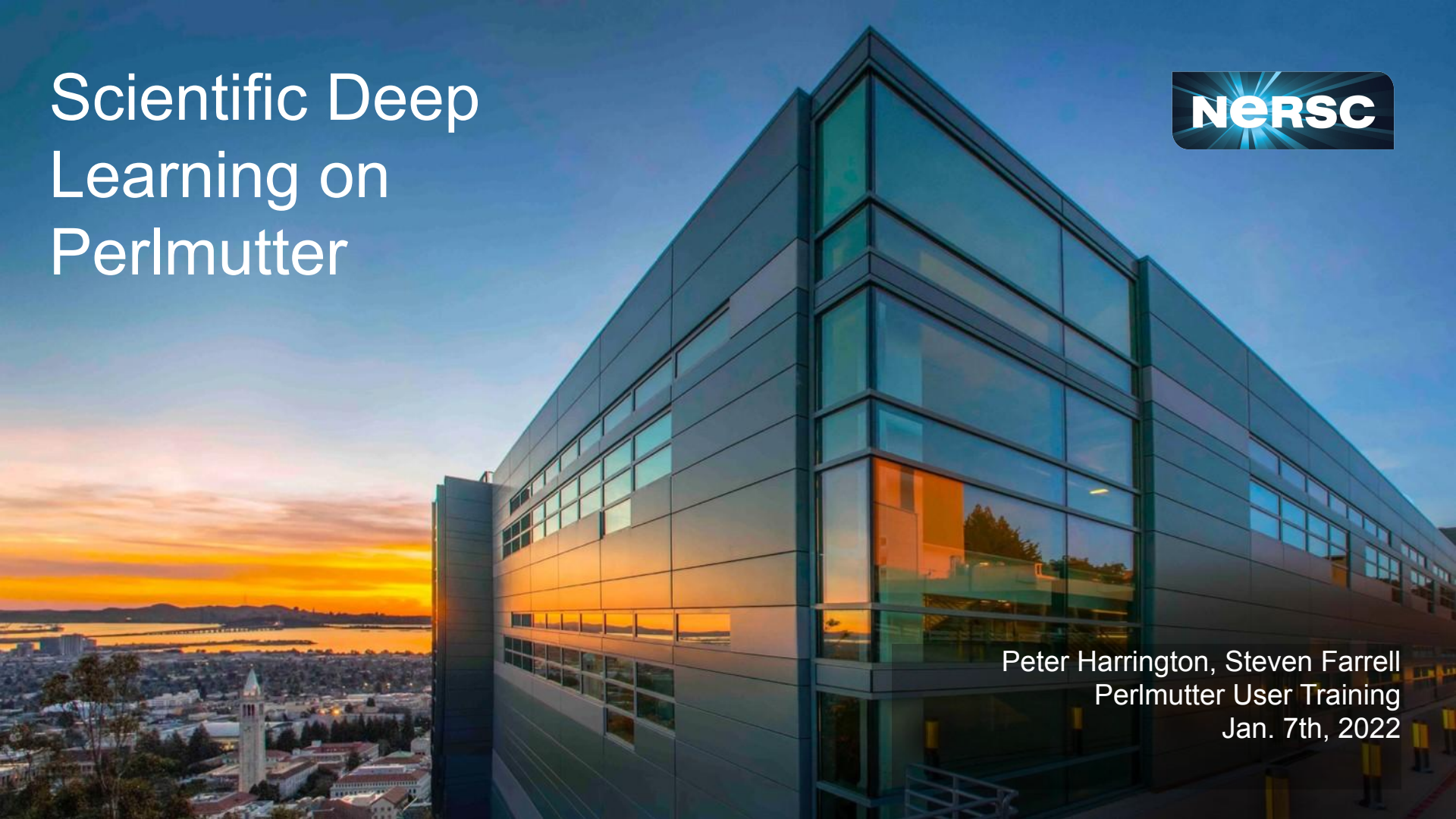
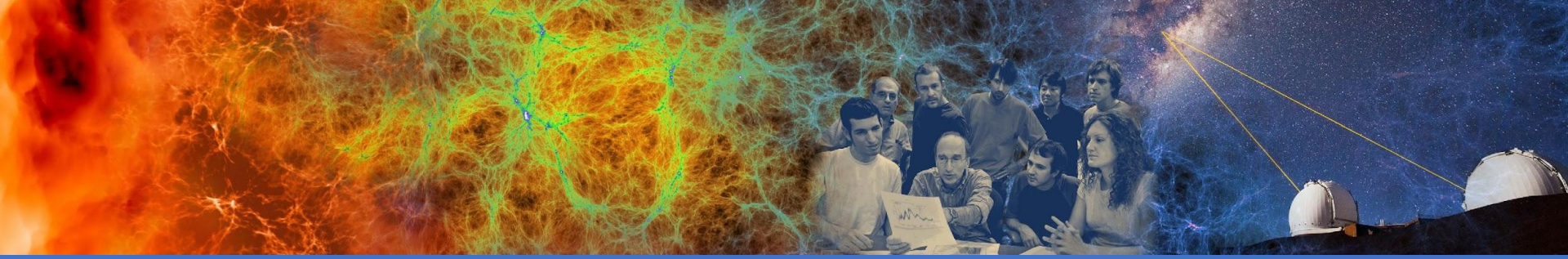
Scientific Deep Learning on Perlmutter



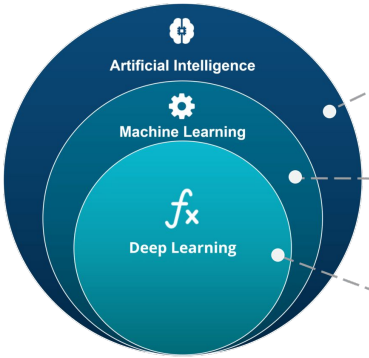
Peter Harrington, Steven Farrell
Perlmutter User Training
Jan. 7th, 2022



Outline

- Deep learning for science @ NERSC
- Deep learning stack on Perlmutter
- How to use DL frameworks on Perlmutter: performance and optimization
- Additional tools & hands-on activity

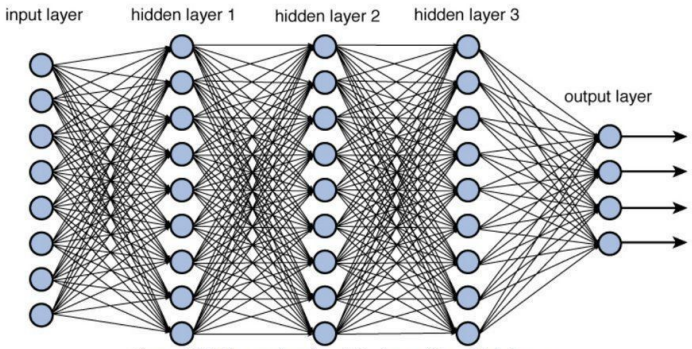
Deep Learning is powered by deep neural networks

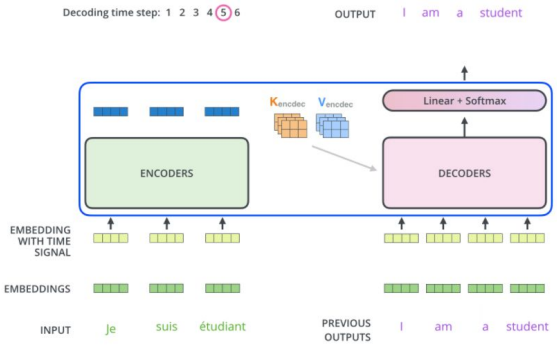


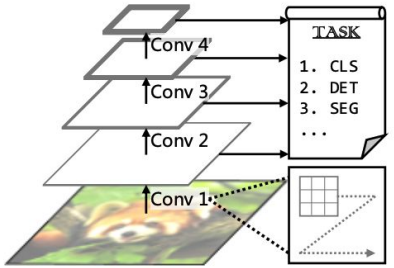
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

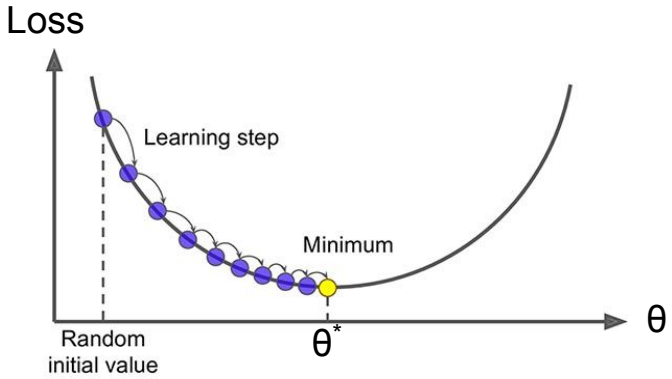




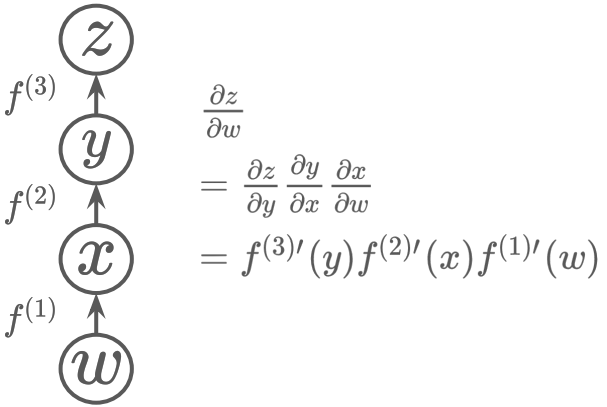


How to train your neural network

- *Loss function:* Compare model prediction to training dataset
- *Gradient Descent:* Dominant method to optimize network parameters to minimize the loss function



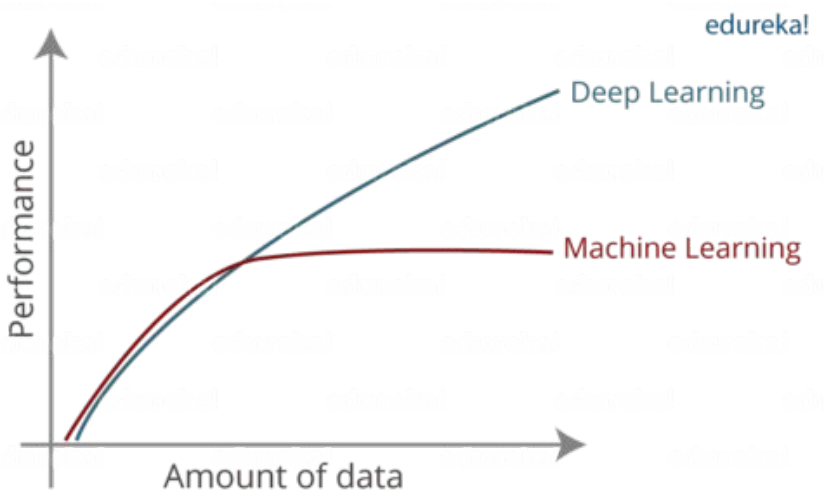
- *Backpropagation:* Propagate updates to parameters through network using chain-rule of calculus



