Tensor Cores and DNN Operation Categories

What are Tensor Cores?

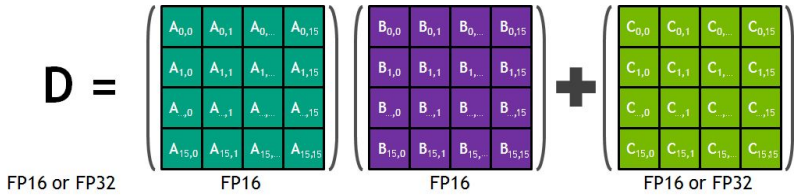
- Programmable **matrix-multiply-and-accumulate** units of V100, A100, H100

- Input is FP16, weight and bias could be FP16 or FP32, …

- 4x4x4 matrix processing array performing D = A*B+C operation

How to know if models use Tensor Core:

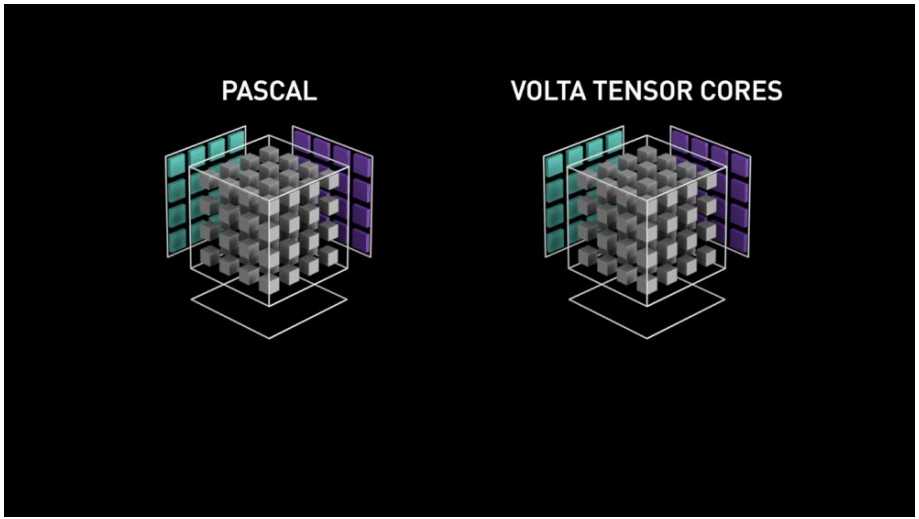
- Inputs is FP16, can be check using apex.pyprof
- Call any kernel function from NVIDIA/cutlass ? eg. Pytorch memory efficient transformers: pytorch/predicated_tile_iterator_residual_last.h
- If it is custom kernel, it often calls wmmaMatmul for kernel loop?



Use profiling tools to check tensor cores usage

Tensor Core technology

- Tensor Cores perform matrix multiply and accumulate (MMA) calculations
- FP16/FP32 mixed-precision Tensor Core allow execute 64 FP16 fused multiply-add operations (FMAs) with FP32 accumulation per clock
- It computes a mixed-precision 4×4×4 matrix multiplication per clock instead of individual element likes CUDA cores.
- A100 Tensor Core adds support for INT8, INT4, Bool, TF32, BF16, and FP64 formats.



A100, V100, P100 comparison

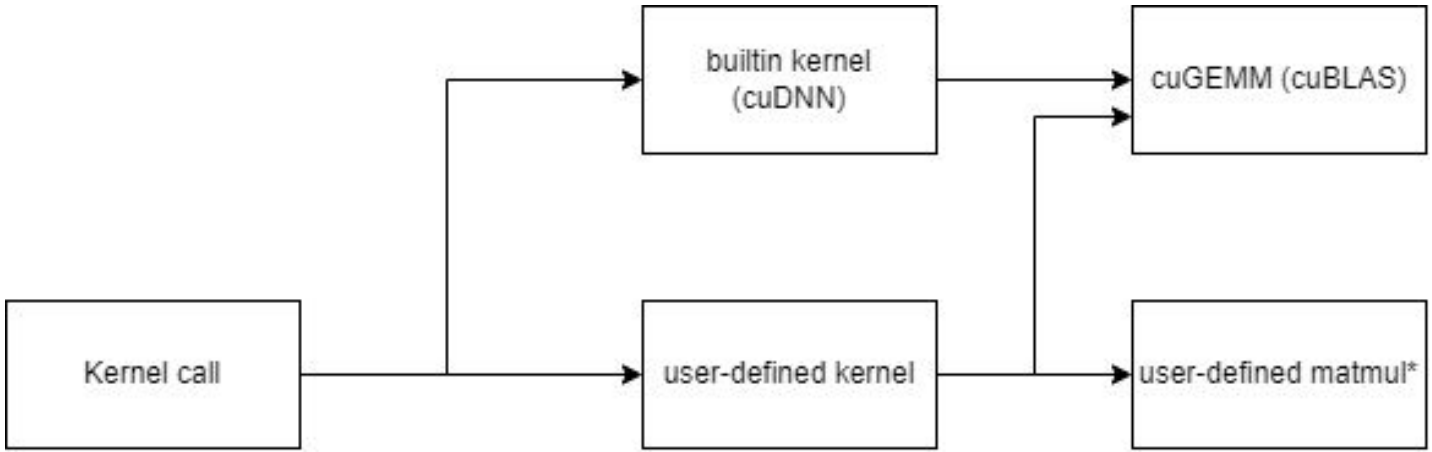
Technical Specification	NVIDIA A100	NVIDIA V100	NVIDIA P100
Cuda Cores	6912	5120	3584
Tensor Cores	432	640	N/A
FLOPS (FP64 Tensor Cores)	19.5 TFLOPS	15.7 TFLOPS	9.3 TFLOPS
Memory Bandwidth	1555 GB/s	900 GB/s	720 GB/s
Memory Size	40 GB HBM2	16 GB HBM2	16 GB HBM2
TDP	400 W	300 W	300 W
Manufacturing Process	7 nm	12 nm	16 nm
Peak Memory Bandwidth	6 TB/s	900 GB/s	732 GB/s
NVLink Bandwidth	600 GB/s	300 GB/s	N/A

How to enable Tensor Cores in C++ (using CuBLAS)

```
// First, create a cuBLAS handle:
cublasStatus_t cublasStat = cublasCreate(&handle);

// Set the math mode to allow cuBLAS to use Tensor Cores:
cublasStat = cublasSetMathMode(handle, CUBLAS_TENSOR_OP_MATH);

// Allocate and initialize your matrices (only the A matrix is shown):
size_t matrixSizeA = (size_t)rowsA * colsA;
T_ELEM_IN **devPtrA = 0;

cudaMalloc((void**)&devPtrA[0], matrixSizeA * sizeof(devPtrA[0][0]));
T_ELEM_IN A  = (T_ELEM_IN *)malloc(matrixSizeA * sizeof(A[0]));

memset( A, 0xFF, matrixSizeA* sizeof(A[0]));
status1 = cublasSetMatrix(rowsA, colsA, sizeof(A[0]), A, rowsA, devPtrA[i], rowsA);

// ... allocate and initialize B and C matrices (not shown) ...

// Invoke the GEMM, ensuring k, lda, ldb, and ldc are all multiples of 8,
// and m is a multiple of 4:
cublasStat = cublasGemmEx(handle, transa, transb, m, n, k, alpha,
                          A, CUDA_R_16F, lda,
                          B, CUDA_R_16F, ldb,
                          beta, C, CUDA_R_16F, ldc, CUDA_R_32F, algo);
```

Matrix multiplication using Tensor Cores

Does Pytorch use cuBLAS?



*** user-defined matmul may not necessarily uses cuBLAS.**

Does Pytorch use cuBLAS?

```
1198    if (params.is_depthwise(input, weight)) {
1199      if (params.use_cudnn_depthwise(input, weight)) {
1200        return ConvBackend::Cudnn;
1201      } else if (params.use_miopen(input, weight, bias_sizes_opt.has_value())) {
1202        return ConvBackend::MiopenDepthwise;
1203      } else {
1204        if (input.ndimension() == 4) {
1205          return ConvBackend::CudaDepthwise2d;
1206        } else if (input.ndimension() == 5) {
1207          return ConvBackend::CudaDepthwise3d;
1208        } else {
1209          // unsupported
1210        }
```

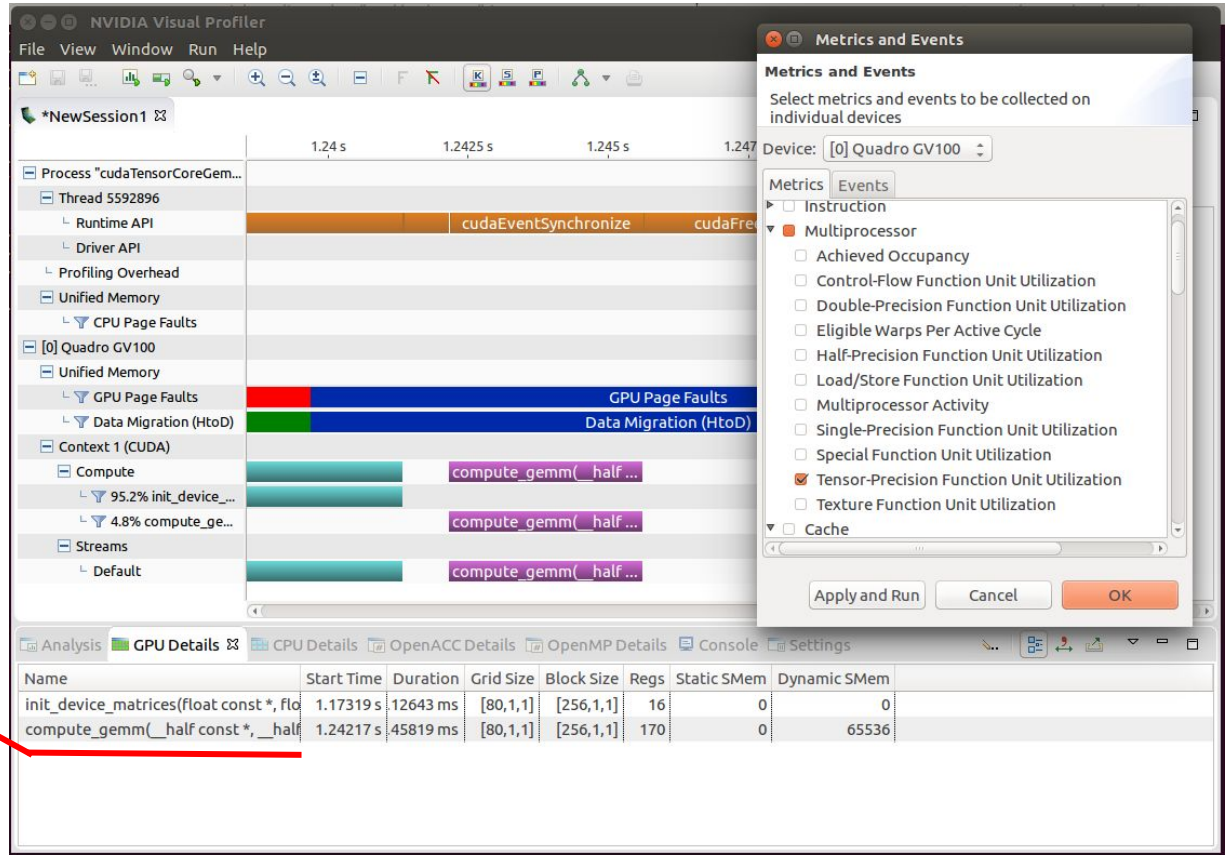
cuDNN/GEMM

CUDA_KERNEL_LOOP
Doesn't use cuDNN

[pytorch/Convolution.cpp at master · pytorch/pytorch (github.com)]

