

HECC Data Science Webinar

9/15/2021

NASA Advanced Supercomputing Division

This meeting will be recorded.

Please be sure to mute your microphone and turn off your camera during the presentations.

The host will be monitoring the chat window for any questions and will relay them to the speaker.

About the Data Science Team

- Shubha Ranjan – Data Science Group Lead
- Data Science Team Members
  - Thaddeus Norman
  - Jonathan Gee
  - Jeff Becker
  - Thai Truong

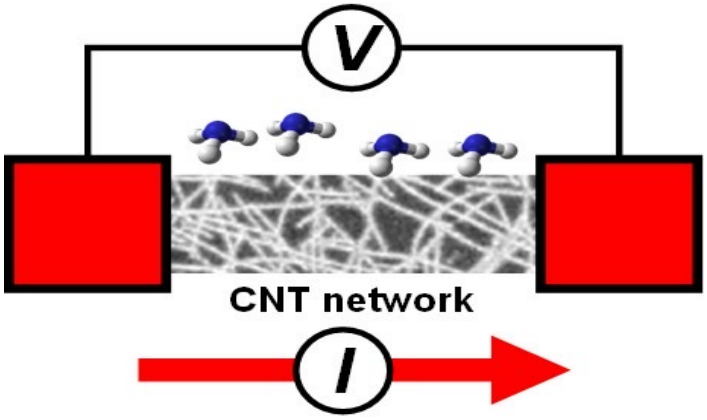
Outline

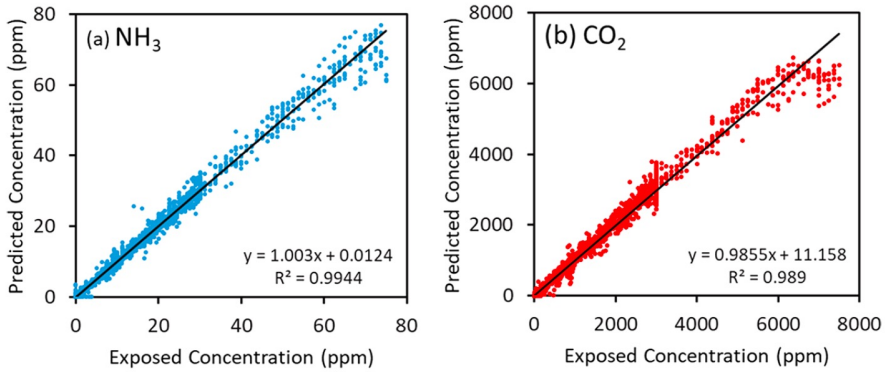
- **Data Science At HECC**
  - Data Science Conda Environments
  - Distributed Python
  - Overview of Dask
  - Overview of Horovod
  - Overview of Jupyter Lab and Web UIs
  - Overview of TensorBoard
  - Overview of Profiling tools

Pilot Projects

- What Are Pilot Projects?
  - Highlight Existing Capabilities
  - Explore New Technology
- How Do You Initiate One?
  - Contact us: dataanalytics@nas.nasa.gov
- Here Are Some Examples:
  - Solar Cell Materials Prediction
  - System Log Aggregation
  - Carbon Nanotube Neural Network Gas Sensor

Carbon Nanotube NN Gas Sensor





Kim et al, ACS Applied Nano Materials vol 2 issue 10 pages 6445-6451

CNT Pros

- Inexpensive
- Easy to fabricate
- Resistance sensitive to gas concentration

CNT Cons

- Non-linear response
- Difficult to model

Neural Network Solution

- Trained model to predict gas concentration from sensor input
- Recommended path for future work

Data Science at HECC

- Data Science Conda Environments
  - TensorFlow: tf1_15, tf2_3, tf2_4
  - PyTorch: pyt1_7, pyt1_8
  - R: r3_6
  - Horovod: horovod
- Deploying Environments on Pleiades
  - Accessing Environments
    - module use -a /swbuild/analytix/tools/modulefiles
    - module load miniconda3/v4
    - source activate *<environment_name>*

Common Python Packages

- Pandas – Data analytics platform
- Dask – Distributed data analytics platform
- Scikit-learn – Machine learning and scientific computing platform
- Matplotlib – Plotting and graphics platform
- Pillow – A fork of Python Image Library (PIL)
- Jupyter Lab – A collaborative and integrated development environment
- To request or suggest a package:
  - Contact: dataanalytics@nas.nasa.gov or support@nas.nasa.gov

Creating a Custom Conda Environment

- Accessing HECC conda environments:
  - module use -a /swbuild/analytix/tools/modulefiles
  - module load miniconda3/v4
  - source activate *environment_name*
- Adding packages for local use:
  - pip install --user *package_name*
- Creating a new conda environment:
  - export CONDA_ENVS_PATH=/nobackup/$USER/.conda/envs
  - export CONDA_PKGS_DIRS=/nobackup/$USER/target_directory
  - conda create –-name *my_env* --clone base

Using Dask

- Distributed Python Environment
  - https://tutorial.dask.org/
  - https://github.com/dask/dask-tutorial
  - https://docs.dask.org/en/latest/setup/hpc.html
- Single Node Interactive Mode:
  - from dask.distributed import Client, LocalCluster
  - cluster = LocalCluster()
  - client = Client(cluster)
- Multiple Node Interactive Mode:
  - mpirun -np <number of cpus> dask-mpi --no-nanny --interface ib0 --local-directory /…/dask_sch/sched.json
  - client = Client(scheduler_file='/…/dask_sch/sched.json')
  - In a separate shell launch Jupyter Notebook or Jupyter Lab

Submitting a Dask Job

- In PBS script:
  - Create or erase Dask scheduler directory
  - Launch Dask using mpi code
- In Python script:
  - Instantiate Dask client using scheduler launched in PBS script

Submitting a Dask Job

In PBS script (batch mode):

```
#clear and create directory for dask scheduler
if [ -d dask_sch ]
then
    echo "removing old scheduler directory"
    rm -r dask_sch
    echo "creating new scheduler directory"
    mkdir dask_sch
else
    echo "creating scheduler directory"
    mkdir dask_sch
fi

#run program
mpirun -np <number of cpus> dask-mpi --no-nanny --interface ib0 --local-directory
/…/dask_sch/sched.json & python <filename.py>
```

In Python Code:

```
from dask.distributed import Client
client = Client(scheduler_file='/…/dask_sch/sched.json')
```

Using Horovod

- Distributed deep learning training framework
  - Used in conjunction with TensorFlow or PyTorch to train models faster when distributed
  - https://github.com/horovod/horovod
  - https://horovod.readthedocs.io/en/stable/summary_include.html
- After setting up pbs script and python bindings, easy to test distribution to many nodes

Submitting a Horovod Job

- In PBS Script
  - load miniconda and cuda 11.0 module and activate horovod environment
  - run script to set environment variables on all nodes
  - run program
- In set_vars.sh
  - Set environment variables needed for horovod to run
- In python code
  - Initialize Horovod
  - Pin processes to 1 GPU
  - Use Horovod distributed optimizer
  - Broadcast Horovod variables

Submitting a Horovod Job



In PBS script (batch mode):

```
#load miniconda and cuda 11.0 module and activate horovod environment
module -a use /swbuild/analytix/tools/modulefiles
module load miniconda3/v4
module load cuda/11.0
source activate horovod

#run script to set environment variables on all nodes
source set_vars.sh <number of GPUs per node>

#run program
mpiexec -np <total number of GPUs> \
-bind-to none -map-by slot -prepend-rank \
python <filename.py>
```

Submitting a Horovod Job



In set_vars.sh:

```
#important environment variables that need to be set
export NCCL_IB_DISABLE=1
#needs to be set on all nodes
export CUDA_VISIBLE_DEVICES=$GPU_LIST

#GPU_LIST set by command line argument, list 0 to specified number of GPUs
GPU_LIST=0
for ((i=1; i<=($1-1); i++));
do
  GPU_LIST=$GPU_LIST,$i
done

#optional environment variables that can help debug
export NCCL_DEBUG=INFO
export HOROVOD_TIMELINE=<folder_path>/<timeline_filename>.json
```

Submitting a Horovod Job

In Python code for TensorFlow:

```
import horovod.tensorflow.keras as hvd

#initialize horovod
hvd.init()

#pin GPUs to one process each
gpus = tf.config.experimental.list_physical_devices('GPU')
    for gpu in gpus:
        tf.config.experimental.set_memory_growth(gpu, True)
    if gpus:
        tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')

#wrap optimizer in a horovod one
opt = tf.optimizers.SGD(0.01)
opt = hvd.DistributedOptimizer(opt)

#compile model with experimental_run_tf_function as False
model.compile(loss=tf.losses.SparseCategoricalCrossentropy(),
              optimizer=opt,
              experimental_run_tf_function=False)

#add horovod broadcast variables callback
callbacks=[hvd.callbacks.BroadcastGlobalVariablesCallback(0)]
```