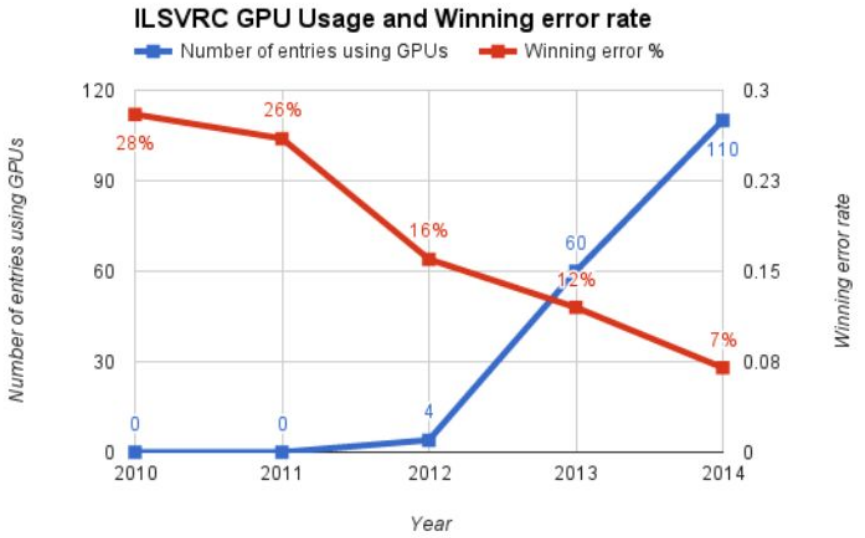
$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} = f^{(3)\prime}(y)f^{(2)\prime}(x)f^{(1)\prime}(w)$$

Why is deep learning so successful?

1) Data: large curated datasets

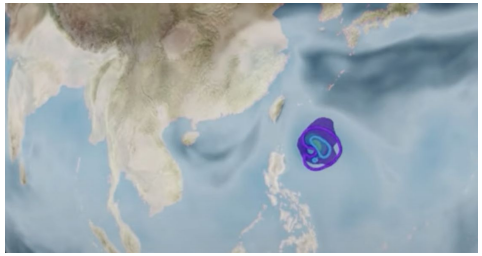


2) GPUs: linear algebra accelerators



3) Algorithmic advances: optimizers, regularization, normalization … etc.

Deep Learning is transforming science

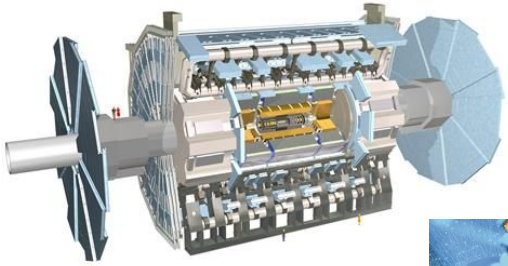


It can enhance various scientific workflows

- Analysis of large datasets
- Accelerating expensive simulations

Adoption is on the rise in the science communities

- Rapid growth in ML+science conferences
- Recognition of AI achievements:
  2018 Turing Award; 2018, 2020 Gordon Bell prizes
- HPC centers awarding allocations for AI,
  optimizing next-gen systems for AI

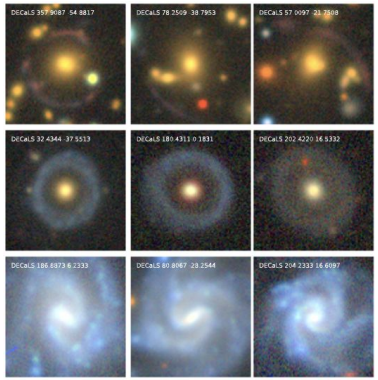


The DOE is investing heavily in AI for science

- Funding calls from ASCR (and other funding agencies), ECP ExaLearn
- Popular, enthusiastic AI4Science town hall series, *300 page report*

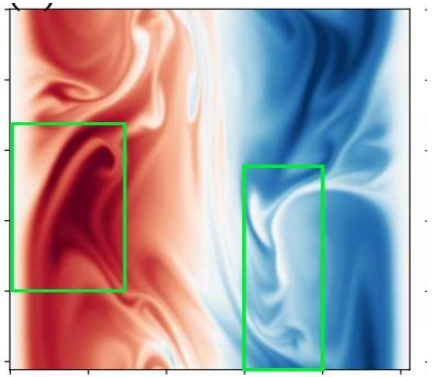
Accelerating science with deep learning

Extract



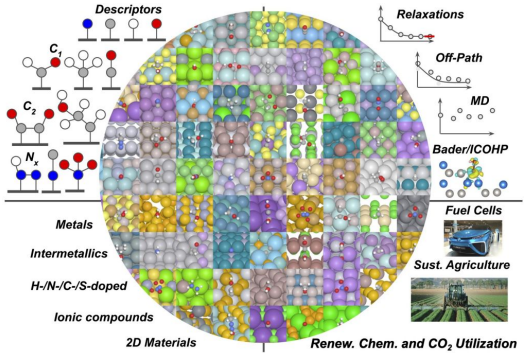
Hayat et al. 2021 arXiv:2012.13083

Enhance



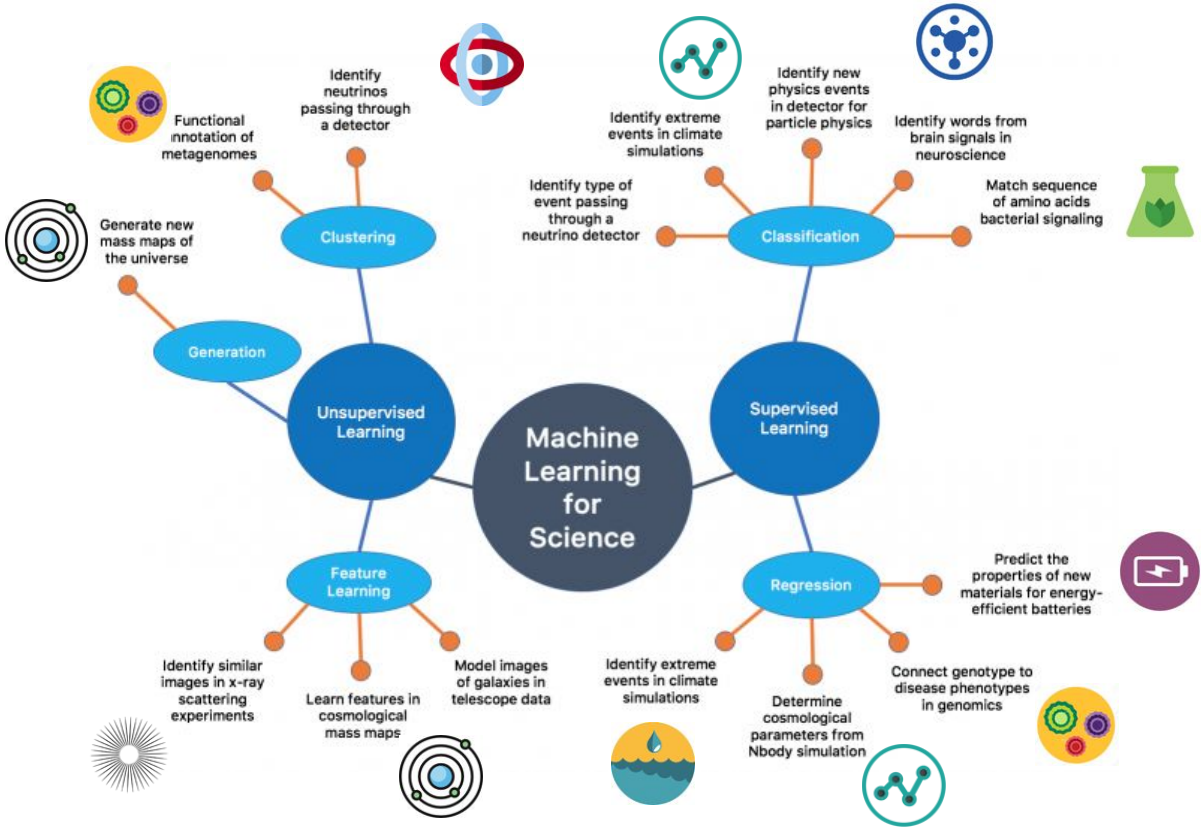
Pathak et al. 2020 arXiv:2010.00072

Explore



Chanussot et al. 2021 arXiv:2010.09990

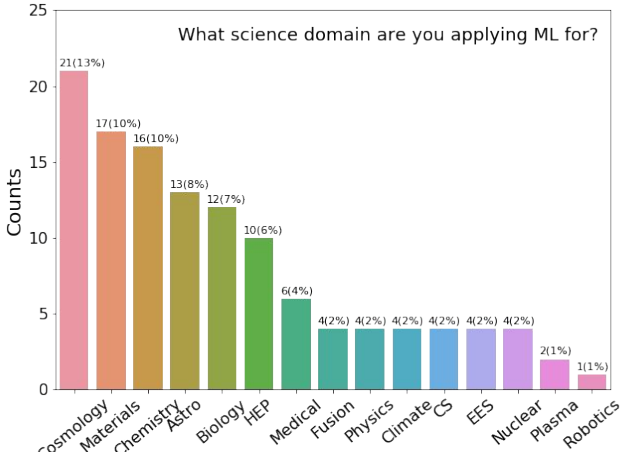
Scientific ML: endless possibilities!