Does Pytorch use cuBLAS? (NVIDIA Visual Profiler)

A kernel call for
half precision input



Does Pytorch use cuBLAS? (pyprof + nsight + pytorch profiling)

[Tensor core usage profiling - Turing architecture · Issue #530 · NVIDIA/apex (github.com)]

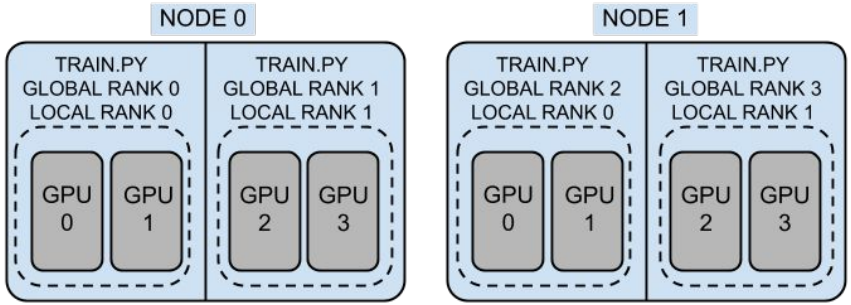
```
$ nvprof -f -o resnet50.sql --profile-from-start off -- python
repro_script.py --fp16

$ python -m apex.pyprof.parse resnet50.sql > resnet50.dict

$ python -m apex.pyprof.prof -c kernel,op,sil,tc,flops --csv
resnet50.dict > resnet50.csv
```

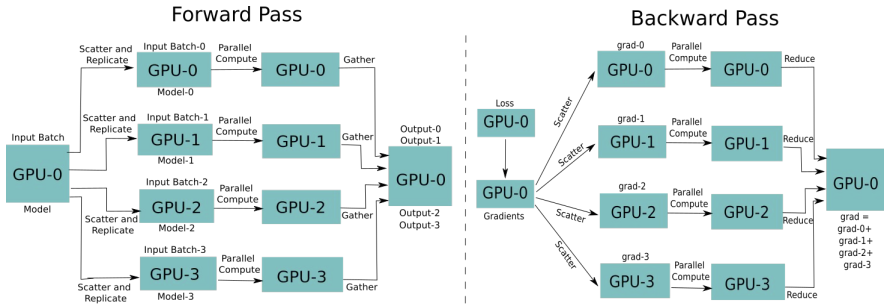
Kernel	Op	Sil(ns)	TC	FLOPs
cudnn::gemm::computeOffsetsKernel	conv2d	7647	-	0
volta_fp16_scudnn_fp16_128x64_relu_medium_nn_v1	conv2d	1099664	-	10838016000.0
batch_norm_transform_input_kernel	batch_norm	675670	-	294912000
elementwise_kernel	relu	465945	-	36864000

How to use multi-GPUs with Tensor Cores enabled

- Quite straightforward, use Pytorch DDP or apex DDP for multi-nodes

- Bandwidth though PCI-e/NV-Link often very fast, inter-nodes requires special connection



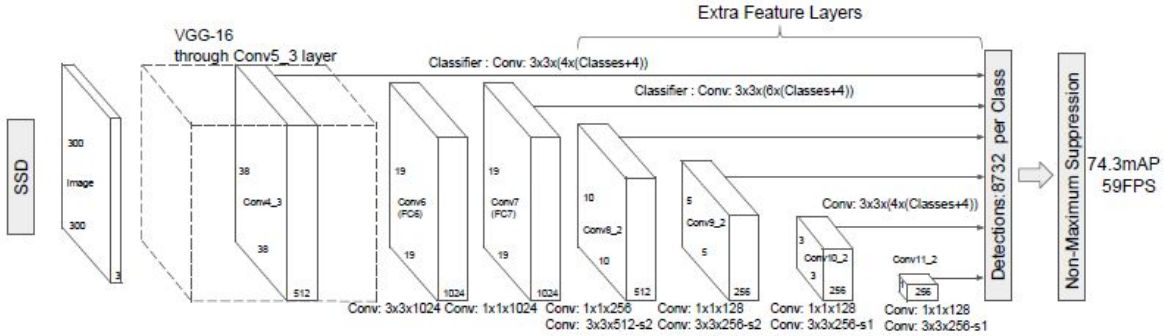


Tensor Cores vs non-Tensor Cores Experiment: SSD300

Experiment setting:https://harsh-fold-50d.notion.site/GPU-Benchmark-Project-c3f1e3a80c7048359c6655706ad09710

- Problem: Object detection.
- Dataset: COCO 2017, 23GB.
- Model: SSD 300 with Resnet50 backbone (Pytorch). 22M params

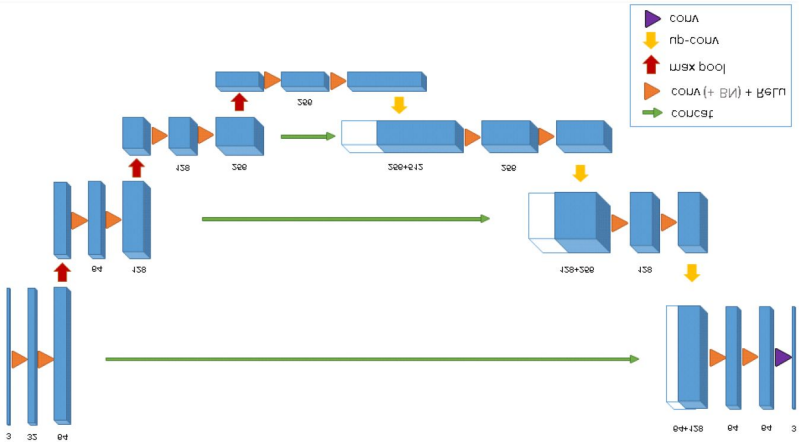
Liu *et al.*



Tensor Cores vs non-Tensor Cores Experiment: U-Net3D

Experiment setting:https://harsh-fold-50d.notion.site/GPU-Benchmark-Project-c3f1e3a80c7048359c6655706ad09710

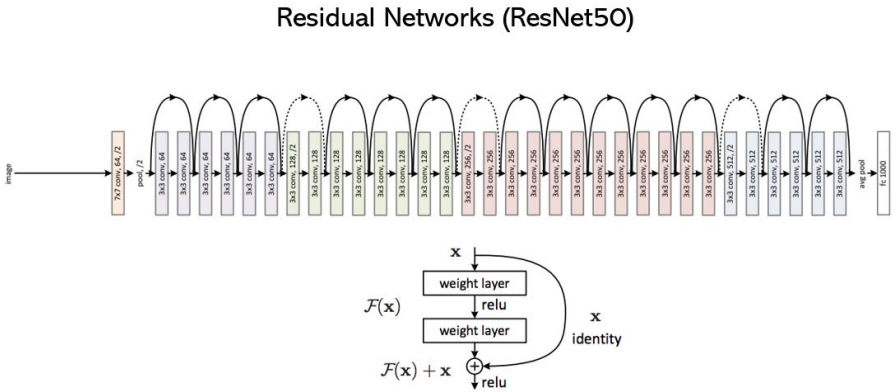
- Problem: 3D Segmentation.
- Dataset: 2019 Kidney Tumor Segmentation Challenge, 17G.
- Model: U-Net3D (Pytorch). 31,2M params



Tensor Cores vs non-Tensor Cores Experiment: Resnet50

Experiment setting:https://harsh-fold-50d.notion.site/GPU-Benchmark-Project-c3f1e3a80c7048359c6655706ad09710

- Problem: Image Classification.
- Dataset: Imagenet.
- Model: Resnet50 (Pytorch). Over 23M params, 4 GFLOPs



GPU Benchmark Project: Features

- Training with amp (auto mixed precision)
- Distributed training with torchrun, amp, DDP(model)
- Profiling using tensorboard, nsight system
- Logging using DLLogging, wandb
- Loading data with Dali (NVLink supported)

GPU Benchmark Project: Metrics

- Benchmarking Time (seconds)
- Throughput (Images/second)
- WanDB metric (GPU Utilization, CPU usage) (check on wandb website)

GPU Benchmark Metrics: Benchmarking Time

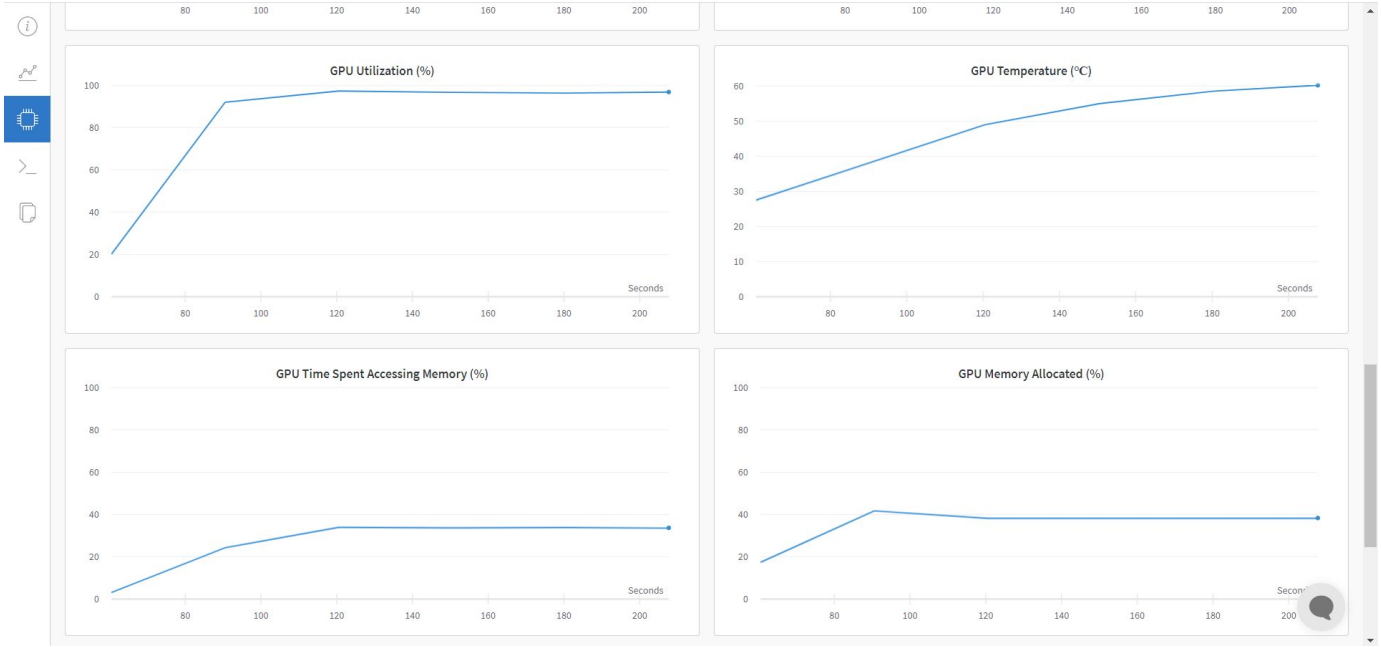
```
# contain benchmarking warmup time and total time


start_epoch_time = time.time()
…
end_epoch_time = time.time() - start_epoch_time
total_time += end_epoch_time
if args.local_rank == 0:
   logger.update_epoch_time(epoch, end_epoch_time)
```

