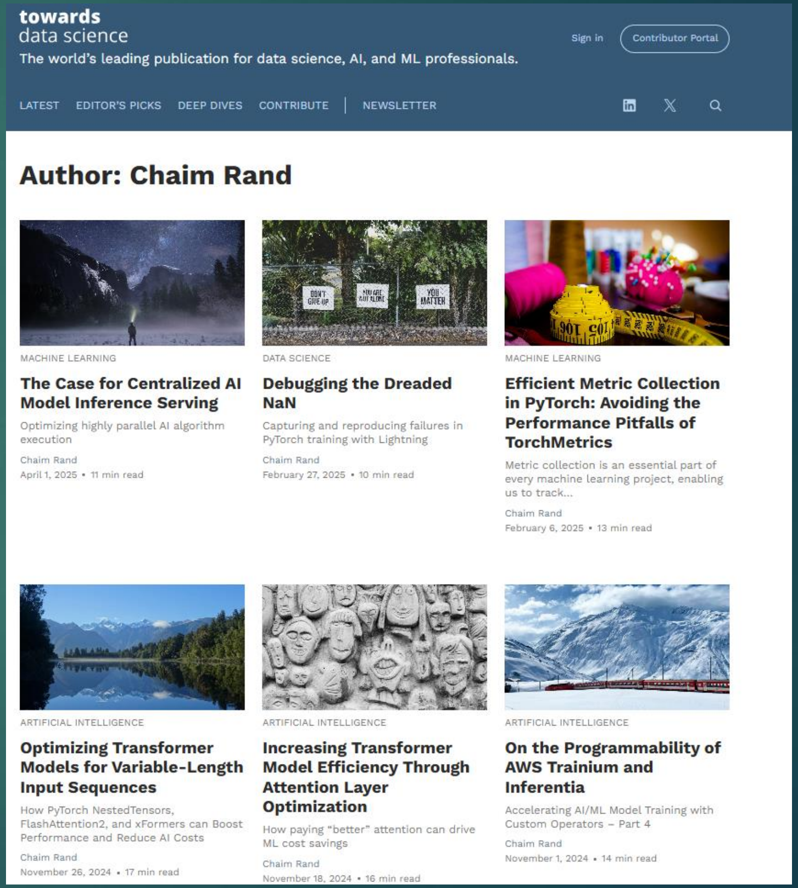
PyTorch Model Performance Analysis and Optimization

Chaim Rand

April25

Chaim Rand

- AI/ML/CV Algorithm Developer
- Areas of interest
  - Cloud Based AI/ML
  - AI/ML Model Performance Optimization
- Blogging Hobbyist
  - https://towardsdatascience.com/author/chaimrand/
  - https://chaimrand.medium.com/



Agenda

- Motivation
- A Typical Model Training Step
- Optimization Methodology
- Examples

