Optimizing AI/ML Workflows in Python for GPUs

By: Daniel Howard **dhoward@ucar.edu**, Consulting Services Group, CISL & NCAR
Date: August 25th, 2022

In this notebook we analyse the overall workflow of typical machine learning/deep learning projects, emphasizing how to work towards optimal performance on GPUs. We will NOT cover theory of or how to implement AI based projects. We will cover:

1. Background on machine learning research in Earth sciences
2. Setting up Python virtual `conda` environments
    - The RAPIDS AI software suite
    - GPU enabled TensorFlow and PyTorch
3. Enabling tuning and profiling with TensorFlow and PyTorch
4. Profiling with DLProf/TensorBoard and performance optimizations for NVIDIA Tensor Cores

Workshop Etiquette

- Please mute yourself and turn off video during the session.
- Questions may be submitted in the chat and will be answered when appropriate. You may also raise your hand, unmute, and ask questions during Q&A at the end of the presentation.
- By participating, you are agreeing to **UCAR's Code of Conduct**
- Recordings & other material will be archived & shared publicly.
- Feel free to follow up with the GPU workshop team via Slack or submit support requests to **rchelp.ucar.edu**
    - Office Hours: Asynchronous support via **Slack** or schedule a time with an organizer

Start a JupyterHub Session

Head to the **NCAR JupyterHub portal** and **start a JupyterHub session on Casper PBS Login Node** and open the notebook at `15_OptimizeAIML/15_OptimizeAIML.ipynb`. Be sure to clone (if needed) and update/pull the NCAR GPU_workshop directory. You are welcome to use an interactive GPU node for the final few cells of this notebook

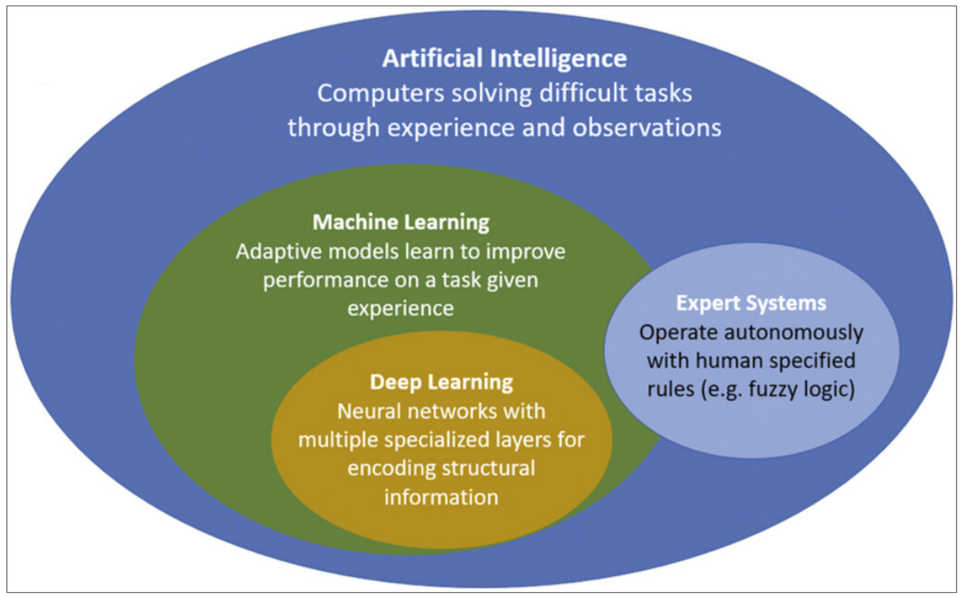
```
# Use the JupyterHub GitHub GUI on the left panel or the below shell commands
git clone git@github.com:NCAR/GPU_workshop.git
git pull
```

Notebook Setup

The `GPU_TYPE=gp100` nodes do not have tensor cores! Thus, the `gpuworkshop` queue is not as useful for this session. Saying as much, please set `GPU_TYPE=v100` and use the `gpudev` or `casper` queue both during the workshop and for independent work. See **Casper queue documentation** for more info.

Machine Learning and Deep Learning?

ML and DL are statistical models that are designed to learn and predict behavior from a large amount of input training data.



The BAMS article "**Outlook for Exploiting Artificial Intelligence in the Earth and Environmental Sciences**" by Boukabara, et al highlights additional applications of AI in the Earth Sciences.

Overiew of an Earth Science AI Workflow - Remote Sensing

Multiple steps are needed to enable AI for Earth Science. **GPUs are critical in the most expensive step, model building and training**, since they perform well with matrix algebra, foundational to ML methods.

