

Large Scale Deep Learning using the High-Performance Computing Library OpenMPI and DeepSpeed

Jennifer Davis
November 3, 2022



Agenda

- Review Parallel and Distributed Computing
- How MPI Works
- Starting your workspace and cloning the project
- DeepSpeed Introduction
- MPI + DeepSpeed
- Jupyter Notebook Code Walk-throughs

We will alternate between 15-20 minutes of lecture and 10 minutes to practice in your notebooks.

What is Parallel Computing?

- Type of computing in which many calculations or processes are carried out simultaneously on several nodes and threads within a CPU or GPU
- Limits
 - Available Parallelism
 - Load Balancing
 - Managing communication between processes
- Pros
 - Faster speeds for data processing and computations
 - Enabling Big Data and Machine Learning

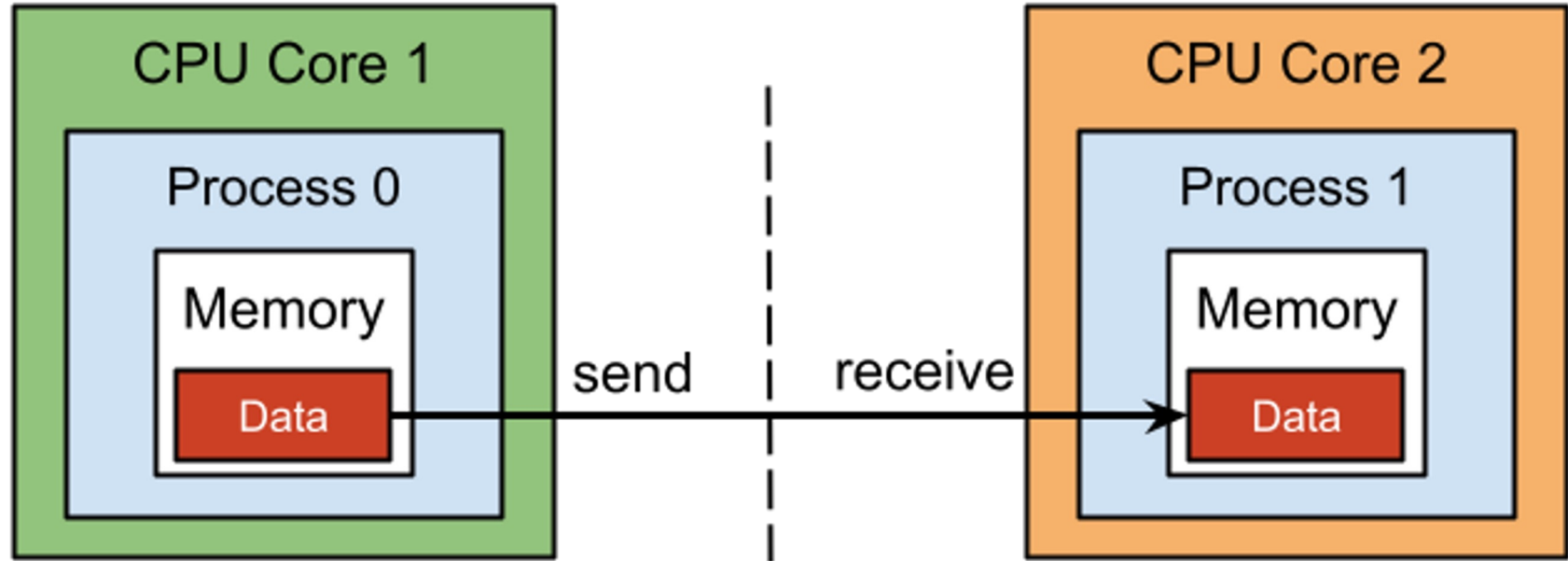
What is Distributed Computing?