Motivation

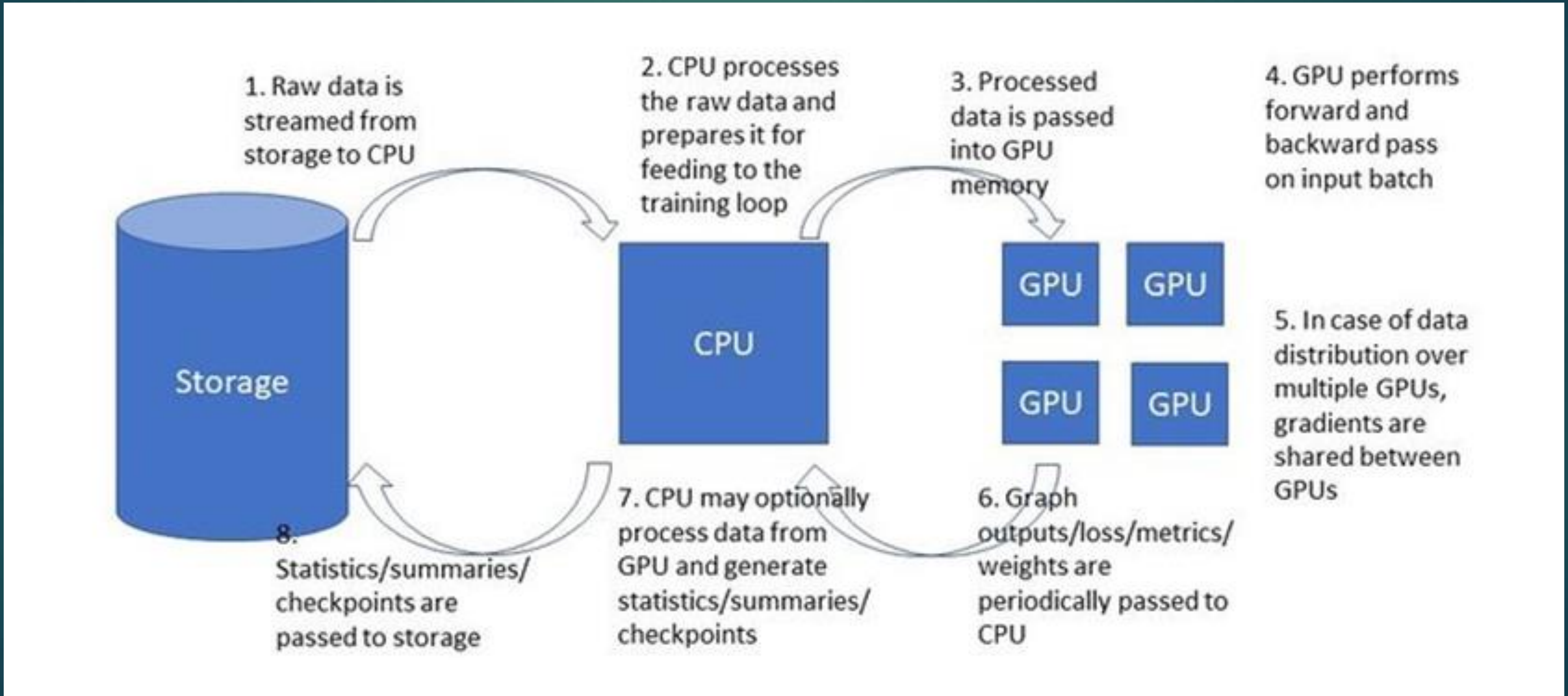
- AI models are resource intensive and expensive to train/run
  - E.g. Amazon EC2 P5 Instance (8 H100) is ~$100 per hour
- ML workloads are prone to performance bottlenecks
- Simple optimization techniques can deliver significant acceleration and cost savings

Key Messages:

→ **AI/ML developers must take responsibility for the runtime performance of their workloads**

→ **You don't need to be a CUDA expert to see results**

Training Pipeline



Optimization Methodology

- Objective - Maximize throughput (samples per second)
- Use performance profilers to measure resource utilization and identify bottlenecks
- → **Integrate into model development lifecycle**