Submitting a Horovod Job



In Python code for PyTorch:

```
import horovod.torch as hvd

#initialize horovod
hvd.init()

#pin GPUs to one process each, place model on GPUs
torch.cuda.set_device(hvd.local_rank())
model.cuda()

#wrap optimizer in a horovod one
optimizer = optim.SGD(model.parameters(), lr=0.01 * lr_scaler)
optimizer = hvd.DistributedOptimizer(optimizer,
                                     named_parameters=model.named_parameters(),
                                     op=hvd.Average)

#horovod broadcast variables
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)
```

Jupyter Lab & Web UIs

- Web UIs and security
  - Web servers allow access to your file system!
  - When using a web server you must be on a compute node!
  - Compute nodes have restricted access
  - Compute nodes will not allow you to download files from the internet
- Interactive Mode
  - qsub –I –l select *node …*
- Reserved Front End
  - pbs_rfe --duration *10*+ --model *bro*

Running Jupyter Lab & Web UIs

- Start conda environments and Web Server:
  - module use -a /swbuild/analytix/tools/modulefiles
  - module load miniconda3/v4
  - source activate tf2_4
  - jupyter lab –no-browser
    - (or command to launch server)
- In a new shell on your local machine:
  - ssh -o "StrictHostKeyChecking ask" \
    -L 18080:localhost:8888 \
    -o ProxyJump=sfe,pfe21 *rfe_name*
- In a browser, connect to http://localhost:18080/

TensorBoard

- TensorBoard is a useful UI tool for visualizing and debugging TensorFlow models
  - Used for easy data visualization, model statistics, and in-depth analysis of the model layers
- One of the extra modules in TensorBoard is the profiling tool
  - Used for in-depth analysis of the training performance by generating a timeline of the model
  - Provides information on where the bottlenecks in training could occur and helps provide suggestions on how to alleviate them
- https://www.tensorflow.org/tensorboard/get_started

TensorBoard Python Bindings

- TensorBoard works as a callback in TensorFlow
  - Assign log directory
    - log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
  - Set TensorFlow callback & assign the histogram frequency
    - tensorboard_callback= tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
    - Histogram number represents the number of logged batches

Running TensorBoard

- You can launch TensorBoard from a PBS node like a Jupyter notebook:
  - module use -a /swbuild/analytix/tools/modulefiles
  - module load miniconda3/v4
  - source activate tf2_4
  - tensorboard --logdir=<directory>
- In a new shell on your local machine:
  - ssh -o "StrictHostKeyChecking ask" \
    -L 8080:localhost:6006 \
    -o ProxyJump=sfe,pfe<21> *rfe_name*
- In a browser, connect to http://localhost:8080/

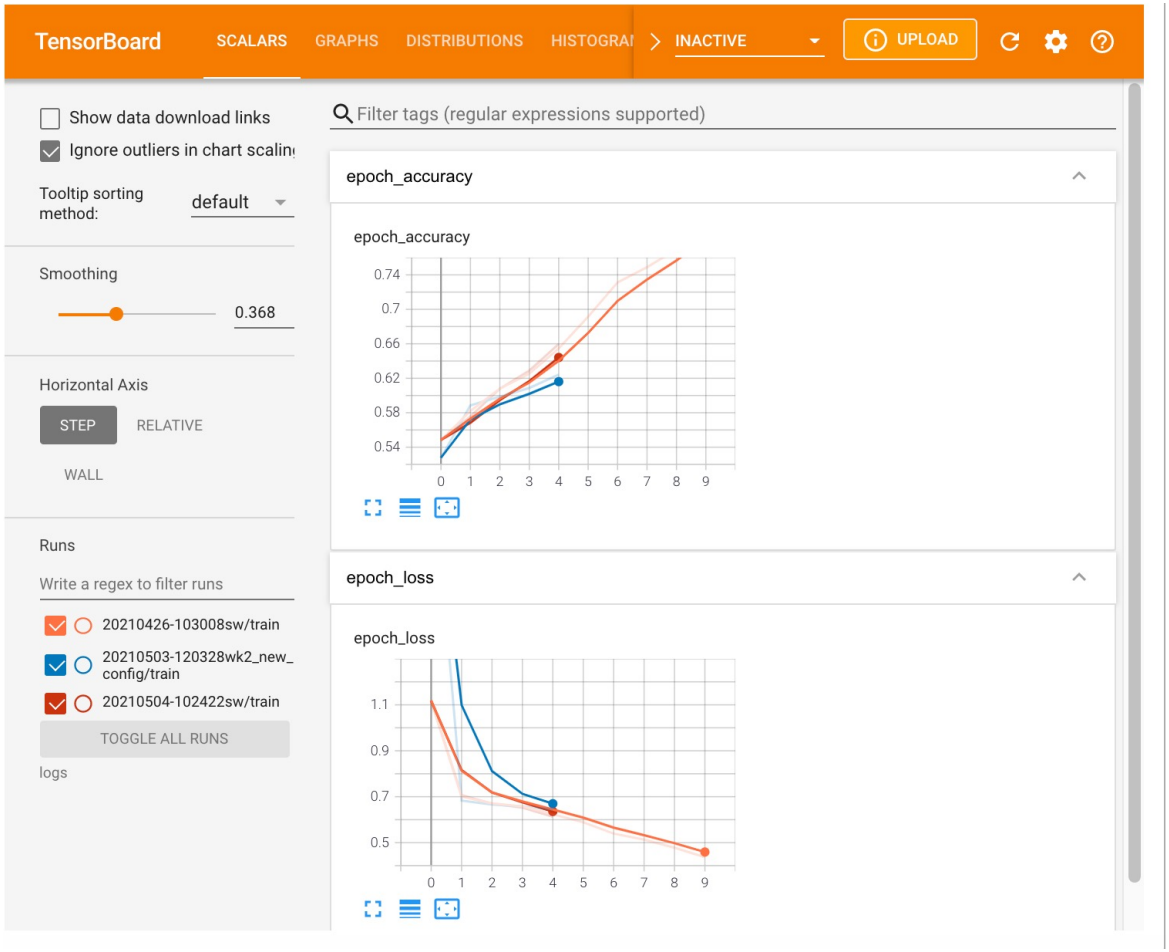
Running TensorBoard In Jupyter Notebook



- Alternatively, you can use TensorBoard in a Notebook with:
  - %load_ext tensorboard
  - %tensorboard --logdir <directory> --load_fast=false