- When multiple computers, CPUs or GPUs are used, tasks can be distributed among the cluster components unevenly and in parallel
- First rule: Don't distribute your workload on clusters unless you see a need to
- Most deep learning and big data machine learning will be more performant on a distributed system
- Reinforcement learning is computationally intense, so distributed computing is helpful
- With very large models (e.g. GPT-2, Megatron)

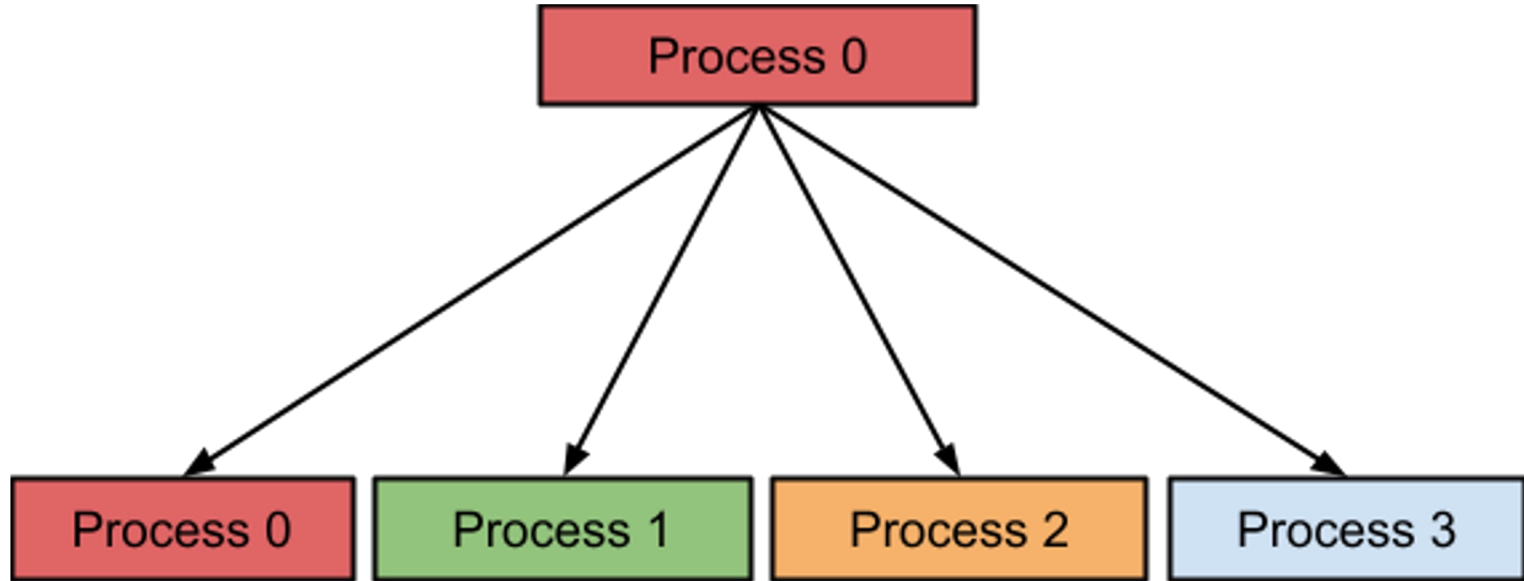
MPI Overview

- When an MPI program initiates, several processes gets created including processes that
 - runs on a node of a cluster,
 - has its own memory space,
 - and communicates with the other processes via MPI point-to-point and collective messaging capabilities.
- Other things to know about MPI
 - complies on different operating systems (windows, linux, mac)
 - supports parallel computation, distributed memory and shared memory
 - each process has its own memory space and executes independently (i.e. one process can fail why others continue to run)
 - Overhead for processes occur only once

MPI Process Communication



MPI Broadcast Distributions (Distributed Computing)



MPI on Demand Use Cases

- Parallel computation
- Parallel File Systems
- Communication / SuperPODs / High Performance Computing
- Image Processing Example
 - Read jpg or png
 - Partition image into parts and distribute to worker processes
 - Collect results
 - Write results into a file