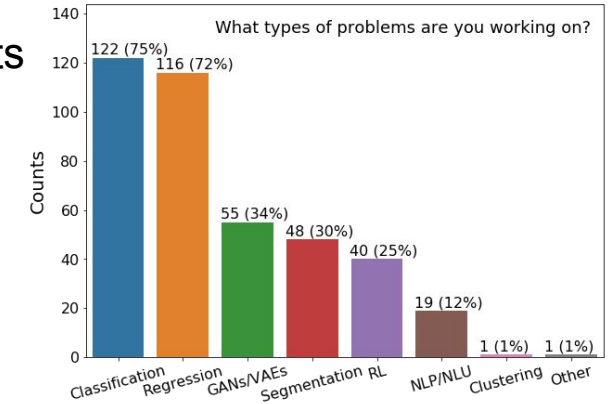


ML@NERSC survey and results

We track Machine Learning trends through our ML@NERSC survey on a 2-year cadence (2018, 2020, **2022**)

- Targets scientific communities which (potentially) use HPC resources (NERSC and non-NERSC users)
- Tracks trends in types of types of problems, workload, model architectures, framework, scaling strategies, hardware and software needs, etc.
- Tracks current use cases of NERSC ML stack and attempts to identify areas for user experience and performance improvements
- Attempts to anticipate future workloads' needs





Deep Learning workloads

Training

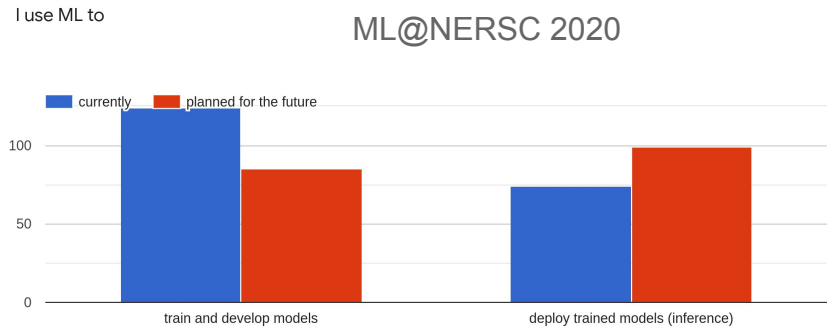
- Iterative, interactive R&D
- Compute, network, and data intensive at large scale

Model selection / development, hyper-parameter optimization

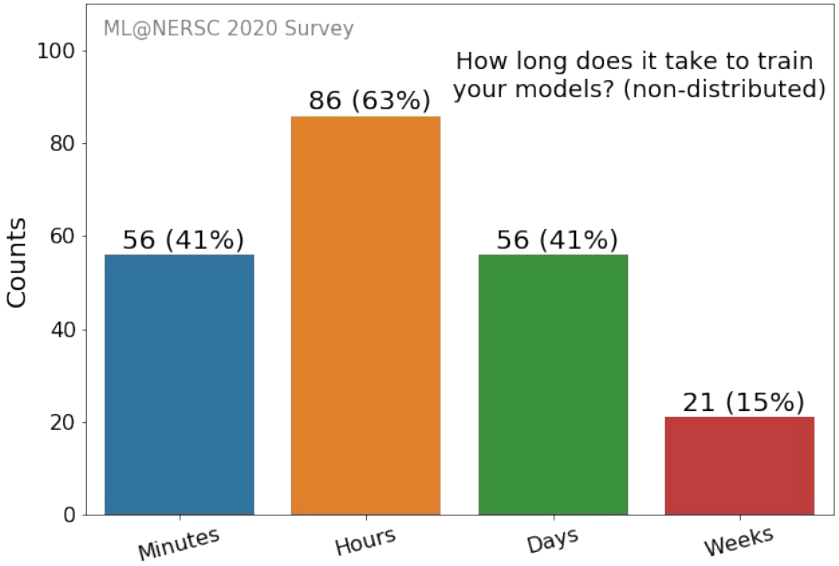
- Massive compute resources
- Searching the model space for the best possible model
- Many parallel training applications

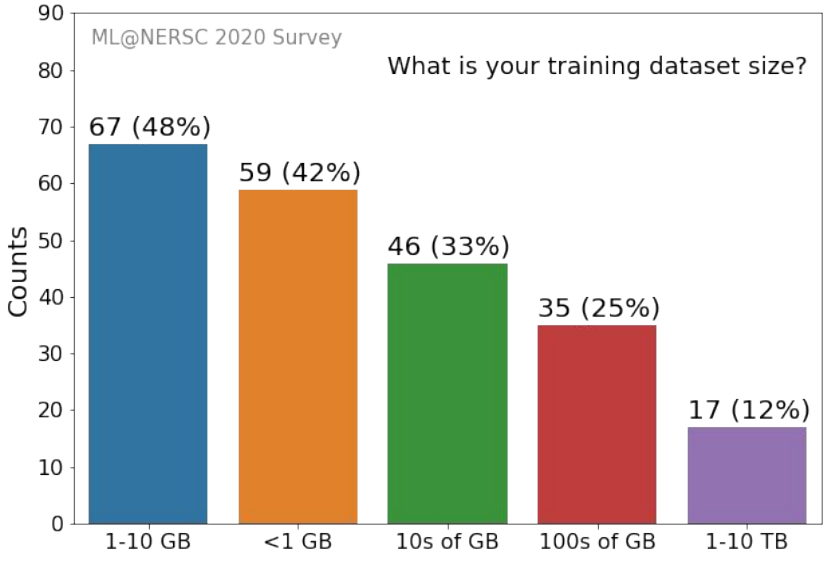
Inference

- Production analytics
- High-throughput
- Offline analytics
- Realtime processing



The need for scale in deep learning R&D

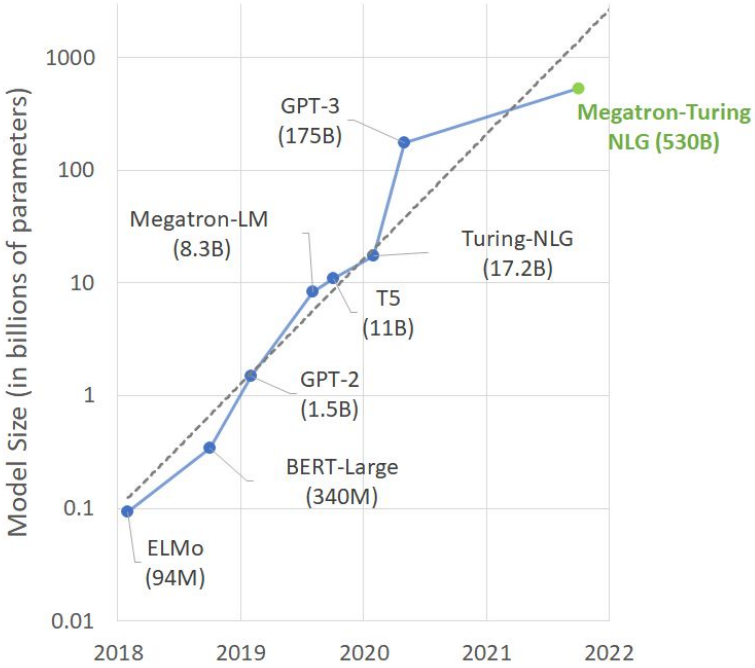




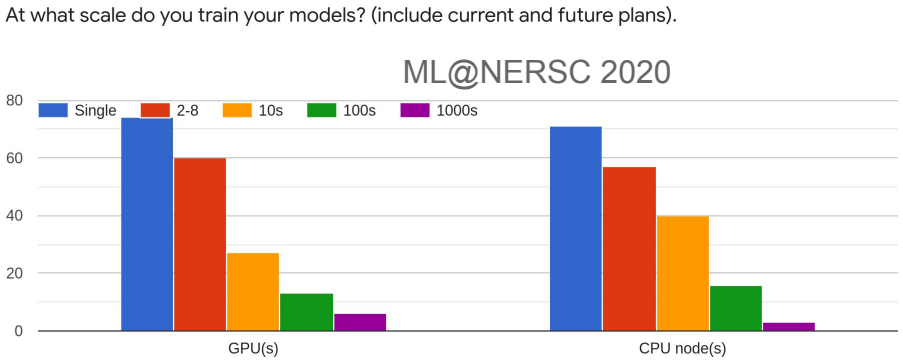
- Rapid prototyping/model evaluation (faster time to solution)
- Problem scale

- Volume of scientific datasets can be large
- Scientific datasets can be complex (multivariate, high dimensional)

More complex tasks, bigger models, more compute

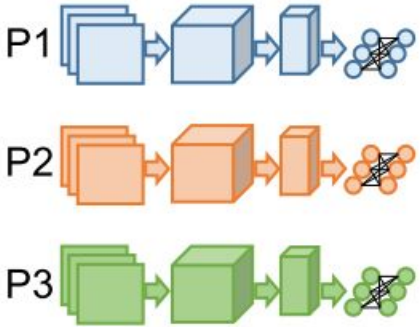


Credit: NVIDIA



Models get bigger and more compute intensive as they tackle more complex tasks

Deep Learning parallelization strategies

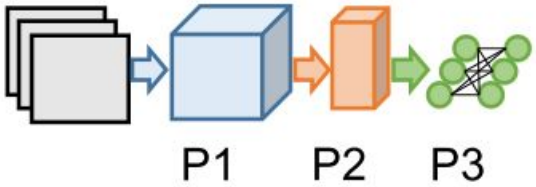


Data Parallelism

Distribute input samples.

Model Parallelism

Distribute network structure (layers).

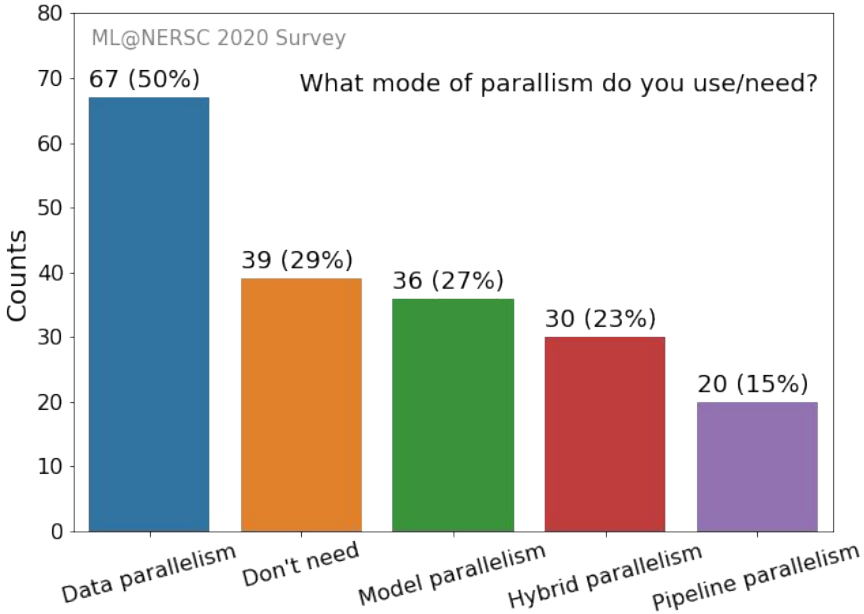


Layer Pipelining

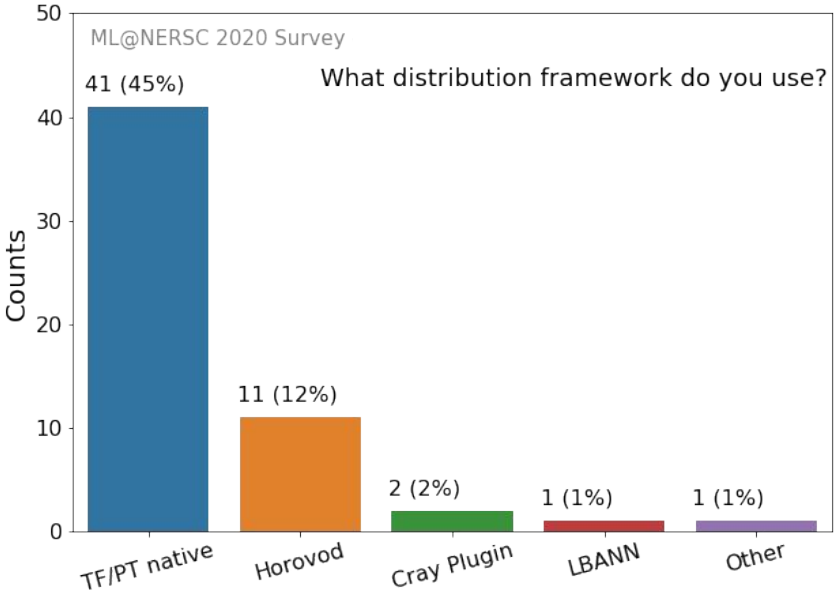
Partition by layer.