- **Profile**

identify bottlenecks in the pipeline and under-utilized resources

- **Optimize**

address bottlenecks and increase resource utilization

- **Repeat**

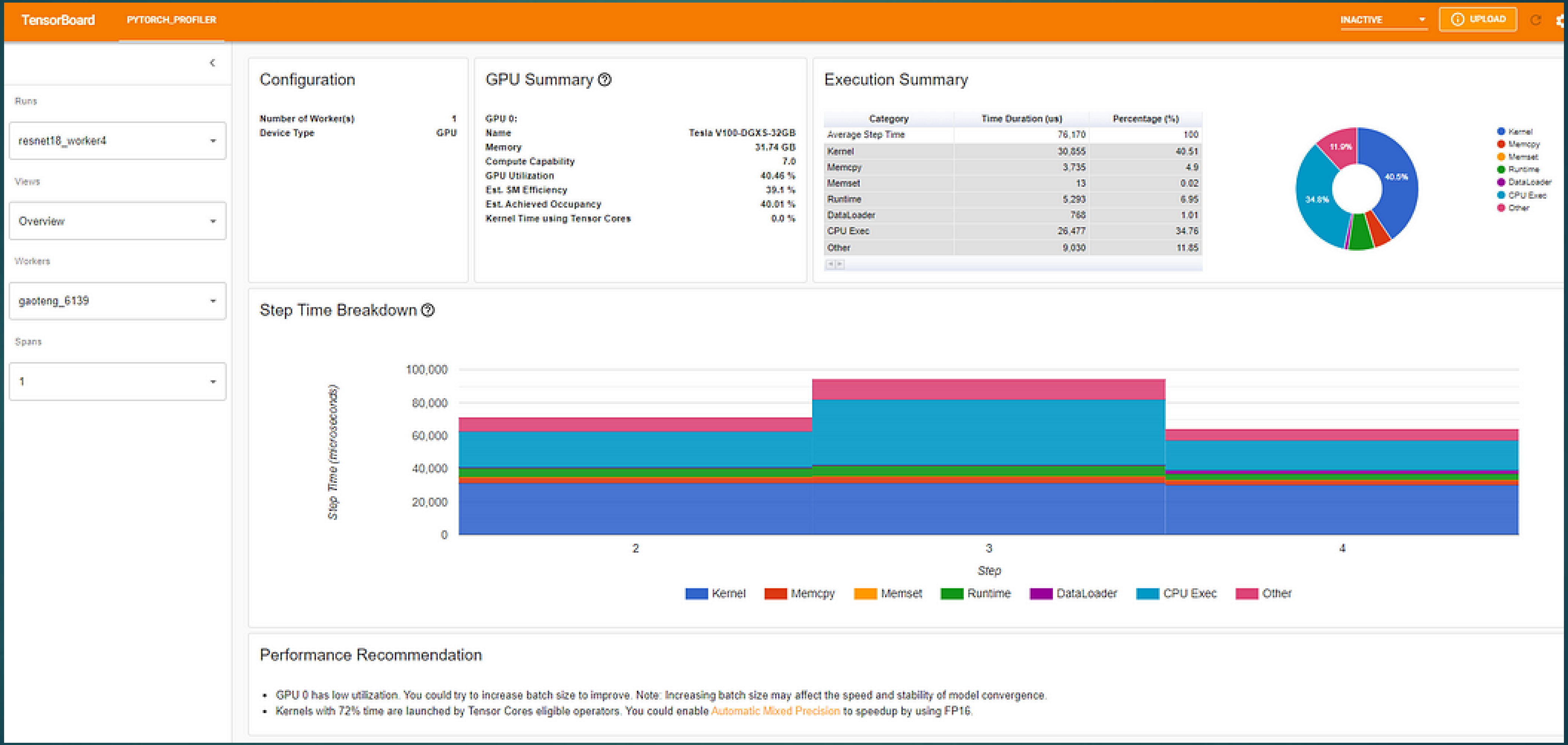
until satisfied with the throughput and resource utilization

PyTorch Profiler

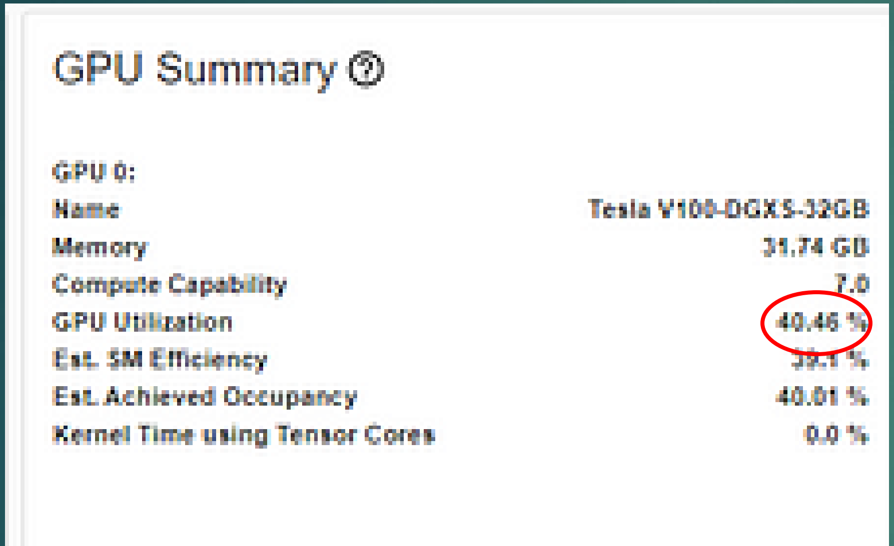
- Relatively easy to use
- View results with TensorBoard (recently deprecated), Perfetto or Chrome

```python
prof = torch.profiler.profile(
        schedule=torch.profiler.schedule(wait=1, warmup=1, active=3, repeat=1),
        on_trace_ready=torch.profiler.tensorboard_trace_handler('./log/resnet18'),
        record_shapes=True,
        with_stack=True)
prof.start()
for step, batch_data in enumerate(train_loader):
    prof.step()
    if step >= 1 + 1 + 3:
        break
    train(batch_data)
prof.stop()
```