Steps to setting up a workspace for MPI

- Name your workspace
 - Use a name you will remember that indicates whether it is distributed or not and the type of tool used (i.e., in this case MPI, DeepSpeed)
- Choose your workspace environment
- Ensure your datasets are available
- Choose your scheduler and compute hardware tier (suggest GPUs as use of CPUs will involve more code modification)
- Choose the number of workers

Cluster Environments on Domino

When using on-demand Clusters in Domino you will have two separate environments:

1. One for the workspace/job
2. One for the cluster workers/executors

We will do a walk-through of how to sign-up, clone the project and start your workspace

1. Our Teaching Assistants are available if you have trouble signing up for Domino Access, cloning your project or starting your workspace
2. Make sure to only use 1 GPU for your cluster

Cluster Settings

- Select the Attach Cluster option
- Set the number of Workers/Executors to 1, and hardware tiers for each to GPU
- Dedicated local storage will be mounted to /tmp and de-provisioned when the cluster is shut down
- Suggest setting the dedicated local storage to 100 GB as you will be downloading a computer vision dataset

Launch New Workspace

Environment & Hardware
Domino Analytic... and Small...

2 Data
demo-DFS-project

3 Compute Cluster
(optional)

Attach Compute Cluster

☐ none ☒ Spark ☒ Ray

Number of Workers 1 Max: 78

Worker Hardware Tier

Small (Kubernetes)
1 core · 1 GiB RAM · \$0.0011/min < 1 MIN

Head Hardware Tier

Small (Kubernetes)
1 core · 1 GiB RAM · \$0.0011/min < 1 MIN

Cluster Compute Environment

Ray 1.3 GPU Base

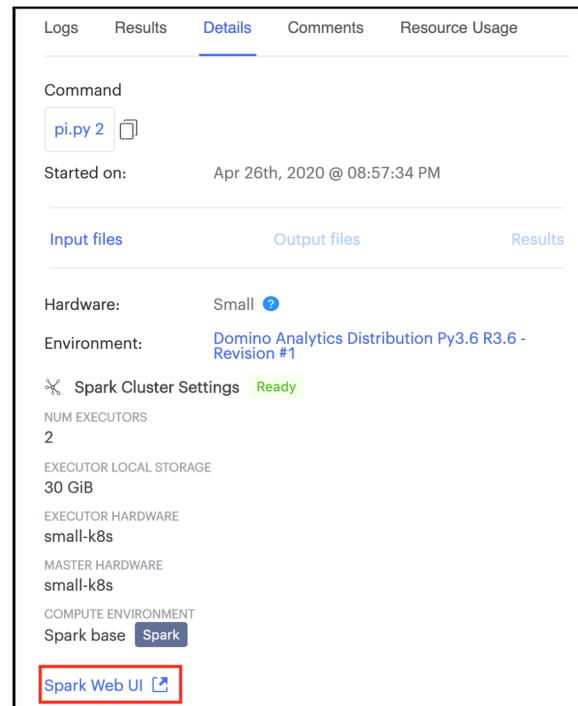
i Your workspace environment must have the correct Ray Client libraries to interact with this cluster. [Read more](#)

☐ **Dedicated local storage per worker** 0 GiB

[Back](#) [Launch](#)

Other ways to Run Distributed Compute: Batch Jobs

- Set up the cluster when running a Job from the quick-action, files page, or Job dashboard
- MPI clusters do not have the an user interface as Spark, Ray and Dask do
- Currently only tested with Jupyter Lab and VSCode



The screenshot displays the 'Details' tab of a job in the Domino Analytics interface. At the top, there are tabs for 'Logs', 'Results', 'Details' (selected), 'Comments', and 'Resource Usage'. Below the tabs, the 'Command' section shows 'pi.py 2' with a copy icon. The 'Started on' timestamp is 'Apr 26th, 2020 @ 08:57:34 PM'. A horizontal separator follows, with links for 'Input files', 'Output files', and 'Results'. Below this, the 'Hardware' is listed as 'Small' with a help icon. The 'Environment' is 'Domino Analytics Distribution Py3.6 R3.6 - Revision #1'. A 'Spark Cluster Settings' section shows a green 'Ready' status. Below this, several configuration details are listed: 'NUM EXECUTORS' is 2, 'EXECUTOR LOCAL STORAGE' is 30 GiB, 'EXECUTOR HARDWARE' is small-k8s, and 'MASTER HARDWARE' is small-k8s. The 'COMPUTE ENVIRONMENT' is 'Spark base' with a 'Spark' button. At the bottom, a 'Spark Web UI' link with an external icon is highlighted with a red rectangle.

