Deep Learning Programming at Scale

Justin Selig, Cerebras SDK Product Manager
Saumya Satish, Machine Learning Product Manager

Abstract

Deep learning has become one of the most important computational workloads of our generation, but it is profoundly computationally intensive. Today, large neural networks are often trained using large clusters of graphics processing units (GPUs). These clusters are expensive, complicated to program, and can take weeks to train a network.

In this paper, we highlight some of the most painful steps involved with distributed deep learning on GPU clusters and describe how Cerebras makes training even massive networks fast, simple and accessible to every ML practitioner, without requiring cluster or parallel programming expertise.

Contents

Introduction

Deep learning has become one of the most important computational workloads of our generation, advancing applications across industries from healthcare to autonomous driving. But it is also profoundly computationally intensive.

To train today's state-of-the-art neural networks, researchers often have to use large clusters of dozens to hundreds of graphics processing units (GPUs). These clusters are expensive to build, complicated to program for, and can still take days to weeks to train a network, dragging the pace of innovation. We founded Cerebras Systems to solve this problem.

Most deep learning work begins in a prototype phase, when researchers want to experiment quickly and iterate often. When an experiment is in its earliest stages and being run with a limited subset of data, a small-scale hardware setup (a workstation with a single GPU, for example) can suffice.

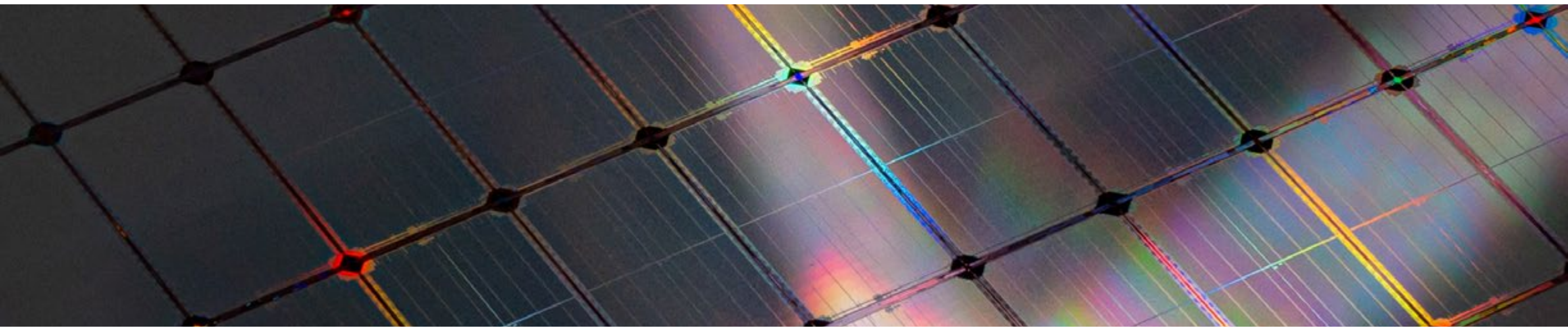
But as research scales up or models move into production, the increased complexity and larger datasets require vastly more compute. At this point, researchers have historically turned to large, scale-out GPU clusters. However, achieving good utilization across an enormous compute cluster is complicated and time consuming. It requires distributing a workload across many small devices, dealing with device memory sizes and memory bandwidth constraints, and carefully managing communication and synchronization overheads. Researchers often find themselves needing to pull in additional layers of software like Horovod and OpenMPI.

Furthermore, it is rare for massively distributed training setups to produce correct results right out of the box. Scale-out efficiency relies on using very large batch sizes, which can affect how models converge. To deal with accuracy drops, researchers run extensive experiment sweeps, tuning learning rates and trying different optimizers, in order to find an optimal training configuration that is then highly tailored to a specific hardware configuration.

So, while it is possible to bring more compute to neural network training by clustering large numbers of GPUs, it is complicated, time-consuming, and difficult for the entire organization - from machine learning (ML) researchers to production ML engineers, infrastructure, and IT teams.