GPU Benchmark Metrics: Throughput

```
    if nbatch >= run_info.benchmark_warmup:
          torch.cuda.synchronize()
          logger.update(run_info.batch_size*run_info.N_gpu, time.time() - start_time)


result_.data[0] = logger.print_result()
if run_info.N_gpu > 1:
    torch.distributed.reduce(result_, 0)
if run_info.local_rank == 0:
    print('Training performance = {} FPS'.format(float(result_.data[0])))
```

GPU Benchmark Metrics: GPU metrics logging

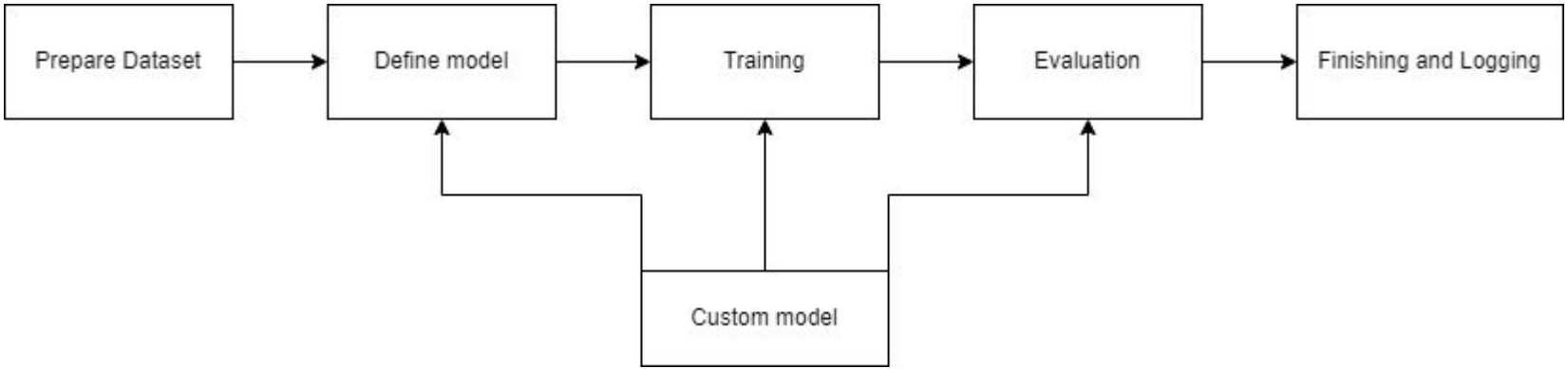
```
import wandb
wandb.init()
```



GPU Benchmark Project: Project Structure

```
Source Code: https://github.com/daovietanh190499/Custom_Benchmarking

< PROJECT ROOT >

   |-- main.py                                    # Main running file <args, config>

   |-- train.py                                   # Train code

   |-- evaluate.py                                # Evaluate code

   |-- requirements.txt                           # Development modules

   |-- Dockerfile                                 # For building image

   |-- logger.py                                  # Logging code
```

GPU Benchmark Project: Pipeline



GPU Benchmark Project: Dockerfile

```
ARG FROM_IMAGE_NAME=nvcr.io/nvidia/pytorch:22.10-py3
FROM ${FROM_IMAGE_NAME}
# Set working directory
WORKDIR /workspace/benchmark
# Copy the model files
COPY . .
# Install python requirements
RUN pip install --no-cache-dir -r requirements.txt
ENV CUDNN_V8_API_ENABLED=1
ENV TORCH_CUDNN_V8_API_ENABLED=1
```

GPU Benchmark Project: Run docker image

```
$ docker build -t nvidia-universal-benchmark .
```

```
$ docker run --rm -it --gpus device=<GPU-ID> --ipc=host -v <RAW-DATA-DIR>:/data -v
<RESULT-DIR>:/results nvidia_universal_benchmark -e WANDB_API_KEY=<WANDB-API-KEY>
```

change <GPU-ID>, <RAW-DATA-DIR>, <RESULT-DIR>, <WANDB-API-KEY>

GPU Benchmark Project: AMP (FP16 mixed-precision)

```
scaler = torch.cuda.amp.GradScaler(enabled=args.amp)

…

with torch.cuda.amp.autocast(enabled=run_info.amp):
     result = model_func(model, data, forward_info)
     loss = loss_func(result)
…
scaler.scale(loss).backward()
scaler.step(optim)
scaler.update()
optim.zero_grad()

#run inside docker bash
$ torchrun --nproc_per_node=1 main.py --batch-size 32 --mode benchmark-training
--benchmark-warmup 100 --benchmark-iterations 300 --data /data —-wandb
```

GPU Benchmark Project: Enable TF32 mixed-precesion

```
torch.backends.cuda.matmul.allow_tf32 = args.allow_tf32
torch.backends.cudnn.allow_tf32 = args.allow_tf32
torch.backends.cudnn.benchmark = True




#run inside docker bash
$ torchrun --nproc_per_node=1 main.py --batch-size 32 --mode benchmark-training
--benchmark-warmup 100 --benchmark-iterations 300 --data /data —-wandb
```

GPU Benchmark Project: Enable Distributed Training

```
args.distributed = False
   if 'WORLD_SIZE' in os.environ:
       args.distributed = int(os.environ['WORLD_SIZE']) > 1
   if args.distributed:
       torch.cuda.set_device(args.local_rank)
       torch.distributed.init_process_group(backend='nccl', init_method='env://')
       args.N_gpu = torch.distributed.get_world_size()
   else:
       args.N_gpu = 1
```

GPU Benchmark Project: Enable Distributed Training

```
    if args.seed is None:
        args.seed = np.random.randint(1e4)
    if args.distributed:
        args.seed = (args.seed + torch.distributed.get_rank()) % 2**32
…
if args.distributed:
        model = DDP(model)
if args.checkpoint is not None:
        if os.path.isfile(args.checkpoint):
            load_checkpoint(model.module if args.distributed else model, args.checkpoint)
…
```

GPU Benchmark Project: Enable Distributed Training

```
if args.distributed:
                obj['model'] = model.module.state_dict()
            else:
                obj['model'] = model.state_dict()
…
if run_info.N_gpu > 1:
        torch.distributed.reduce(result_, 0)



#run inside docker bash
$ torchrun --nproc_per_node=2 main.py --batch-size 32 --mode benchmark-training
--benchmark-warmup 100 --benchmark-iterations 300 --data /data —-wandb

#change nproc_per_node for distributed training
```

Benchmark Result: SSD300

Benchmark results of A100 (batch size 32):

Benchmarking Scenario	Time (seconds per 100 iterations after warming up)	Throughput (images/sec)
A100 + FP16 (Mixed-Precision) + Tensor Cores	12.560	509.568
A100 + TF32 (Only for operations that support TF32) + Tensor Cores	21.567	296.746
A100 + FP32 + no Tensor Cores	54.642	117.127

Benchmark Result: SSD300

Benchmark results of V100 (batch size 32):

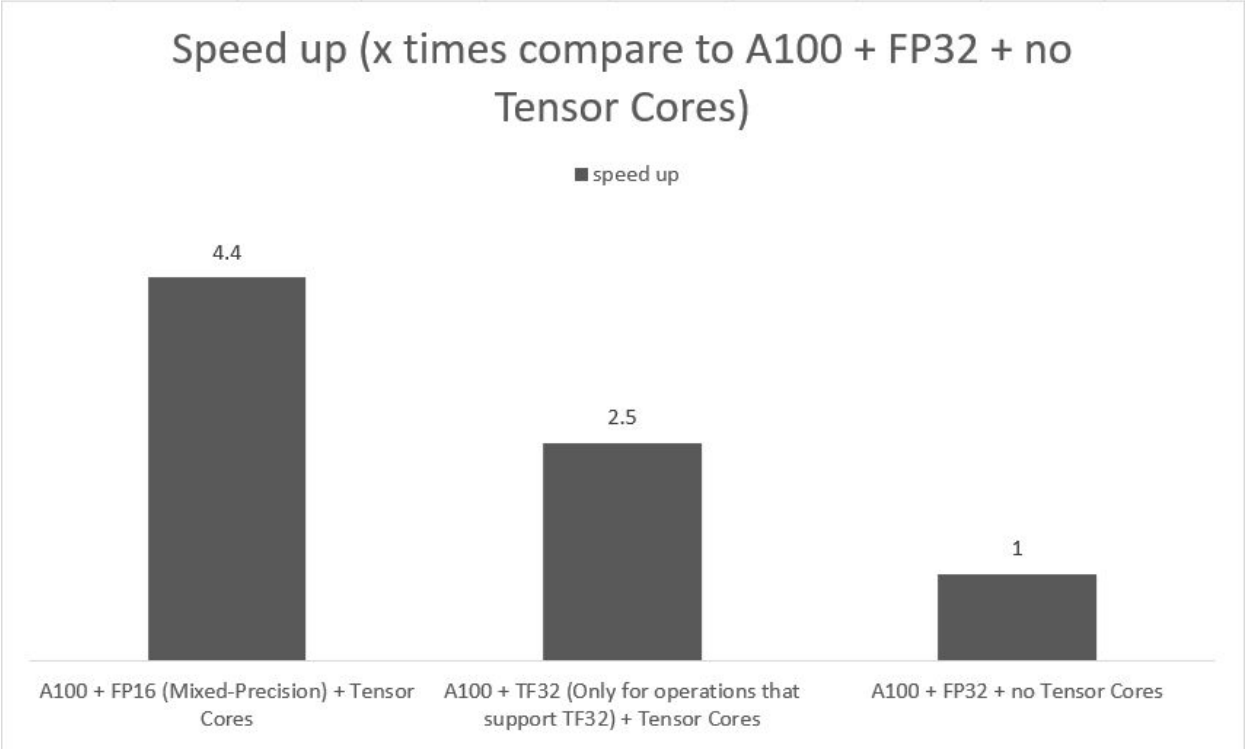
Benchmarking Scenario	Time (seconds per 200 iterations after warming up)	Throughput (images/sec)
V100 + FP16 (Mixed-Precision) + Tensor Cores	22.856	280.019
V100 + FP32 + no Tensor Cores	69.965	91.474