Logs Results **Details** Comments Resource Usage

Command
pi.py 2

Started on: Apr 26th, 2020 @ 08:57:34 PM

Input files Output files Results

Hardware: Small

Environment: Domino Analytics Distribution Py3.6 R3.6 - Revision #1

Spark Cluster Settings Ready

NUM EXECUTORS
2

EXECUTOR LOCAL STORAGE
30 GiB

EXECUTOR HARDWARE
small-k8s

MASTER HARDWARE
small-k8s

COMPUTE ENVIRONMENT
Spark base Spark

Spark Web UI

Setting up your workspace

MPI Examples (Parallelism vs. Distributed): Calculating Pi

- MPI distributes a computation using a message passing interface and shared memory
- We will calculate Pi two ways using MPI (two separate files for submission to the cluster)
- How the code is written matters: calculates pi more quickly if run in distributed mode rather than simple parallelism

Calculating Pi: A Hello World Example for Distributed Computing

- The number pi is a ratio obtained from defining the area with a circle. If the diameter and the circumference of a circle are known, the value of pi will be as $\pi = \text{circumference of a circle} / \text{diameter of a circle}$.
- We can take several samples from the calculations of pi and reduce them to a final number in order to estimate pi more closely
- For this example we will use the Leibniz formula (which is easily distributed) and which you can find out more about it [here](#).

Calculate Pi Using Leibniz Method with and without MPI

- First we will submit the code file 'as-is' and time it
- Next we will submit the code 'as-is' to the cluster
- Notice whether we are using simple parallelism or running in a distributed mode
- What differences do you see

```
import numpy as np
import math

N=10**8

h = 1.0 / N; s = 0.0
for i in range(N):
    x = h * (i + 0.5)
    s += 4.0 / (1.0 + x**2)
    estimated_pi = s * h

print(estimated_pi)
```


Hello World Example to Calculate Pi with Large Numbers in a Distributed Fashion

```
from mpi4py import MPI
import numpy
import sys

comm = MPI.COMM_SELF.Spawn(sys.executable,
                           args=['cpi.py'],
                           maxprocs=3)

N = numpy.array(10**8, 'i')
comm.Bcast([N, MPI.INT], root=MPI.ROOT)
PI = numpy.array(0.0, 'd')
comm.Reduce(None, [PI, MPI.DOUBLE],
            op=MPI.SUM, root=MPI.ROOT)
print(PI)

comm.Disconnect()
```

```
#!/usr/bin/env python
from mpi4py import MPI
import numpy

comm = MPI.Comm.Get_parent()
size = comm.Get_size()
rank = comm.Get_rank()
name = MPI.Get_processor_name()

N = numpy.array(0, dtype='i')
comm.Bcast([N, MPI.INT], root=0)
h = 1.0 / N; s = 0.0
for i in range(rank, N, size):
    x = h * (i + 0.5)
    s += 4.0 / (1.0 + x**2)
PI = numpy.array(s * h, dtype='d')
comm.Reduce([PI, MPI.DOUBLE], None,
            op=MPI.SUM, root=0)

comm.Disconnect()
```

MPI Distributed Code Does the Following

- Broadcasts the Numpy points across workers
- Collects calculations from the workers
- Reduces the answer to a final approximation of pi
- Let's take a look at our notebook tutorial to learn more