At Cerebras Systems, we believe that state-of-the-art deep learning should be simple and accessible to every ML practitioner, without requiring cluster or parallel programming expertise. Powered by wafer-scale technology, the Cerebras CS-2 system densifies the compute and memory of an entire cluster onto a single chip in a single device. This enables ML researchers and practitioners to achieve cluster-scale performance with the programming ease of a single machine. And for extreme-scale models, Cerebras systems can be clustered "invisibly", meaning that performance scales linearly with any code changes.

In the sections below, we highlight some of the most painful steps involved with distributed deep learning on GPU clusters and describe an easier, faster way to achieve deep learning at scale on Cerebras systems.



Traditional approach: distributed training with a GPU cluster

What does it take today to scale training across a large cluster of GPUs?

Model distribution and cluster orchestration
ML Frameworks like PyTorch and TensorFlow have made it straightforward to build and run a model on a single GPU. But eventually you hit a performance wall and need to scale out to support full-scale experiments that use large amounts of data. At this point you are faced with the challenge of figuring out how to distribute your model across many GPUs.

Multi-GPU workload distribution requires thinking beyond the scope of a single neural network model and exploiting parallelism across devices. Typically, users start by changing their model code to train data-parallel across one or many multi-GPU machines, and use framework augmentations like Distributed TensorFlow or PyTorch Distributed for software configuration.

These frameworks make scaling out less painful than trying to manually implement data, or model-parallel execution, but it still takes time to tune the setups for them, and for them to learn. And getting the model running is just the first step – it is only rarely that a distributed model will run at good utilization and target accuracy right out of the box.

Device and cluster orchestration
Trying to run distributed deep learning workload at higher device utilization becomes more involved. It's a complex challenge to harness maximum functional performance from many individual small processors, each with specific device constraints, which all need to be carefully managed and orchestrated.

Doing this involves figuring out how to allocate compute between devices, how to think about device memory sizes and memory constraints, and how to deal with the communication and synchronization overheads among them. This is the point at which many users bring in yet more frameworks like Horovod and libraries like OpenMPI – additional software to help with workload distribution, inter-process communication, and both inter- and intra-node communication.

But model parallelization is not deep learning research or engineering; it is supercomputer cluster engineering, and a fundamentally complex parallel programming problem. Even with the best tools, it is extremely time consuming, and often requires the expertise of supporting IT, HPC, and ML engineering teams. While this work is meant to speed up training across more GPUs, it is instead itself one of the main reasons why training state-of-the-art models remains such a slow and difficult process.

Convergence and tuning
Successfully distributing a model across a cluster takes more than tuning the cluster setup and orchestrating the software; it requires researchers to change their actual model implementations.

As GPU clusters scale to 10s to 100s to even 1,000s of individual GPUs over many multi-GPU servers, researchers are often forced to use extremely large batch sizes to mitigate massive communication overheads and to achieve device utilization. But extreme batch size training often has significant impact on model convergence. It can cause the total number of epochs needed to increase considerably and can even result in drops in model accuracy.

Achieving a fast, distributed model implementation that still converges to target accuracy can take days, weeks, or even longer. Researchers often need to run dozens of experiments to find the right combination of hyperparameters (e.g. batch size, learning rate and momentum), optimizers, and more to achieve convergence and target accuracy at scale.

Meanwhile, wall-clock training time also scales very sub-linearly (Figure 1). For example, A single DGX-A100 server equipped with eight A100 GPUs, completed the second phase on the MLPerf benchmarking dataset in 20 minutes, while a massive cluster of 540 DGX-A100 servers took 13.5 seconds. In other words, a 540x increase in the number of GPUs only delivered 89x speed-up.[1]