Benchmark Result: SSD300

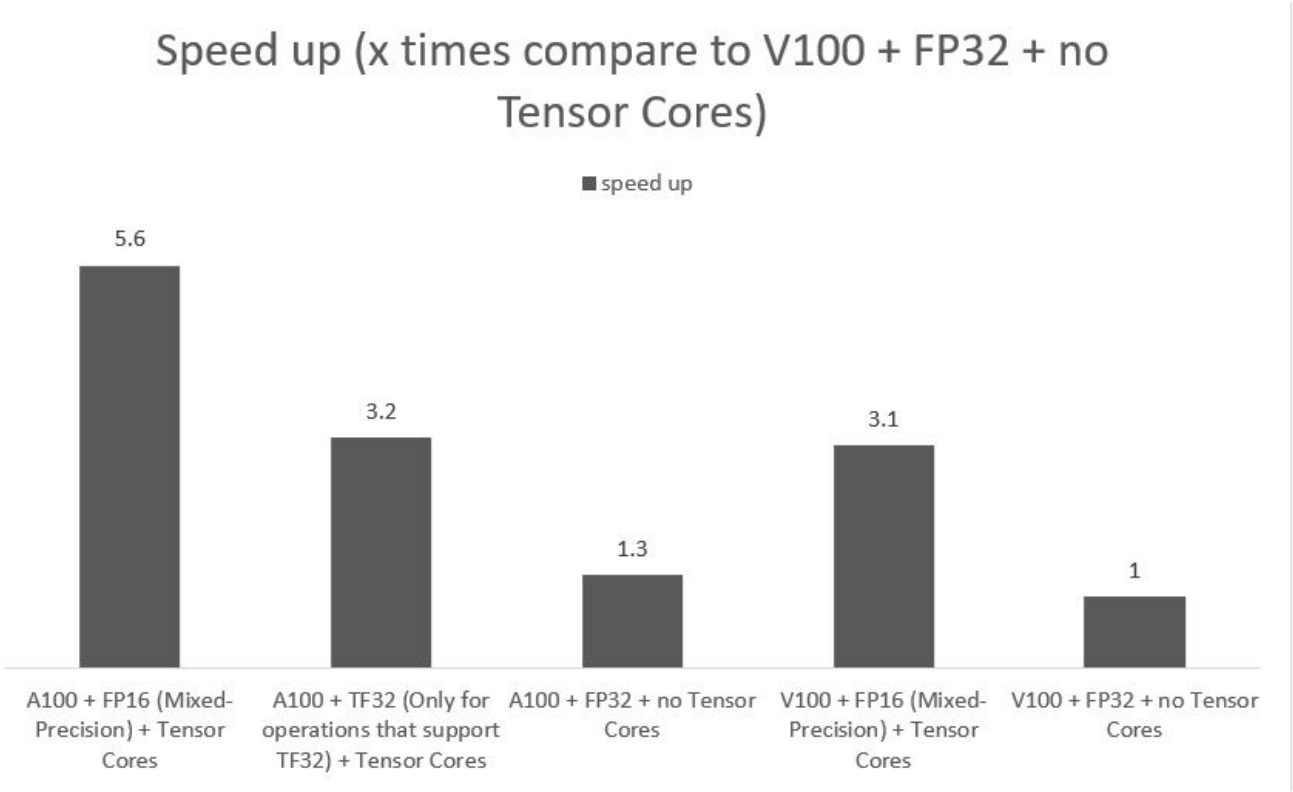
Benchmark results of P100 (batch size 32):

Benchmarking Scenario	Time (seconds per 200 iterations after warming up)	Throughput (images/sec)
P100 + FP16 (Mixed-Precision) + no Tensor Cores	108.821	58.812
P100 + FP32 + no Tensor Cores	593.810	10.778

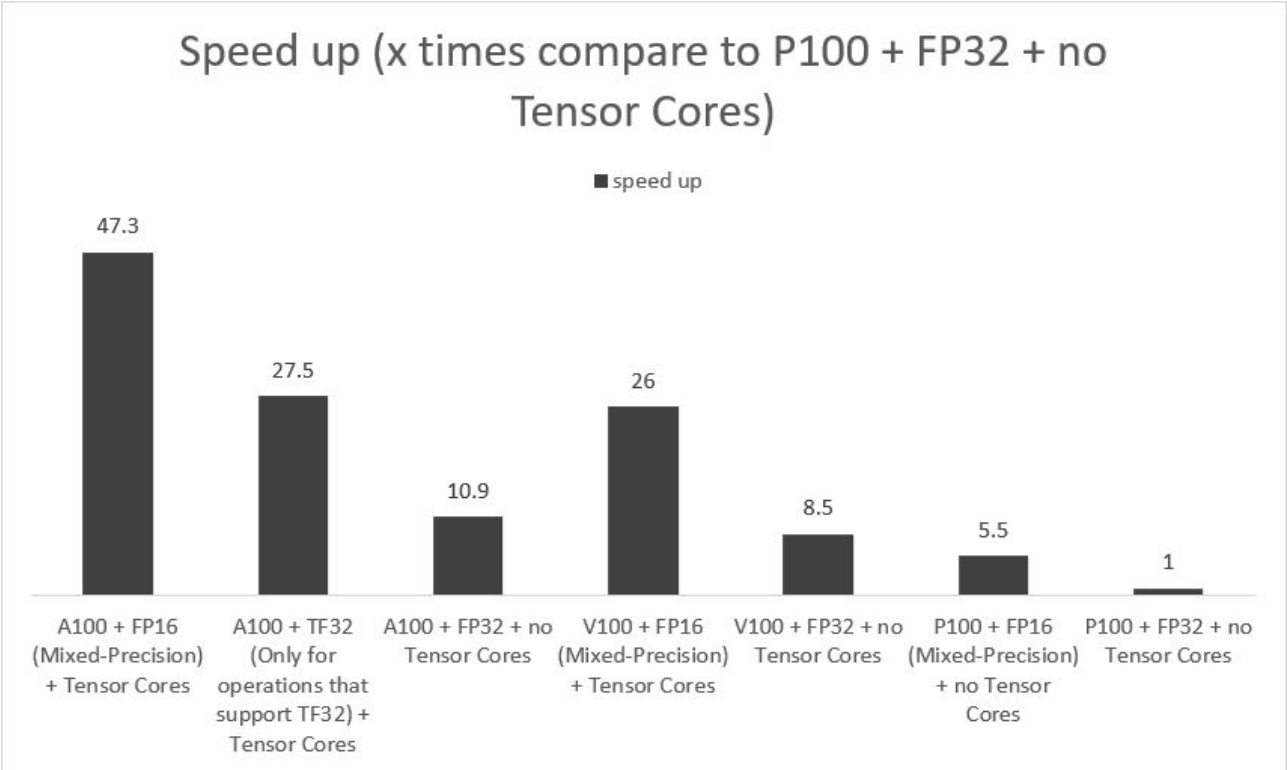
Benchmark Result: SSD300



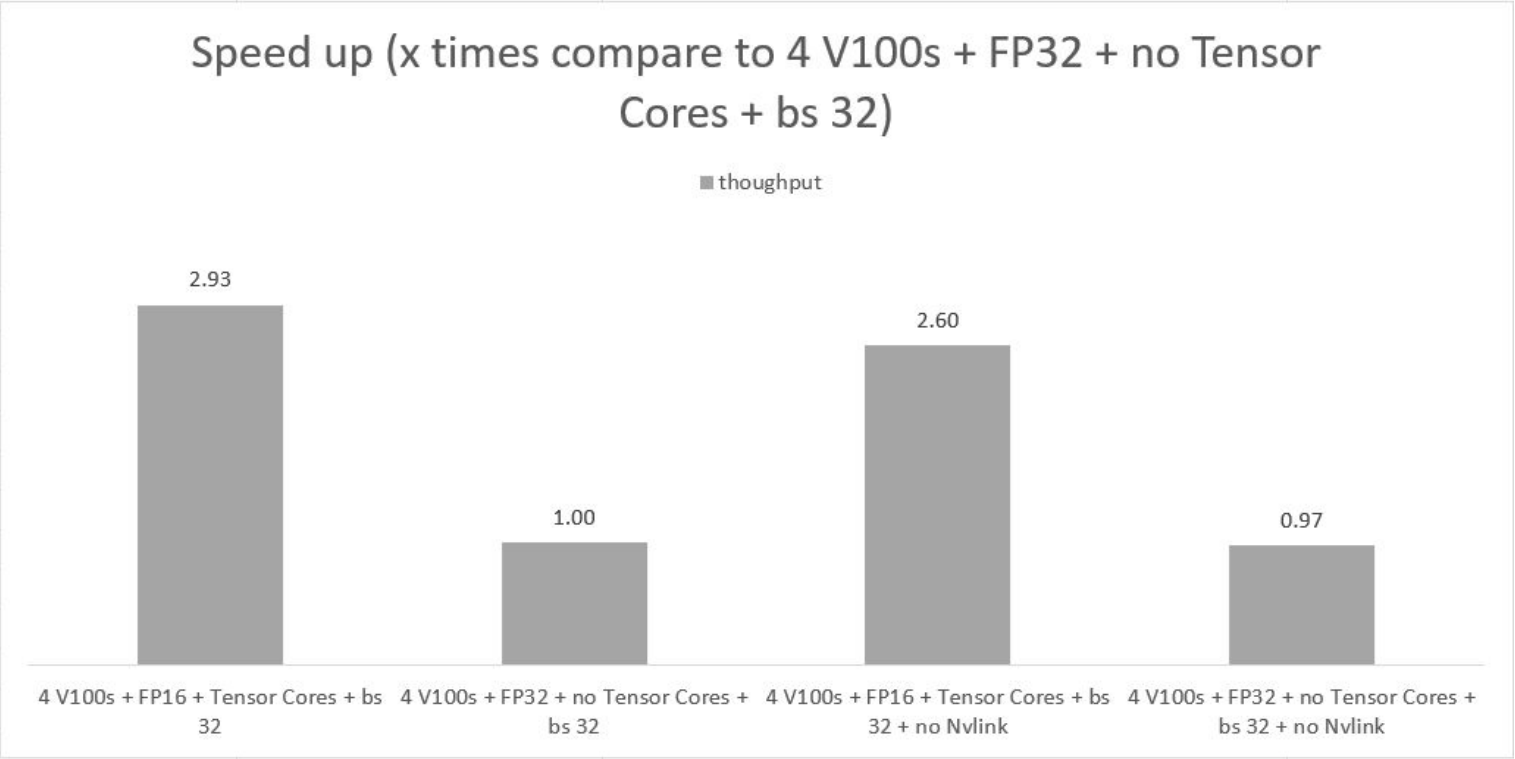
Benchmark Result: SSD300



Benchmark Result: SSD300



Multi-GPUs & NVLink performance (HGX - 4xV100): SSD300



Benchmark Result: U-Net3D

Benchmark results of A100 (batch size 2):

Benchmarking Scenario	Time (seconds per 100 iterations after warming up)	Throughput (images/sec)
A100 + FP16 (Mixed-Precision) + Tensor Cores	23.855	16.768
A100 + TF32 (Only for operations that support TF32) + Tensor Cores	40.057	9.986
A100 + FP32 + no Tensor Cores	195.583	2.045

Benchmark Result: U-Net3D

Benchmark results of V100 (batch size 2):