Fig. credit: Ben-Nun and Hoefler arXiv:1802.09941

Deep Learning parallelization strategies



Data parallelism is the most common strategy in practice, especially for inter-node scaling.

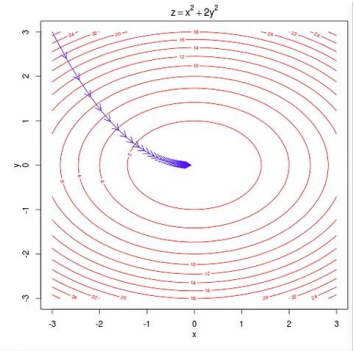


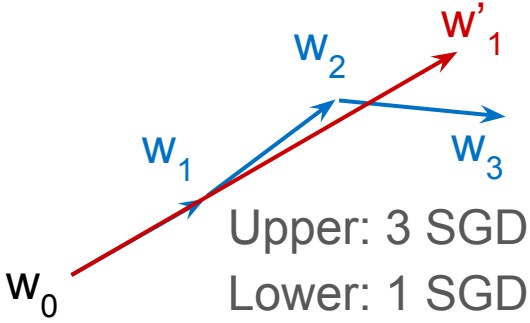
TensorFlow and PyTorch support data and intra-node pipeline parallelism natively. Horovod is the leading non-native distribution framework. All support MPI and/or NCCL backends.

Data-parallel training considerations

Weak scaling: converge faster by taking fewer, bigger, faster steps

- i.e., more GPUs, larger batch sizes, larger learning rates





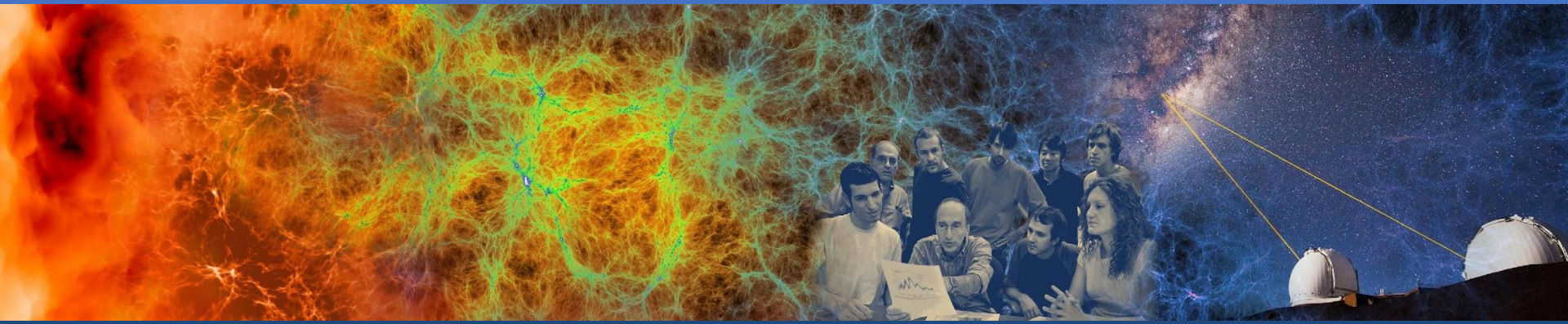
Upper: 3 SGD steps w. learning-rate = η

Lower: 1 SGD step w. learning-rate = 3 * η

Caveat: for stability & convergence, requires tuning

- Warm-up+scale learning rate, adaptive optimizers, etc
- See our SC21 “Deep Learning at Scale” tutorial for more tips

Deep Learning on Perlmutter: Software stack and best practices



NERSC

BERKELEY LAB

U.S. DEPARTMENT OF ENERGY | Office of Science

Deep Learning on Perlmutter

Our goal is to provide a functional, performant system for scientific DL workloads

- Hardware, software, tools, and methods
- For a highly diverse set of scientific domains and application types

How do we do that?

- by deploying optimized software in partnership with vendors
- by testing and evaluating system performance through benchmarking
- by helping users through consulting tickets
- documentation and training for best practices (like today)
- through our science engagements and own research projects

Perlmutter: next-gen system for science, *awesome for deep learning!*

Cray Shasta system with 3-4x capability of Cori
Phase 1: 12 GPU cabinets with 4x NVIDIA Ampere GPU nodes. Total >6000 GPUs!
35 PB of All-Flash storage
Phase 2 (mid-2021): 12 AMD CPU-only cabinets
Cray Slingshot high performance network

Need for Speed: Researchers Switch on World's Fastest AI Supercomputer

Six thousand NVIDIA A100 GPUs deliver four exaflops of mixed-precision performance to help NERSC advance science.

May 27, 2021 by DION HARRIS

https://blogs.nvidia.com/blog/2021/05/27/nersc-perlmutter-ai-supercomputer/

Perlmutter deep learning software stack overview

General strategy:

- Provide functional, performant installations of the most popular frameworks and libraries
- Enable flexibility for users to customize and deploy their own solutions

Frameworks:

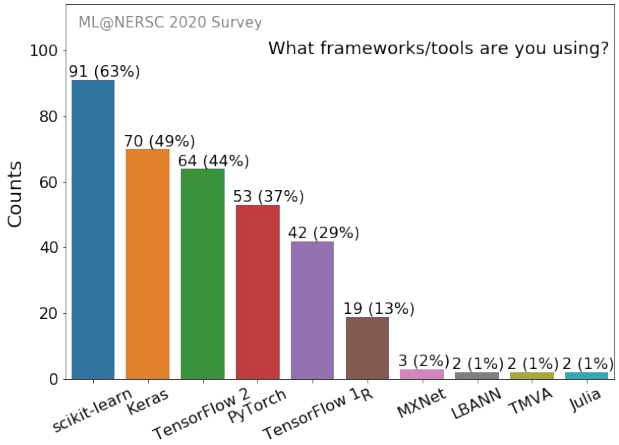
TensorFlow Keras PyTorch

Distributed training libraries:

- Horovod
- PyTorch distributed

Productive tools and services:

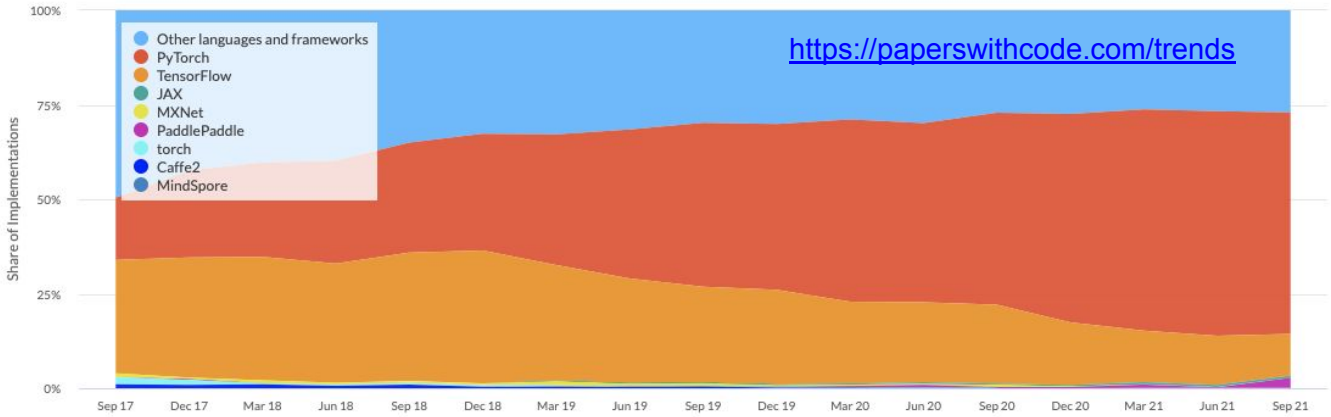
- Jupyter, Shifter



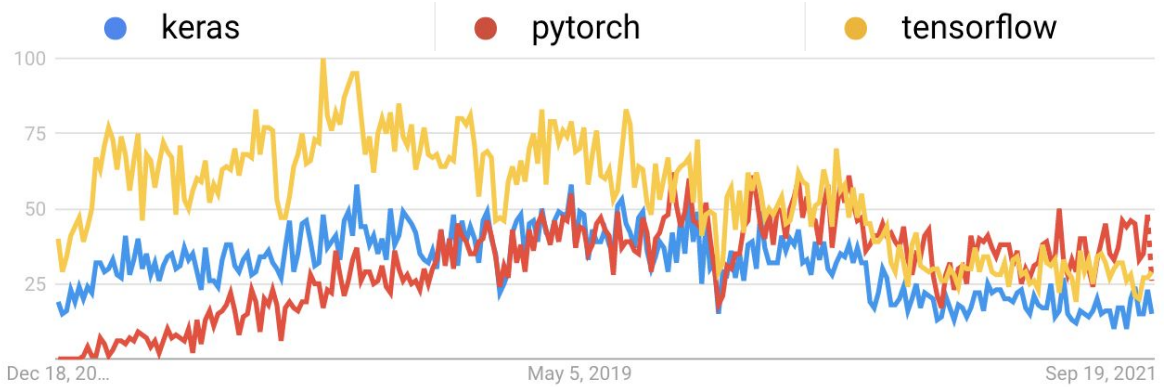
https://docs.nersc.gov/machinelearning/

Frameworks trends

Repositories on PapersWithCode: (research works with published code)



Google Search trends:



How to use the Perlmutter DL software stack

We have modules you can load which contain python and DL libraries:

```
module load tensorflow/2.6.0
module load pytorch/1.10.0
```

Check which software versions are available with:

```
module avail pytorch
```

You can install your own packages on top to customize:

```
pip install --user MY-PACKAGE
```

Or, clone a conda environment from our modules:

```
conda create -n my-env --clone /path/to/module/installation
```

Or, create custom conda environments from scratch:

```
conda create -n my-env MY-PACKAGES
```

More on how to customize your setup can be found in the docs (TensorFlow, PyTorch).

Containerized DL: using Shifter on Perlmutter

NERSC currently supports containers with Perlmutter via Shifter

- Easy, performant: Top500 HPL number was from a container!

