-time simulations. This unlocks currently inaccessible timescales of slow microstructure transformation processes that are critical for understanding material behavior and function.**

**Our dataflow algorithm runs an Embedded Atom Method (EAM) simulation at rates over 270,000 timesteps per second for problems with up to 800k atoms. This corresponds to a nearly 180-fold speedup versus the Frontier GPU-based Exascale platform. It simultaneously achieves an over 30-fold improvement in energy efficiency. This demonstrated performance is unprecedented for general-purpose processing cores. With further parallelization of the algorithm, we project performance in excess of one million timesteps per second for 200,000 atoms. This projected perfor-**



# CS-2 accelerates molecular dynamics for metallic alloys

- Embedded Atom Model (EAM) is a molecular dynamics method with an interatomic potential suited for modelling metallic systems
- Strong scaling applies more than one core per simulated atom
- Simulation timestep **1,000x faster than today's SOTA**
- **2 years on Exascale** done in **1 day on a CS-2**
- Investigate long time-scale system properties previously infeasible to compute
- Larger molecular systems can scale to cluster of Cerebras nodes with same timestep performance
- Extensions for biomolecules possible



# Getting Access and Running

# SDK Access and Next Steps

Get local access to the SDK simulator!

- Email developer@cerebras.net for access

Join the Cerebras Developer Community

- Forums at discourse.cerebras.net

discourse.cerebras.net

View our public SDK examples GitHub repository

- See github.com/Cerebras/csl-examples

Run on ANL's systems with appliance mode

- See https://sdk.cerebras.net/appliance-mode

Questions? leighton.wilson@cerebras.net

cerebras.net/developers/sdk-request

# Roadmap

# Optimized Models in Q2 2024: LLM Focus

| Model type | Model architecture | Model examples |
|---|---|---|
| **Decoder-only Transformers** | • Sequential (e.g GPT) and parallel (e.g. GPT-J attention and feed-forward blocks<br>• Attention types: vanilla multi-head (GPT), MQA (Llama 7B), GQA (Llama-2 70B)<br>• Activation functions: relu, gelu (GPT), swiglu (Llama), etc<br>• Positional encodings: learned (GPT), fixed, RoPE (Llama) | • Llama / Llama 2 / Llama 3<br>• Mistral7B<br>• GPT-2 / GPT-3<br>• GPT-J / GPT-NeoX<br>• MPT<br>• Falcon<br>• Bloom<br>• JAIS<br>• StarCoder<br>• SantaCoder<br>• BTLM |
| **Encoder-only Transformers** | • BERT-style | • BERT Base/Large<br>• BERT SQuAD, SST, MNLI, NER |
| **Encoder-decoder Transformers** | • Vanilla transformer and variants | • Transformer<br>• T5 |

# Optimized Models in Q2 2024: LLM Focus (cont.)

| Model type | Model architecture | Model examples |
|---|---|---|
| **Multimodal** | • LLaVA-style: decoder-style, multiple image encoders, LLM backbones<br>• PaLI-style: encoder-decoder | • LLaVA 1.5<br>• AnyMAL<br>Eyes Wide Shut |
| **Embeddings & Alignment** | • BERT-style embedding models<br>• Alignment with DPO with supported LLM backbones | • DPR<br>• DPO |

# GenAI Training & Fine-Tuning Capabilities through 2025

| H1 2024 | H2 2024 | 2025 |
| --- | --- | --- |
| **Extensive LLM support:** Mistral, LLaMa, GPT, MPT, BERT, etc.<br><br>**Mixture of Experts LLMs**<br>• E.g. Mixtral 8x7B and 8x22B<br><br>**Multimodal**<br>• Visual Question Answering<br>• Inputs:<br>  ○ Text, code<br>  ○ HQ images, charts, graphs<br>• Outputs:<br>  ○ Text | **GenAI+ (MM + MoE)**<br>• Inputs:<br>  ○ + DNA<br>  ○ + Video encoders<br>• Output Generation:<br>  ○ + Images<br>• Multi-modal MoEs | **GenAI-Next**<br>• All-to-all modalities<br>• New GenAI architectures, e.g.<br>  • SSMs<br>  • Mixing convolutions with GPT architectures |

# Summary

- **Now (Q2): Industry-leading performance on state-of-the-art LLMs, and early Multimodality VQA**
  - LLMs: Llama 3, Mistral, GPT3, MPT, JAIS, Falcon
  - Multimodal: Visual Question Answering (input modalities of: text, images, charts, graphs, code)
- **In July'24: Mixture of Experts & Multimodality**
  - MoE LLMs: Mixtral 8 x 7B, Mixtral 8 x 22B
  - Multimodal: Fast pre-training of HQ image encoders & higher performance
- **By EOY: Full optimization of MoE and Multimodality features**
  - Multimodal MoE models
  - Multimodal with video and DNA input, and image generation output

# Q & A, Final remarks

# Accelerate Scientific Discovery with Us at ALCF

- **We're passionate about driving innovation:** Our mission is to create the ultimate platform for large-scale scientific AI and open science.
- **Unleash your research potential:** Experience unparalleled performance, effortless scaling, and superior efficiency for faster breakthroughs.
- **ALCF's CS-2 systems empower you:** Discover the transformative impact they have on open scientific research.
- **Effortless access to cutting-edge AI:** Tap into powerful features like expanded model size, distributed compute, greater context length, and sparsity optimization.

**Let's collaborate on groundbreaking science!** Share your ambitious projects and ideas – we're eager to support your success.

# How to join ALCF

The ALCF welcomes open research projects seeking access to ANL production systems.

- Project teams are encouraged to submit their applications through the Director's Discretionary Allocation Program (DDAP) page.
- Our DDAP program page provides information on how to apply for access to Cerebras Systems and other production systems available.
- Rolling proposals are accepted from project teams at any time.
- Notification of proposal status is typically provided within 1-2 weeks of submission.
- The ALCF's DDAP program is committed to supporting innovative research initiatives and empowering project teams to achieve their goals.

- DDAP program page ---> https://www.alcf.anl.gov/science/directors-discretionary-allocation-program

# How to contact Cerebras?

- Email us at developer@cerebras.net
- Sign up for our monthly newsletter at info.cerebras.net/subscribe
- Join our Discord at discord.gg/hZp5MUyw
- Join our Discourse at discourse.cerebras.net/
- LinkedIn - linkedin.com/company/cerebras-systems/
- Twitter - twitter.com/CerebrasSystems

# Thank you