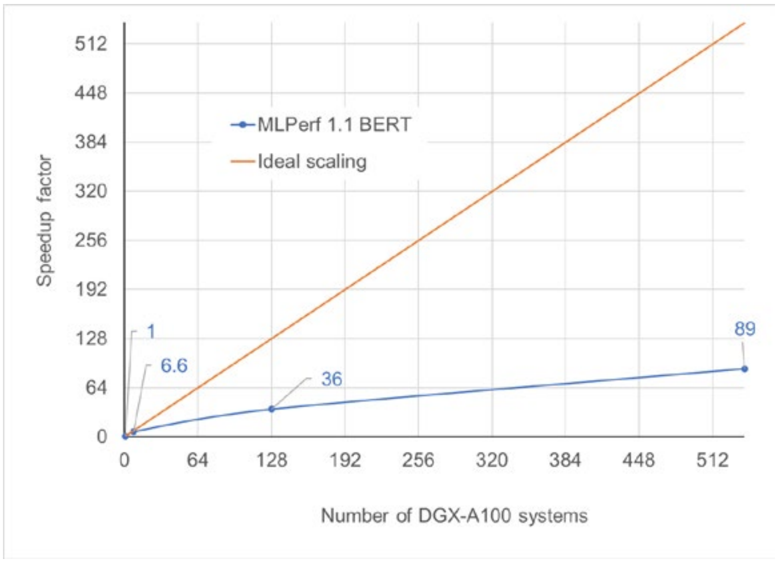
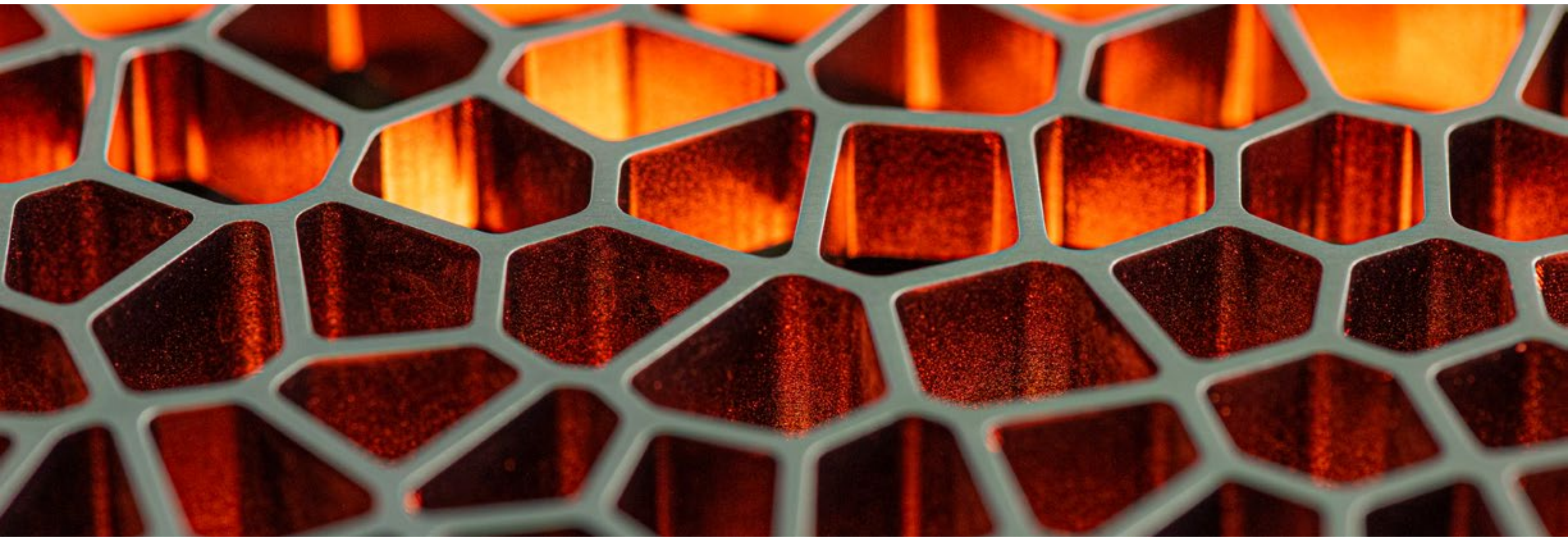


Figure 1. Non-linear scaling performance of GPU clusters (MLPerf 1.1 benchmark published results)

As the need for compute grows, researchers running distributed GPU implementations must deal with increasing software and model convergence complexities, all while getting diminishing returns in performance.