Jupyter Notebook Walk-through: Calculate Pi

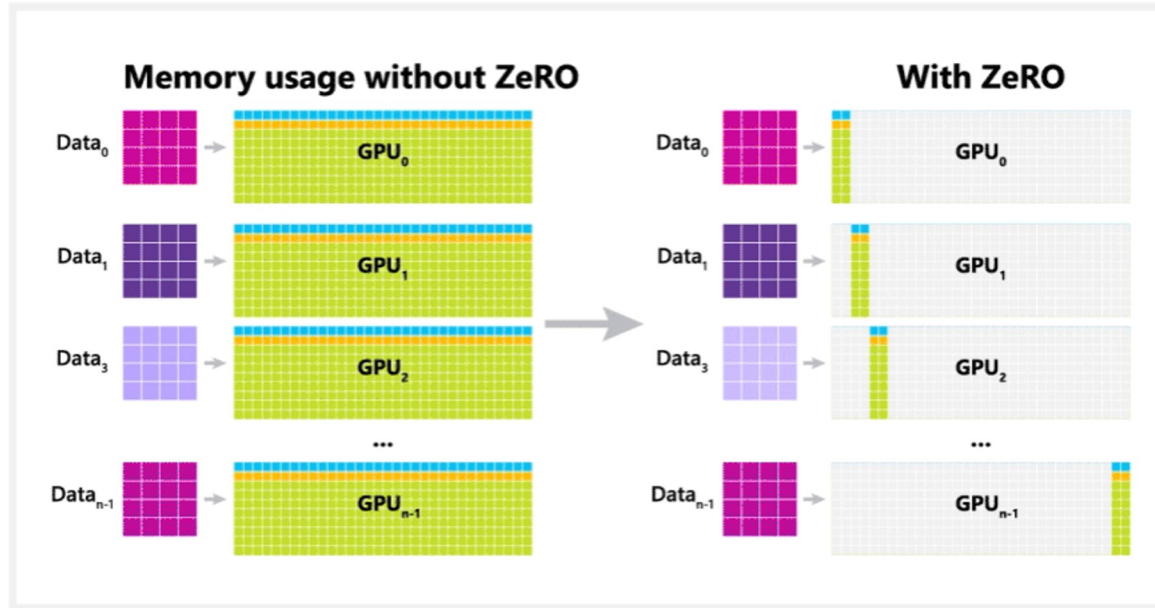
What is DeepSpeed?

- “DeepSpeed is a **deep learning optimization library that makes distributed training easy, efficient, and effective**. 10x Larger Models | 10x Faster Training | Minimal Code Change.” - Microsoft
- DeepSpeed is a distributed computing engine which acts as a wrapper around pytorch neural network models and includes check-pointing
- DeepSpeed contains:
 - ZeRO Optimizations,
 - Scheduler
 - Autotuning, optimizers
 - Is appropriate for quantized models as well as massive, trained models such as GPT-2 and Megatron
- DeepSpeed uses MPI to distribute, broadcast and employ workers

Why use a MPI Cluster with DeepSpeed?

- Larger datasets
- Massive Models
- Efficient scheduling by DeepSpeed on MPI
- Load balancing by DeepSpeed on MPI
- Faster computation times

DeepSpeed + ZeRO



DeepSpeed Use Cases

- Training massive models using transfer learning or from scratch (GPT-2, Megatron)
- Inference on massive models (e.g. forward propagations)
- Native use of Quantized of Models
- Hypermeter Tuning

How DeepSpeed Integrates into the AI Science Lifecycle

- Model Development in Pytorch, Pytorch Lightning or Transformers
- Model Training – use DeepSpeed for scaling and speeding up training to reasonable times, can use billions of parameters for training
- Model Inference – use DeepSpeed forward propagation from a trained model to infer outputs

Standard Transformers and DeepSpeed+Transformers

Transformers Library:

- Python library of language models
- Built on Pytorch
- Models are made to train on single GPU or CPU
- High memory requirements for most models, so hardware may be more expensive

DeepSpeed + Transformers Library:

- Python library which optimizing training using MPI and the ZeRO training formulas based on Distributed Data Parallel
- Built to complement Pytorch and integrates tightly with transformers library and pytorch lightning
- Built to run on MPI so can use cluster to share data load
- Speeds up most calculations significantly
- Can handle larger parameter models without TPU requirements

