

# Profiling Deep Learning

Denis Boyda, Corey Adams, Filippo Simini

2022 ALCF Computational Performance Workshop, May 24-26

# Outline

- What is profiling
- Motivation to profile
- Some frequent problems and solutions
- Common Profiler APIs
- Understanding profiler output
- (new) PyTorch Profiler view



# What is profiling?

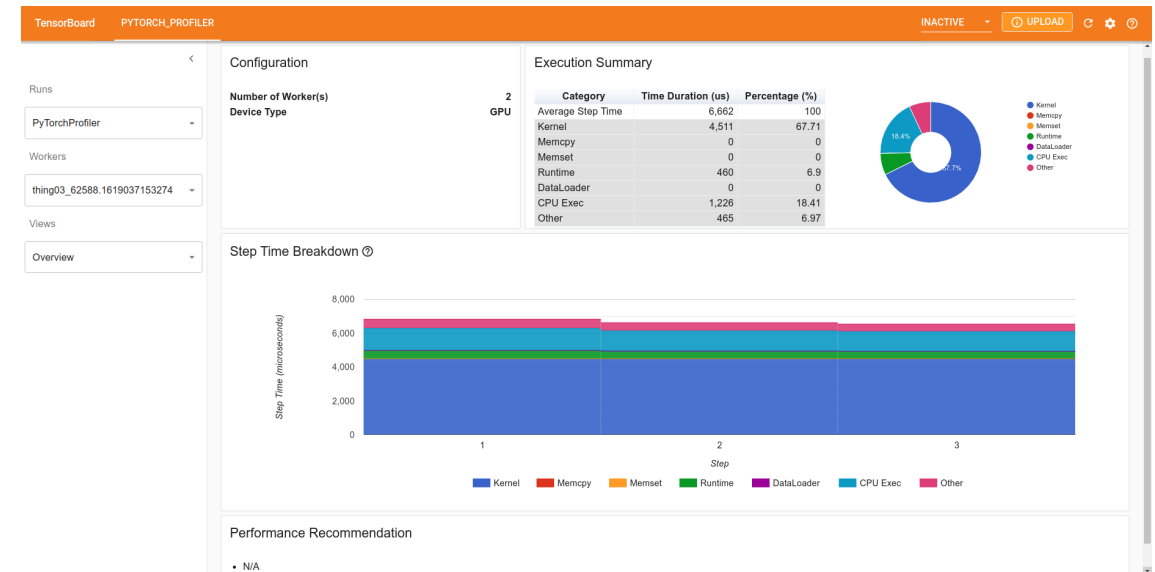
Profiling is a diagnostic tool

- Measuring time/memory cost
- Identifying bottleneck
- Tracking call stack

```
181 function calls (169 primitive calls) in 3.434 seconds
```

Ordered by: cumulative time

```
ncalls  tottime  percall  cumtime  percall  filename:lineno(function)
10/1    0.000    0.000    3.434    3.434  module.py:866(_call_impl)
1       0.078    0.078    3.434    3.434  v2.py:21(forward)
1       0.000    0.000    2.817    2.817  <_array_function__ internals>:2(argwhere)
4/1    0.000    0.000    2.817    2.817  {built-in method numpy.core.multiarray_umath.implement_array_function}
1       0.005    0.005    2.817    2.817  numeric.py:537(argwhere)
2       0.000    0.000    2.812    1.406  fromnumeric.py:52(_wrapfunc)
1       0.000    0.000    1.476    1.476  <_array_function__ internals>:2(nonzero)
1       0.000    0.000    1.476    1.476  fromnumeric.py:1816(nonzero)
1       1.476    1.476    1.476    1.476  {method 'nonzero' of 'numpy.ndarray' objects}
1       0.000    0.000    1.335    1.335  <_array_function__ internals>:2(transpose)
1       0.000    0.000    1.335    1.335  fromnumeric.py:601(transpose)
1       0.000    0.000    1.335    1.335  fromnumeric.py:39(_wrapit)
1       0.000    0.000    1.335    1.335  _asarray.py:14(asarray)
1       1.335    1.335    1.335    1.335  {built-in method numpy.array}
1       0.348    0.348    0.348    0.348  {method 'cuda' of 'torch._C._TensorBase' objects}
1       0.188    0.188    0.188    0.188  {method 'cpu' of 'torch._C._TensorBase' objects}
1       0.003    0.003    0.003    0.003  {method 'item' of 'torch._C._TensorBase' objects}
```



# Motivation to profile and optimization

What is the purpose of profiling?

- finding performance issues
- optimizations and tuning
- fine-tuning

Do you need to deeply profile and optimize your script?

- training vs evaluation
  - Are you developing your model or you already know the solution?
  - Is your accuracy increasing with scaling?
- 
- Is speed up worth your affords?
    - Is GPU utilization high?
    - Do you expect high improvements?
    - Are you planning to scale?



**“Premature optimization is the root of all evil!”**  
**@Donald Knuth**

# Some optimization advices

- large kernel utilization
  - check which functions take a majority of compute
- slow IO
  - Use designated PT/TF data loaders
  - Use multithreading
  - Move preprocessing to GPU ([Nvidia DALI](#))
- (extra) copy CPU ↔ GPU
  - Move all operations to device
    - Rewrite your code with using only PT/TF tensors
    - Use Numba / CuPy
  - Overlap copy and computation
  - Use asynchronous copy
- low GPU utilization
  - Increase batch size
- too high precision
  - Reduce precision to float/half?
  - use autotuning mixed precision in PT