And finally, the resulting distributed cluster implementation of a deep learning model is brittle. Training time-to-accuracy on a cluster is a function of the model, data, batch size, learning rate, and number of devices. This means that if a researcher needs to change their data dimensions, dataset, model architecture, or neural network layer operations / optimizers, they need to reset their work and re-do the functional hyperparameter tuning, debugging, and performance tuning experiments.

In summary, scaling deep learning training on a GPU cluster is time-intensive, complex, and often results in brittle, suboptimal solutions.



The Cerebras Solution

At Cerebras, we built our systems to eliminate the challenges associated with scaling deep learning models across a GPU cluster.

Powered by the largest chip in the world (over 56x larger than the next-largest chip), the second-generation wafer-scale engine (WSE-2), the CS-2 system densifies the compute power of 850,000 AI-optimized cores onto a single chip. Keeping everything together on silicon means the CS-2 delivers not only enormous compute and on-chip memory, but also orders of magnitude more memory and interconnect bandwidth than a GPU. These purpose-built performance advantages allow the CS-2 to accelerate deep learning models far beyond what processors optimized for other applications, such as GPUs, are capable of.

A single CS-2 provides the wall-clock compute performance of an entire cluster of GPUs made up of dozens to hundreds of individual processors, at a fraction of the space and power.

For organizations, this means faster insights at lower cost. For the ML researcher, this means achieving cluster-scale performance with the programming ease of a single device. With the CS-2, researchers can accelerate state-of-the-art models without spending the days to weeks on the setup and tuning needed to run distributed training on large clusters of small devices.

Invisible Clustering with Weight Streaming

As we've seen, because we don't split problems up to multiple small chips, we set developers free from parallel programming and cluster tuning. Each Cerebras system provides cluster-scale AI compute resource with the programming ease of a laptop.

And the gigantic models in our near future – impossible to train today – can be seamlessly spread across multiple Cerebras systems by just changing a configuration file. Users simply compile the neural network mapping for a single CS-2 system, and the Cerebras software takes care of the rest.

Cerebras makes training massive models practical. Our "weight streaming" execution mode provides a seamless scaling path from $\text{BERT}_{\text{LARGE}}$, to GPT-J, to GPT-3 and all the way up to models with more than 100 trillion parameters! Weight streaming works by inverting our normal execution mode: instead of storing weights on chip, we store activations on the wafer, one massive layer at a time, streaming the weights layer-by-layer to and from the WSE (Figure 2).[2]

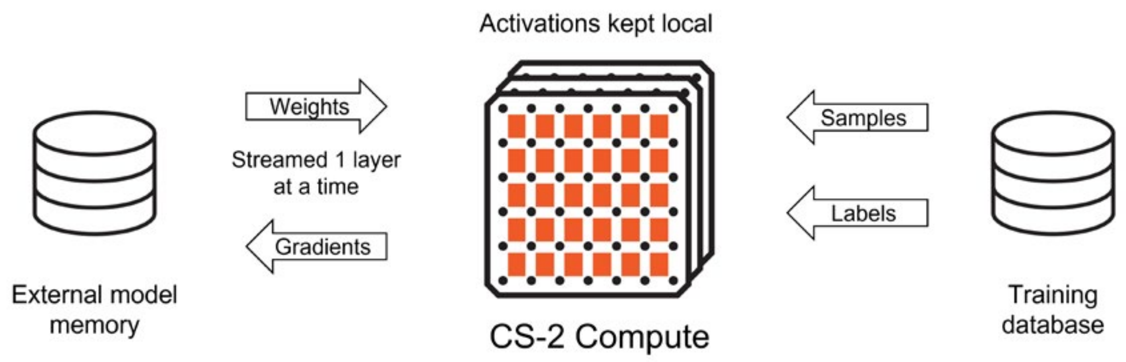


Figure 2. Weight streaming execution mode system diagram.

