Case Study

This notebook shows how to run Debugger's profiling functionality. We use a tutorial from PyTorch website that fine-tunes a pre-trained Mask R-CNN model for instance segmentation on a new dataset: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html (https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html)

First let's download the dataset

In [ ]:
```
! wget https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip
! unzip PennFudanPed.zip
! mv PennFudanPed entry_point/PennFudanPed
```

Run single GPU training with profiling enabled

First we define the profiler configuration that instructs Debugger to sample system metrics at 500ms and collect detailed metrics (dataloading times, python profiling, GPU operators...) from step 10 to 12.

In [68]:
```
from sagemaker.debugger import ProfilerConfig, FrameworkProfile, DetailedProfilingConfig, DataloaderProfilingConfig, PythonProfilingConfig, PythonProfiler

profiler_config = ProfilerConfig(
    system_monitor_interval_millis=100,
    framework_profile_params=FrameworkProfile(
    detailed_profiling_config=DetailedProfilingConfig(
        start_step=10,
        num_steps=2
        )
    )
)
```

Next we start the SageMaker training job. Debugger will auomatically apply the necessary configuration in the training script and model, to obtain the profiling data. The instance segmentation model training is defined in `train.py` . Let's take a look at the training script:

In [2]:
```
! cat entry_point/train.py
```

```
import os
os.system('pip install pycocotools')
import math
import numpy as np
import torch
import torch.utils.data
from PIL import Image
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor
from torchvision.models.detection.rpn import AnchorGenerator
from engine import evaluate
import utils
import transforms as T
import smdebug.pytorch as smd
from smdebug import modes

class PennFudanDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms
        self.imgs = list(sorted(os.listdir(os.path.join(root, "PNGImages"))))
        self.masks = list(sorted(os.listdir(os.path.join(root, "PedMasks"))))

    def __getitem__(self, idx):
        # load images ad masks
        img_path = os.path.join(self.root, "PNGImages", self.imgs[idx])
        mask_path = os.path.join(self.root, "PedMasks", self.masks[idx])
        img = Image.open(img_path).convert("RGB")
        mask = Image.open(mask_path)

        mask = np.array(mask)
        obj_ids = np.unique(mask)
        obj_ids = obj_ids[1:]

        masks = mask == obj_ids[:, None, None]

        num_objs = len(obj_ids)
        boxes = []
        for i in range(num_objs):
            pos = np.where(masks[i])
            xmin = np.min(pos[1])
            xmax = np.max(pos[1])
            ymin = np.min(pos[0])
            ymax = np.max(pos[0])
            boxes.append([xmin, ymin, xmax, ymax])

        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.ones((num_objs,), dtype=torch.int64)
        masks = torch.as_tensor(masks, dtype=torch.uint8)

        image_id = torch.tensor([idx])
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

        target = {}
```

```
        target["boxes"] = boxes
        target["labels"] = labels
        target["masks"] = masks
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd

        if self.transforms is not None:
            img, target = self.transforms(img, target)

        return img, target

    def __len__(self):
        return len(self.imgs)

def get_transform(train):
    transforms = []
    transforms.append(T.ToTensor())
    if train:
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)



def train(batch_size, checkpoint_freq, num_epochs):

    num_classes = 2
    model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True, rpn_nms_thresh=1, rpn_pre_nms_top_n_train=5000)

    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    in_features_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
    hidden_layer = 256

    model.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask,
                                                       hidden_layer,
                                                       num_classes)
    model = torch.nn.DataParallel(model)
    model.to('cuda')

    dataset = PennFudanDataset('PennFudanPed', get_transform(train=True))
    dataset_test = PennFudanDataset('PennFudanPed', get_transform(train=False))


    indices = torch.randperm(len(dataset)).tolist()
    dataset = torch.utils.data.Subset(dataset, indices[:-50])
    dataset_test = torch.utils.data.Subset(dataset_test, indices[-50:])

    data_loader = torch.utils.data.DataLoader(
        dataset, batch_size=batch_size, shuffle=True, num_workers=4,
        collate_fn=utils.collate_fn)

    data_loader_test = torch.utils.data.DataLoader(
        dataset_test, batch_size=batch_size, shuffle=False, num_workers=4,
        collate_fn=utils.collate_fn)

    params = [p for p in model.parameters() if p.requires_grad]
    optimizer = torch.optim.SGD(params, lr=0.005,
```

```
                            momentum=0.9, weight_decay=0.0005)

lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)

hook = smd.Hook.create_from_json_file()



for epoch in range(num_epochs):

    hook.set_mode(modes.TRAIN)
    model.train()
    metric_logger = utils.MetricLogger(delimiter="  ")
    metric_logger.add_meter('lr', utils.SmoothedValue(window_size=1, fmt='{value:.6f}'))
    header = 'Epoch: [{}]'.format(epoch)

    if epoch == 0:
        warmup_factor = 1. / 1000
        warmup_iters = min(1000, len(data_loader) - 1)

        lr_scheduler = utils.warmup_lr_scheduler(optimizer, warmup_iters, warmup_factor)

    for iteration, (images, targets) in enumerate(data_loader):
        images = list(image.to('cuda') for image in images)
        targets = [{k: v.to('cuda') for k, v in t.items()} for t in targets]

        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())

        loss_dict_reduced = utils.reduce_dict(loss_dict)
        losses_reduced = sum(loss for loss in loss_dict_reduced.values())

        loss_value = losses_reduced.item()

        optimizer.zero_grad()
        losses.backward()

        optimizer.step()

        if lr_scheduler is not None:
            lr_scheduler.step()

        metric_logger.update(loss=losses_reduced, **loss_dict_reduced)
        metric_logger.update(lr=optimizer.param_groups[0]["lr"])

        if iteration%checkpoint_freq == 0:
            utils.save_on_master({
            'model': model.state_dict(),
            'optimizer': optimizer.state_dict()
            },
            'model_{}.pth')

    lr_scheduler.step()

    hook.set_mode(modes.EVAL)
    evaluate(model, data_loader_test, device='cuda')
```

```
if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description='PyTorch Detection Training')

    parser.add_argument('--batch_size', default=2)
    parser.add_argument('--checkpoint_freq', default=10)
    parser.add_argument('--num_epochs', default=20)
    args = parser.parse_args()

    train(args.batch_size, args.checkpoint_freq, args.num_epochs)
```