Summary View

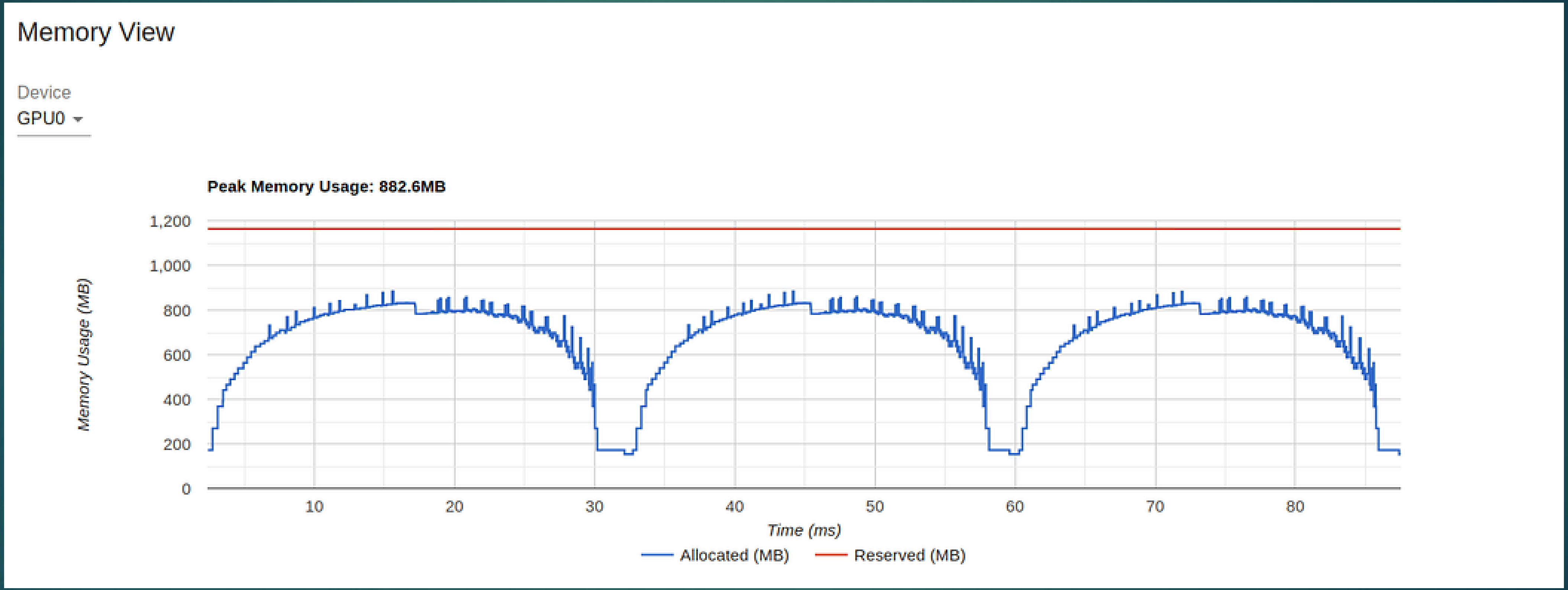


Summary View

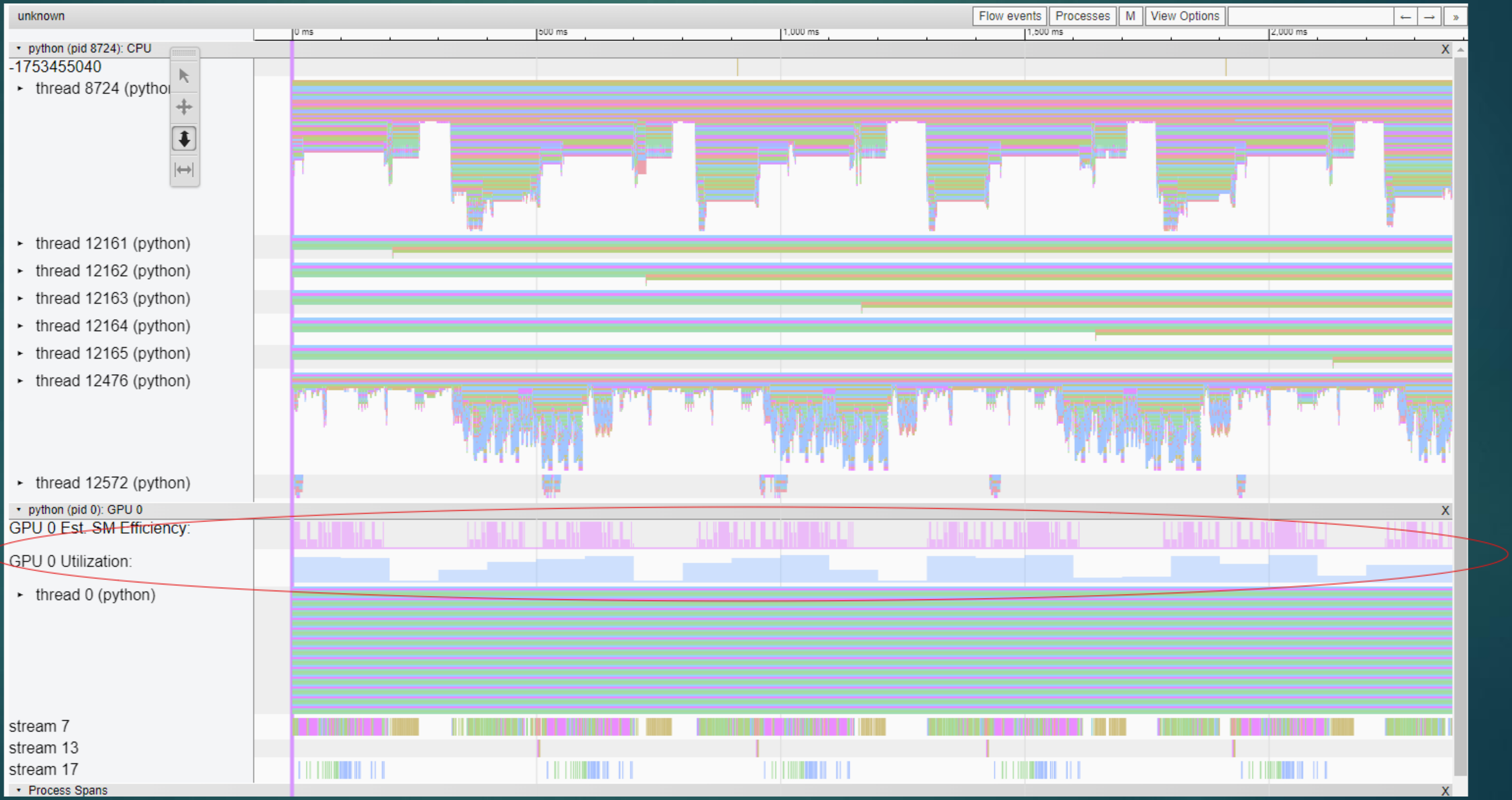


Challenge: Aim for > 80-90% utilization (exceptions apply)

Memory View


Memory View
Device
GPU0
Peak Memory Usage: 882.6MB
1,200
1,000
800
600
400
200
0
Memory Usage (MB)
10
20
30
40
50
60
70
80
Time (ms)
Allocated (MB)
Reserved (MB)

Trace Viewer



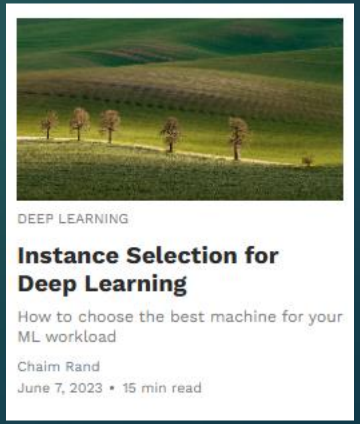
Optimization Tips, Tricks, and Techniques (TTTs)