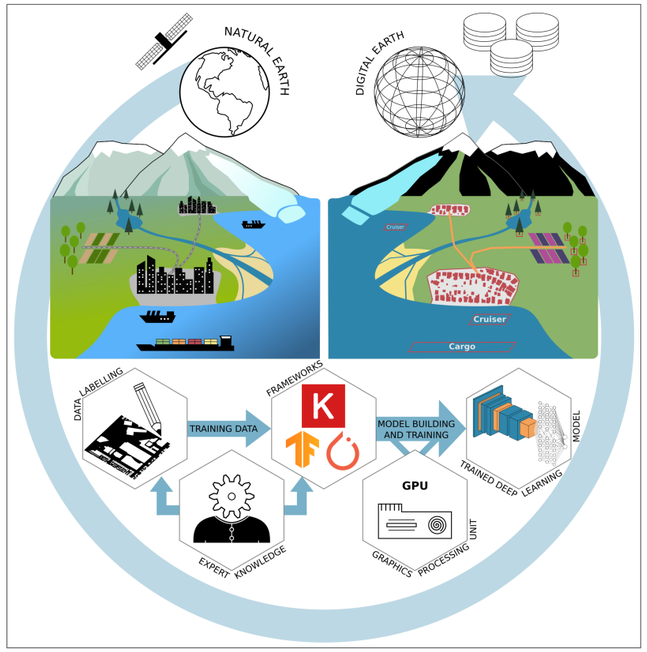
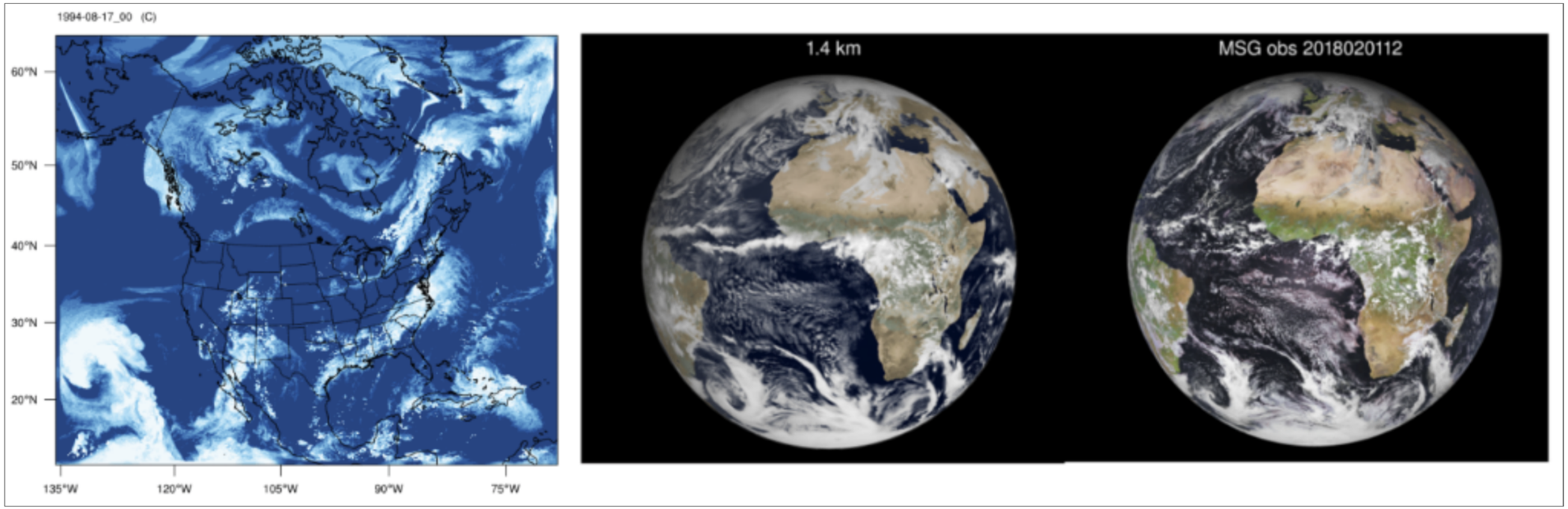


Image: **Object Detection and Image Segmentation with Deep Learning on Earth Observation Data: A Review—Part II: Applications** by Hoeser, et al

Why Use AI for Earth Science?

Earth Science is largely built on physics based theories and dynamical interactions with the biosphere. Today, these models have scaled to enormous sizes, consuming significant computational resources and data storage.



4km global runs of **E3SM** (left) over 100 forecast years uses 120M core-hours and 250 GB/forecast day, or 12 PB. 1km ECMWF runs (right), as **in this article** and by Nils Wedi **keynote at ESMD 2020**.
AI offers an opportunity to reduce computational resources required. Feel free to consult **A Review of Earth Artificial Intelligence** for current "Grand Challenges"

Surrogate Models

Novel ways can be explored to use Earth Science data to reduce required computational resources. A **surrogate model** in machine learning is a **statistical model** designed to more efficiently approximate the output of a physics based model.

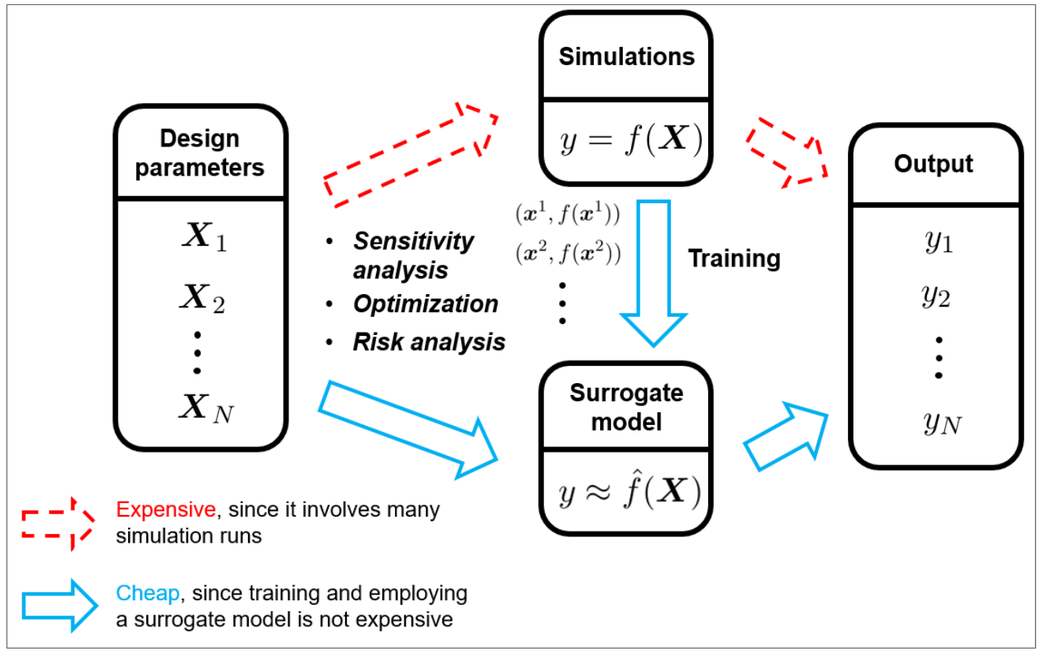
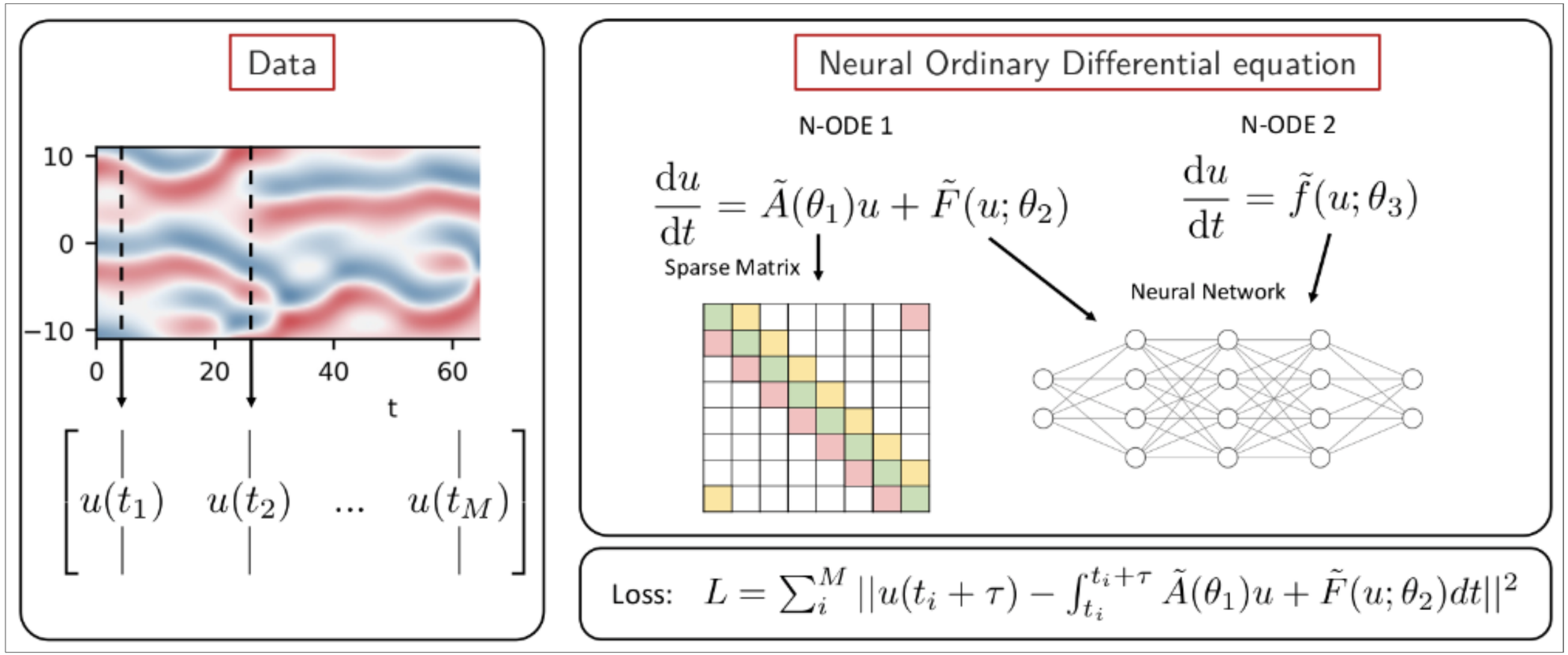