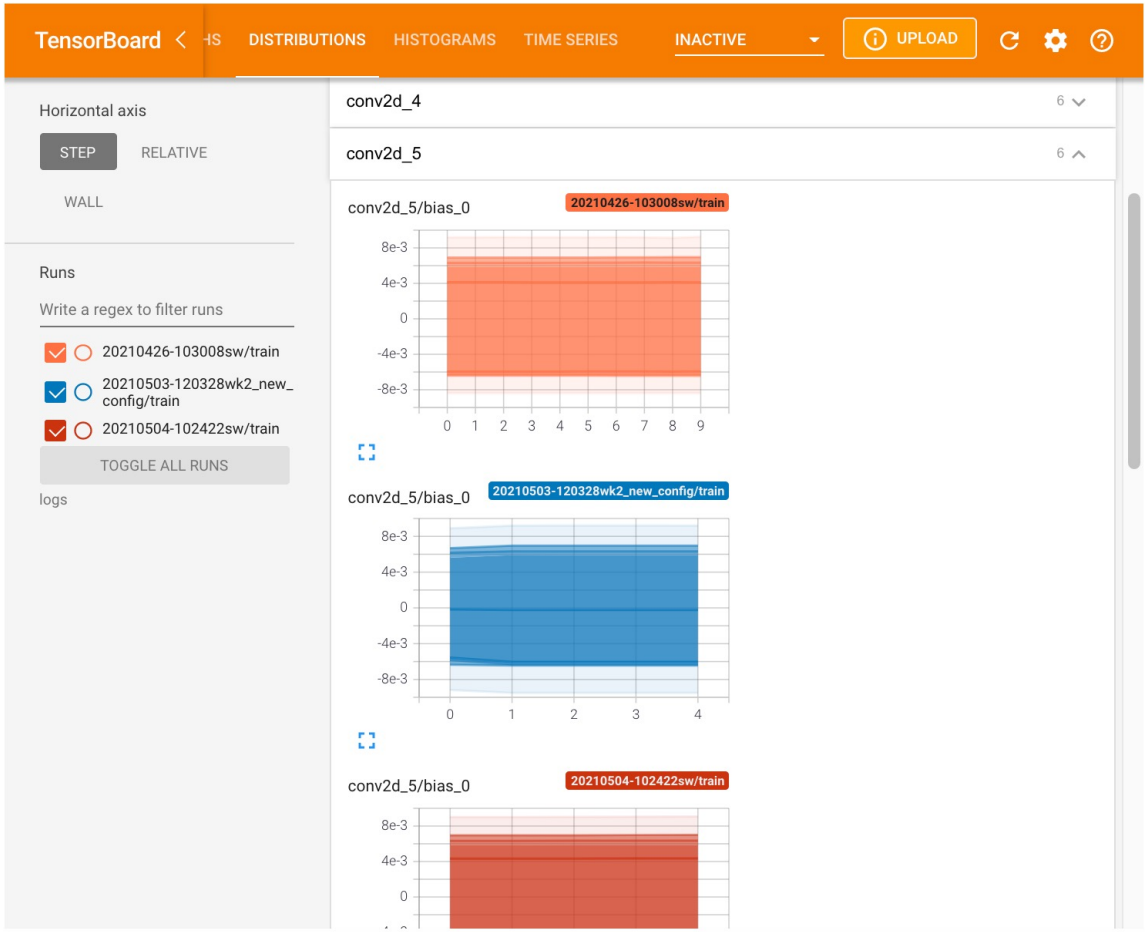
TensorBoard Usage

- Basic model results comparison using scalar tab:



TensorBoard Usage

- Analysis of weights within each layer with distribution tab:



TensorBoard Profiling



- TensorBoard has a built-in profiling tool that can be helpful for debugging and optimizing training
- TensorFlow profiling can identify a few common bottlenecks
  - Data loading
  - Collective Reduce
  - GPU usage

TensorBoard Profiling for TensorFlow



- The profiling tool can be accessed through the same TensorBoard window, but needs additional arguments in the callback to record data:
  - tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, profile_batch='<start,end>')
- Records data from the first epoch
- Some of the graphs use Google Chart Libraries which require internet access which is not available on compute nodes
  - We recommend transfer log data to local machine to get full outputs
- https://www.tensorflow.org/guide/profiler

Running TensorBoard Locally



- Transfer log data to local machine
- Run TensorBoard or Jupyter notebook locally:
  - tensorboard --logdir=<directory>
- Required packages:
  - TensorFlow >=2.4
  - TensorBoard >= 2.5
  - Jupyterlab
- If you want to use TensorBoard with PyTorch:
  - pytorch >=1.8
  - torch-tb-profiler

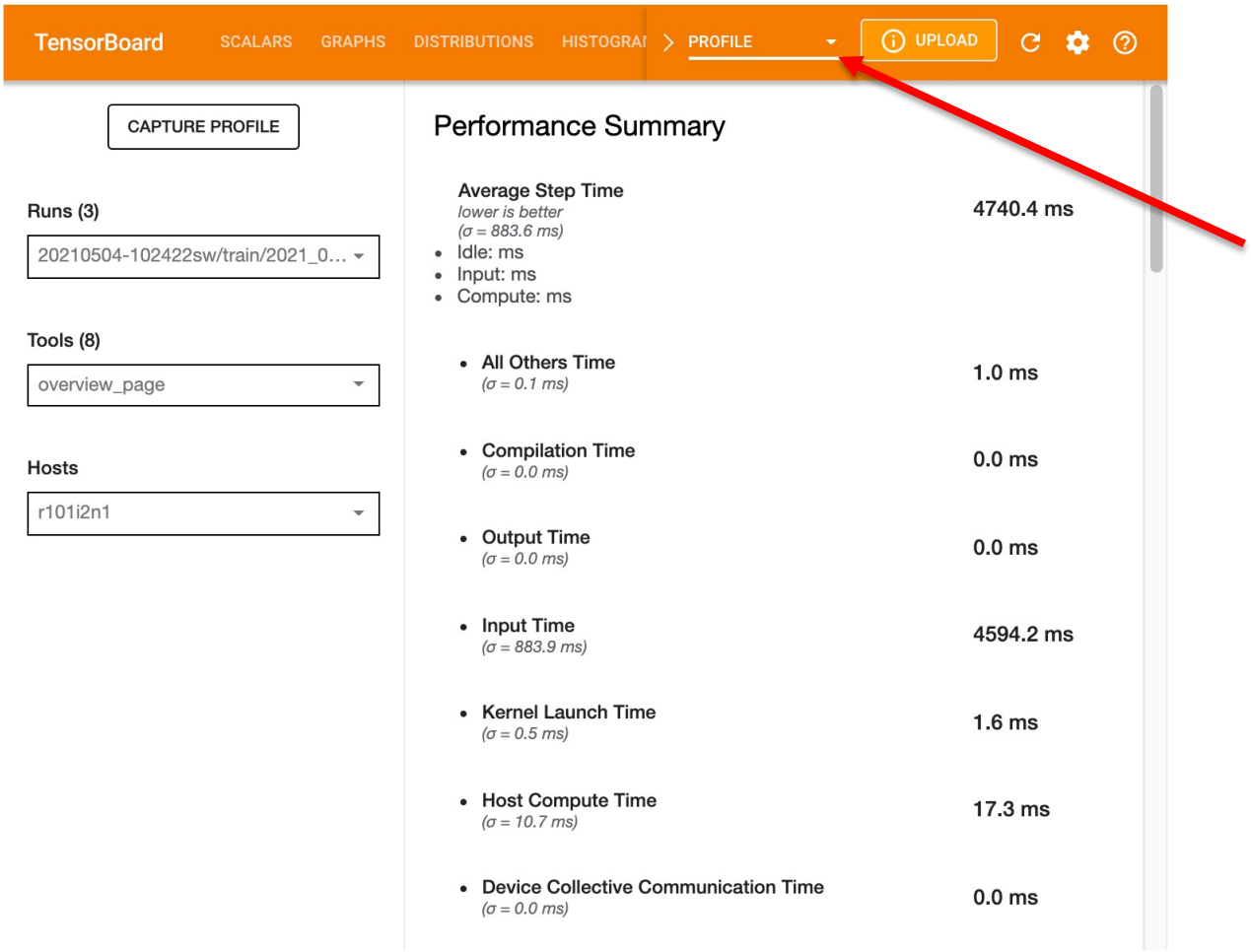
TensorBoard Profiling



- TensorBoard Profiling provides different levels of depth for system resource utilization:
  - Overview – summarizes average step time, can help easily identify a training bottleneck
  - kernel_stats, pod_viewer, tensorflow_stats – breakdown of computational graph during training, can help find which part of the model is slow
  - Memory_profile – monitors memory usage and can help debug OOM errors
  - trace_viewer – full timeline of every op and device that executed them
  - tf_data_bottleneck_analysis – experimental tool to help optimize data throughput