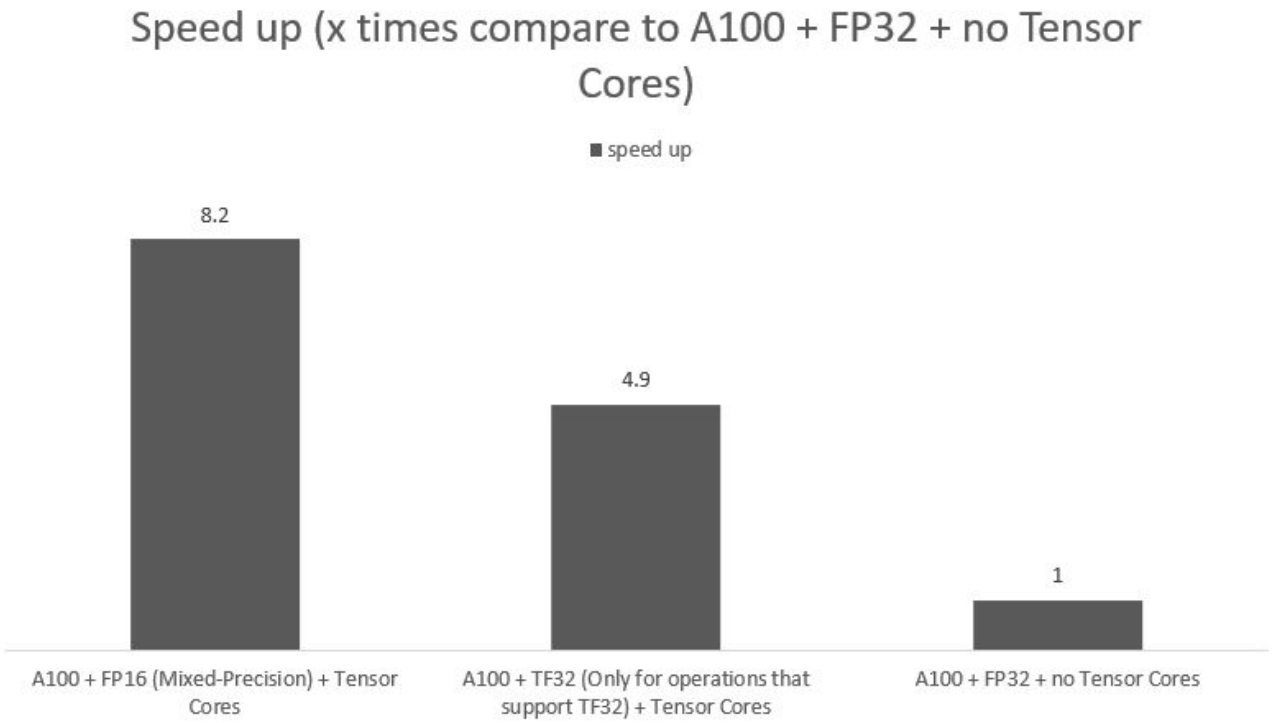
Benchmarking Scenario	Time (seconds per 200 iterations after warming up)	Throughput (images/sec)
V100 + FP16 (Mixed-Precision) + Tensor Cores	39.989	10.003
V100 + FP32 + no Tensor Cores	158.625	2.522

Benchmark Result: U-Net3D

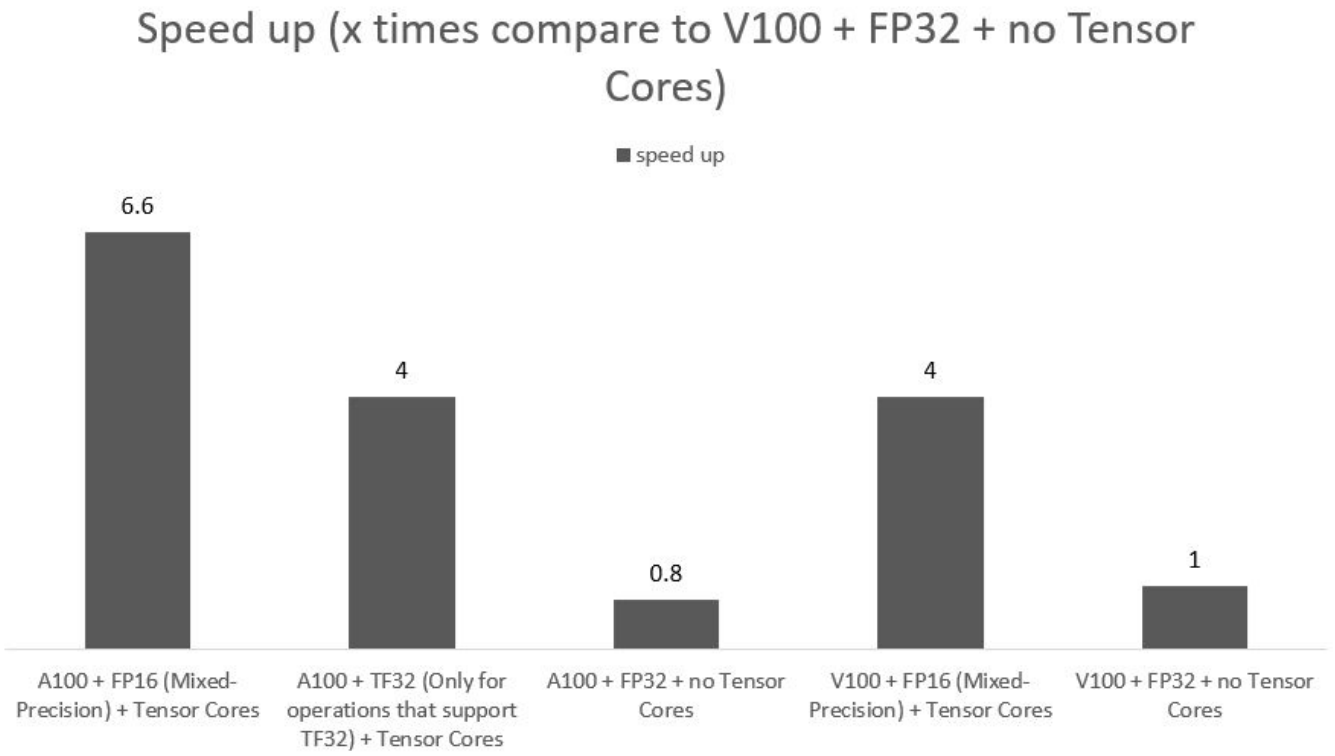
Benchmark results of P100 (batch size 2):

Benchmarking Scenario	Time (seconds per 200 iterations after warming up)	Throughput (images/sec)
P100 + FP16 (Mixed-Precision) + no Tensor Cores	RuntimeError: cuDNN error: CUDNN_STATUS_EXECUTION_FAILED	
P100 + FP32 + no Tensor Cores	251.700	1.589

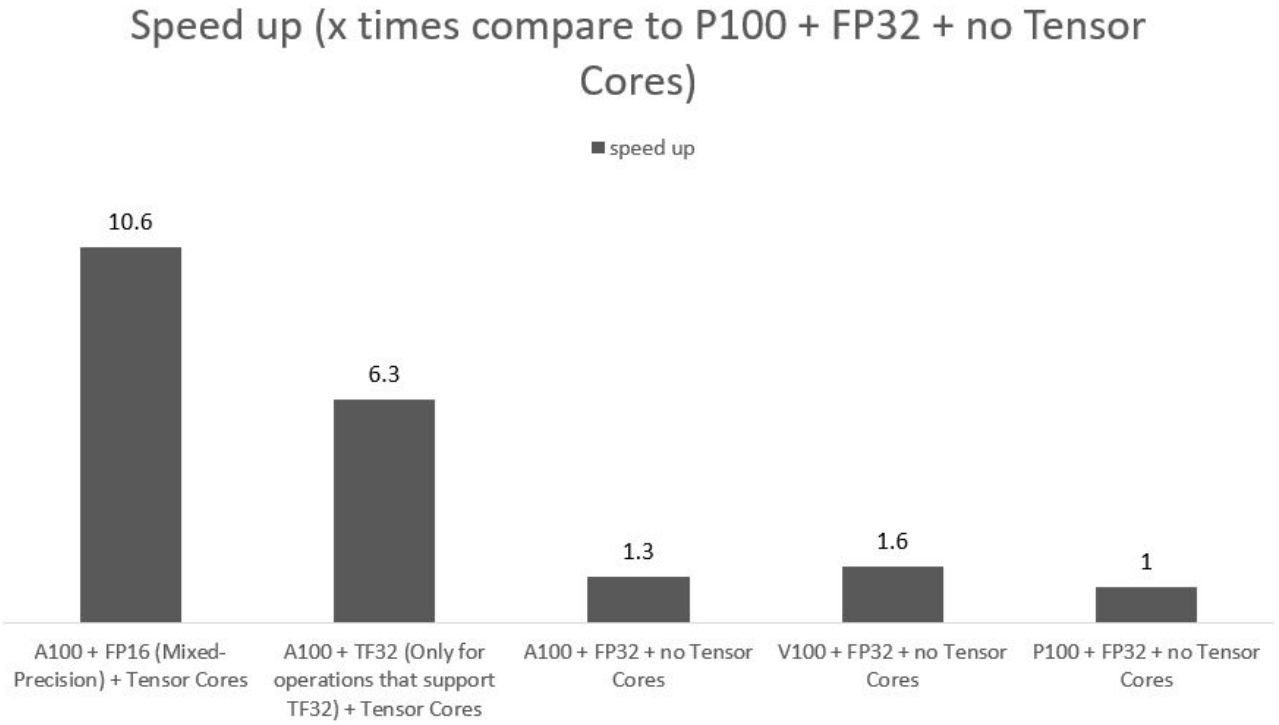
Benchmark Result: U-Net3D



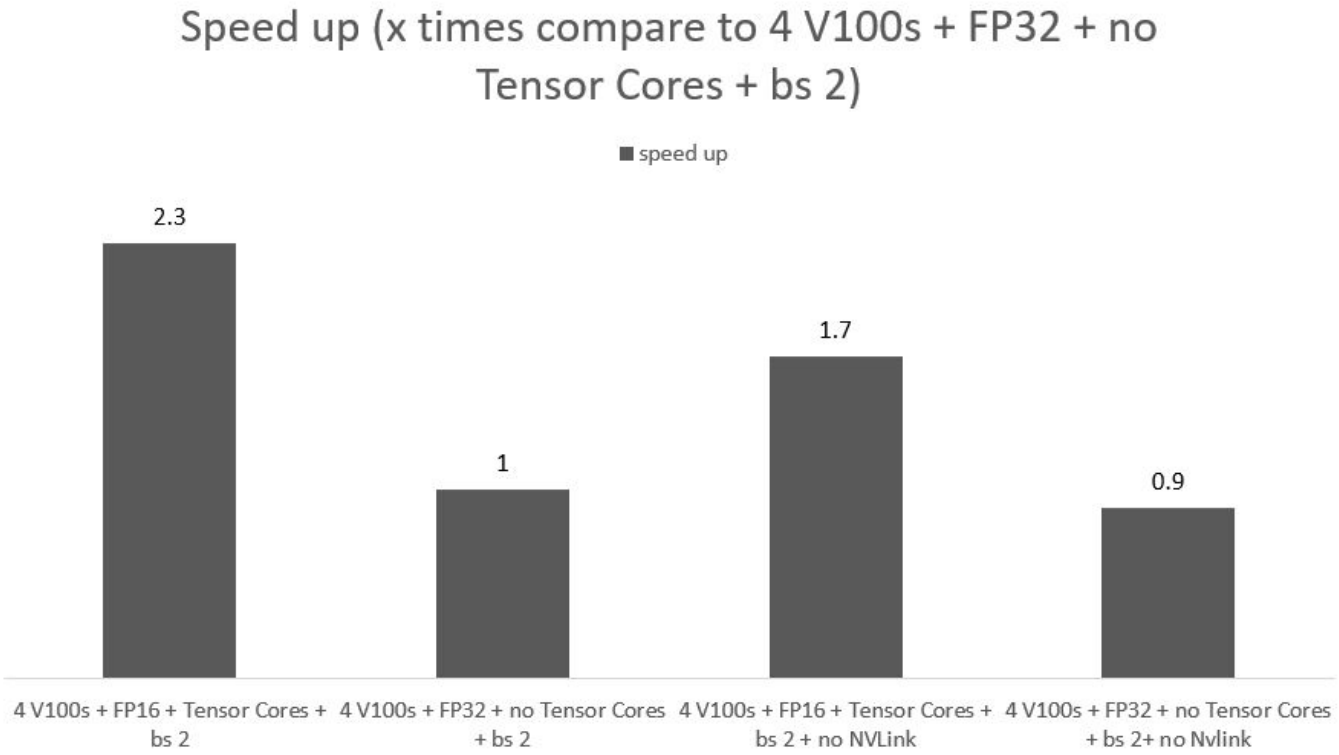
Benchmark Result: U-Net3D



Benchmark Result: U-Net3D



Multi-GPUs & NVLink performance (HGX - 4xV100): U-Net3D



Benchmark Result: ResNet50

Benchmark results of A100 (batch size 512):