In [69]:

```
import sagemaker
from sagemaker.pytorch import PyTorch


image_uri = f"763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-training:1.6.0-gpu-py36-cu110-ubuntu18.04"

estimator = PyTorch(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    image_uri=image_uri,
    instance_type='ml.p3.2xlarge',
    source_dir='entry_point',
    entry_point='train.py',
    profiler_config=profiler_config
)

estimator.fit(wait=False)
```

Once the job is running we can use `smdebug` library to access and query the data as the training is still in progress. We can now plot the profiling data such as timeline charts, heatmaps or download the profiler report from Amazon S3.

In [ ]:

```
import smdebug

from smdebug.profiler.analysis.notebook_utils.training_job import TrainingJob

jobname=estimator.latest_training_job.job_name
tj = TrainingJob(jobname, 'us-west-2')
```

To check if the system and framework metrics are available from the S3 URI

In [ ]:

```
tj.wait_for_sys_profiling_data_to_be_available()
tj.wait_for_framework_profiling_data_to_be_available()
```

To create system and framework reader objects after the metric data become available

In [3]:

```
system_metrics_reader = tj.get_systems_metrics_reader()
framework_metrics_reader = tj.get_framework_metrics_reader()
```

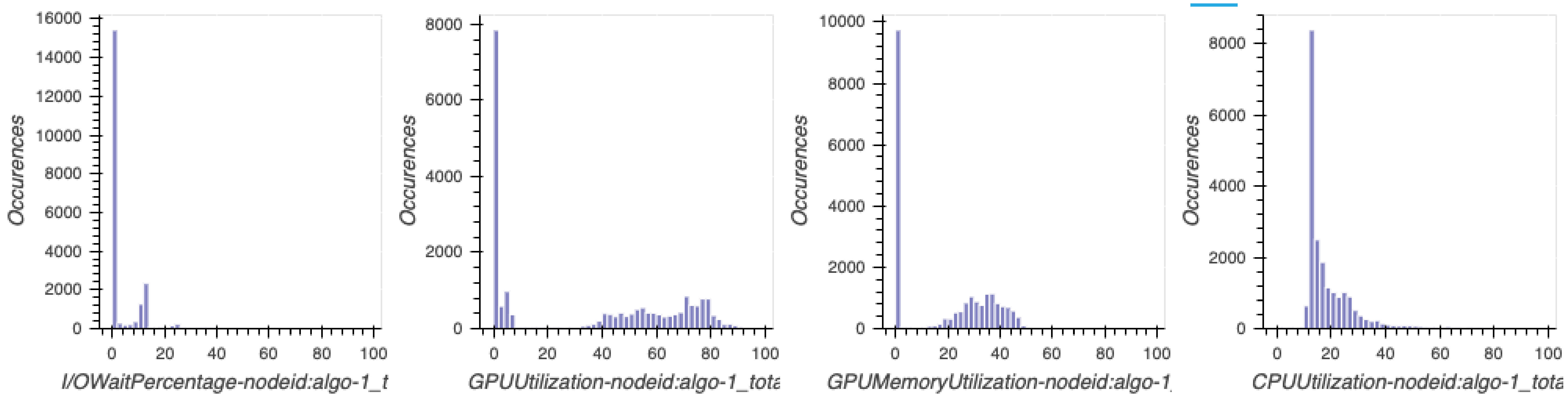
Real-time visualizations of profiling metrics

Histogram visualizations of system metrics:

In [4]:
```
from smdebug.profiler.analysis.notebook_utils.metrics_histogram import MetricsHistogram

metrics_histogram = MetricsHistogram(system_metrics_reader)
metrics_histogram.plot(
    starttime=0,
    endtime=system_metrics_reader.get_timestamp_of_latest_available_file(),
    select_dimensions=["CPU", "GPU", "I/O"],
    select_events=["total"]
)
```

```
[2021-06-02 21:02:03.935 ip-172-16-59-176:14011 INFO metrics_reader_base.py:134] Getting 37 event files
Found 549961 system metrics events from timestamp_in_us:0 to timestamp_in_us:1622663280000000
select events:['total']
select dimensions:['CPU', 'GPU', 'I/O']
filtered_events:{'total'}
filtered_dimensions:{'I/OWaitPercentage-nodeid:algo-1', 'GPUUtilization-nodeid:algo-1', 'GPUMemoryUtilization-nodeid:algo-1', 'CPUUtilization-nodeid:algo-
1'}
```

[(https://bokeh.org/)](https://bokeh.org/)