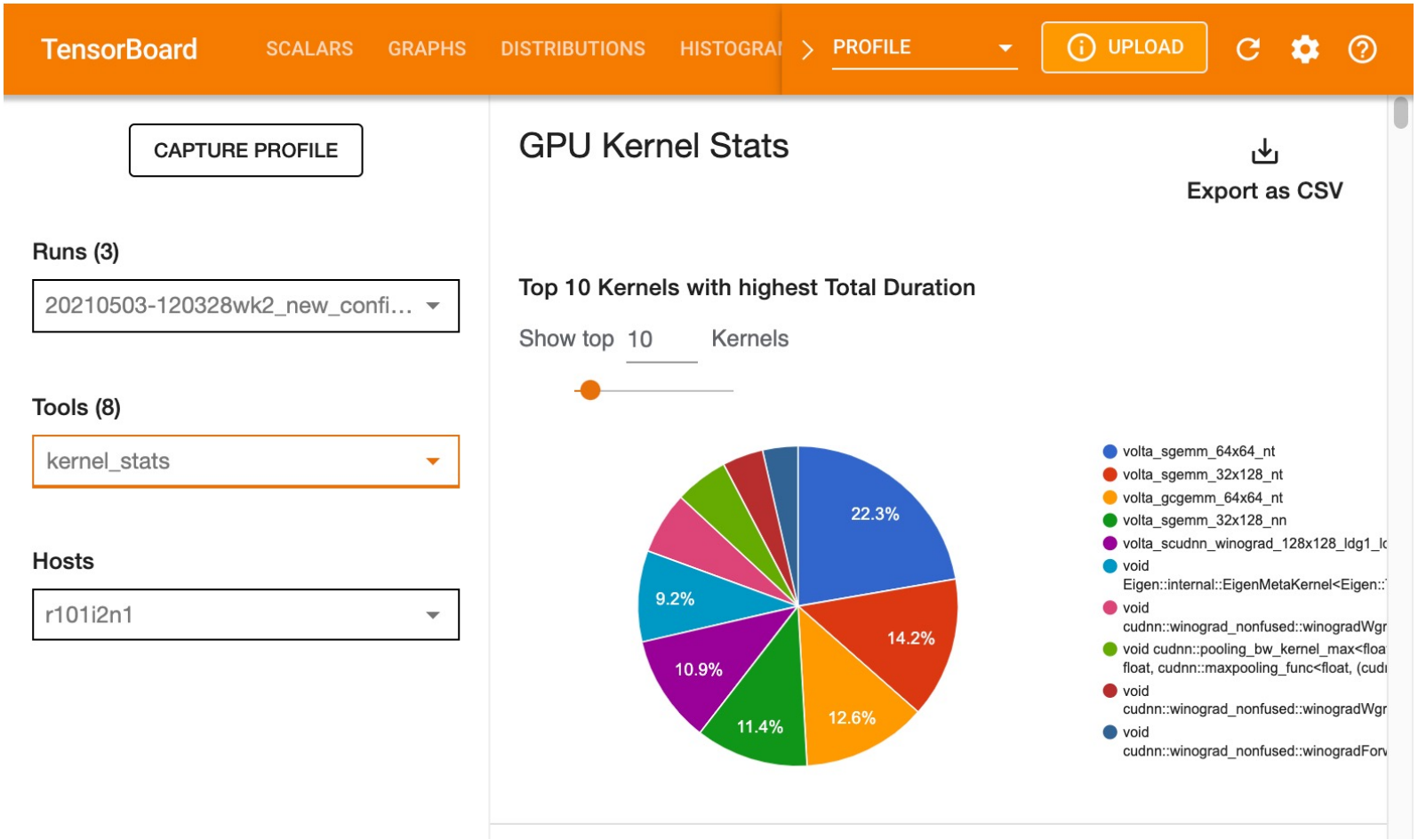
Overview

- Access profiling tool from the dropdown menu in the top right



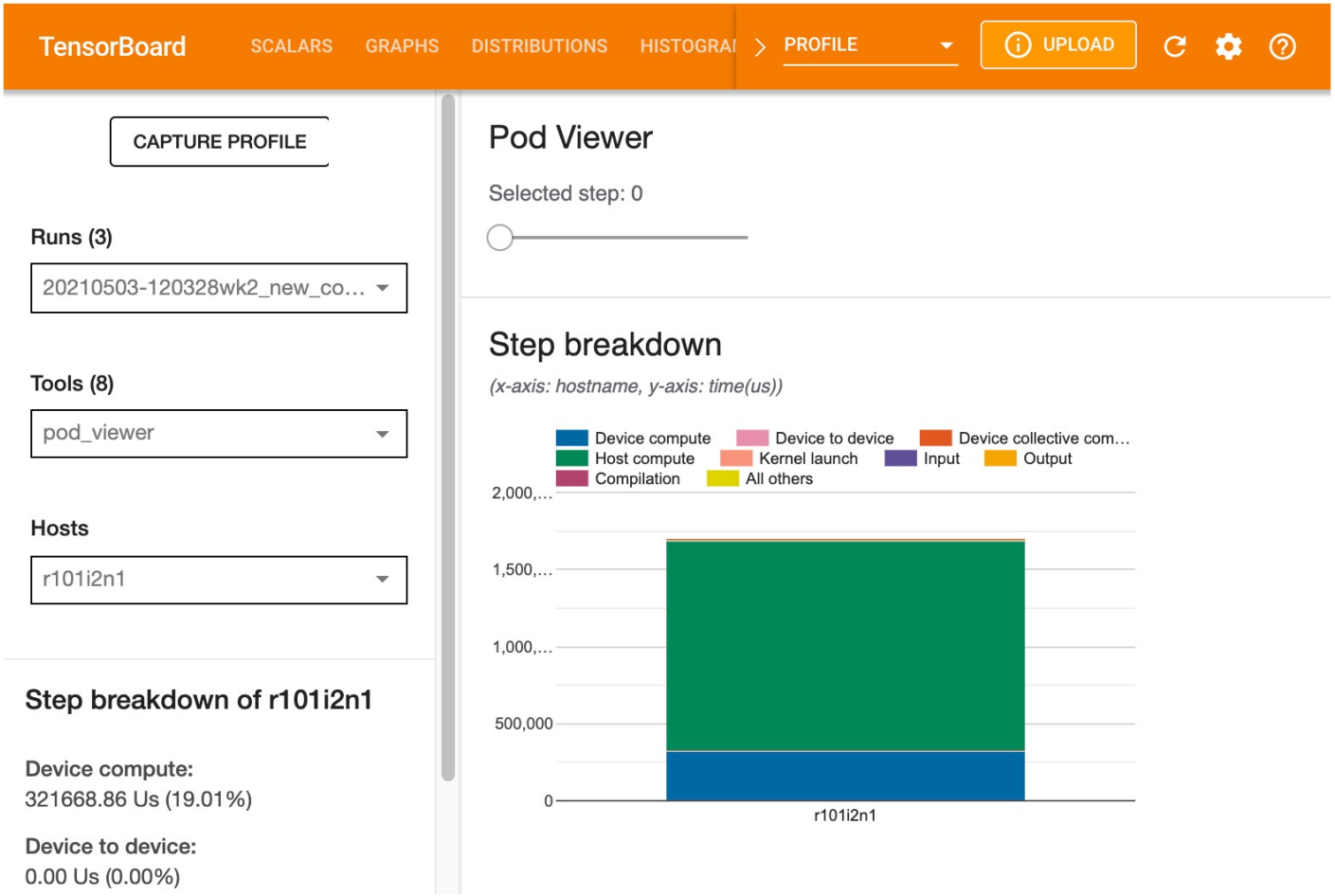
Kernel Stats

Shows breakdown of GPU usage



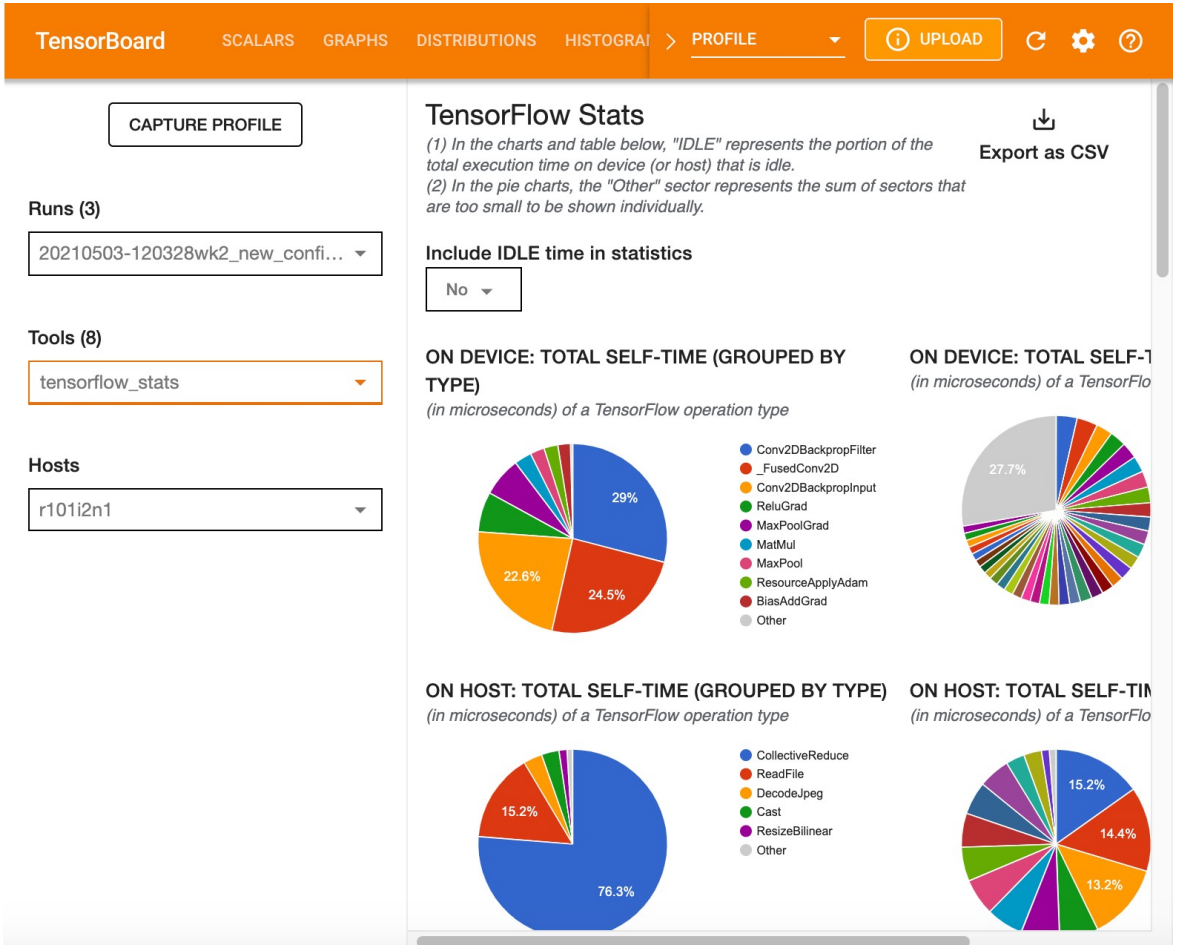
Pod Viewer

Shows breakdown of device usage in each step



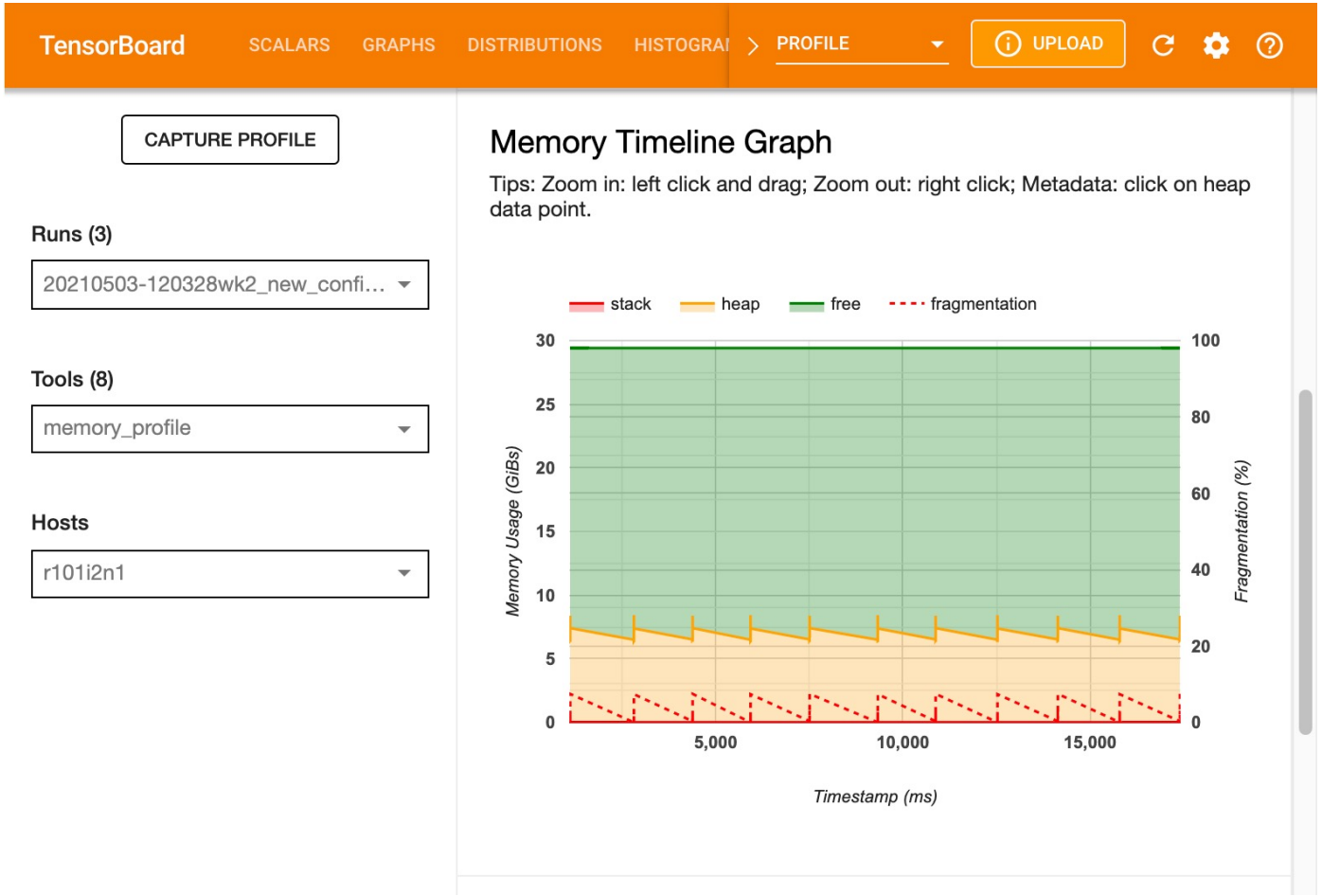
TensorFlow Stats

Shows breakdown of usage across the GPU and CPU



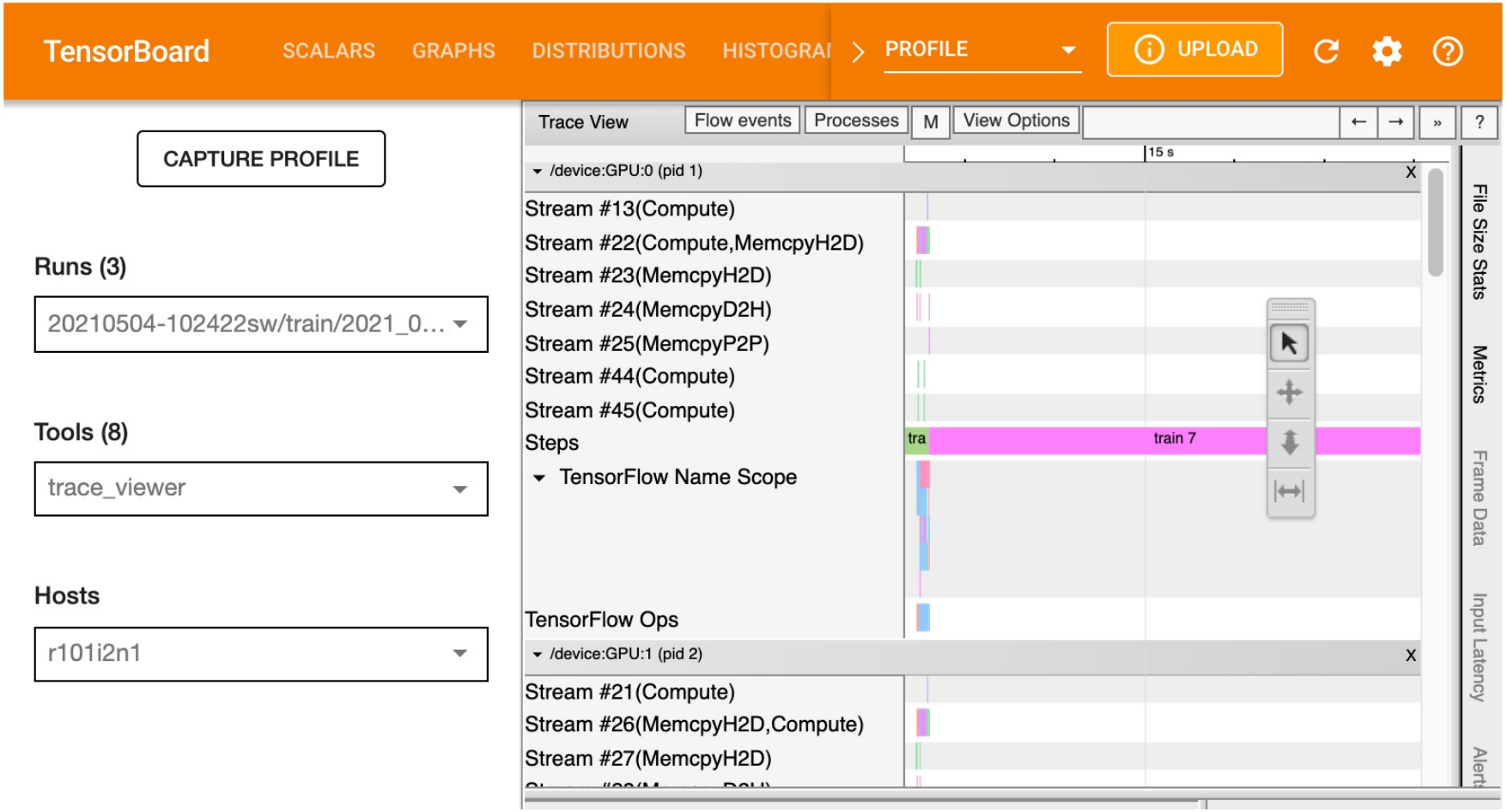
Memory Profile

Shows memory usage of selected device



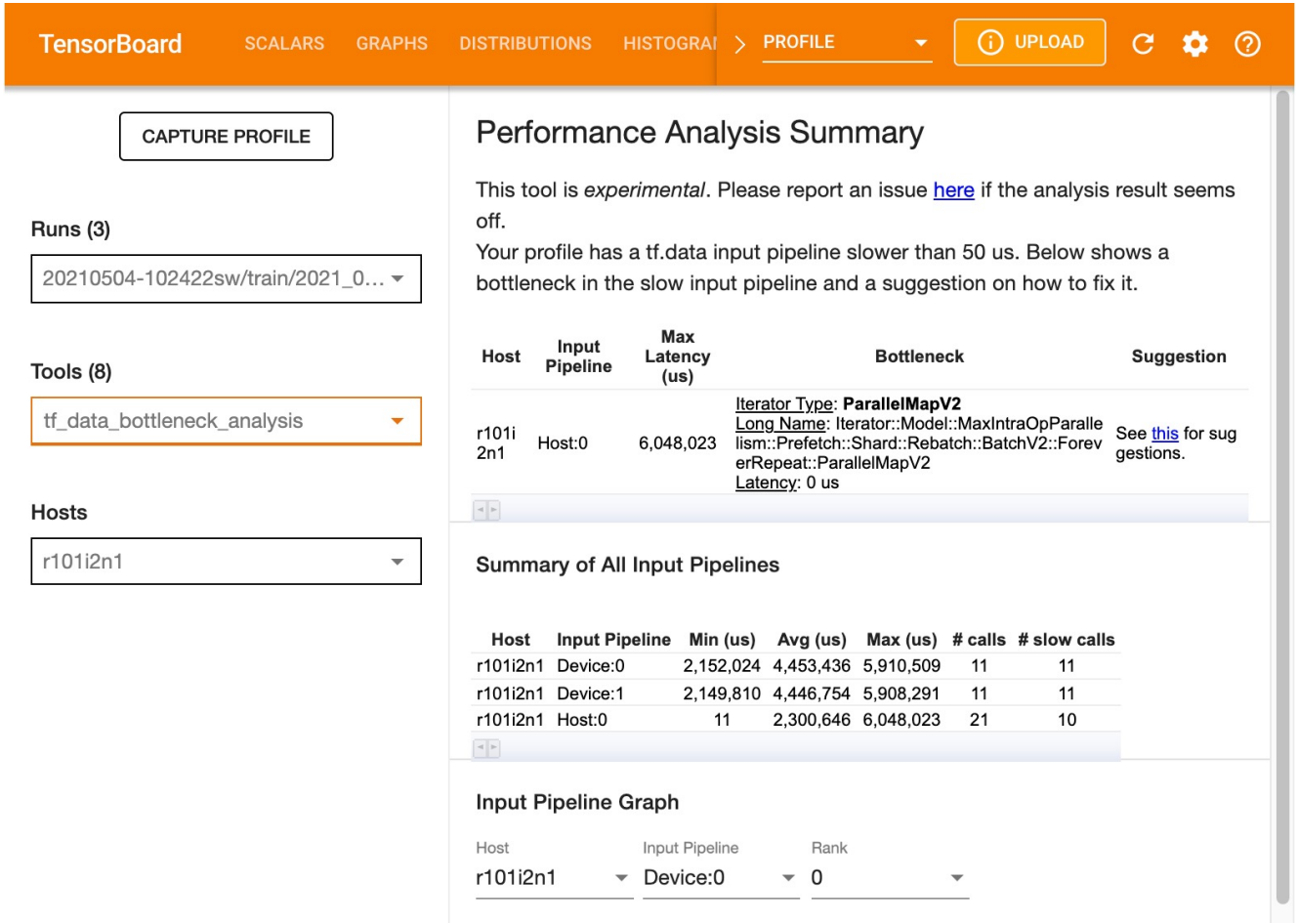
Trace Viewer