TTT1 - Parallelize CPU and GPU Activities

- By default, PyTorch runs the data preprocessing pipeline and the GPU training step *sequentially*
  - Leads to resource idle time, AKA "GPU starvation"
- Solution: set num_workers > 0 in PyTorch dataloader to run data prep (on CPU) and training (on GPU) in parallel
  - Recommended to set num_workers to the number of CPU cores
  - Tune for optimal performance

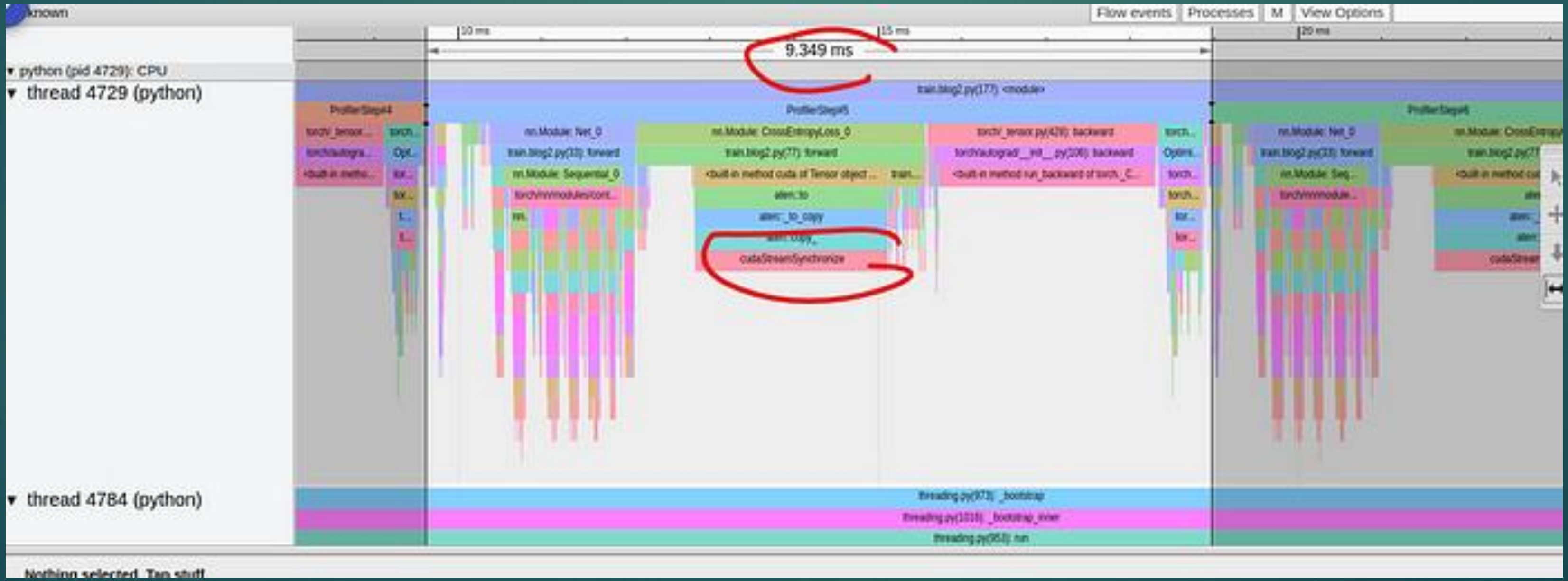
TTT2 – Optimize Instance Selection

- Optimizing instance selection for your workload to increase cost performance
- E.g., Don't use a multi-GPU machine (e.g., Amazon EC2 p5) if you intend to train on 1 GPU
- E.g., Choose an instance with an optimal CPU/GPU compute ratio
- E.g., Consider AI-specific ASICS (e.g., Google Cloud TPU, AWS Trainium/Inferentia)
- E.g., You may require machines with direct GPU2GPU links



TTT3 – Minimize Host to Device Sync Events

- Generally, corelates with GPU idle time
- Maximize asynchrony and minimize dependence between the CPU and GPU