To see images currently available:

```
shifterimg images | grep pytorch
```

To pull desired docker images onto Perlmutter:

```
shifterimg pull <dockerhub_image_tag>
```

To use interactively:

```
shifter --module gpu --image=nvcr.io/nvidia/pytorch:21.08-py3
```

Use Slurm image shifter options for best performance in batch jobs:

```
#SBATCH --image=nersc/pytorch:1.5.0_v0
srun shifter python my_python_script.py
```

Best Practices for DL + Shifter on Perlmutter

NVIDIA provides containers optimized for deep learning on GPUs with

- Pytorch or TensorFlow+Horovod
- Optimized drivers, CUDA, NCCL, cuDNN, etc
- Many different versions available

We also provide images based on NVIDIA's, which have a few useful extras

You can also build your own custom containers (easy to build on top of NVIDIA's)

Notes

- Customization: from inside the container, do `pip install --user MY-PACKAGE` (make sure to set `$PYTHONUSERBASE` to a custom path for the desired container)

- NVIDIA NGC containers use OpenMPI, which requires specific options if you require MPI. Instructions: https://docs.nersc.gov/development/shifter/gpus/#shifter-mpich-module

General guidelines for deep learning at NERSC

NERSC documentation: https://docs.nersc.gov/machinelearning/

Use our provided modules/containers if appropriate

- They have the recommended builds and libraries tested for functionality and performance
- We can track usage which informs our software support strategy

For developing and testing your ML workflows

- Use interactive QOS or Jupyter for on-demand compute resources
- Visualize your models and results with TensorBoard or Weights & Biases

For performance tuning

- Next section of these slides

On Perlmutter, refer to these pages for known issues:

- https://docs.nersc.gov/current/
- https://docs.nersc.gov/machinelearning/known_issues/

If you need additional help, open a ticket: https://help.nersc.gov/

Guidelines - TensorFlow distributed training

TensorFlow at NERSC docs:
https://docs.nersc.gov/machinelearning/tensorflow/

For distributed training, we recommend using Horovod

- Easy to use and launch with SLURM
- Can use MPI and NCCL as appropriate
- Horovod examples: https://github.com/horovod/horovod/tree/master/examples

TensorFlow has some nice built-in profiling capabilities

- TF profiler in TF 2: https://www.tensorflow.org/guide/profiler
- Keras TensorBoardCallback in TF 1

Guidelines - PyTorch distributed training

PyTorch at NERSC docs:
https://docs.nersc.gov/machinelearning/pytorch/

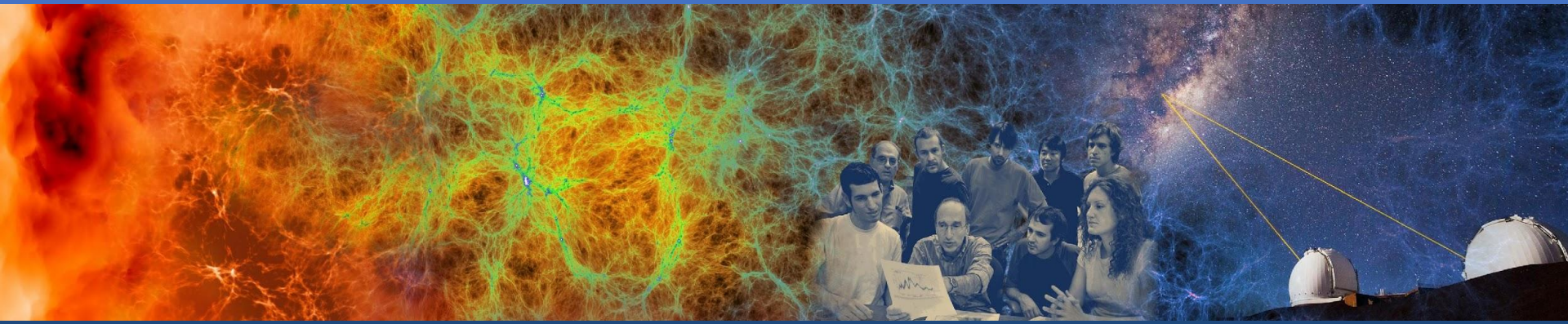
For distributed training, use PyTorch's DistributedDataParallel

- Simple model wrapper, native to Pytorch
- Works on CPU and GPU
- Highly optimized for distributed GPU training
- Docs: https://pytorch.org/tutorials/beginner/dist_overview.html

Distributed backends

- On Perlmutter, use the NCCL backend for optimized GPU communication

Deep Learning on Perlmutter: Performance & benchmarking



Deep Learning Performance on Perlmutter

Good performance for DL workloads on Perlmutter is essential

- for fast iteration in R&D for individual scientists
- for production workloads with computational constraints (e.g. realtime)
- to optimize overall system throughput for *all NERSC users*

This is true regardless of your type of workload

- Single GPU vs. 1000s of GPUs
- Jupyter notebooks or batch scripts

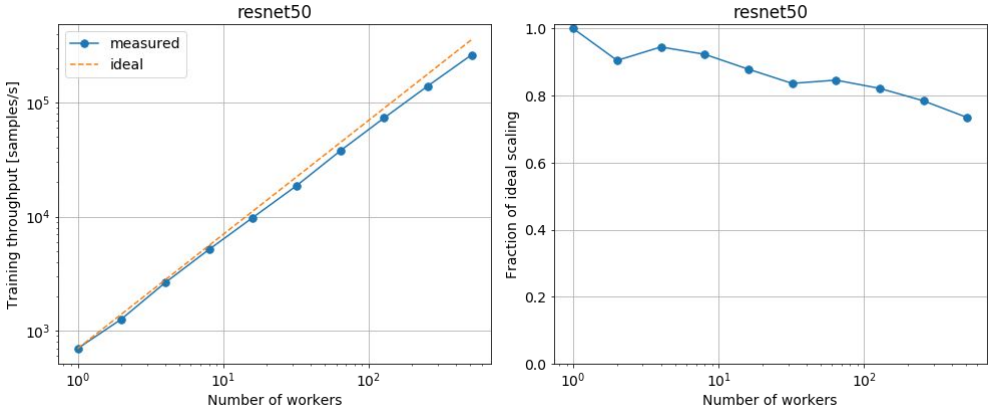
Ideally, the DL frameworks/tools would give both maximal flexibility, ease of use, and performance out-of-the-box

- Not always the case; there can be performance limitations/pitfalls
- It is always useful to spend a little time evaluating the performance of your workload; you could have a lot to gain

How do we evaluate system performance?

Running various tests and benchmarks

- NCCL tests
- Torchvision benchmarks



MLPerf HPC - DL benchmarking for HPC science from MLCommons

- Measures time-to-train models as well as system throughput (models/min)
- The v1.0 submission round had 3 scientific applications:
  - DeepCAM - climate segmentation
  - CosmoFlow - 3D CNN regression
  - OpenCatalyst - GNN predicting energy+forces in atomic system
- We submitted highly competitive results for v1.0 with Perlmutter Phase 1
  - Leading time-to-train result for OpenCatalyst, sub-leading results for CosmoFlow+DeepCAM
  - Largest scale GPU throughput measurement (5120 GPUs)
  - See the full results here: https://mlcommons.org/en/training-hpc-10/

What can cause performance problems for DL?

At the single GPU level

- Spending too much time in (single-threaded) Python code
  - Keep as much of the work as possible on the GPU and/or in numerical libraries.
- Poorly-performing input data pipelines
  - probably the most common source of DL performance problems
  - relatively straightforward to diagnose (e.g. low GPU utilization), sometimes easy to fix
- Unoptimized GPU kernels

At the multi-GPU and multi-node levels

- Network communication bottlenecks
  - Poorly configured communication libraries - can be easy to fix
  - Poorly optimized communication patterns - may be able to tweak library settings
- Load imbalance for irregular-sized scientific data samples
- Parallel file system
  - DL random read patterns are not very friendly to large parallel filesystems like Lustre

How can you diagnose performance problems?

Start simple, e.g. check GPU utilization

- Use nvidia-smi, gpustat, or another monitoring tool like Weights & Biases

Example using nvidia-smi in your sbatch script

```
# Run nvidia-smi in the background, log to CSV
nvidia-smi -l 1 \
    --query-gpu=timestamp,name,index,utilization.gpu,memory.used \
    --format=csv > nvsmi.csv &
NVSMI_PID=$!

# Run your training
srun python train.py ...

# Terminate nvidia-smi
kill $NVSMI_PID
```

- If utilization is low, you're not making good use of the GPU. Investigate deeper to figure out why

How can you diagnose performance problems?

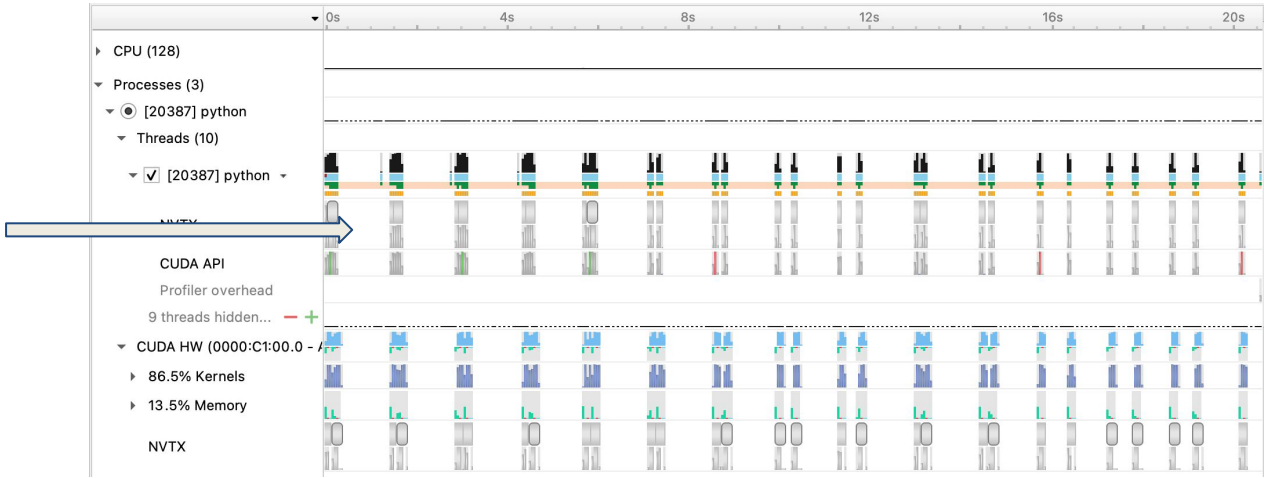
Use a profiler to gain deeper insights

- Nsight-systems is a highly standard NVIDIA tool which can collect and visualize the execution timeline to enable insights
  - E.g., you can see visually how the GPU is waiting for data from CPU
  - Understanding the timeline can take a little bit of practice, though
- Nsight-compute is a powerful tool for collecting kernel-level information about your application
  - E.g., if you want to look at performance of individual kernels, make roofline plots, etc.
  - Challenging to use unless you're a performance expert
- DL-framework-specific tools are getting better all the time, and try to provide high-level recommendations:
  - TensorFlow profiler, PyTorch profiler, NVIDIA's DLProf