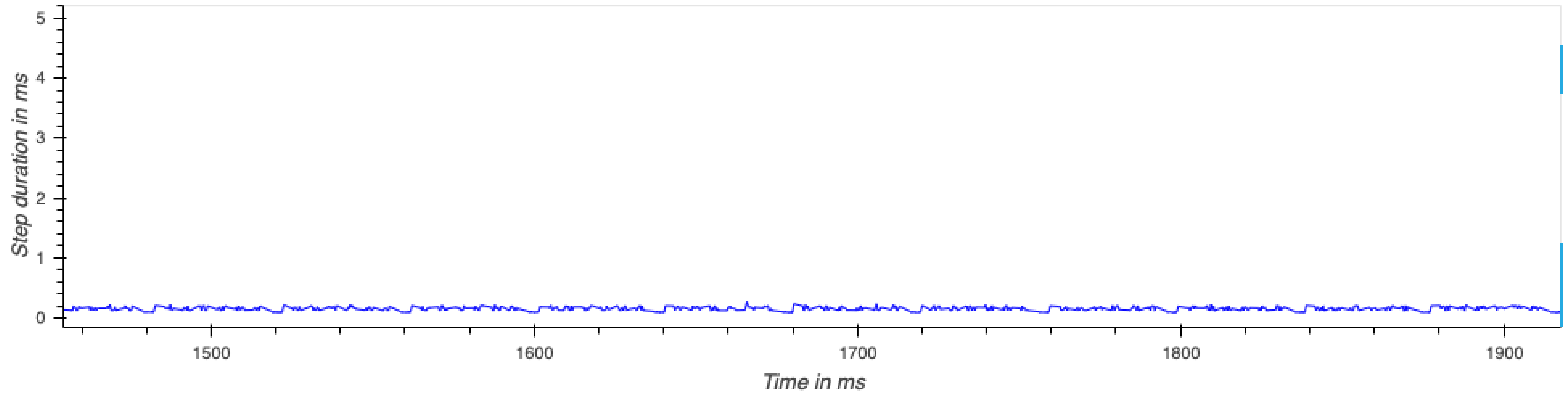



```
filtered_dimensions:{'I/OWaitPercentage-nodeid:algo-1', 'GPUUtilization-nodeid:algo-1', 'GPUMemoryUtilization-nodeid:algo-1', 'CPUUtilization-nodeid:algo-
1'}
```

Step timeline chart shows the step duration (time for forward and backward pass) as the training is in progress. X-axis shows the training job duration and y axis indicates the step duration.

```
In [5]: from smdebug.profiler.analysis.notebook_utils.step_timeline_chart import StepTimelineChart

        view_step_timeline_chart = StepTimelineChart(framework_metrics_reader)
```

```
[2021-06-02 21:06:18.682 ip-172-16-59-176:14011 INFO metrics_reader_base.py:134] Getting 33 event files
[2021-06-02 21:06:25.344 ip-172-16-59-176:14011 INFO trace_event_file_parser.py:197] Start time for events in uSeconds = 1622661370398837
[2021-06-02 21:06:40.915 ip-172-16-59-176:14011 INFO metrics_reader_base.py:134] Getting 33 event files

BokehUserWarning: ColumnDataSource's columns must be of the same length. Current lengths: ('metric', 4250), ('step', 4249), ('x', 4249), ('y', 4249)
```