Weight Streaming Delivers Linear Performance Scaling

We have demonstrated that the weight streaming architecture delivers linear performance scaling as more CS-2 systems are added to the cluster (Figure 3). This is in stark contrast to the strongly sub-linear scaling observed with conventional GPU-based systems (Figure 1).

As Figure 3 shows, the bigger the model, the further the linear trend persists to larger cluster sizes. Note that the 10x in the legend indicates the speed up we achieve from a conservative 90% sparsity. The multiple lines indicate results for models with different aspect ratios. This data shows that it's possible to train a model with a trillion parameters in just a few days.

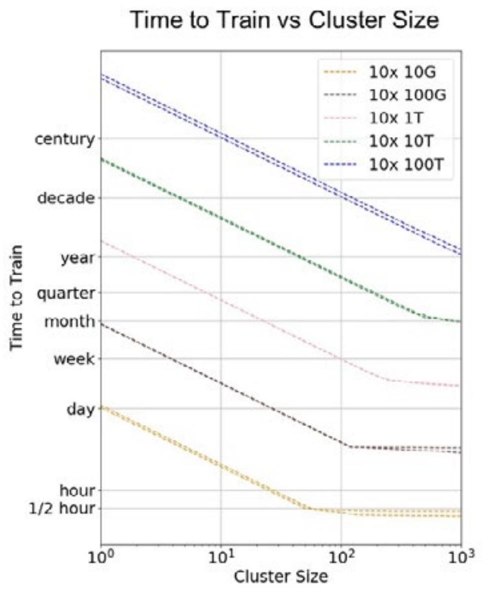


Figure 3. Cerebras CS-2 cluster demonstrates linear scaling performance with increasing model size

This remarkable result shows that the simplicity of the Cerebras programming model ensures that the expertise acquired today developing code for our architecture will pay dividends far into the future.

Learn more about Cerebras weight streaming technology

Scaling Up and Out: Training Massive Models on Cerebras Systems using Weight Streaming (technical blog)[3]

Training Giant Neural Networks Using Weight Streaming on Cerebras Wafer-Scale Systems (white paper)[2]

Programming the CS-2

Because the CS-2 delivers cluster-scale acceleration in a single device, users can scale up performance and dramatically reduce time-to-solution, all while keeping their programming model simple. Data scientists and ML researchers can focus on working with their data, model, and application, rather than sinking time into addressing the challenges and idiosyncrasies of multi-device cluster orchestration and optimization.

Researchers can program the CS-2 using familiar ML frameworks like TensorFlow and PyTorch. After that, the Cerebras Graph Compiler (CGC) handles everything else to automatically translate the user's neural network graph into an optimized executable for the 850,000 cores of the CS-2 (Figure 4).

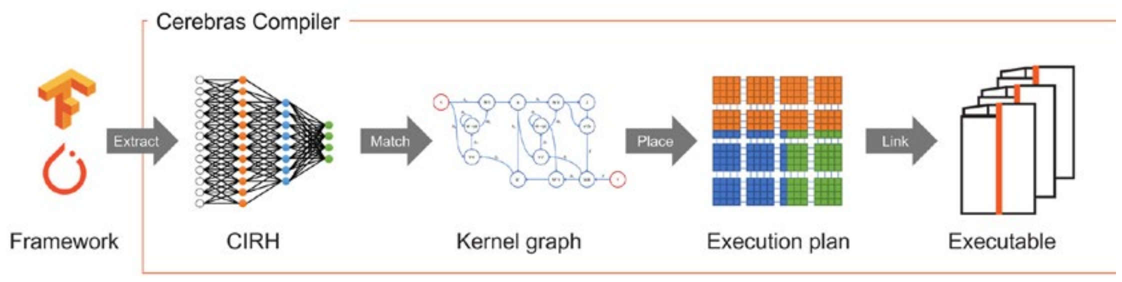


Figure 4. Diagram showing how the Cerebras Graph Compiler turns framework code into executables for the Cerebras SC-2 system.

Bringing an application to life on the CS-2 is as simple as adding a few lines of code. Because the CS-2 is such a powerful single system, no additional work is needed to scale the network across multiple small devices or to deal with communication and synchronization issues. Users can quickly experiment with alternative model architectures, hyperparameters, and different batch sizes by changing just a few lines of code.