Benchmarking Scenario	Time (seconds per 100 iterations after warming up)	Throughput (images/sec)
A100 + FP16 (Mixed-Precision) + Tensor Cores	11.986	1052.12
A100 + TF32 (Only for operations that support TF32) + Tensor Cores	15.494	826.56
A100 + FP32 + no Tensor Cores	27.457	466.30

Benchmark Result: ResNet50

Benchmark results of V100 (batch size 512):

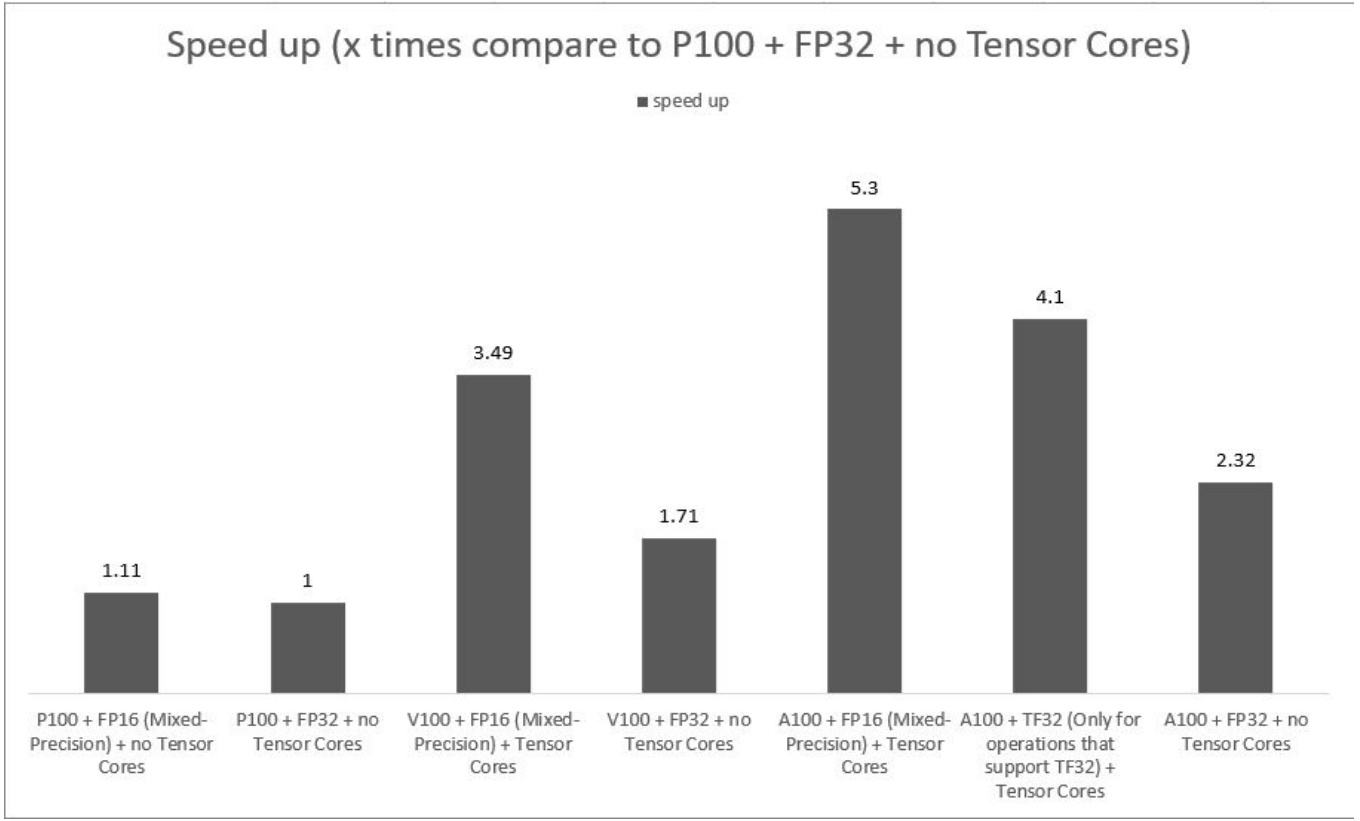
Benchmarking Scenario	Time (seconds per 200 iterations after warming up)	Throughput (images/sec)
V100 + FP16 (Mixed-Precision) + Tensor Cores	18.221	703.25
V100 + FP32 + no Tensor Cores	37.321	343.02

Benchmark Result: ResNet50

Benchmark results of P100 (batch size 512):

Benchmarking Scenario	Time (seconds per 200 iterations after warming up)	Throughput (images/sec)
P100 + FP16 (Mixed-Precision) + no Tensor Cores	57.695	221.25
P100 + FP32 + no Tensor Cores	63.473	201.75

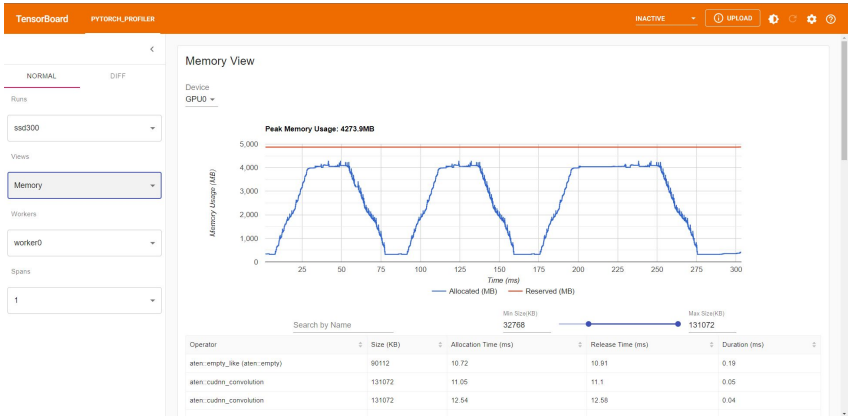
Benchmark Result: ResNet50



GPU Profiling

Using Pytorch Profiler, you can:

- Trace back which kernel takes time and memory
- Which kernel is using Tensor Core and percentage of Tensor Cores usage
- Memory and computation overhead of each functions



GPU Profiling Source Code

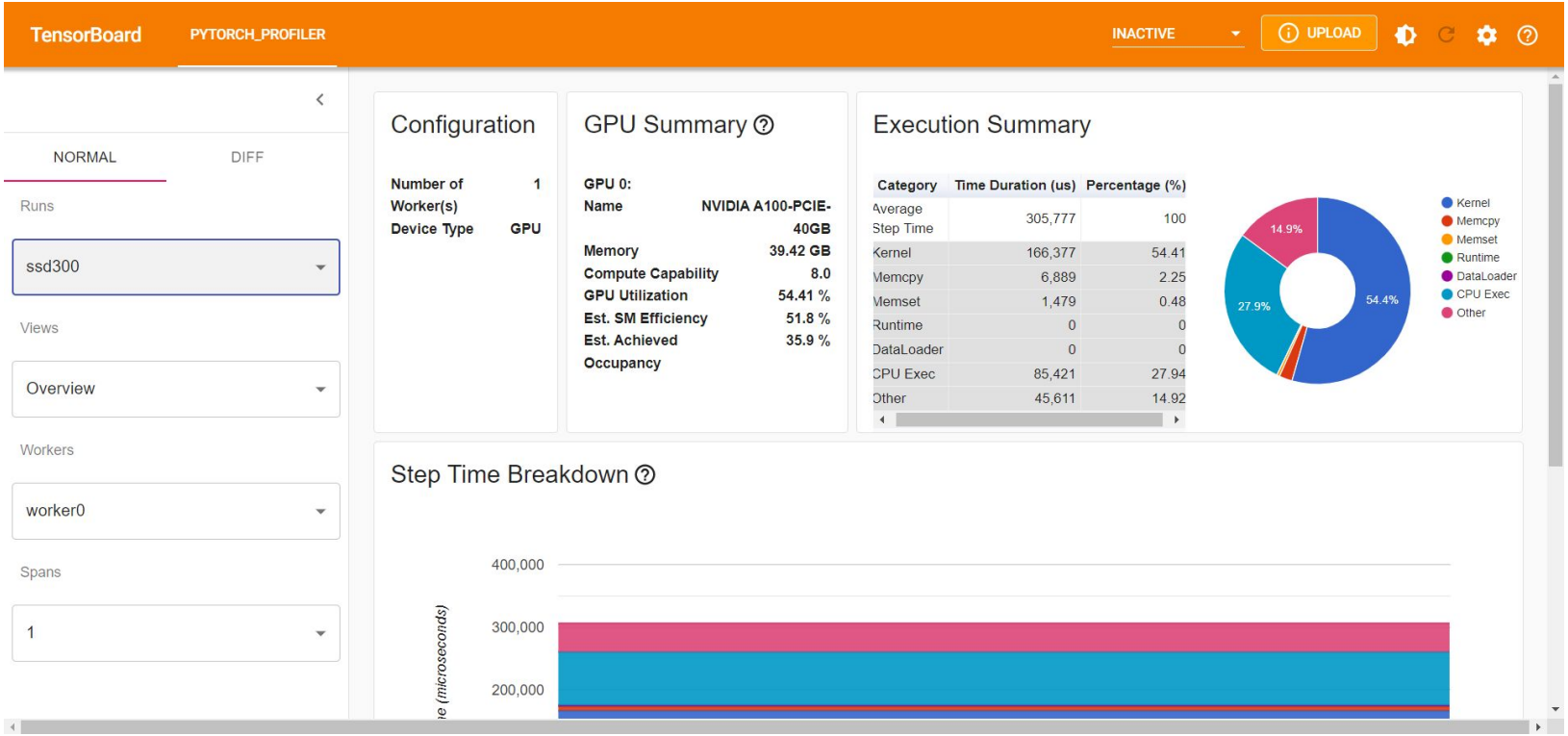
```
with torch.profiler.profile(
     schedule=torch.profiler.schedule(wait=1,  warmup=1, active=3, repeat=2),
     on_trace_ready=torch.profiler.tensorboard_trace_handler('./log/<your model>'),
      record_shapes=True,
      profile_memory=True,
      with_stack=True) as prof:
    for _ in range(10):
        outputs = model(inputs)
        prof.step()
    prof.stop()
```

```
#Log result:
https://github.com/daovietanh190499/Custom_Benchmarking/releases/download/v1.0.0/log.zip
```