Nsight Systems example

Nsight Systems can let you see what your application is doing in a nice interactive timeline view, which can help elucidate performance issues

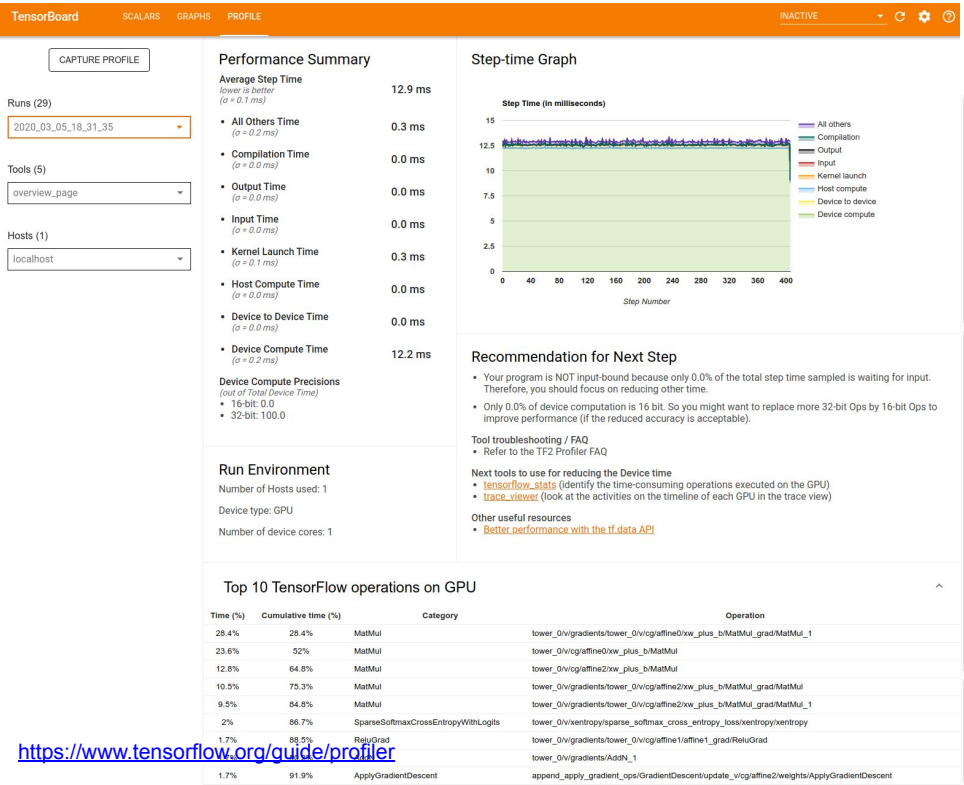
e.g., gaps in cuda execution due to data loading



Refer to our full SC21 Deep Learning at Scale tutorial for a very nice real-world walkthrough: https://github.com/NERSC/sc21-dl-tutorial#profiling-with-nsight-systems

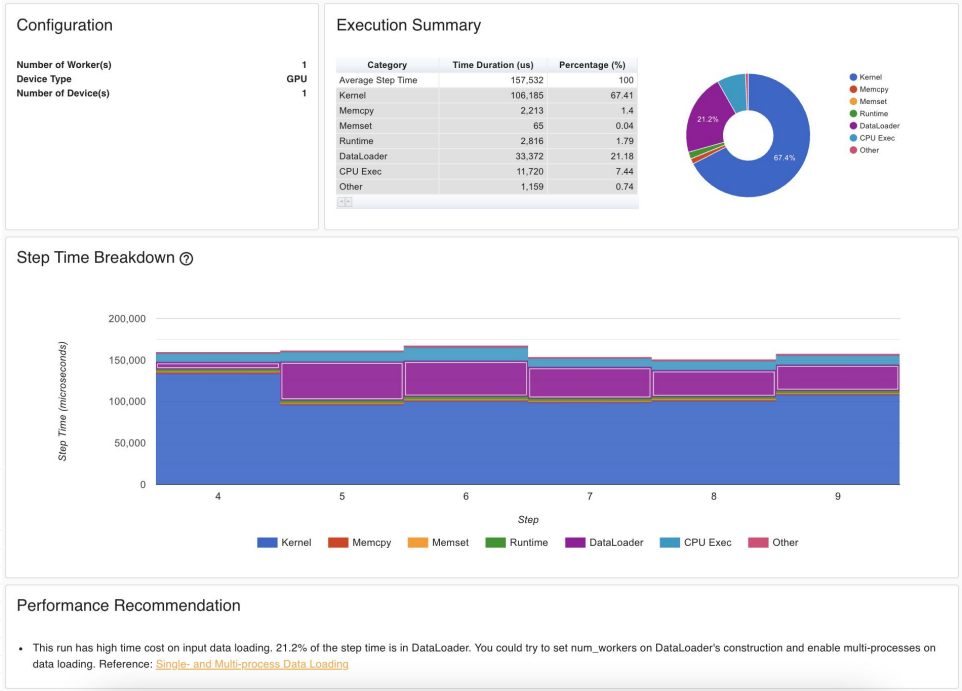
TensorFlow and PyTorch profilers

The framework profilers try to give you nice, actionable, summary information about performance



You can view the results in TensorBoard

https://pytorch.org/blog/introducing-pytorch-profiler-the-new-and-improved-performance-tool/



https://www.tensorflow.org/guide/profiler

Tips for improving performance

Tune your data loading pipeline

- Adjust num_workers, use pin_memory (PyTorch)
- If I/O (from lustre) is a bottleneck, consider staging data onto nodes
  - Use per-process memory, or /tmp (126 GB shared by all workers on node)
  - Larger datasets may require partitioning across nodes to fit
- Consider NVIDIA DALI library for GPU-accelerated data transformations/augmentations, parallel host-to-device streams

Tune single-GPU performance

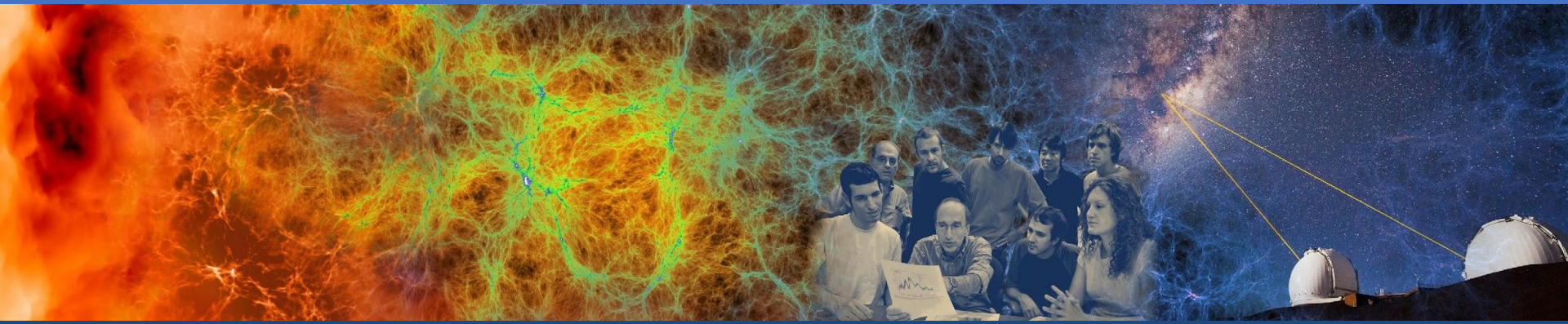
- Try mixed-precision training
- Try JIT compiling your model
- For PyTorch, try Apex fused optimizers

Tune distributed performance

- For a fixed global batch size, scaling to more GPUs trades off efficiency for runtime - tune for your needs
- Tune communication backend settings (e.g. pytorch bucket size)

Refer to our full SC21 tutorial for more: https://github.com/NERSC/sc21-dl-tutorial

Deep Learning on Perlmutter: Workflow tools



NERSC

BERKELEY LAB

U.S. DEPARTMENT OF ENERGY | Office of Science

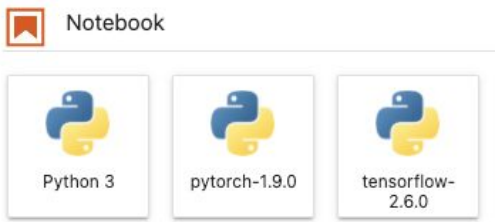
Jupyter for deep learning

JupyterHub service provides a rich, interactive notebook ecosystem on Cori

- Very popular service with hundreds of users
- A favorite way for users to develop ML code

Users can run their deep learning workloads

- on dedicated Perlmutter GPU nodes
- using our pre-installed DL software kernels
- using their own custom kernels



	Shared CPU Node	Shared GPU Node	Exclusive GPU Node	Exclusive Large Memory Node	Configurable GPU	Configurable DGX
Perlmutter	start		start		start	
Cori	start	start		start	start	
Resources	Use a node shared with other users' notebooks but outside the batch queues.		Use your own node within a job allocation using defaults.		Use multiple compute nodes with s	
Use Cases	Visualization and analytics that are not memory intensive and can run on just a few cores.		Visualization, analytics, machine learning that is compute or memory intensive but can be done on a single node.		Multi-node analytics jobs, jobs in reservations, custom project charging, and more.	

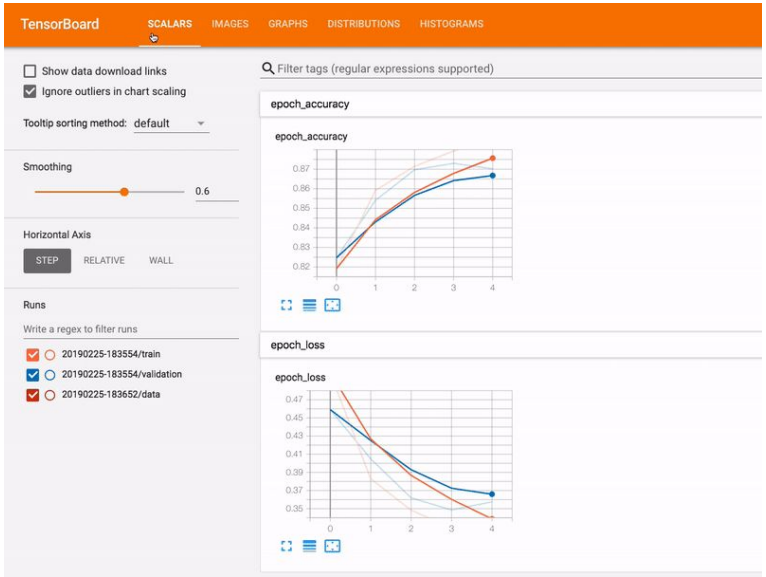
TensorBoard at NERSC

TensorBoard is the most popular tool for visualizing and monitoring DL experiments, widely adopted by TensorFlow and PyTorch communities.
We recommend running TensorBoard in Jupyter using nersc-tensorboard helper module.

```
import nersc_tensorboard_helper
%load_ext tensorboard
%tensorboard --logdir YOURLOGDIR --port 0
```

then get an address to your TensorBoard GUI:

```
nersc_tensorboard_helper.tb_address()
```



Hyper-parameter optimization (HPO) solutions

Model selection/tuning are critical for getting the most out of deep learning

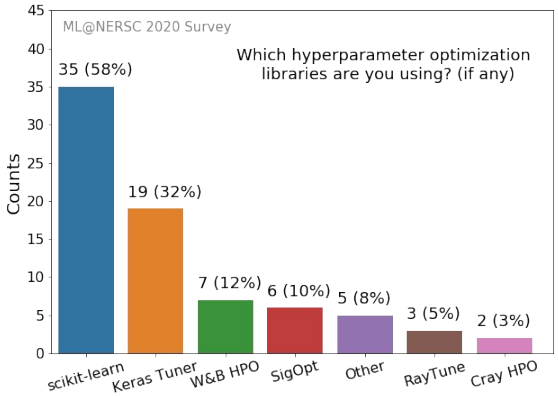
- Many methods and libraries exist for tuning your model hyper-parameters
- Usually very computationally expensive because you need to train many models => Good for large HPC resources

Users can use whatever tools work best for them

- Ask us for help if needed!