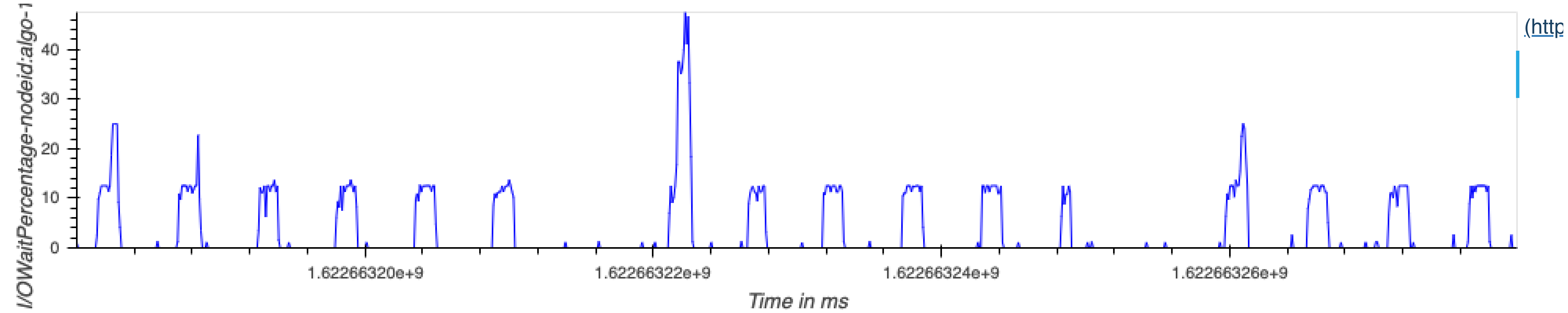


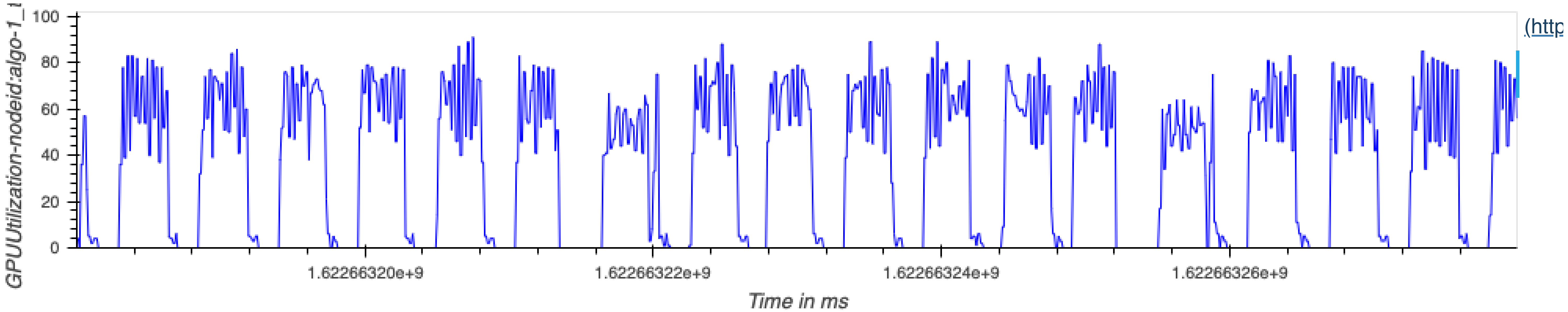
(http

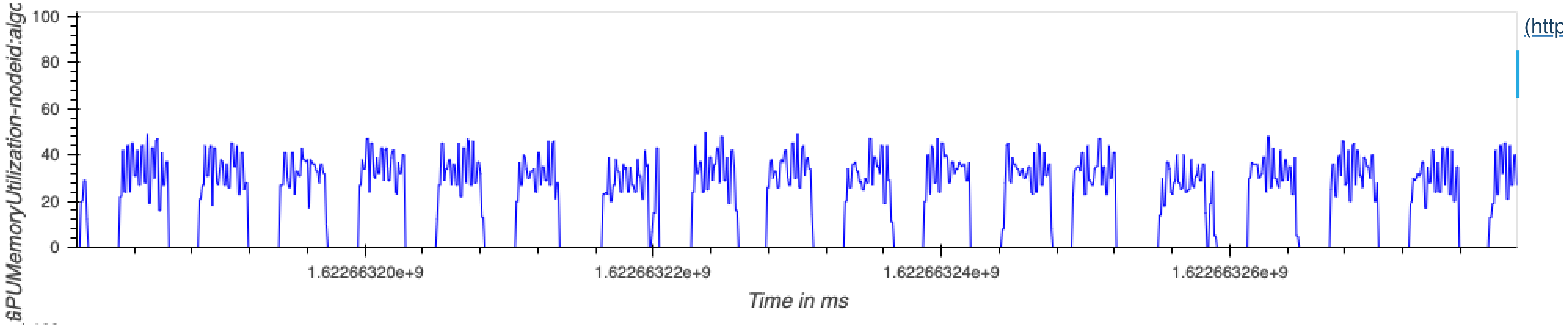
Timeseries charts show the utilization across training job duration

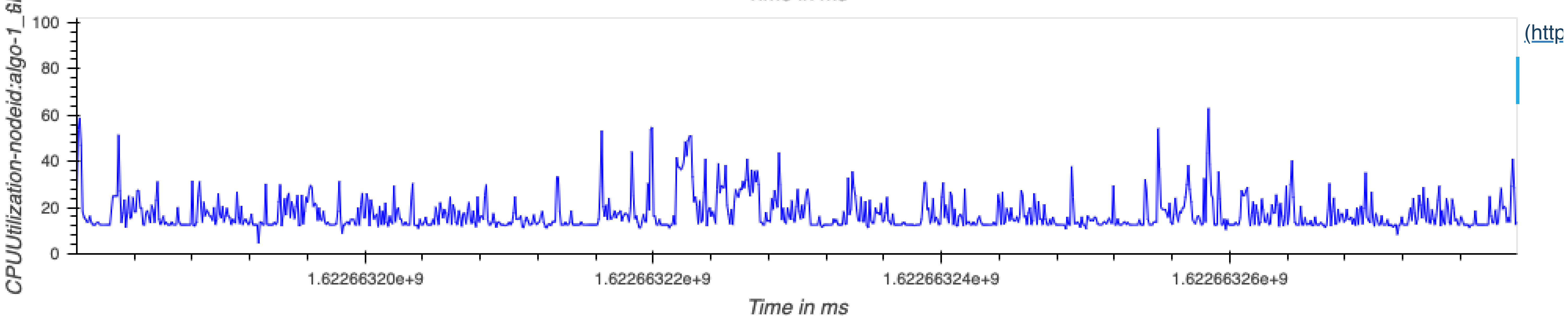
In [6]:
```
from smdebug.profiler.analysis.notebook_utils.timeline_charts import TimelineCharts

view_timeline_charts = TimelineCharts(
    system_metrics_reader,
    framework_metrics_reader,
    select_dimensions=["CPU", "GPU", "I/O"], # optional
    select_events=["total"]                  # optional
)
```

```
[2021-06-02 21:07:25.463 ip-172-16-59-176:14011 INFO metrics_reader_base.py:134] Getting 37 event files
select events:['total']
select dimensions:['CPU', 'GPU', 'I/O']
filtered_events:{'total'}
filtered_dimensions:{'I/OWaitPercentage-nodeid:algo-1', 'GPUUtilization-nodeid:algo-1', 'GPUMemoryUtilization-nodeid:algo-1', 'CPUUtilization-nodeid:algo-1'}
```