Image: **Introduction to Surrogate Modeling**, Shuai Guo. See "**Learning Nonlinear Dynamical Systems from Data Using Scientific Machine Learning**" by Maulik, ANL.

Neural Ordinary Differential Equations

For example, a stabilized neural ODE can be designed to accurately simulate shocks and chaotic dynamics.



See paper by Linot, et al "**Stabilized Neural Ordinary Differential Equations for Long-Time Forecasting of Dynamical Systems**".

Physics Informed Neural Networks (PINNs)

Other applications to consider are **Physics Informed Neural Networks**. PINNs attempt to embed known physics relationships into the design of a machine learning model. This may include defining the Navier-Stokes conservation laws as conditions to minimize in a ML model's loss function.

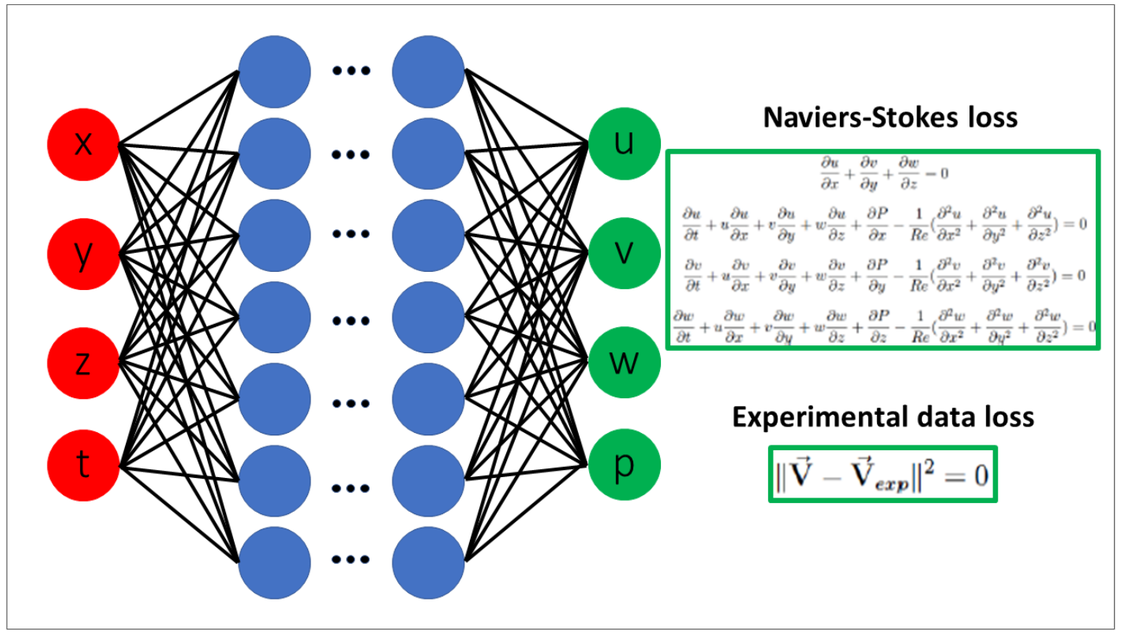


Image: **Wikipedia - Physics Informed Nearal Networks**

Resources for Engaging and Learning AI in Earth Sciences

Feel free to reach out to **rchelp@ucar.edu** if you want assistance recreating environments for any below code examples.