RAY tune Weights & Biases

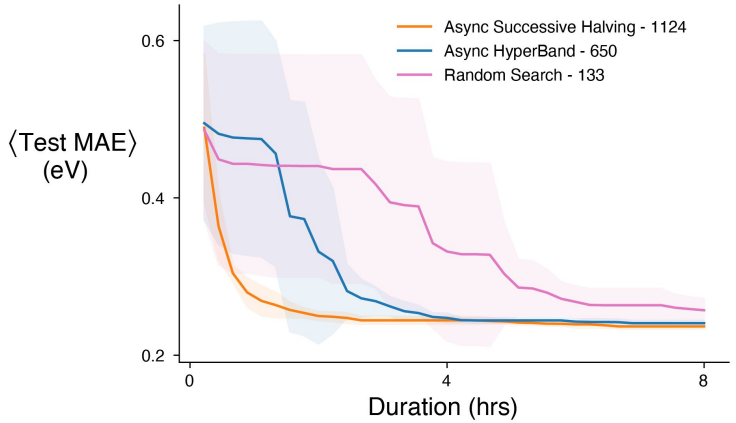
SIGOPT OPTUNA



HPO Example: Ray Tune

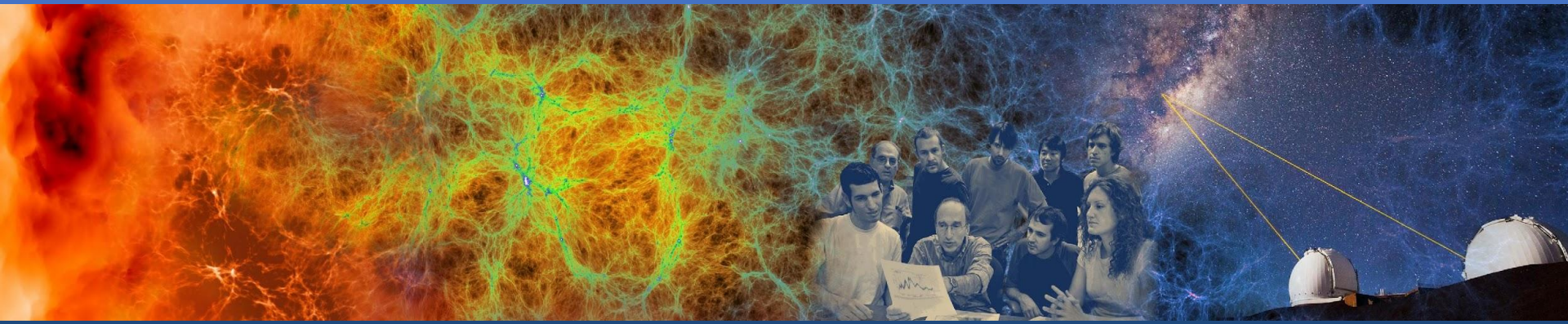
Tune is an open-source Python library for experiment execution and hyperparameter tuning at any scale.

- Supports any ML framework
- Implements state of the art HPO strategies
- Natively integrates with optimization libraries (HyperOpt, BayesianOpt, and Facebook Ax)
- Integrates well with Slurm
- **Handles trials micro scheduling on multi-gpu-node resources** (no GPU binding boilerplate needed)



Example of Multi-node HPO using RayTune used by NESAP team to optimize Graph Neural Network models for catalysis applications (Brandon Wood et al.)

Outreach & additional resources



Training events

The Deep Learning for Science School at Berkeley Lab (https://dl4sci-school.lbl.gov/)

- Comprehensive program with lectures, demos, hands-on sessions, posters
- You can view the full 2019 material (videos, slides, code) online: https://sites.google.com/lbl.gov/dl4sci2019
- 2020 webinar series – recorded talks: https://dl4sci-school.lbl.gov/agenda

The Deep Learning at Scale Tutorial

- Jointly organized with NVIDIA (& Cray in previous years)
- Presented at SC18-21, ECP Annual 2019, ISC19
- Detailed lectures + hands-on material:
  - Distributed training, profiling & optimization on Perlmutter
  - Basis for today's hands-on exercises
- See the full SC21 material here

NERSC Data Seminar Series:
https://github.com/NERSC/data-seminars

Conclusions

Deep learning for science is here and growing

- Powerful capabilities
- Enthusiastic community
- Increasing HPC workloads

Perlmutter has a productive, performant software stack for deep learning

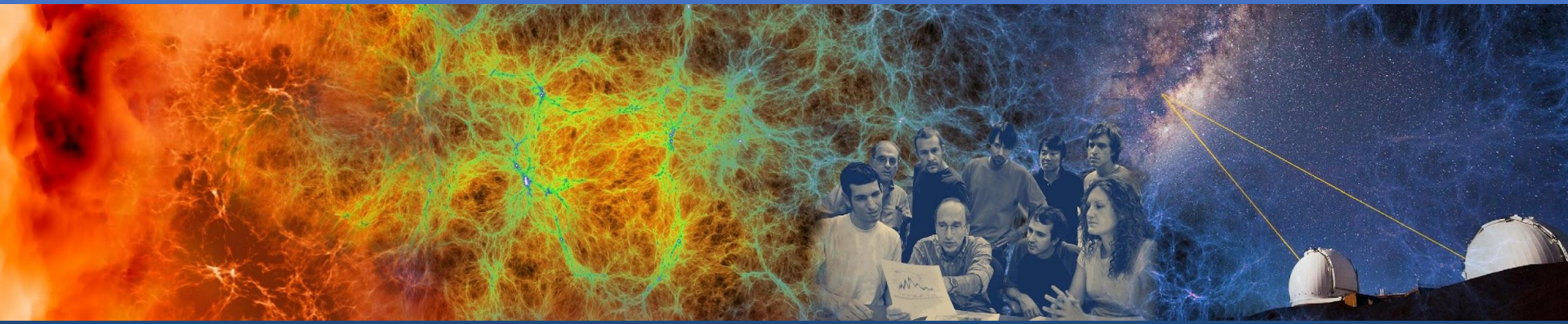
- Optimized frameworks and solutions for small to large scale DL workloads
- Support for productive workflows (Jupyter, HPO)

Join the NERSC Users Slack

Time for questions, then setup for hands-on!

Thank you

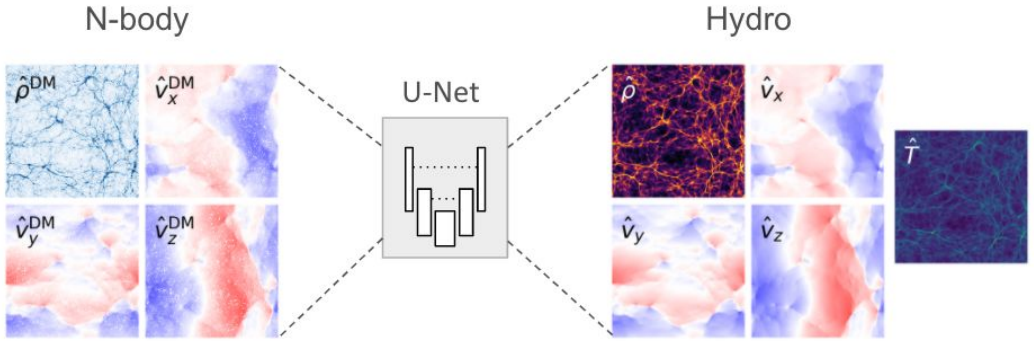
Hands-on exercises: background



Deep learning science example:

“N-body to Hydro” model for cosmology

Adapted from “Fast, high-fidelity Lyman-α forests with convolutional neural networks”, https://arxiv.org/abs/2106.12662

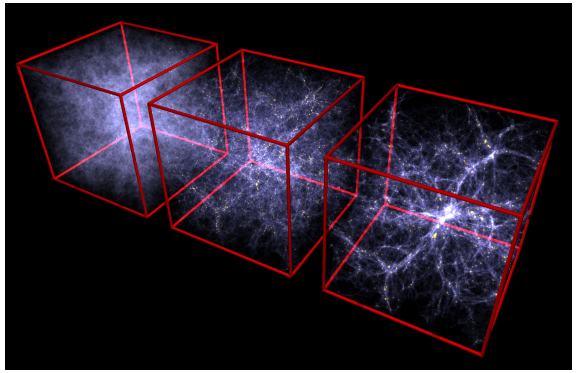


Science problem: cosmological simulations

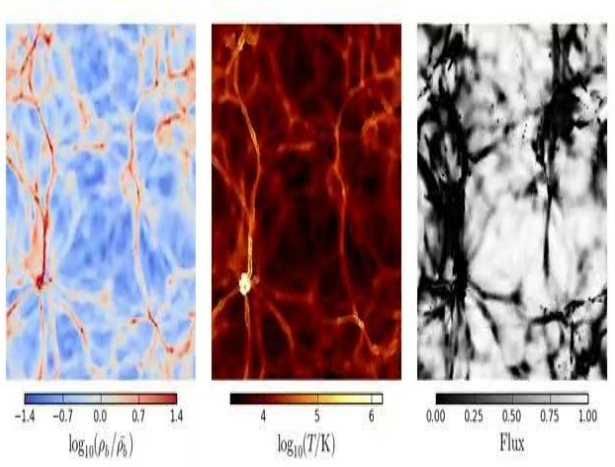
Dark matter is abundant, and essential to structure formation, but can't see it!