With a CS-2, end-to-end model development tasks such as model setup, hyperparameter optimization, scaling, and performance optimization can be done in hours or days, as compared to the weeks they can take on a traditional GPU cluster.

PyTorch Example
Our PyTorch interface library is a simple wrapper for PyTorch program exposed through API calls that is easy to add as few extra lines of code for an existing PyTorch implementation. The integration is via lazy tensor backend with XLA to capture the full graph of a model and map it optimally onto the WSE-2.

This code snippet illustrates how easy it is for a user to adapt their PyTorch code.

Learn more about Cerebras PyTorch integration

Supporting PyTorch on the Cerebras Wafer-Scale Engine (technical blog)[4]

Getting Started with PyTorch BERT Models on the Cerebras CS-2 System (walkthrough)[5]

```
import torch
import cerebras.framework.torch as cbtorch

cbtorch.initialize(
        cs_ip=cs_ip,
         …   )

model = cbtorch.module(model_fn( ))
dataloader = cbtorch.dataloader(data_fn())
optimizer = cbtorch.optimizer(optimizer)

with cbtorch.Session(data_loader, modes.TRAIN):
           super().train(data_loader)
with cbtorch.Session(data_loader, modes.EVAL):
           super().evaluate(data_loader)
```

TensorFlow Example
The `CerebrasEstimator` is a wrapper class developed by our team for TensorFlow. Users simply import `CerebrasEstimator`, then define their model function, input function, relevant parameters, and training script as usual, using standard TensorFlow semantics.

The entire process is captured below:

```
from cerebras.tf.cs_estimator import CerebrasEstimator
from cerebras.tf.run_config import CSRunConfig

    est_config = CSRunConfig(
        cs_ip=params[“cs_ip”],
        cs_config=cs_config,
    )
    est = CerebrasEstimator(
        model_fn=model_fn,
        model_dir=`./out`
        config=est_config,
        params=params,
        use_cs=True
    )
    est.train(input_fn, max_steps=100000, use_cs=True)
```

`CerebrasEstimator` is subclassed from the official TensorFlow Estimator class, to keep the workflow easy and familiar. The user need only instantiate the `CerebrasEstimator`, provide an IP address for their Cerebras system, and set a flag `use_cs=True` to direct training or inference to the CS-2. At runtime, `CerebrasEstimator.train()` will automatically invoke the CGC and handle the rest of what is needed to prepare a model for the CS-2.

Time-to-solution advantage

The CS-2 system's unique combination of tremendous performance and single-node simplicity not only avoids parallel programming complexity, but also unlocks much faster time-to-solution, from research concept to model-in-production.

In a typical GPU cluster setup, an ML engineer may spend days or weeks choosing and tuning hyperparameters to achieve acceptable device utilization while maintaining model accuracy at extreme batch sizes.

Because the CS-2 is a single powerful device, there is no such batch size requirement. On CS-2, researchers can train models with high utilization at any batch size, avoiding the need to run additional hyperparameter sweeps to avoid accuracy drops. This means that users can achieve not only enormous acceleration right out of the box, but can also increase model convergence to target accuracy.

In partnership with one of our life sciences customers, we compared the time-to-solution for a domain-specific BERT NLP model development project from concept to production using a GPU cluster versus using a Cerebras system (Figure 5).