# Common profiler APIs

- Context manager

```
with profiler.profile() as prof:  
    model(inputs)
```

```
with torch.profiler.profile(  
    activities=[torch.profiler.ProfilerActivity.CPU,  
               torch.profiler.ProfilerActivity.CUDA],  
    schedule=torch.profiler.schedule(wait=0, warmup=1, active=3),  
    on_trace_ready=torch.profiler.tensorboard_trace_handler(dir_name)  
    ) as prof:
```

```
for _ in range(8):  
    model(inputs)
```

- Python module

```
python -m line_profiler train_GAN.py.lprof
```

```
python -m torch.utils.bottleneck example2/v3.py
```


- Decorator

```
@profile  
def train_loop(batch_size, n_training_iterations, models, opts, global_size):  
  
    logger = logging.getLogger()  
  
    rank = hvd.rank()  
    for i in range(n_training_iterations):
```

# Understanding profiler output

no subfunction calls included

time including all subroutines calls

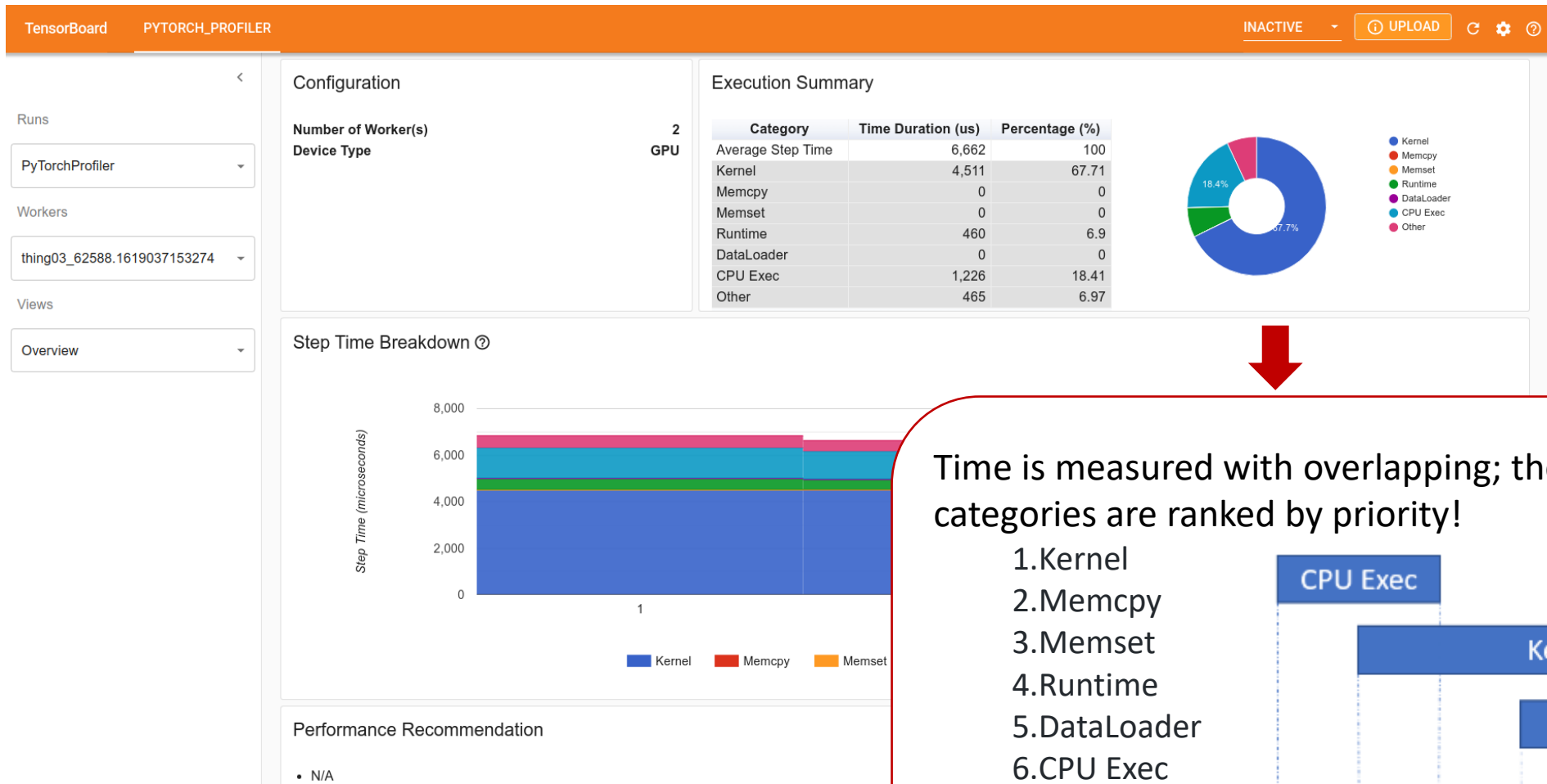


Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::nonzero	96.00%	15.948ms	96.08%	15.962ms	15.962ms	1
LABEL1: linear pass	0.82%	136.417us	2.67%	442.910us	442.910us	1
aten::addmm	0.78%	130.082us	0.93%	153.704us	38.426us	4
LABEL2: masking	0.34%	56.329us	96.98%	16.112ms	16.112ms	1
aten::threshold	0.26%	43.308us	0.33%	54.649us	13.662us	4

Self CPU time total: 16.613ms

- additional metrics can be shown
- reference to the line in source files can be added
- one can sort by different metrics

# (new) PyTorch Profiler view



Time is measured with overlapping; the categories are ranked by priority!

1. Kernel
2. Memcpy
3. Memset
4. Runtime
5. DataLoader
6. CPU Exec
7. Other

The Gantt chart illustrates overlapping execution on a timeline from 1 to 7 seconds. CPU Exec runs from 1s to 3s. Memcpy runs from 4s to 6s. Kernel runs from 2s to 7s, overlapping with both CPU Exec and Memcpy.



**See hands-on exercises**