Need to model "observables" from visible matter, e.g. luminous gas + galaxies

Large-scale-structure forms mostly via dark matter:



Gas dynamics affected by small-scale hydrodynamic interactions:



Hydrodynamic reconstruction from N-body simulations

Modeling full system computationally demanding

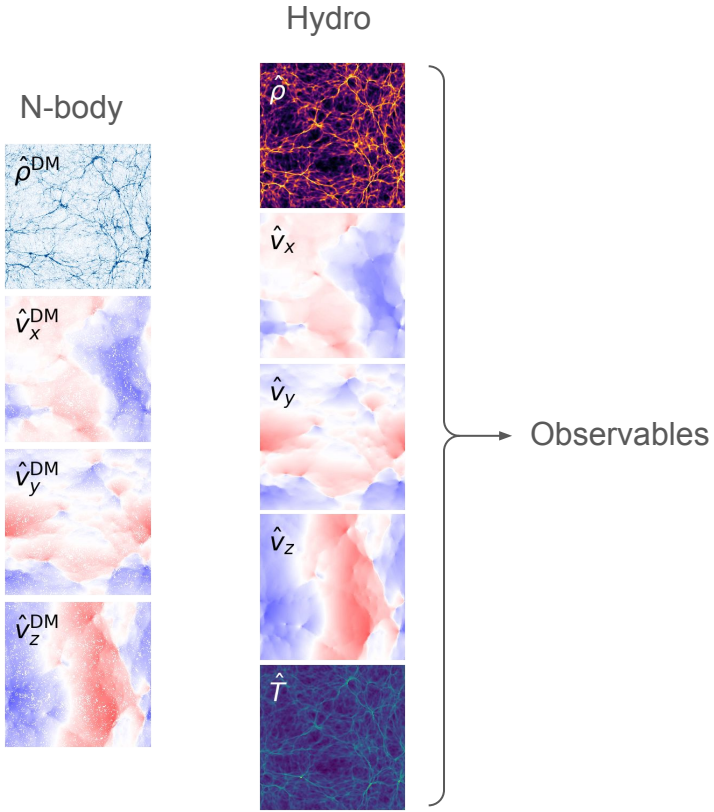
- Multiphysics fluids solver on HPC systems

Simpler: N-body simulations (dark matter only)

- Quick to run, ignore hydrodynamics
- Still capture large-scale structure

Long-standing goal:

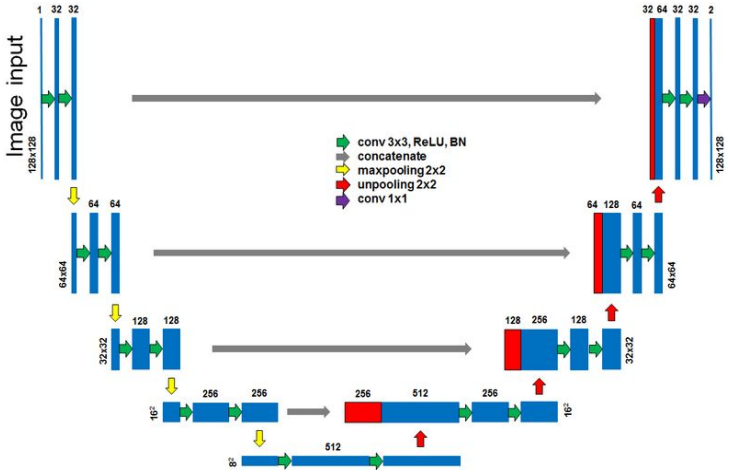
Reconstruct hydrodynamic fields from N-body

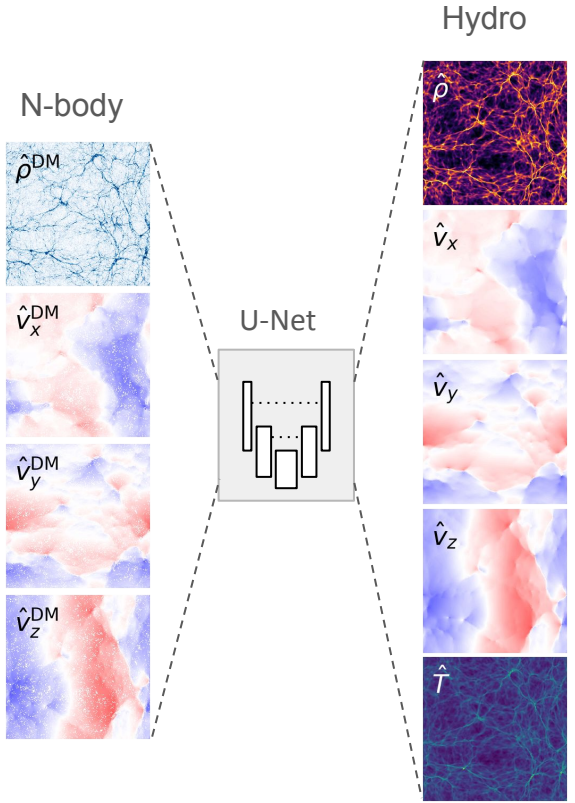


Hydrodynamic reconstruction from N-body simulations

U-Net architecture:

- Convolution layers (down/up-sampling)
- Skip connections across scales

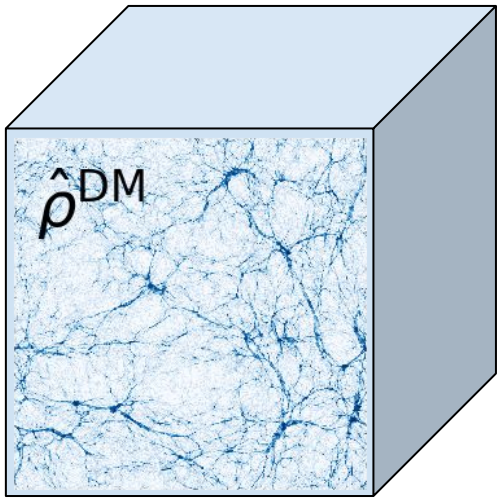


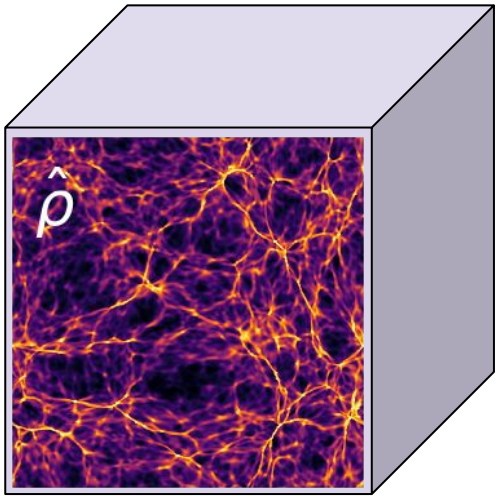


Dataset: N-body + Hydro simulations

Volume of data in simulations presents a challenge:

- 4 input fields, 5 output fields (densities, temperatures, velocities)
- Spatial grid is very large (1024^3 - 2048^3)
  - Train with smaller crops, or sub-volumes

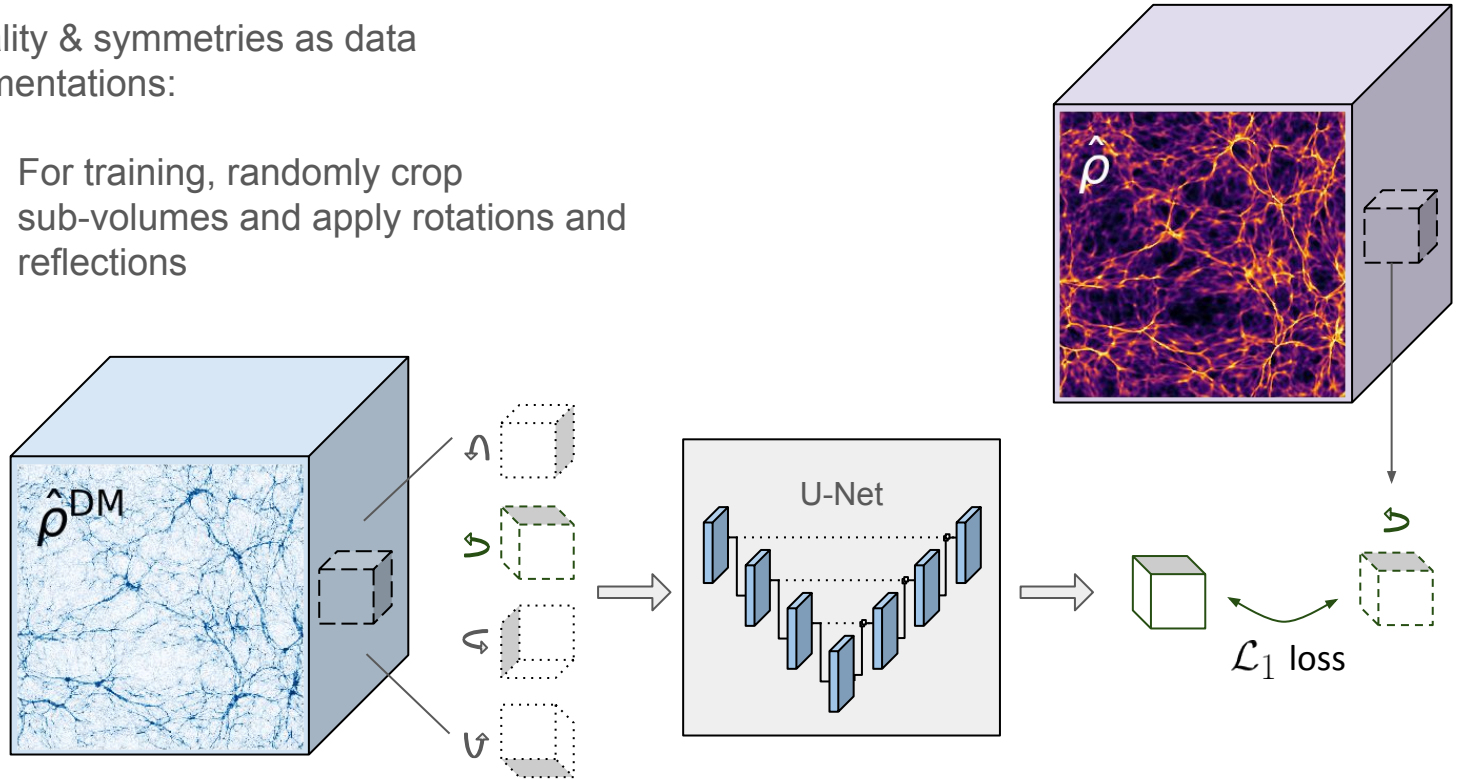




Dataset: N-body + Hydro simulations

Locality & symmetries as data augmentations:

- For training, randomly crop sub-volumes and apply rotations and reflections



Today's code

We will be using PyTorch today

- Pythonic, easy to integrate with other python code
- Good performance and distributed training with support for MPI and NCCL

The example code we'll be using is in the github repository:

https://github.com/NERSC/ml-pm-training-2022

Readme has detailed instructions!

Access to Perlmutter is via NERSC JupyterHub:

https://jupyter.nersc.gov