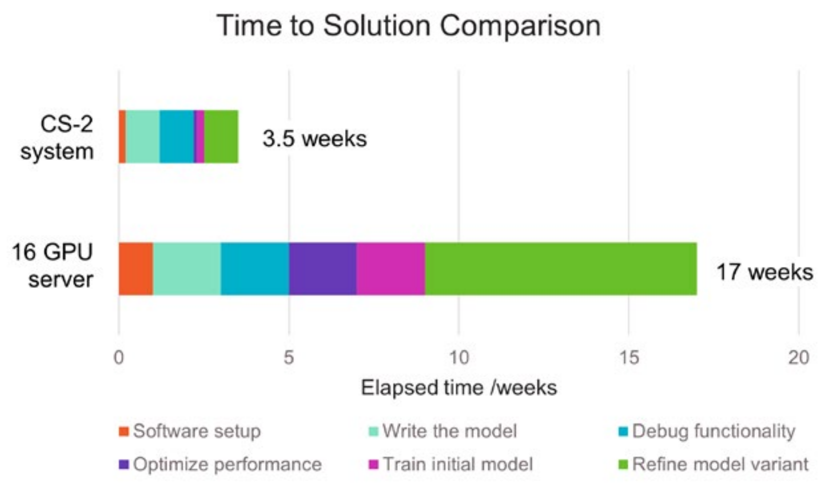


Figure 5. Overall time-to-solution for a Cerebras system vs customer GPU cluster. End-to-end development time was reduced by almost 5x.

We considered the same model and dataset, and included steps for software setup: model definition, functional debugging, performance optimization, initial model training, and subsequent experiments to develop a production-ready implementation.

This work showed that the Cerebras solution reduced end-to-end time to production solution from 17 weeks on a GPU cluster to 3.5 weeks on a Cerebras system. Programming and compute time were reduced by more than three months, saving our customer significant engineering costs and allowing them to accelerate new AI innovation.

Explore more customer applications and use cases

GlaxoSmithKline and Cerebras are Advancing the State of the Art in AI for Drug Discovery (joint customer blog)[6]

Accelerating NLP Model Training and Enabling Higher Accuracy for Financial Services Applications (case study)[7]

Conclusion

Deep Learning will continue to be one of the most important computational workloads of our time. Today's traditional systems are creating drag on the pace of innovation across countless industries.

At Cerebras Systems, we believe that state-of-the-art deep learning should be simple and accessible. We have been able to densify the compute and memory of an entire cluster's worth of compute onto a single chip in a single device. We have created an easier, faster way to achieve deep learning at scale.

To learn more or to see a demo, please contact us at cerebras.net/get-demo.

References

1 MLPerf training 1.1 results https://mlcommons.org/en/news/mlperf-training-v11/

2 Stewart Hall, Rob Schreiber, Sean Lie, "Training Giant Neural Networks Using Weight Streaming on Cerebras Wafer-Scale Systems", Cerebras Systems, 2021 https://f.hubspotusercontent30.net/hubfs/8968533/Virtual%20Booth%20Docs/CS%20Weight%20Streaming%20White%20Paper%20111521.pdf

3 Michael James, "Scaling Up and Out: Training Massive Models on Cerebras Systems using Weight Streaming", Cerebras Systems, 2021 https://www.cerebras.net/blog/scaling-up-and-out-training-massive-models-on-cerebras-systems-using-weight-streaming/

4 Emad Barsoum, "Supporting PyTorch on the Cerebras Wafer-Scale Engine", Cerebras Systems, 2022 https://www.cerebras.net/blog/supporting-pytorch-on-the-cerebras-wafer-scale-engine/

5 Antonio Kim, Ryan Reece, "Getting Started with PyTorch BERT Models on the Cerebras CS-2 System", Cerebras Systems, 2022 https://www.cerebras.net/blog/getting-started-with-pytorch-bert-models-on-the-cerebras-cs-2-system/

6 Kim Branson, Meredith Trotter, Stephen Young, Natalia Vassilieva, "GlaxoSmithKline and Cerebras are Advancing the State of the Art in AI for Drug Discovery", GlaxoSmithKline and Cerebras Systems, 2022 https://www.cerebras.net/blog/glaxosmithkline-and-cerebras-are-advancing-the-state-of-the-art-in-ai-for-drug-discovery/

7 Sanjana Mallya, Cindy Orozco Bohorquez, Natalia Vassilieva, "Accelerating NLP Model Training and Enabling Higher Accuracy for Financial Services Applications", Cerebras Systems, 2022 https://f.hubspotusercontent30.net/hubfs/8968533/Cerebras-Financial-Institution-NLP-case-study.pdf