Overview of DeepSpeed Functions: Training Set-up

See all training functions in `deepspeed` by going to documentation
[DeepSpeed Training Initialization](#)

Highlights / Gotcha's:

- `deepspeed.initialize` – initialize arguments for training parameters, models etc. Does not automatically run training on a cluster!
- `deepspeed.init_distributed` – looks for pytorch backend (remember we set the back-end in our code), and can perform MPI discovery
- `--hostfile` should be used to provide the MPI host file doc

Overview of DeepSpeed Functions: Inference Set-up

See all the functions in `deepspeed` by typing [DeepSpeed Inference Engine](#)

Highlights / Gotcha's:

- `deepspeed.init_inferences`– initializes DeepSpeed inference engine
- `deepspeed.InferenceEngine.forward`– implements forward propogation
- `--hostfile` should be used to provide the MPI host file doc

Examples in the Second Tutorial

Workshop Examples

- Inference to summarize text using DeepSpeed
- Inference to translate text to French using DeepSpeed

Supplemental – these take too long to run but you can try them after the workshop

- Training a model to summarize news articles using CNN/Daily Mail dataset
- Training a model to recognize protein sequences and conformations
- Will show output but we will not run during the workshop

DeepSpeed Configuration Files and Auto-Tuning

```
▼ root:
  ► fp16:
  ► bf16:
  ► optimizer:
  ► scheduler:
  ▼ zero_optimization:
    stage: 3
    ► offload_optimizer:
    ► offload_param:
      overlap_comm: true
      contiguous_gradients: true
      sub_group_size: 1000000000
      reduce_bucket_size: "auto"
      stage3_prefetch_bucket_size: "auto"
      stage3_param_persistence_threshold: "auto"
      stage3_max_live_parameters: 1000000000
      stage3_max_reuse_distance: 1000000000
      stage3_gather_16bit_weights_on_model_save: true
    gradient_accumulation_steps: "auto"
    gradient_clipping: "auto"
    steps_per_print: 2000
    train_batch_size: "auto"
    train_micro_batch_size_per_gpu: "auto"
    wall_clock_breakdown: false
```

- Basic Parameters
- Stage 3 ZeRO Training
- Can fine-tune
 - fp16
 - bf16
 - scheduler
 - optimizer

Proteomics Example: 8 minutes to do Transfer Learning a Massive Model (>2 Billion parameters)

```
[1]: %%time
!deepspeed run_summarization_og.py \
--model_name_or_path Rostlab/prot_t5_xl_uniref50 \
--output_dir /mnt/data/DeepSpeed_Tutorial \
--overwrite_output_dir True \
--max_train_samples 5 \
--max_eval_samples 5 \
--max_source_length 512 \
--max_target_length 128 \
--val_max_target_length 128 \
--do_train --do_eval \
--do_predict \
--num_train_epochs 1 \
--warmup_steps 500 \
--predict_with_generate \
--save_steps 0 \
--eval_steps 1 \
--group_by_length \
--train_file train.csv \
--test_file train_small.csv \
--validation_file train_small.csv \
--text_column text_column \
--summary_column summary_column \
--overwrite_cache True \
--cache_dir /mnt/data/DeepSpeed_Tutorial \
--preprocessing_num_workers 9 \
--do_predict \
--deepspeed ds_config.json
```

[2022-06-02 21:28:36.053] [WARNING] [runner.nv:159:fetch hostfile] Unable to find hostfile. will proceed with