1. OLCF AI 4 Science Fluid Flow Tutorial (**GitHub**) - Uses **MiniWeatherML**
2. OpenHackathons GPU Bootcamp (**GitHub**) - **HPC AI Examples** for PINNs, CFD, and Climate
3. NSF AI Institute for Research on Trustworthy AI in Weather, Climate, and Coastal Oceanography (**AI2ES.org**) - **Education Materials** and **2022 Trust-a-thon GitHub**
4. Argonne ALCF
    - **2021 Simulation, Data, and Learning Workshop for AI** (**GitHub**)- Detailed **DL profiling tutorial notebooks** & **video**
    - **2022 Introduction to AI-driven Science on Supercomputers** (**GitHub**)
5. Data Driven Atmospheric and Water Dynamics Beucler Lab (U. of Lausanne - Switzerland)
    - **Getting Started with Machine Learning** curated resource list
6. **NOAA Workshop on Leveraging Artificial Intelligence in Environmental Sciences** - 4th Workshop free to register, virtual Sept 6-9 2022
7. National Academies - 2022 workshop **Machine Learning and Artificial Intelligence to Advance Earth System Science: Opportunities and Challenges**
8. **Climate Informatics** community - **Conferences** and **Hackathons**
9. Book - **Deep learning for the Earth Sciences -- A comprehensive approach to remote sensing, climate science and geosciences**

10. **climatechange.ai** - Global initiative to catalyze impactful work at the intersection of climate change and machine learning.

How to Manage Python Software for ML and DL Models

The Python ecosystem already provides many robust pre-built software packages and libraries which are continually maintained. **Learning about and employing the Python ecosystem well can simplify the process of using machine learning tools**.
The kernel `GPU_Workshop` already has many useful packages plus others (notably **Horovod** for distributed deep learning) which you are welcome to explore on your own beyond this workshop.
Run the below cell to get a listing of all packages installed in the `GPU_Workshop` conda environment.

In [ ]:
```
!mamba list -p /glade/work/dhoward/conda/envs/GPU_Workshop/
```

Setting Up Conda Environments

Since ensuring compatibility and reproducibility is difficult across python package environments, **you are encouraged to maintain your own personalized `conda` virtual environments**. Nonetheless, NCAR provides a base set of commonly used Python packages via the **NCAR Package Library (NPL)**. NPL does include the faster package management tool `mamba` which uses the same command syntax as `conda`.
If you prefer to install your own and not use `module load conda`, we encourage **Mambaforge**. In general, `mamba` is safe to use compared to `conda`. To update all non-pinned packages in an environment, you can use `mamba update --all`.

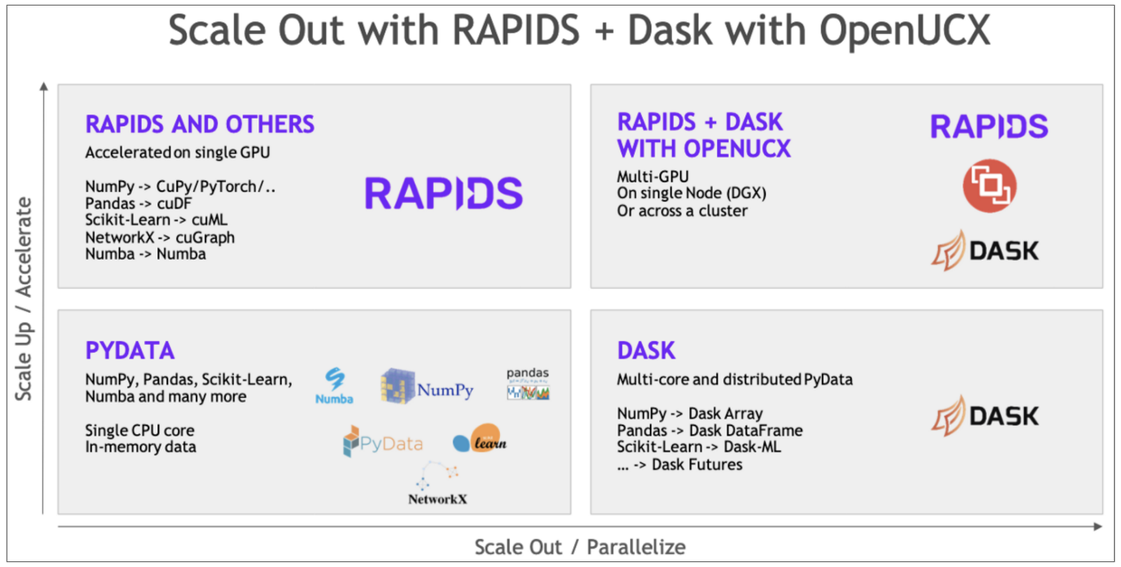
Choosing Conda Channels

To source packages, the channel `conda-forge` is recommended and set as priority on Casper but other channels you may consider are `ncar`, `nvidia`, `rapidsai`, `intel`, `pytorch`, and `anaconda` among others.

- Learn to manage channels **here** using your `$HOME/.condarc` file
- Define pinned packages, ie packages that should stay at a specific version or use a specific build type, via the `/path/to/env/conda-meta/pinned` file

RAPIDS AI Environment

`rapidsai` channel provides **RAPIDS**, an open source, NVIDIA maintained suite for end-to-end data science and analytics pipelines on GPUs. Feel free to explore RAPIDS **Getting Started Notebooks**.



Scale Up with RAPIDS tools and **Scale Out with Dask/UCX or Horovod** tools.

Python Packages and RAPIDS Equivalents

Function	CPU	GPU/RAPIDS
Data handling	pandas	cuDF **
Machine learning	scikit-learn	cuML **
Graph analytics	NetworkX	cuGraph
Geospatial	GeoPandas/SciPy	cuSpatial
Signals	SciPy.signal	cuSignal
Image Processing	scikit-image	cuCIM
Function	**CPU**	**GPU**
Numerical Computing	NumPy	CuPy **
JIT Kernels	Numba	Numba **
Stream Processing	Streamz	cuStreamz

Install RAPIDS environment

Setting flexible channel priority via `conda config --set channel_priority flexible` or in `~/.condarc`, follow install directions **here** or by running:

```
conda create -n rapids-22.08 -c rapidsai -c nvidia -c conda-forge  \
rapids=22.08 python=3.9 cudatoolkit=11.5
```

Installing Customized Python Packages

For more personalized environments, an example process to setup a `conda` environment on Casper is below:

```
module load conda
# Creates environment in /glade/work/$USER/conda-envs/my-env-name or a fully specified path
mamba create -n my-env-name
mamba activate my-env-name

# The Python version installed here will automatically be pinned
# Recommend to not use the latest Python version (3.10+) given compatibility issues
mamba install python=3.9*

# Ensures we get MKL optimized packages to run on Casper's Intel CPUs
mamba install numpy scipy pandas scikit-learn xarray "libblas=*=*mkl"
# Ensures common packages provide MPI support (typically defaults to OpenMPI).
# Useful to pin packages in `/path/to/env/conda-meta/pinned` file.
mamba install mpi4py fftw=*=mpi* h5py=*=mpi* netcdf4=*=mpi*
```

To highlight, adding `<package-name>=<version>=<build-type>` is important to ensure you install the most relevant and performant version for your needs.
For example, `libblas=*=*mkl` guarantees you get the Intel MKL optimized versions of packages that utilize the BLAS library. The `*` is a wildcard for the latest version or other build specifications/hashes.