Full timeline of all ops on each device



TF Data Bottleneck Analysis

Provides suggestions on how to optimize data loading



TensorBoard Profiling for PyTorch



- New versions (>1.8) of PyTorch support TensorBoard
  - Not as in-depth as TensorFlow's output, but still can be helpful
- PyTorch profiler created in a context manager:

```
with torch.profiler.profile(
        schedule=torch.profiler.schedule(wait=1, warmup=1, active=3, repeat=2),
        on_trace_ready=torch.profiler.tensorboard_trace_handler('./log/resnet18_2'),
        record_shapes=True,
        profile_memory=True,
        with_stack=True
) as prof:
        <train model>
        prof.step()
```

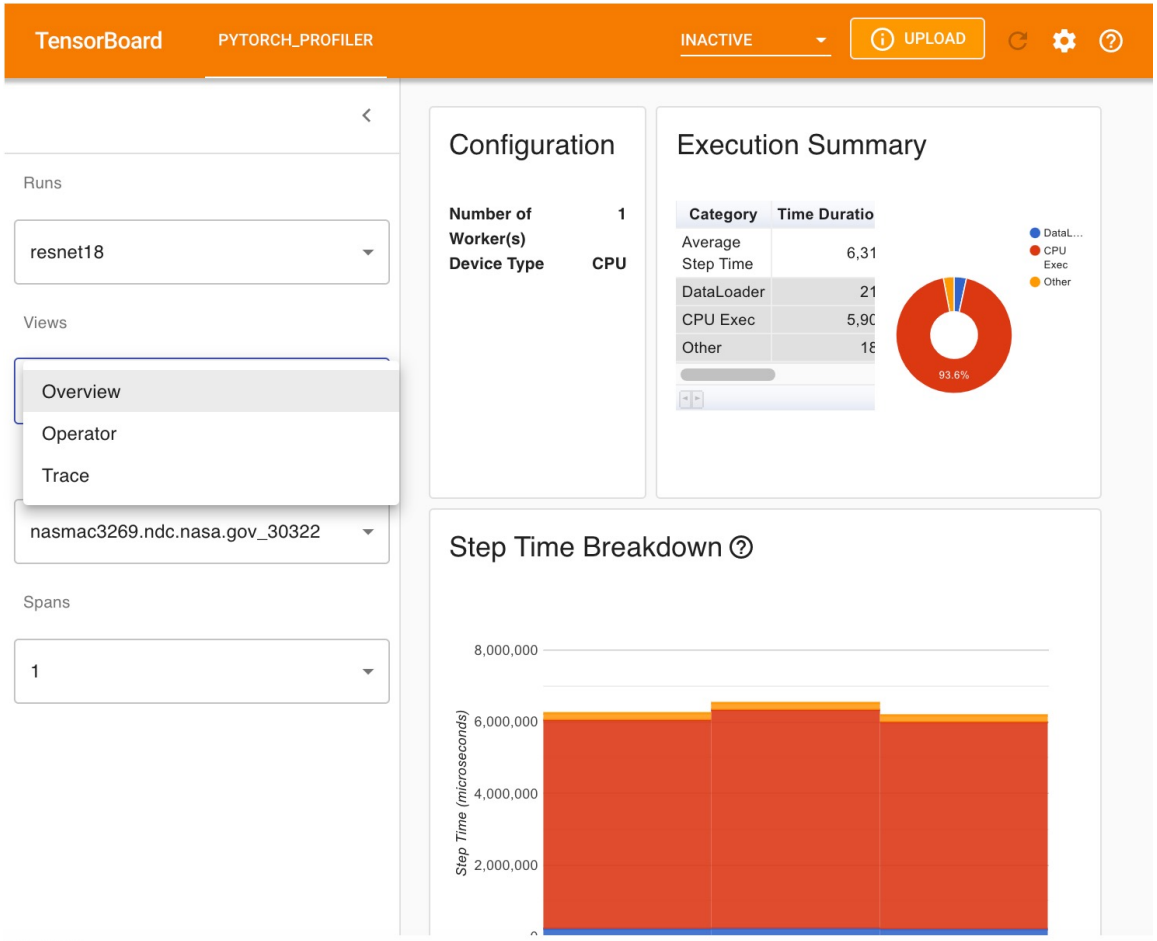
TensorBoard Profiling for PyTorch



- Pytorch profiler only has a few tabs:
  - Overview – summarizes average step time
  - Operator – displays every op and gives a breakdown on the most time-consuming ones
  - Trace - full timeline of every op and device that executed them
  - Memory – breakdown of memory usage for each op
- Does not provide a suggestion on what is bottlenecking training like TensorFlow does