Proteomics Example: Transfer Learning a Massive Model Output

```

[2022-06-02 21:29:52,914] [INFO] [engine.py:3229:save_model] Saving model weights to /mnt/d:
**** train metrics ****
epoch                = 1.0
train_loss           = 0.4004
train_runtime        = 0:00:10.26
train_samples        = 5
train_samples_per_second = 0.487
train_steps_per_second = 0.097
06/02/2022 21:31:48 - INFO - __main__ - *** Evaluate ***
[INFO|trainer.py:2653] 2022-06-02 21:31:48,935 >> ***** Running Evaluation *****
[INFO|trainer.py:2655] 2022-06-02 21:31:48,935 >> Num examples = 5
[INFO|trainer.py:2658] 2022-06-02 21:31:48,935 >> Batch size = 8
100%|████████████████████████████████████████████████████████████████████████████████| 1/1 [00:01<00:00, 1.31s/it]
**** eval metrics ****
epoch                = 1.0
eval_gen_len         = 127.0
eval_loss            = 0.1718
eval_rouge1          = 100.0
eval_rouge2          = 100.0
eval_rougeL          = 100.0
eval_rougeLsum       = 100.0
eval_runtime         = 0:02:27.31
eval_samples         = 5
eval_samples_per_second = 0.034
eval_steps_per_second = 0.007
06/02/2022 21:34:16 - INFO - __main__ - *** Predict ***
[INFO|trainer.py:2653] 2022-06-02 21:34:16,357 >> ***** Running Prediction *****
[INFO|trainer.py:2655] 2022-06-02 21:34:16,357 >> Num examples = 5
[INFO|trainer.py:2658] 2022-06-02 21:34:16,357 >> Batch size = 8
0%|████████████████████████████████████████████████████████████████████████████████| 0/1 [00:00<?, ?it/s]***** predict met
predict_gen_len      = 127.0
predict_loss         = 0.1718
predict_rouge1       = 100.0
predict_rouge2       = 100.0
predict_rougeL       = 100.0
predict_rougeLsum    = 100.0
predict_runtime      = 0:02:24.07
predict_samples      = 5
predict_samples_per_second = 0.035
predict_steps_per_second = 0.007
100%|████████████████████████████████████████████████████████████████████████████████| 1/1 [00:01<00:00, 1.59s/it]
[2022-06-02 21:36:44,695] [INFO] [launch.py:210:main] Process 1430 exits successfully.
[2022-06-02 21:36:45,695] [INFO] [launch.py:210:main] Process 1431 exits successfully.
[2022-06-02 21:36:46,697] [INFO] [launch.py:210:main] Process 1429 exits successfully.
[2022-06-02 21:36:46,697] [INFO] [launch.py:210:main] Process 1432 exits successfully.
CPU times: user 5.38 s, sys: 2.13 s, total: 7.51 s

```

Intermediate Tutorial: Code Walk-through

Computer Vision Example: Zero Redundancy Optimization Advantages

- Optimizes memory while improving speed of training very large models (100 Billion + parameters)
- Allows scaling on GPUs with high computational granularity and efficiency
- Original analysis on memory and communication volume suggested DeepSpeed can scale models beyond 1 Trillion parameters on available GPUs

Advanced Tutorial: Computer Vision Example

- Explore using DeepSpeed to perform training and inference on the CIFAR dataset
- Explore running DeepSpeed code on MPI versus the DeepSpeed framework
- Explore the output of our CIFAR example
- Demonstrate DeepSpeed on ZeRO, ZeRO2 and ZeRO3 algorithms

Advanced Tutorial: Code Walk-through

Field Data Science Practice

- Assists at all phases of the project
 - Ideation
 - Research and development
 - Execution
- Expert data scientists and field engineers help you leverage Domino for success
 - Dedicated team of field data scientists and data engineers focus on your success
- We provide consultation services:
 - Best practices for data science on Domino
 - Developing Proof of Concepts and Minimum Viable Products
 - Execution of complex projects to solve business problems
 - Knowledge transfer upon project completion



FDS and FE Team



**Josh
Poduska,**
Chief Field
Data Scientist



Jennifer Davis,
Staff Field
Data Scientist



**Melanie
Veale, Field**
Data Scientist



**Odette
Harary, Field**
Data Scientist



**Deepika
Vadlamudi,**
Field Engineer

Thank you!

More questions or comments:

Jennifer Davis

jennifer.davis@dominodatalab.com

or LinkedIn:

<https://www.linkedin.com/in/drjenniferdavis/>