GPU Enabled Python Packages and Tools

ML libraries **pytorch** and **tensorflow** require additional steps to ensure they are installed with GPU support.

```
mamba install cudatoolkit cudnn cupy nvtx
# Make sure package wheel ID includes *cuda* to verify GPU support
mamba install pytorch=1.12.1=cuda112*
# Don't use tensorflow-gpu package as package solver is inconsistent in condo-forge channel
# TF recommends pip install for latest official version but conda-forge versions also work
mamba install tensorflow=2.9.1=cuda112*

# Enables added profiling capabilities, only available via pip and PyPI or NVIDIA's package index
pip install nvidia-pyindex
pip install nvidia-dlprof nvidia-dlprof-pytorch-nvtx
pip install tensorboard_plugin_profile
```

Each library's documentation linked above has more info about installation options. As of this workshop, TensorFlow guarantees support up to CUDA v11.2 and PyTorch up to CUDA v11.6 so we specified builds with `=cuda112*`. Run `mamba search <package>` to view all available packages given available channels.

TensorFlow recommends installation via `pip` for their offical versions but the community does tend to maintain similar quality releases via `conda-forge`. Combining `pip` with `conda/mamba` installs should be avoided if possible due to greater difficulty in maintaining environments.

Horovod for Distributed Deep Learning

For distributed deep learning with **Horovod** instead of Dask, see below or **Horovod installation documentation** for how to use pip to install Horovod from PyPI on Casper.

```
module load cuda/11.7 gnu/10.1.0
mamba install pip gxx_linux-64 cmake nccl
export HOROVOD_NCCL_HOME=$CONDA_PREFIX
export HOROVOD_CUDA_HOME=$CUDA_HOME
HOROVOD_GPU_OPERATIONS=NCCL pip install horovod[tensorflow,keras,pytorch]
horovodrun --check-build
```

Note the specification of `HOROVOD_GPU_OPERATIONS=NCCL` to use NVIDIA's Collective Communication Library. An `MPI` option is also selectable for CUDA-aware MPI libraries. Find more details about Horovod's GPU tensor operations and **GPU install options here**.
A useful tutorial for Horovod was given as part of the **Argonne Training Program on Extreme-Scale Computing** (ATPESC) - **Data Parallel Deep Learning**

Sharing Package Environments

Once your environment is setup, you can share or give access to your Python virtual environments, which is vitally important to consider towards **enabling reproducible science**.

1. On a shared cluster, share a path to your environment, see `mamba env list`. Make sure you provide read access plus write access if you want others to be able to modify the environment. Then run `mamba activate /path/to/env`
2. Others may instead clone a readable environment with `mamba create --name cloned_env --clone /path/to/original_env`
3. To distribute your environment, run `mamba env export > my-env.yml`. Others can then install this environment with `mamba env create -f /path/to/yaml-file`

Running a Profiler on TensorFlow and PyTorch Models

Both `tensorflow` and `pytorch` have built in tools and `tensorboard` GUI interface for DL profiling, which typically run profiles during the training portion of a deep learning model. Base guides for using these built-in tools follow:

- PyTorch
    - **Profiler Tutorial**
    - **Building a Benchmark Tutorial**
    - **PyTorch Profiler with TensorBoard Tutorial**
- TensorFlow
    - **TensorFlow Profiler Guide**
    - **TensorBoard Profiler Analysis Guide**
- TensorBoard - **Callbacks API Class**

Easy Ways to Implement TensorFlow and PyTorch Profilers

PyTorch

```
model = models.resnet18().cuda()
inputs = torch.randn(5, 3, 224, 224).cuda()

with profile(activities=[
        ProfilerActivity.CPU, ProfilerActivity.CUDA], record_shapes=True) as prof:
    with record_function("model_inference"):
        model(inputs)

print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))
```

The `record_shapes` parameter ensures the profiler collects data on the data pipeline types, notably tensor shapes.

TensorFlow - See **API** for additional options

```
import tensorflow as tf

tf.profiler.experimental.start('/path/to/log/output/')

# ... training loop ...

tf.profiler.experimental.stop()
```

Using NVIDIA Tools for Profiling DL Models

The tools `nsys` and `ncu` are similarly adaptable to run against DL Python codes. The **`dlprof` tool** was previously developed to run `nsys` on DL models then output a TensorBoard interface. However, `dlprof` is no longer being developed in favor of the previous built in profiling methods.

- PyTorch
    - DNN Layer annotations are disabled by default
    - Use `with torch.autograd.profiler.emit_nvtx():`
    - Manually with `torch.cuda.nvtx.range_(push/pop)`
    - TensorRT backend is already annotated
- TensorFlow
    - Annotated by default with NVTX, *only in **NVIDIA NGC containers** or `nvidia-pyindex` TF 1.X containers*
        - `export TF_DISABLE_NVTX_RANGES=1` to disable for production
    - For TensorFlow 2.X, must manually inline NVTX ranges or use `dlprof --mode=tensorflow2 ...`