TTT3 – Minimize GPU-CPU Sync Events

- Generally, corelates with GPU idle time
- Maximize asynchrony and minimize dependence between the CPU and GPU
- E.g., Instead of:

```
t = torch.arange(1024).to('cuda')  # Creates on CPU, then copies to GPU
```

Use:

```
t = torch.arange(1024, device='cuda')  # Creates directly on GPU
```

TTT4 – Get to Know torch.compile

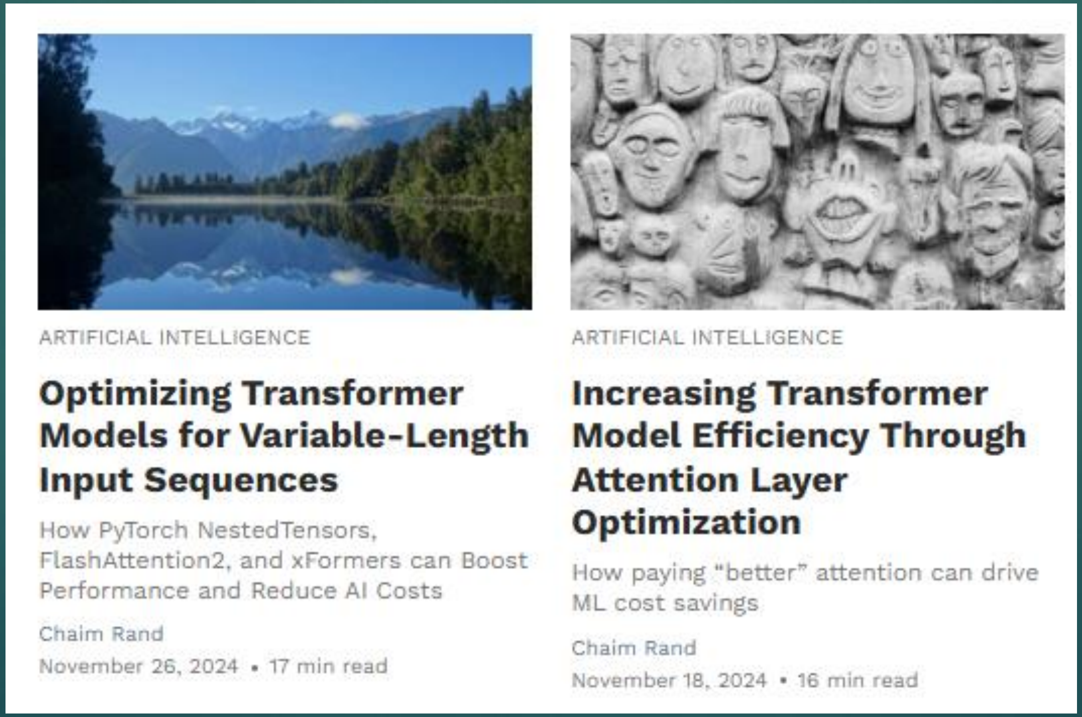
- Introduced in PyTorch2.0, torch.compile can significantly optimize graph execution
  - Contrary to eager execution mode which runs line by line
- Compiles model into optimized computation graph
  - Fuses kernels
  - Enables out-of-order execution
  - Reduces kernel launch overhead
- E.g.: `model = torch.compile(model, fullgraph=True)`

TTT5 – Use Automatic Mixed Precision

- Using lower precision floating point types (fp16, bfloat16, fp8) can significantly optimize GPU utilization
  - Reduced memory usage allows for increasing batch size
  - Modern accelerators include dedicated engines for lower precision floats
- Importantly, reducing the bit-precision can impact model output quality
  - PyTorch Automatic Mixed Precision (AMP) strategies selectively applies precision reduction only where it minimally impacts accuracy.
  - Be sure to evaluate its effect on your specific workload before full adoption

TTT6 – Use Optimized Model Components

- E.g., as transformer-based architectures increase in popularity, it is important to take advantage of optimized attention layer solutions



Key Takeaways

- AI/ML developers must take responsibility for the runtime performance of their models
- Integrate profiling analysis and optimization into AI/ML development workflow
- You do not need to be a CUDA expert to boost runtime performance and reduce AI/ML costs