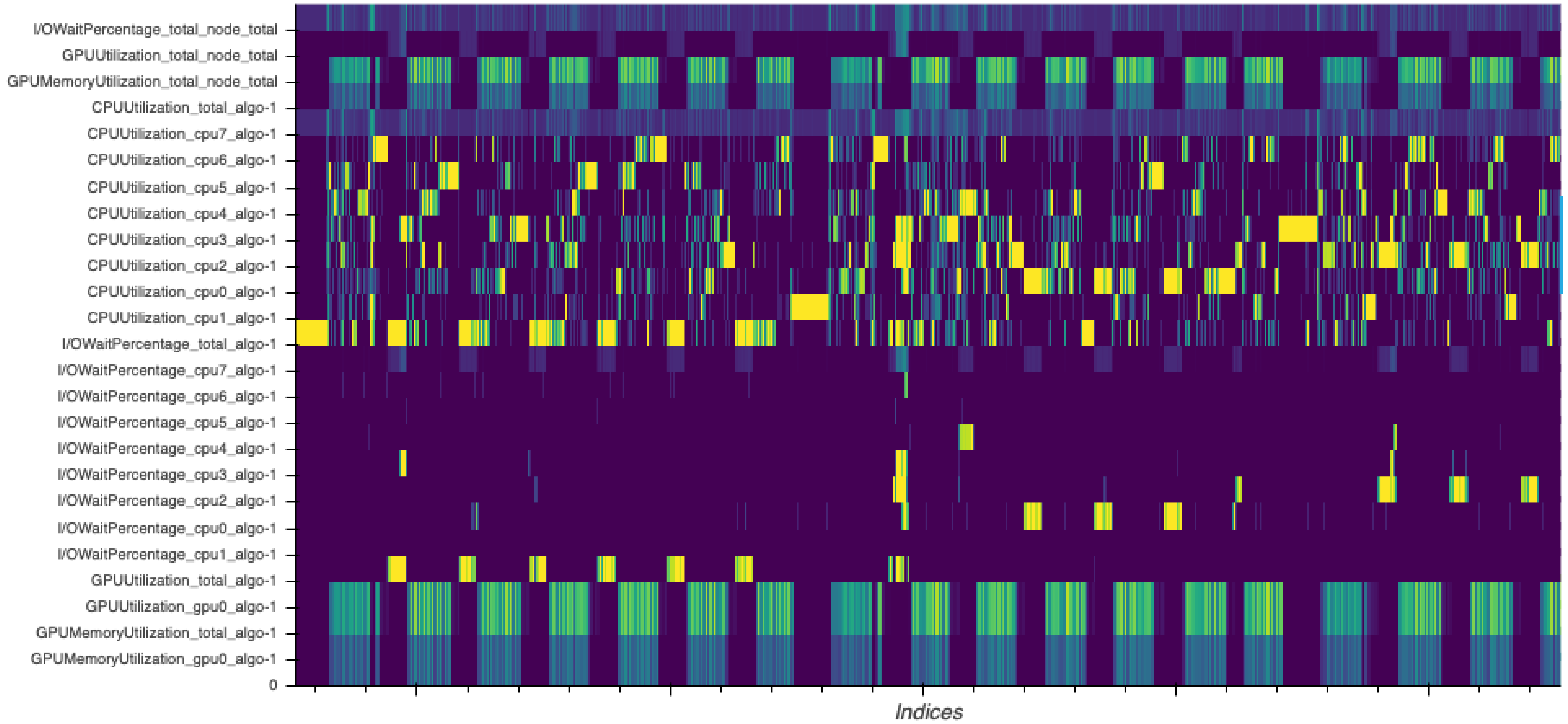




The heatmap visualization helps to more easily identify bottlenecks where utilization on GPU is low and CPU utilizaion is high.

In [7]:
```
from smdebug.profiler.analysis.notebook_utils.heatmap import Heatmap

view_heatmap = Heatmap(
    system_metrics_reader,
    framework_metrics_reader,
    select_dimensions=["CPU", "GPU", "I/O"],
    plot_height=450
)
```

```
[2021-06-02 21:08:01.206 ip-172-16-59-176:14011 INFO metrics_reader_base.py:134] Getting 37 event files
select events:['.*']
select dimensions:['CPU', 'GPU', 'I/O']
filtered_events:{'cpu3', 'ReceiveBytesPerSecond', 'TransmitBytesPerSecond', 'gpu0', 'cpu2', 'WriteThroughputInBytesPerSecond', 'cpu0', 'total', 'IOPS', 'Re
adThroughputInBytesPerSecond', 'cpu4', 'cpu5', 'cpu7', 'cpu1', 'MemoryUsedPercent', 'cpu6'}
filtered_dimensions:{'GPUMemoryUtilization', 'CPUUtilization', 'I/OWaitPercentage', 'GPUUtilization'}
```



(http

We can now generate the merged timeline (merged_timeline.json), that gives a detailed view of system and framework metrics. We can load the json file into the Chrome trace viewer.

In [ ]:
```
import time
from smdebug.profiler.analysis.utils.merge_timelines import MergedTimeline

combined_timeline = MergedTimeline(tj.profiler_s3_output_path, output_directory="./")
combined_timeline.merge_timeline(0, time.time())
```

We can look at the profiling report (profiler-report-1.html) where we can see that the training suffered 37\% of the time from CPU bottlenecks and 21\% of the time from IO bottlenecks.

Run multi GPU training with profiling enabled and PyTorch's DataParallel

Next we run a multi GPU training jobs for which we use PyTorch's DataParallel.

In [ ]:
```
import sagemaker
from sagemaker.pytorch import PyTorch


image_uri = f"763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-training:1.6.0-gpu-py36-cu110-ubuntu18.04"

estimator = PyTorch(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    image_uri=image_uri,
    instance_type='ml.p3.8xlarge',
    source_dir='entry_point',
    entry_point='train.py',
    profiler_config=profiler_config
)

estimator.fit(wait=False)
```