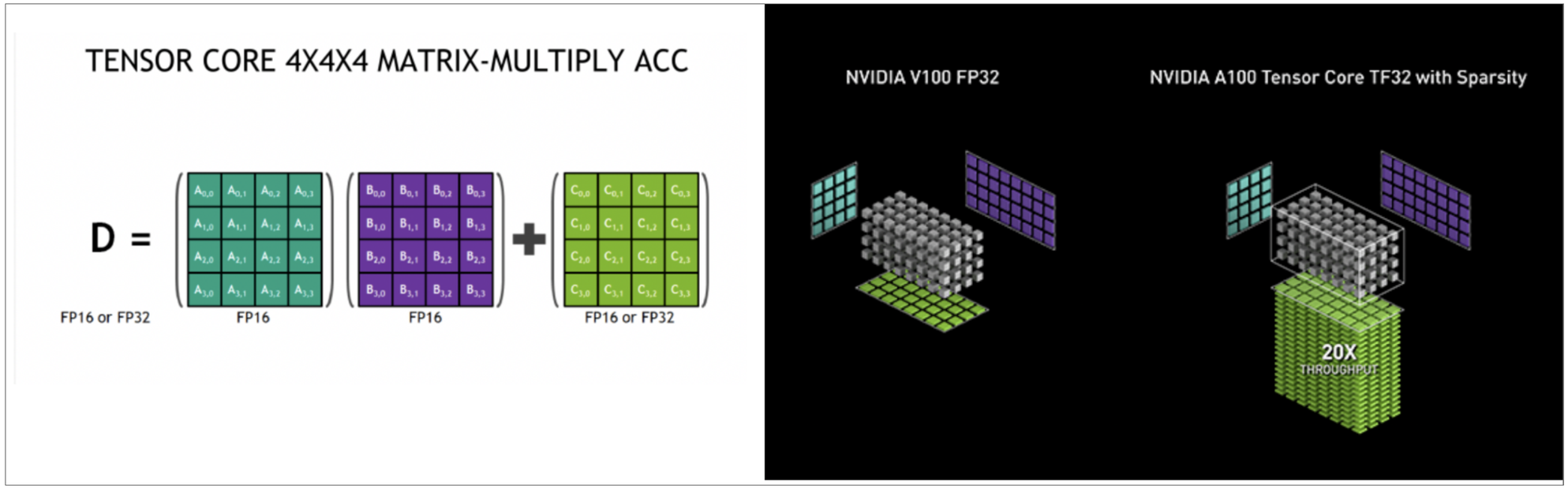
NVIDIA provides their own guides, such as **NVIDIA Deep Learning Performance**. A small example using the `nsys`/`ncu` tools and `dlprof` with DL models can be found **here**. `dlprof` can still work well in NVIDIA **NGC Containers** but compatibility elsewhere is not well supported.

Common Performance Considerations

1. I/O
    - Use designated TF/PT data loaders
        - TensorFlow - **Better Performance with the `tf.data` API**
        - PyTorch - **Datasets & Dataloaders**
    - Multithreading, eg **Multi-Worker Training with Keras**
2. CPU to/from GPU data copies
    - Rewrite code with TF/PT tensors or use CuPy, etc
    - Overlap copy and computation
3. Batch size - Increase batch size up to GPU is saturated
4. Precision (Background: See Theo Mary's **Mixed Precision Artithmetic** talk at London Math Society)
    - Consider mixed precision, **NVIDIA Mixed Precision Training Guide**
    - Automatic Mixed Precision (AMP) settings
        - **PT Guide**: `scaler = torch.cuda.amp.GradScaler()`
        - **TF Guide**: `policy = mixed_precision.Policy('mixed_float16'); mixed_precision.set_global_policy(policy)`
    - Ensure usage of Tensor Cores with Mixed Precision

TensorFlow provides a comprehensive guide, **Optimize TensorFlow GPU performance with the TensorFlow Profiler**

Performance Improvements with Tensor Cores



Per NVIDIA's recommendation on **Optimizing for Tensor Cores**, setting parameters such as matrix dimension sizes, batch sizes, convolution layer channel counts, etc. as **multiples of 8** is optimal due to tensor core shape constraints.
Utilizing mixed precision and tensor cores effectively can lead to **theoretical throughput performance** of **9.70 TeraFLOPS** for FP64 arithmetic up to **78.0 TeraFLOPS** for FP16 arithmetic on A100 GPUs.

Profiler Runs of a Geomagnetic Field LSTM Model

This Long Short-Term Memory (LSTM) example comes courtesy of the **Trustworthy AI for Environmental Science Trust-a-thon**. You can follow the original example, with data preparation and explanation of how the LSTM model is implemented in the **source notebook**.
To begin, let's first download data to use for training and validation of our LSTM model.

In [ ]:

```
%%capture captured_io
%%bash

# Download data we need. If a directory "data/" already exists, we'll assume the data are already downloaded.
#      The above "magic" statements are used to capture shell in/out and to run the following Bash commands.
if [ ! -d "data" ]; then
  wget --verbose https://ngdc.noaa.gov/geomag/data/geomag/magnet/public.zip
  wget --verbose https://ngdc.noaa.gov/geomag/data/geomag/magnet/private.zip
  unzip public.zip
  unzip private.zip
  mkdir -v data
  mv -v public private data/
  mv -v public.zip private.zip data/
fi
# Uncomment for debugging if you have trouble downloading:
#print(captured_io)
```

Profile the `magnet_lstm_tutorial.py` Python Script

The full Geomagnetic Field LSTM model is condensed into the Python file **magnet_lstm_tutorial.py**. Recall that profiling does not require analyzing the full runtime of most models. In DL, most operations are highliy repetitive so the profiler only needs to sample a small portion of the runtime. **Minimizing the time for profile runs can speed up the iterative development process**.

- **TODO Line 237**: Adjust parameter `n_epochs=1` in order to minimize profiling time.
- **TODO Line 295/301**: Add TensorBoard callbacks as defined below.

```
tboard_callback = keras.callbacks.TensorBoard(
    log_dir = "profile_results", histogram_freq = 1, profile_batch = '500,520')

...

model.fit(
    ...,
    callbacks = [tboard_callback]
)
```

Question: How else could you minimize runtime of a "profile run" but still maintain model configuration parameters equivalent to production runs?

In [ ]:

```
%%bash
qsub pbs_job.sh
```

In [ ]:

```
%%bash
# Run this cell if in an interactive GPU node job
#module load cuda/11.7 &> /dev/null
#python magnet_lstm_tutorial.py
```