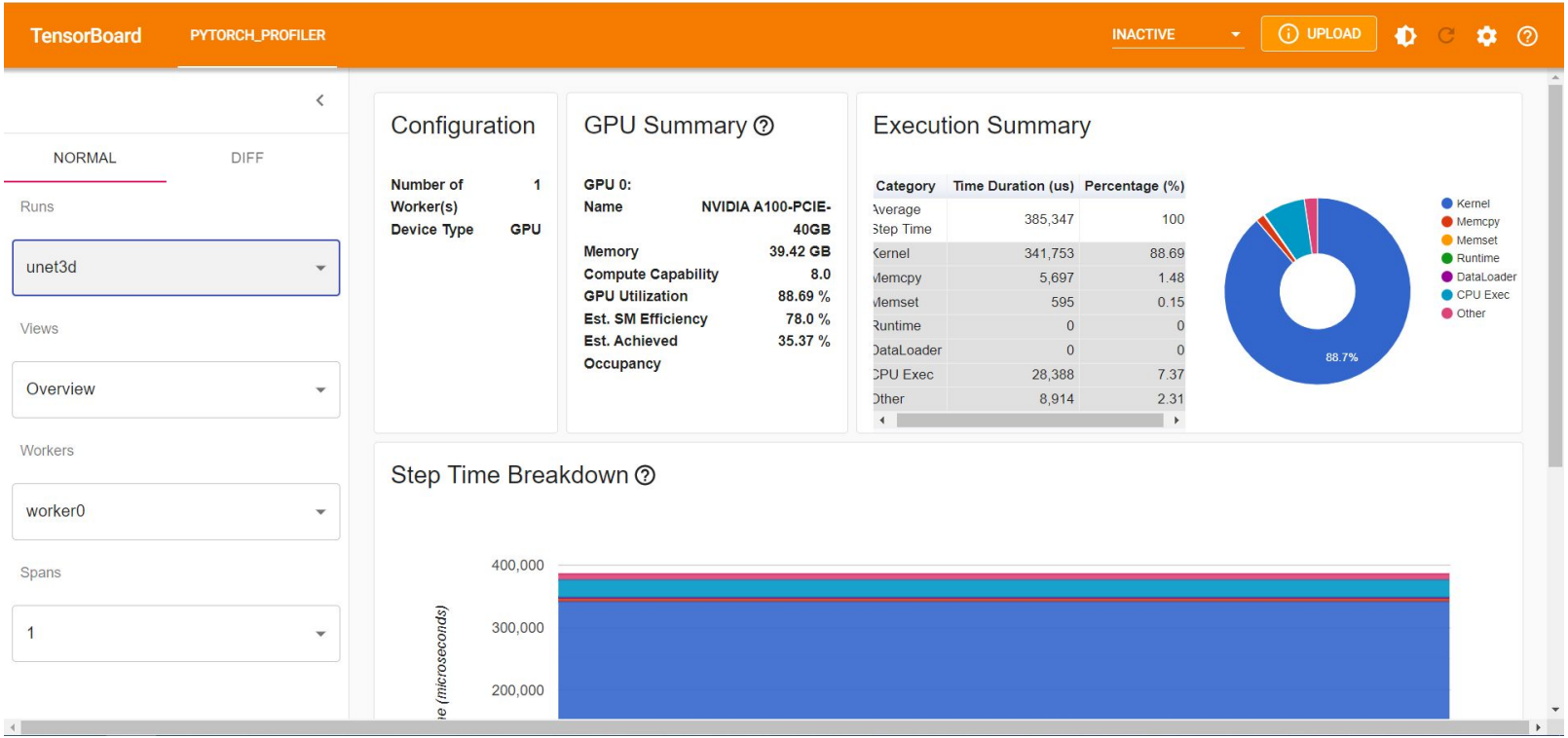
GPU Utilization, Memory, Computation and IO

	GPU Utilization (%)	Memory (%)	Computation (%)	IO (%)
A100 + FP16 + Tensor Cores SSD300	**24.07**	**4.91**	**27.46**	**27.2**
A100 + TF32 + Tensor Cores SSD300	**51.6**	**15.23**	**18.91**	**27.2**
A100 + FP32 + no Tensor Cores SSD300	**99**	**36.05**	**11.06**	**27.2**
A100 + FP16 + Tensor Cores Unet3D	**64.47**	**15.61**	**6.36**	**27.5**
A100 + TF32 + Tensor Cores Unet3D	**91.53**	**32.94**	**6.24**	**27.5**
A100 + FP32 + no Tensor Cores Unet3D	**100**	**28.84**	**6.34**	**27.5**

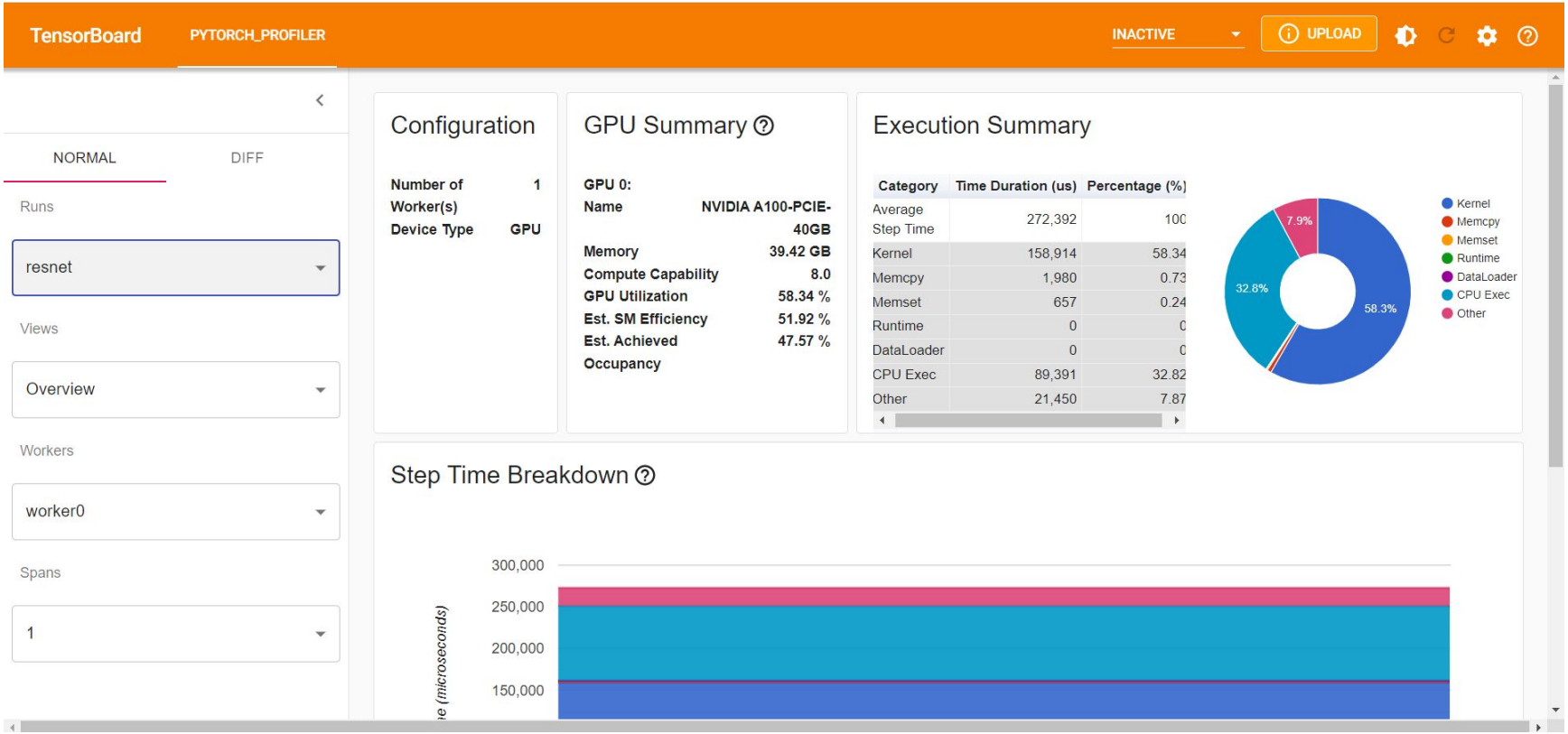
GPU Overview (A100 mixed-precision): SSD300



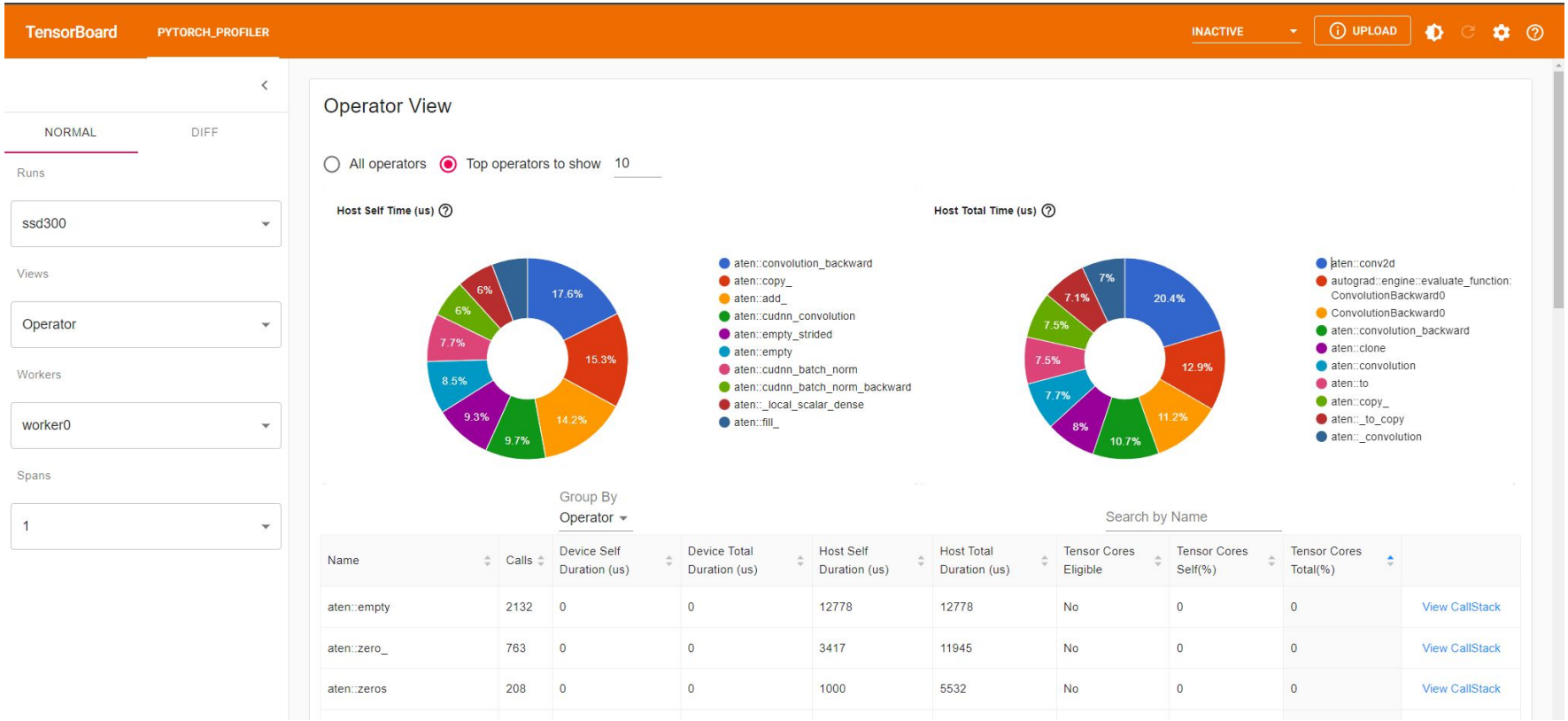
GPU Overview (A100 mixed-precision): U-Net3D