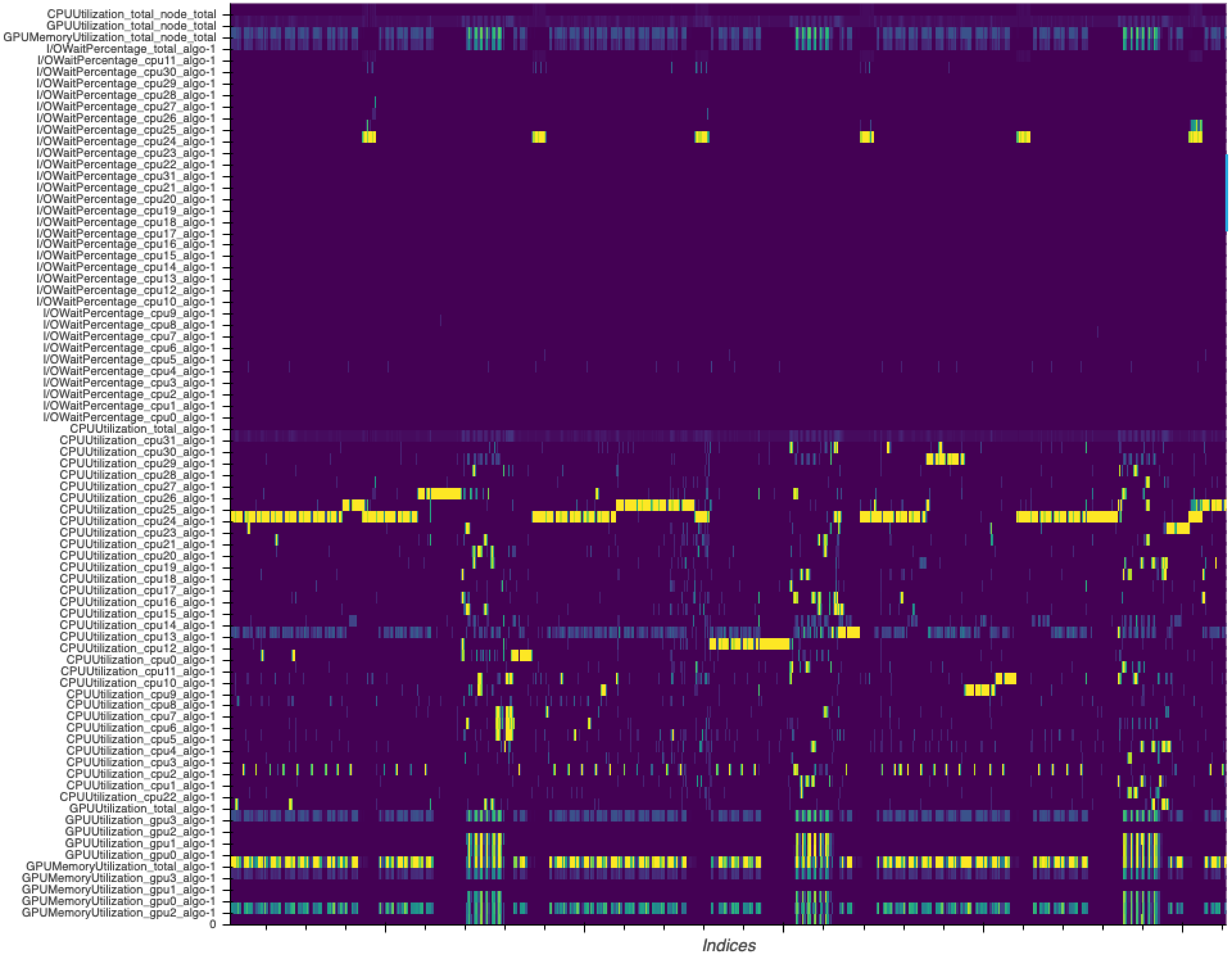
In [ ]:
```
jobname=estimator.latest_training_job.job_name
tj = TrainingJob(jobname, 'us-west-2')
```

In [ ]:
```
tj.wait_for_sys_profiling_data_to_be_available()
tj.wait_for_framework_profiling_data_to_be_available()
system_metrics_reader = tj.get_systems_metrics_reader()
framework_metrics_reader = tj.get_framework_metrics_reader()
```

Now let's compare the heatmap to previous training run

In [10]:
```
from smdebug.profiler.analysis.notebook_utils.heatmap import Heatmap

view_heatmap = Heatmap(
    system_metrics_reader,
    framework_metrics_reader,
    select_dimensions=["CPU", "GPU", "I/O"],
    plot_height=750
)
```

```
[2021-06-02 21:08:45.281 ip-172-16-59-176:14011 INFO metrics_reader_base.py:134] Getting 31 event files
select events:['.*']
select dimensions:['CPU', 'GPU', 'I/O']
filtered_events:{'ReceiveBytesPerSecond', 'cpu18', 'TransmitBytesPerSecond', 'gpu3', 'gpu0', 'cpu19', 'cpu17', 'IOPS', 'cpu13', 'cpu31', 'cpu10', 'cpu25', 'gpu1', 'cpu28', 'total', 'cpu24', 'ReadThroughputInBytesPerSecond', 'cpu9', 'cpu27', 'cpu30', 'cpu11', 'cpu2', 'MemoryUsedPercent', 'gpu2', 'cpu6', 'cpu2
6', 'cpu15', 'WriteThroughputInBytesPerSecond', 'cpu22', 'cpu14', 'cpu16', 'cpu7', 'cpu3', 'cpu1', 'cpu12', 'cpu8', 'cpu20', 'cpu0', 'cpu29', 'cpu23', 'cpu
4', 'cpu5', 'cpu21'}
filtered_dimensions:{'GPUMemoryUtilization', 'CPUUtilization', 'I/OWaitPercentage', 'GPUUtilization'}
```



(http

From the profiling report (profiler-report-2.html) we can see that CPU bottlenecks increased to 74\%. Training time reduced from 2116 to 1747 seconds. So despite having 4 GPUs, training does not even run twice faster.

Run multi-GPU training with profiling enabled and PyTorch's DistributedDataParallel

In [76]:
```
import sagemaker
from sagemaker.pytorch import PyTorch


image_uri = f"763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-training:1.6.0-gpu-py36-cu110-ubuntu18.04"

estimator = PyTorch(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    image_uri=image_uri,
    instance_type='ml.p3.8xlarge',
    source_dir='entry_point',
    entry_point='distributed_launch.py',
    profiler_config=profiler_config,
    hyperparameters={"training_script":"train_ddp.py",  "nproc_per_node": 4}
)

estimator.fit(wait=False)
```