Open the Profile Report in TensorBoard

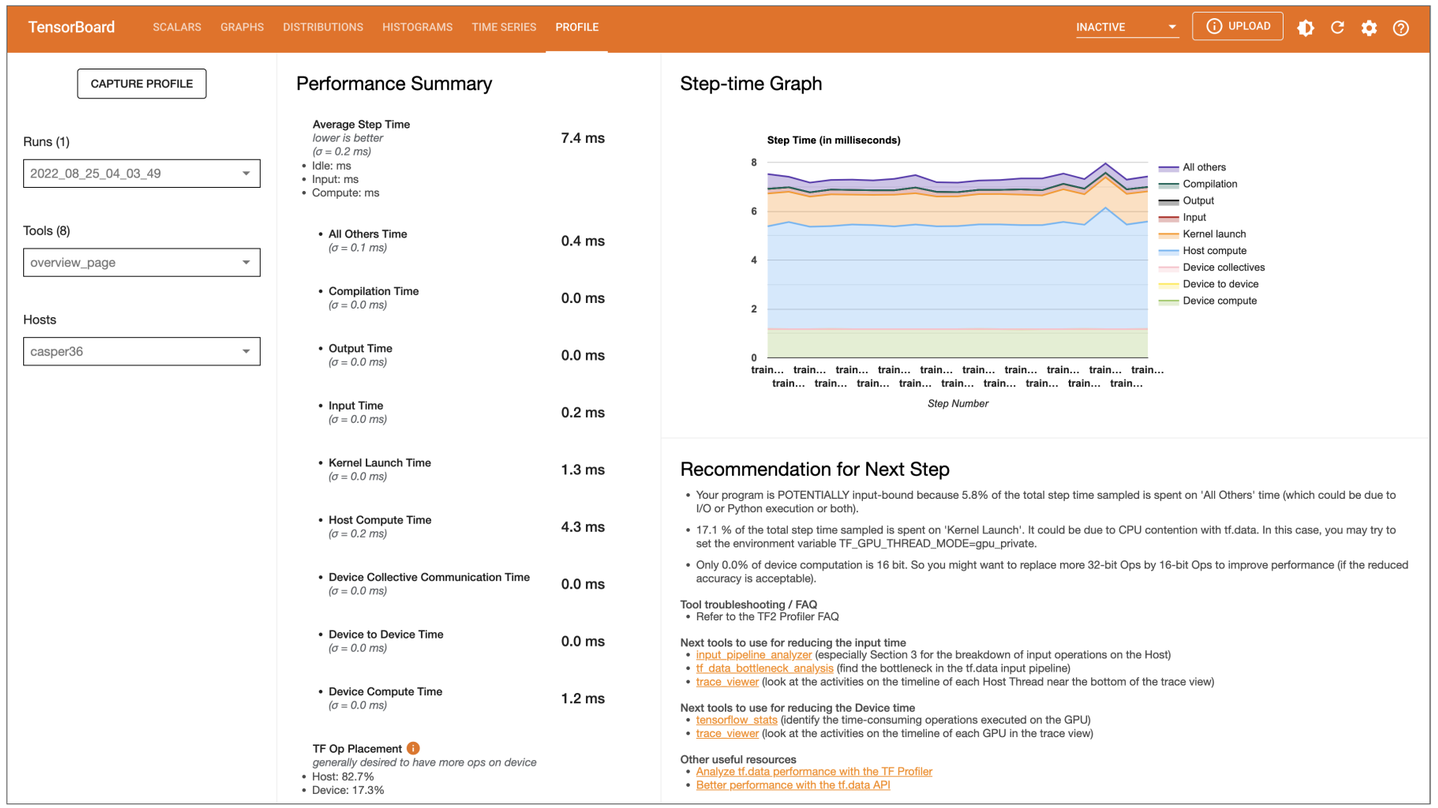
Typically you'll run this on the login node of Casper and will need to do some `ssh` port forwarding to access the server. You can generally follow these steps:

1. `ssh -L$PORTA:localhost:$PORTB $USER@casper.ucar.edu`
2. `module load conda`
3. `mamba activate /glade/work/dhoward/conda/env/GPU_Workshop`
4. `cd /path/to/log/output`
5. `tensorboard --port $PORTB --bind_all --logdir </path/to/log/output/>` and wait for message that says server has started
6. Open browser on your laptop and go to `localhost:$PORTA`

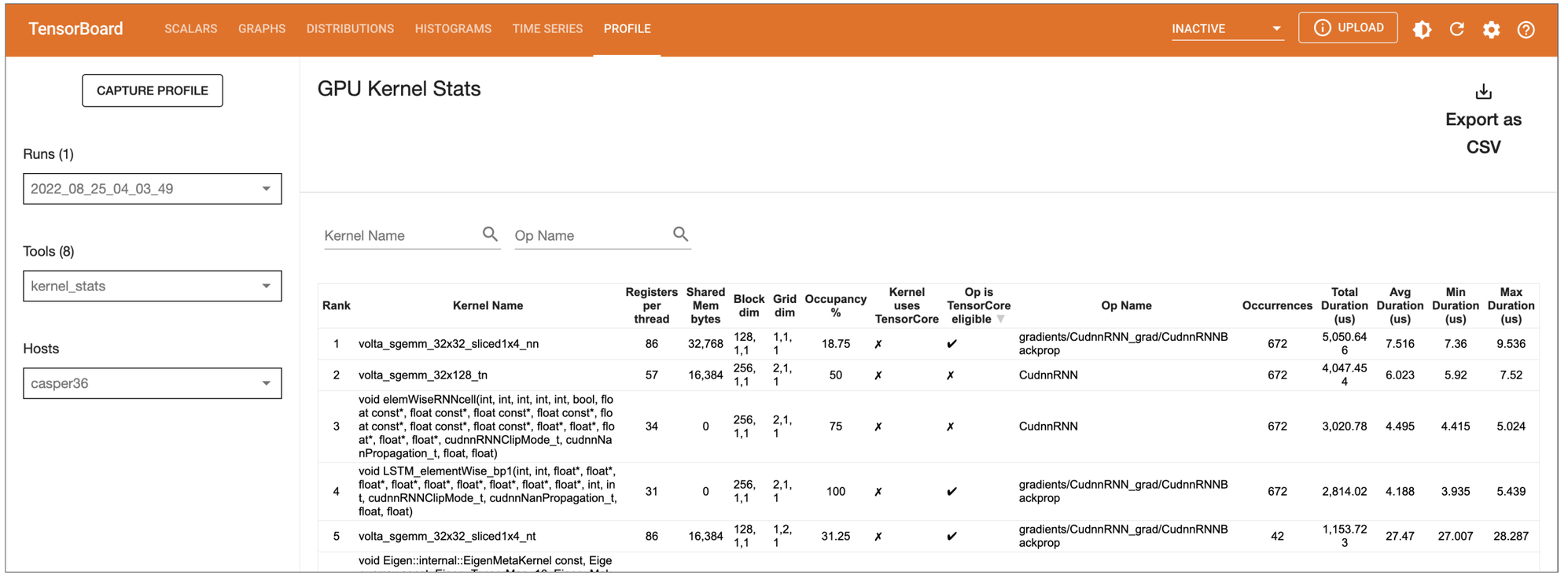
In [ ]:

```
# If on a local machine with GPU, use these commands to open the profile.
# Otherwise, port forwarding is needed on Casper
# %load_ext tensorboard
# %tensorboard --logdir=profile_results
```