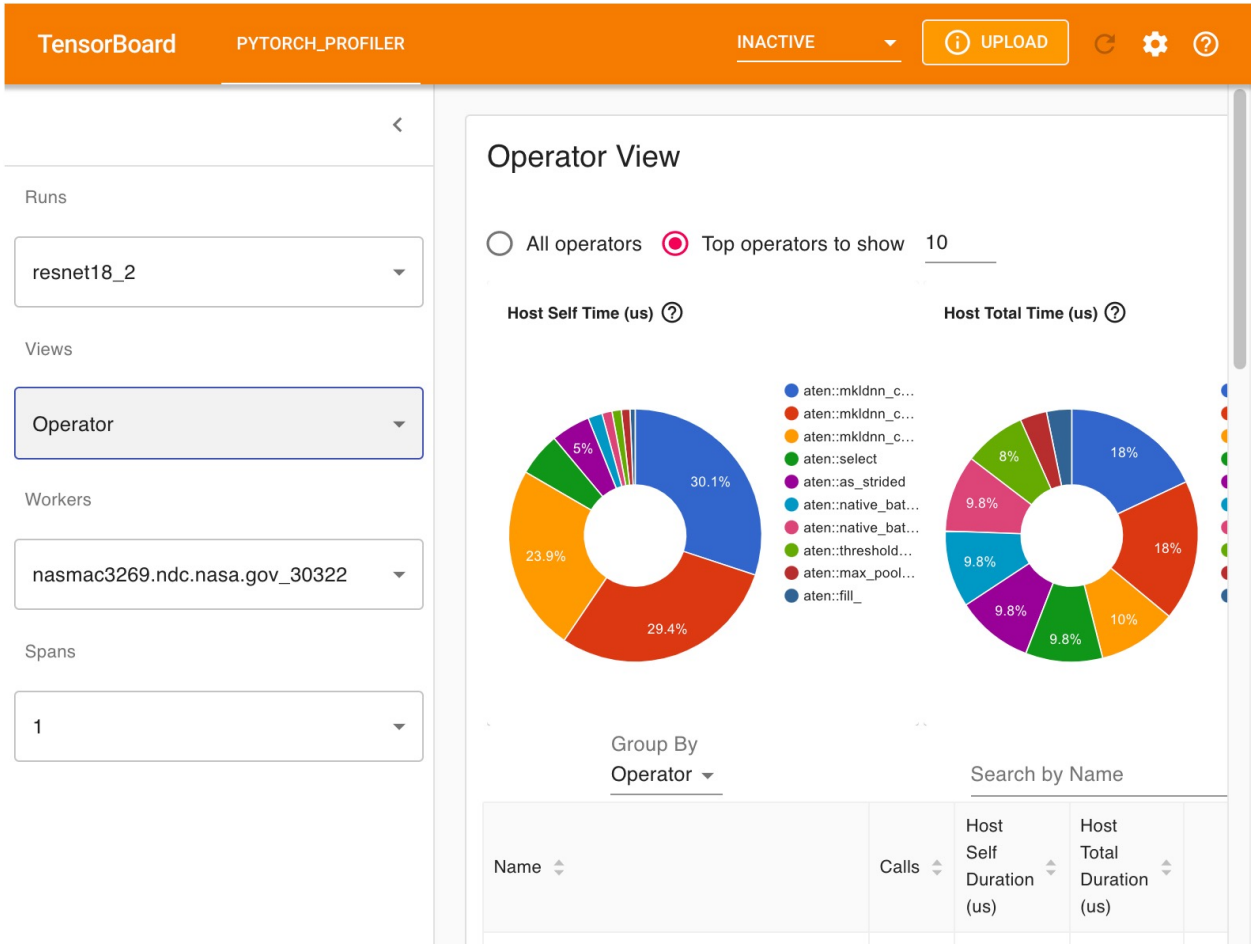
Overview

- Loading the log file will display the PyTorch profiling tool



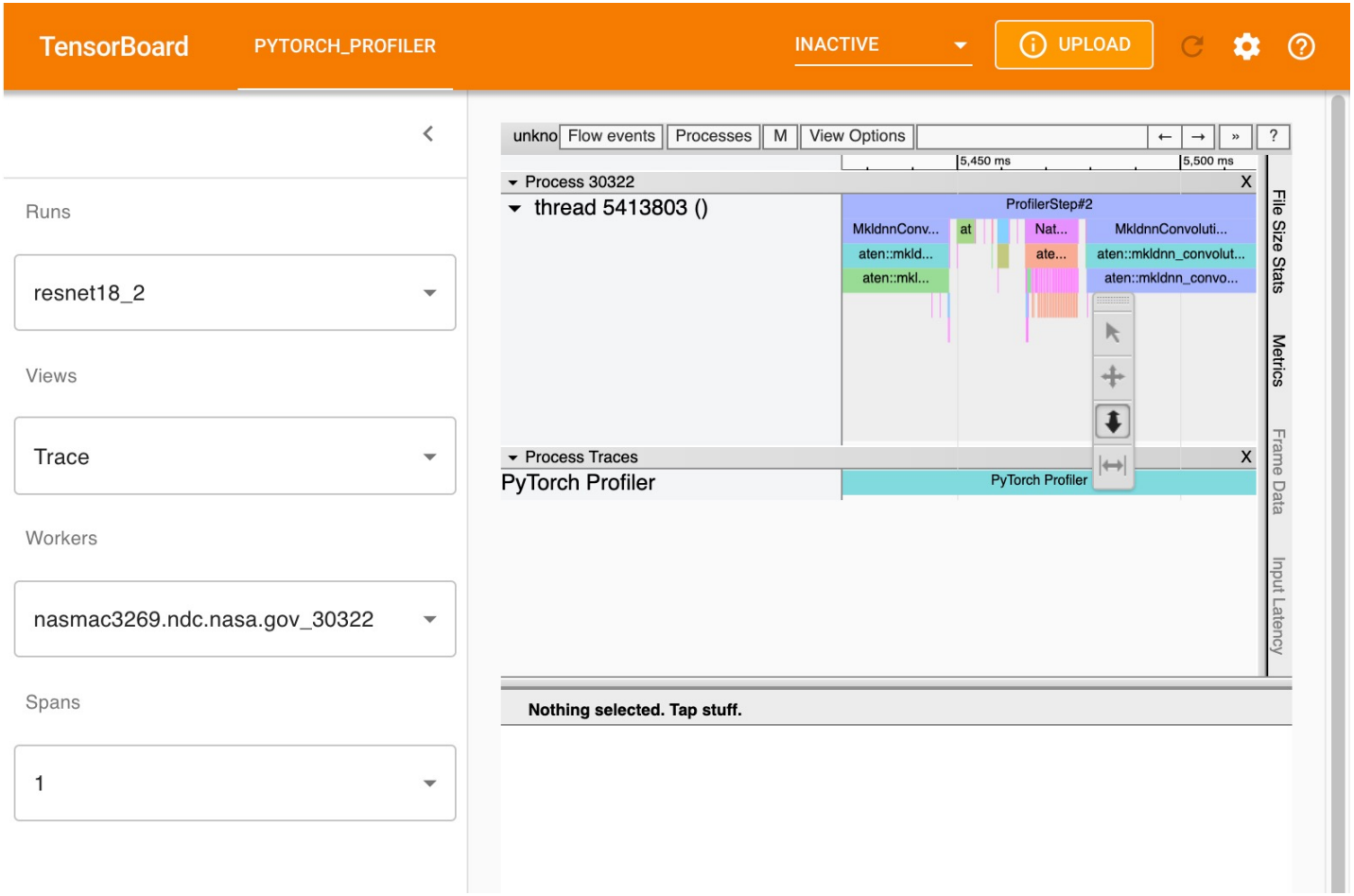
Operator View

Shows breakdown of usage, similar to TensorFlow stats page



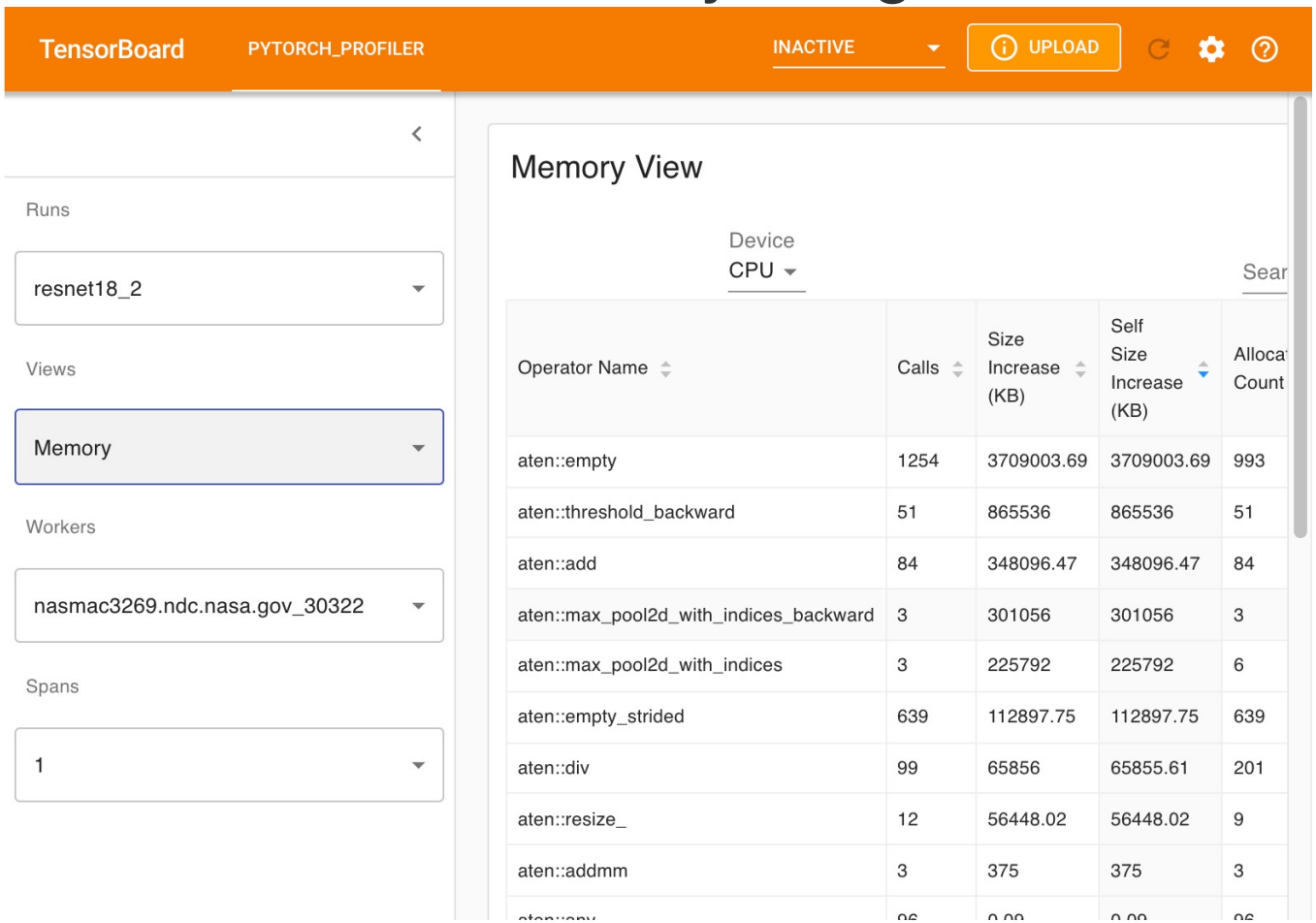
Trace View

Full timeline of all ops on each device



Memory View

Shows breakdown of memory usage

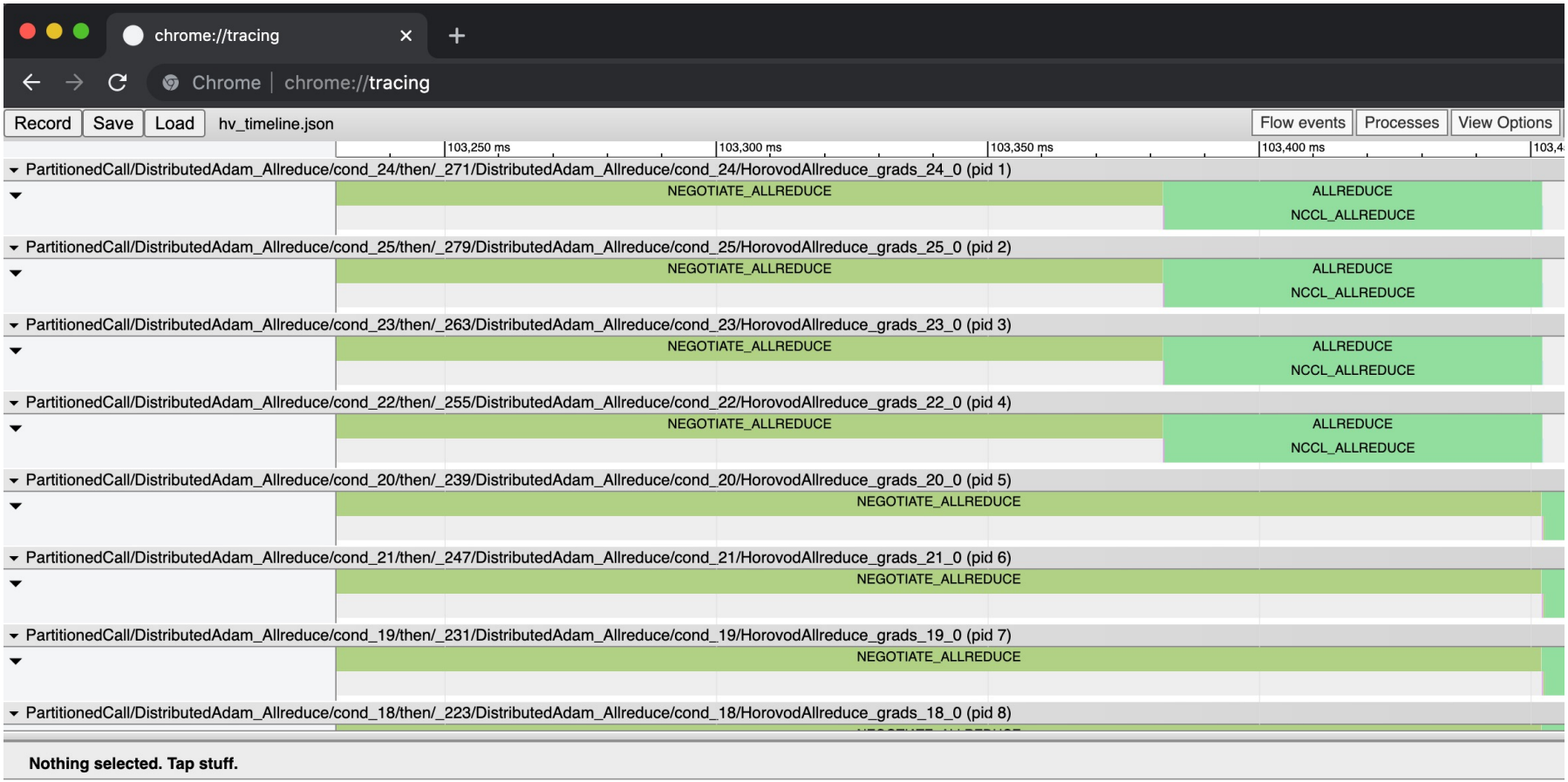


Profiling for Horovod

- Horovod does not support Tensorboard profiling
- Horovod can generate a timeline of all ops, which is the same information as the trace viewer in TensorBoard
  - Timeline saved by setting the HOROVOD_TIMELINE environment variable
    - export HOROVOD_TIMELINE=<filename>.json
- <filename>.json has to be moved to a local machine to view in a Chrome browser with the built-in tracing tool

Profiling for Horovod

- Viewable in a chrome browser with chrome://tracing/



Solving Common Bottleneck Issues



- Recommendations for some common data distribution/training bottlenecks
  - Data loading
  - Collective Reduce
  - GPU usage

Data Loading Bottleneck



- Data loading is the most common type of bottleneck
  - Change the way you load data into your model using tf.data.dataset
  - Prefetch, interleave map functions, and cache data
  - https://www.tensorflow.org/guide/data_performance#best_practice_summary
  - One of the main things to watch out for is properly sharding data when distributing models
  - When sharding data, try to shard by files rather than data

Collective Reduce Bottleneck

- Bottleneck during communication between nodes
  - Distributing the model might not be the best approach
  - Try increasing the batch size to do more computations on each device before doing collective reduce
  - Check the amount of data you have per epoch
  - For small amounts of data per epoch, probably better to run on single node

GPU Usage Bottleneck



- Other common techniques when data loading and communication are not the problem
  - Increase the number of GPUs used per node
  - Increase the number of nodes used
  - Decrease batch size if the ratio of GPU usage to other operations is very large

Summary

- Horovod and Dask can be used to distribute python code
  - Horovod is used specifically for training deep learning models
  - Dask is used for general distributed python programming
- Secure Jupyter Lab by using RFE (Reserved Front-End) or PBS interactive mode
- TensorBoard profiling can be used to debug and optimize model training
- We're here to help you get you started with Machine Learning

Questions?

A PDF and recording of this webinar will be available within 48 hours at:
http://www.nas.nasa.gov/hecc/support/training.html

Suggestions for future webinar topics are welcome

Contact us any time

dataanalytics@nas.nasa.gov