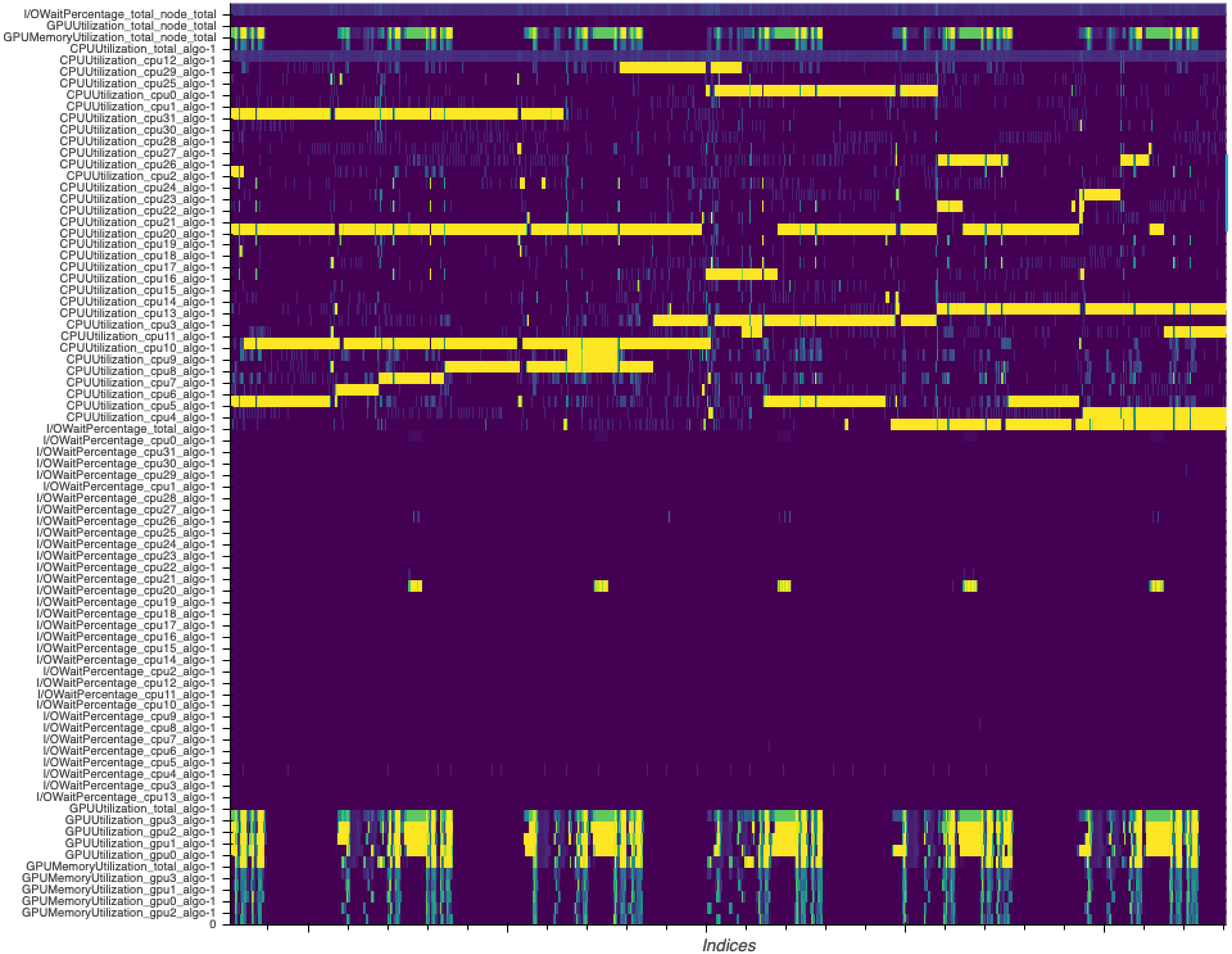
In [ ]:
```
jobname=estimator.latest_training_job.job_name
tj = TrainingJob(jobname, 'us-west-2')
```

In [15]:
```
tj.wait_for_sys_profiling_data_to_be_available()
tj.wait_for_framework_profiling_data_to_be_available()
system_metrics_reader = tj.get_systems_metrics_reader()
framework_metrics_reader = tj.get_framework_metrics_reader()
```

```
Profiler data from system is available
```

In [16]:
```
from smdebug.profiler.analysis.notebook_utils.heatmap import Heatmap

view_heatmap = Heatmap(
    system_metrics_reader,
    framework_metrics_reader,
    select_dimensions=["CPU", "GPU", "I/O"],
    plot_height=750
)
```

```
[2021-06-02 21:16:33.690 ip-172-16-59-176:14011 INFO metrics_reader_base.py:134] Getting 19 event files
select events:['.*']
select dimensions:['CPU', 'GPU', 'I/O']
filtered_events:{'ReceiveBytesPerSecond', 'cpu18', 'TransmitBytesPerSecond', 'gpu3', 'gpu0', 'cpu19', 'cpu17', 'IOPS', 'cpu13', 'cpu31', 'cpu10', 'cpu25', 'gpu1', 'cpu28', 'total', 'cpu24', 'ReadThroughputInBytesPerSecond', 'cpu9', 'cpu27', 'cpu30', 'cpu11', 'cpu2', 'MemoryUsedPercent', 'gpu2', 'cpu6', 'cpu2
6', 'cpu15', 'WriteThroughputInBytesPerSecond', 'cpu22', 'cpu14', 'cpu16', 'cpu7', 'cpu3', 'cpu1', 'cpu12', 'cpu8', 'cpu20', 'cpu29', 'cpu0', 'cpu23', 'cpu
4', 'cpu5', 'cpu21'}
filtered_dimensions:{'GPUMemoryUtilization', 'CPUUtilization', 'I/OWaitPercentage', 'GPUUtilization'}
```



(http

From the profiling report (profiler-report-3.html) we can now see that training time is only 1054 seconds. So switching from `DataParallel` to `DistributedDataParallel` improved training time.

Optimize training script and re-run training

We now run an optimized version of the training script: with default NMS threshold and with reduced model checkpointing.