Analyzing Profiles in TensorBoard

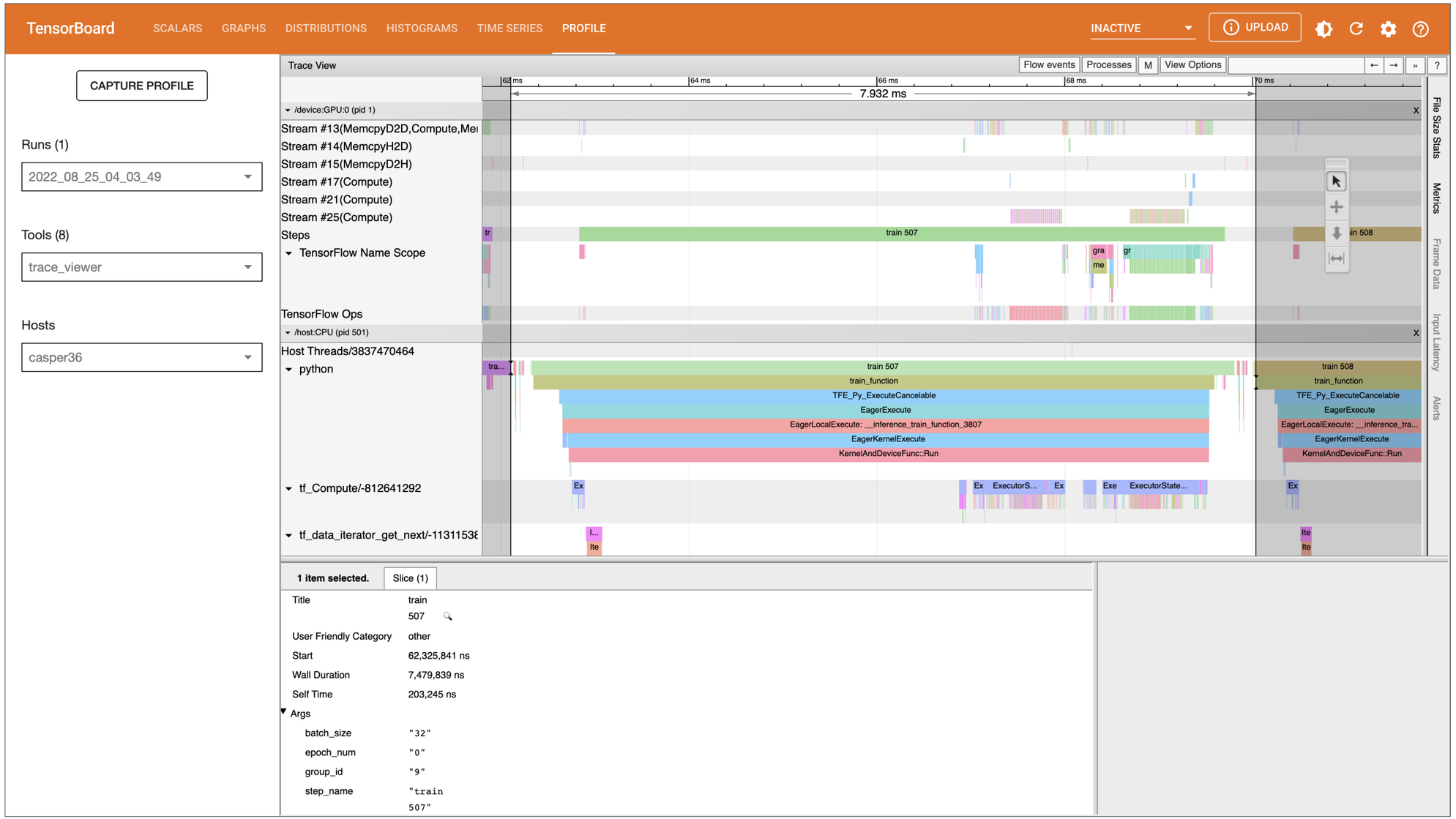


Performance improvement heuristics are often provided with links to more detailed information.



Rank	Kernel Name	Registers per thread	Shared Mem bytes	Block dim	Grid dim	Occupancy %	Kernel uses TensorCore	Op is TensorCore eligible	Op Name	Occurrences	Total Duration (us)	Avg Duration (us)	Min Duration (us)	Max Duration (us)
1	volta_sgemm_32x32_sliced1x4_nn	86	32,768	128, 1,1	1,1, 1	18.75	✗	✔	gradients/CudnnRNN_grad/CudnnRNNBackprop	672	5,050.646	7.516	7.36	9.536
2	volta_sgemm_32x128_tn	57	16,384	256, 1,1	2,1, 1	50	✗	✗	CudnnRNN	672	4,047.454	6.023	5.92	7.52
3	void elemWiseRNNcell(int, int, int, int, int, bool, float const*, float const*, float const*, float const*, float const*, float const*, float*, float*, float*, float*, cudnnRNNClipMode_t, cudnnNanPropagation_t, float, float)	34	0	256, 1,1	2,1, 1	75	✗	✗	CudnnRNN	672	3,020.78	4.495	4.415	5.024
4	void LSTM_elementWise_bp1(int, int, float*, float*, float*, float*, float*, float*, float*, float*, float*, int, int, cudnnRNNClipMode_t, cudnnNanPropagation_t, float, float)	31	0	256, 1,1	2,1, 1	100	✗	✔	gradients/CudnnRNN_grad/CudnnRNNBackprop	672	2,814.02	4.188	3.935	5.439
5	volta_sgemm_32x32_sliced1x4_nt	86	16,384	128, 1,1	1,2, 1	31.25	✗	✔	gradients/CudnnRNN_grad/CudnnRNNBackprop	42	1,153.723	27.47	27.007	28.287
	void Eigen::internal::EigenMetaKernel const, Eige													

Important to emphasize **where Tensor Core use is eligible** in your model and determine if it's appropriate for employing reduced precision.



Use the `trace_viewer` to get an overall timeline of your deep learning model.