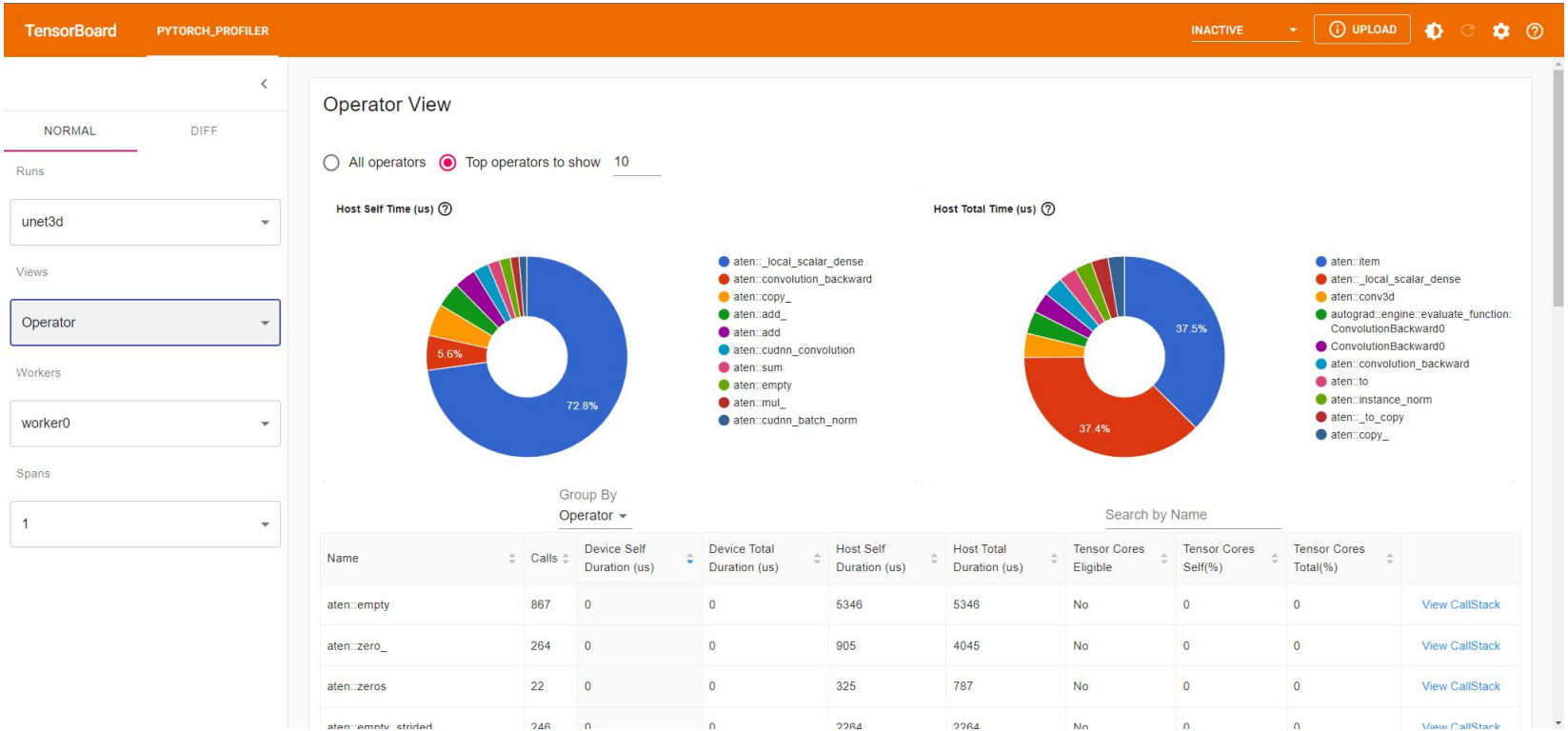
GPU Overview (A100 mixed-precision): ResNet50



GPU Computation and IO (A100 mixed-precision): SSD300

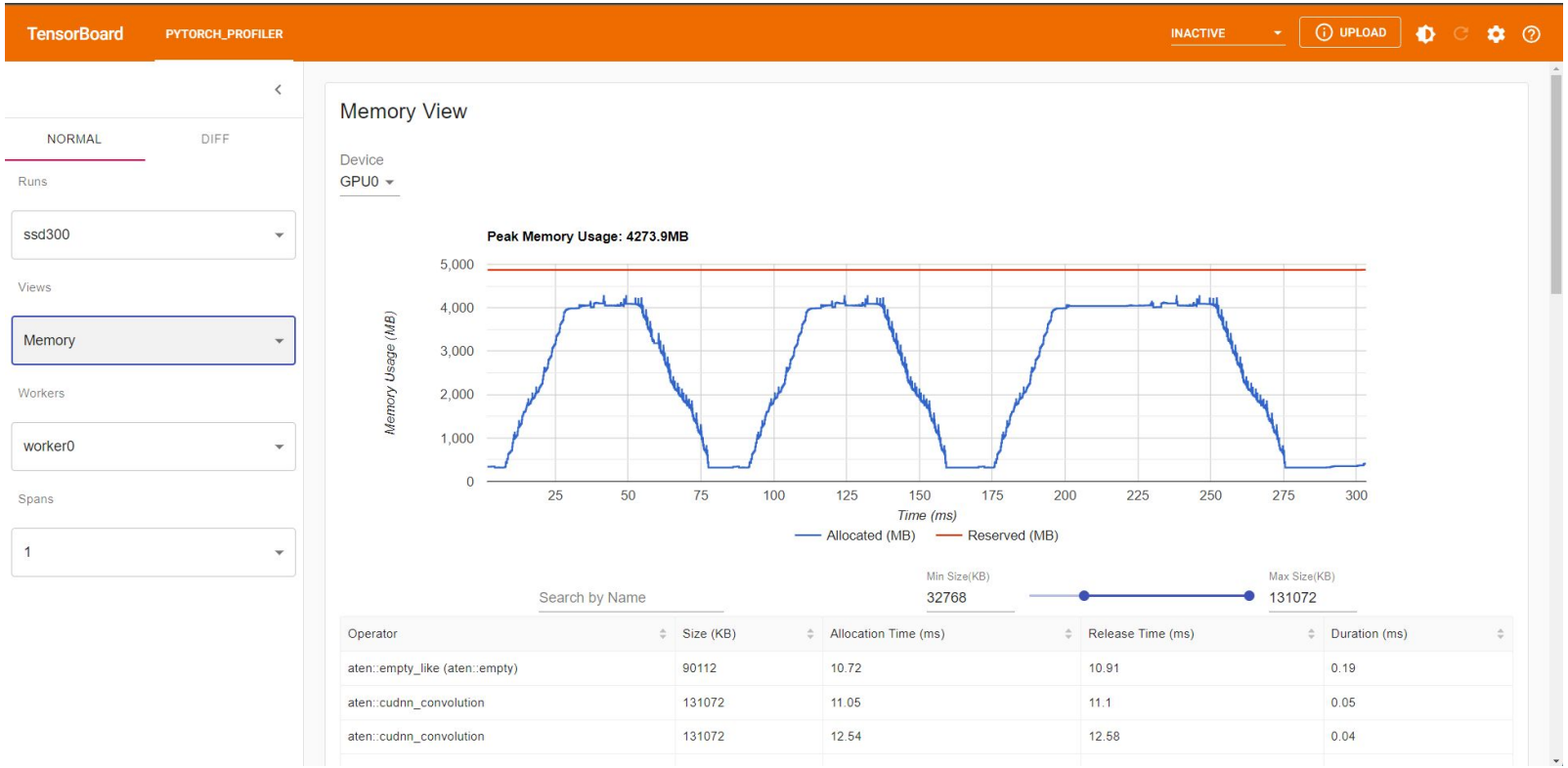


GPU Computation and IO (A100 mixed-precision): U-Net3D

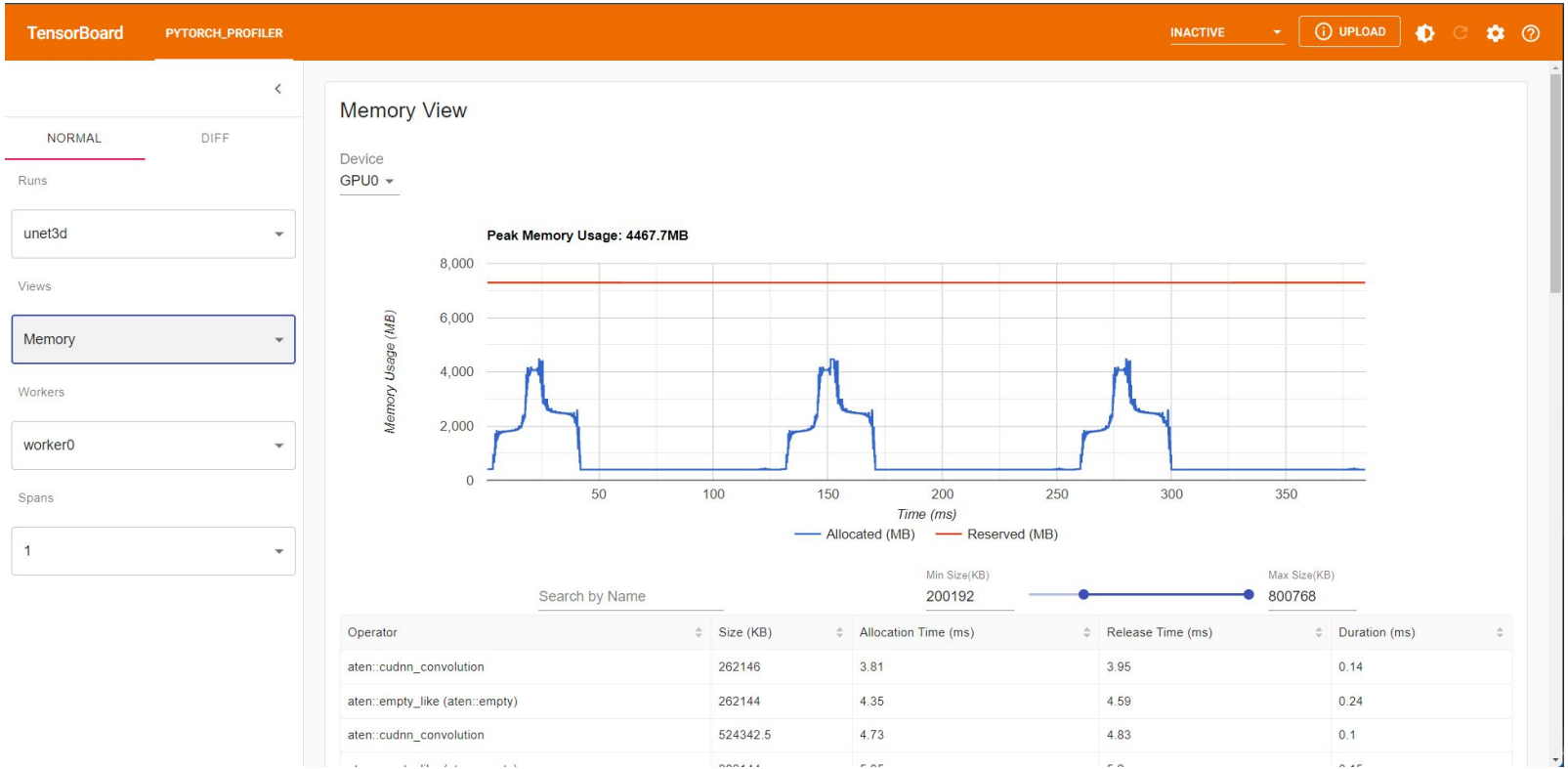


GPU Computation and IO (A100 mixed-precision): ResNet50

GPU Memory (A100 mixed-precision): SSD300



GPU Memory (A100 mixed-precision): U-Net3D



GPU Memory (A100 mixed-precision): ResNet50