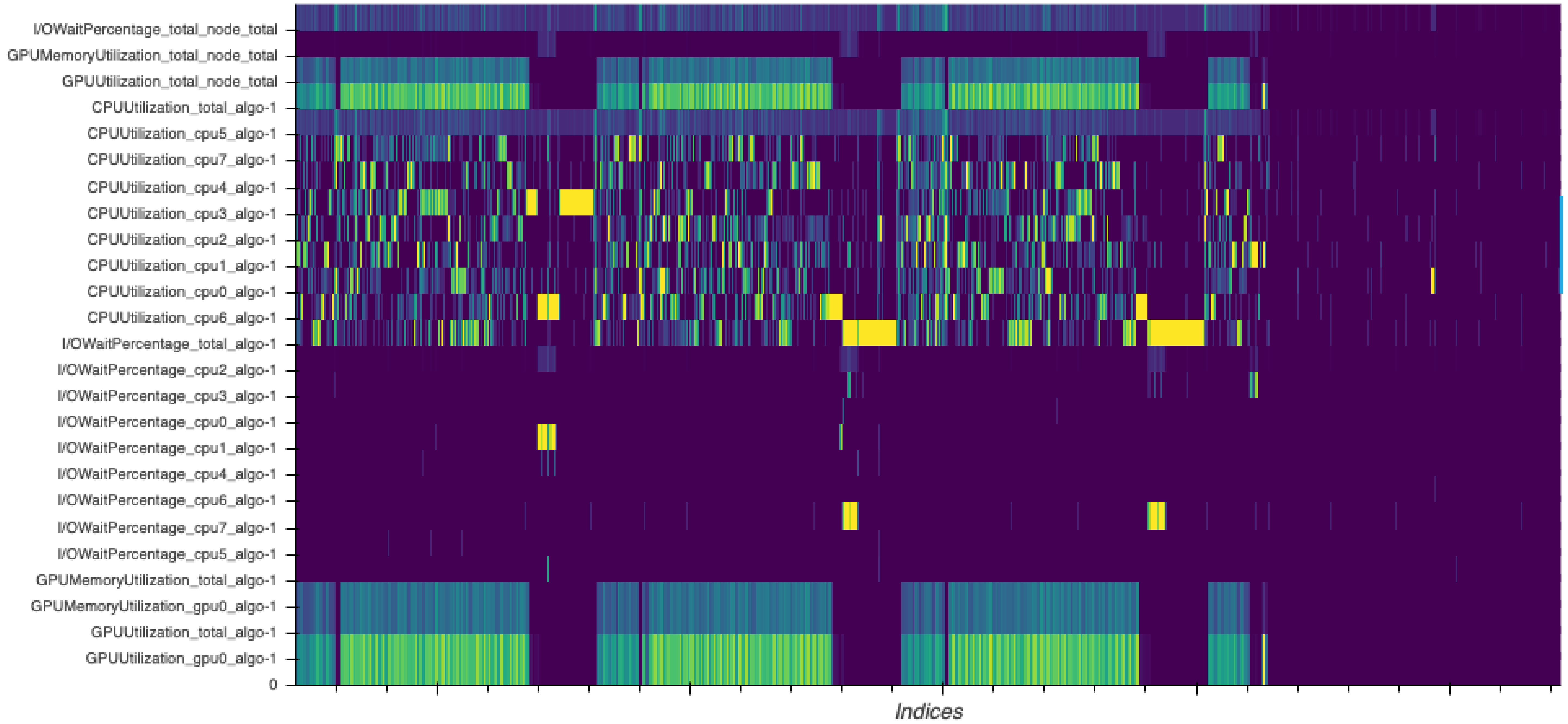
In [81]:
```
import sagemaker
from sagemaker.pytorch import PyTorch


image_uri = f"763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-training:1.6.0-gpu-py36-cu110-ubuntu18.04"

estimator = PyTorch(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    image_uri=image_uri,
    instance_type='ml.p3.2xlarge',
    source_dir='entry_point',
    entry_point='train_optimized.py',
    profiler_config=profiler_config
)

estimator.fit(wait=False)
```

In [ ]:
```
jobname=estimator.latest_training_job.job_name
tj = TrainingJob(jobname, 'us-west-2')
```

In [ ]:
```
tj.wait_for_sys_profiling_data_to_be_available()
tj.wait_for_framework_profiling_data_to_be_available()
system_metrics_reader = tj.get_systems_metrics_reader()
framework_metrics_reader = tj.get_framework_metrics_reader()
```

In [20]:
```python
from smdebug.profiler.analysis.notebook_utils.heatmap import Heatmap

view_heatmap = Heatmap(
    system_metrics_reader,
    framework_metrics_reader,
    select_dimensions=["CPU", "GPU", "I/O"],
    plot_height=450
)
```

```
[2021-06-02 21:18:02.916 ip-172-16-59-176:14011 INFO metrics_reader_base.py:134] Getting 25 event files
select events:['.*']
select dimensions:['CPU', 'GPU', 'I/O']
filtered_events:{'cpu3', 'ReceiveBytesPerSecond', 'TransmitBytesPerSecond', 'gpu0', 'cpu2', 'WriteThroughputInBytesPerSecond', 'cpu0', 'total', 'IOPS', 'ReadThroughputInBytesPerSecond', 'cpu4', 'cpu5', 'cpu7', 'cpu1', 'MemoryUsedPercent', 'cpu6'}
filtered_dimensions:{'GPUMemoryUtilization', 'CPUUtilization', 'I/OWaitPercentage', 'GPUUtilization'}
```



(http

From the [profiler report (profiler-report-4.html)](profiler-report-4.html) we can now see that training job duration is 1412 seconds, so by removing performance bottlenecks we were able to reduce training from 2116 second to 1412 seconds.