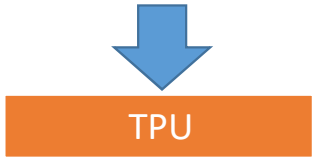
GPUs (NVIDIA)
vs.
TPUs (Google)

A Benchmark Study using
MNIST data set

Invoking TPU in GCP (Not in COLAB); COLAB has only 2 choice of TPUs (v5, v6)

```
martinworkmbcet@t1v-n-47be96ca-w-0:~$ python3
Python 3.8.10 (default, Mar 18 2025, 20:04:55)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> from torch import nn
>>> from torchvision import datasets
>>> from torchvision.transforms import ToTensor
>>>
>>> import torch.optim as optim
>>> import torch_xla.core.xla_model as xm
>>> import torch_xla.distributed.parallel_loader as pl
>>> import torch_xla.distributed.xla_multiprocessing as xmp
>>> from torch.utils.data import DataLoader
>>>
>>> device = xm.xla_device()
WARNING:root:XRT configuration not detected. Defaulting to preview PJRT runtime. To silence this
 device or configure XRT. To disable default device selection, set PJRT_SELECT_DEFAULT_DEVICE=0
WARNING:root:For more information about the status of PJRT, see https://github.com/pytorch/xla/bl
WARNING:root:libtpu.so and TPU device found. Setting PJRT_DEVICE=TPU.

>>> print(f"Using device:",device)
Using device: xla:0
```



Simple MNIST dataset Training

```
# Data loading setup (Standard Dataloader)
training_data = datasets.MNIST(root="data", train=True, download=True, transform=ToTensor())
test_data = datasets.MNIST(root="data", train=False, download=True, transform=ToTensor())
train_dataloader = DataLoader(training_data, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 12.7MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 334kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 3.19MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 23.3MB/s]
```

We used SAME MNIST dataset for the Benchmark

Simple MNIST dataset Training – TPU's device mapping

```
device = xm.xla_device()
```

```
# --- Execution Loop ---
from datetime import datetime
t1=datetime.now()
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")

    # 1. Wrap the train DataLoader for efficient transfer
    train_para_loader = pl.ParallelLoader(train_dataloader, [device])
    train(train_para_loader.per_device_loader(device), train_dataloader, model, loss_fn, optimizer)

    # 2. Wrap the test DataLoader
    test_para_loader = pl.ParallelLoader(test_dataloader, [device])
    test(test_para_loader.per_device_loader(device), test_dataloader, model, loss_fn)

print("Done!")
t2=datetime.now()
print("Benchmark Time",t2-t1)
```

device: xla:0 (**xla** means ***Accelerated Linear Algebra*** Compiler)

Simple MNIST dataset Training – GPU's device mapping

```
device = torch.device("cuda")
```

```
def train(dataloader, model, loss_fn, optimizer, device):
    size = len(dataloader.dataset)
    model.train()

    # Iterate over the standard DataLoader
    for batch, (X, y) in enumerate(dataloader):

        # *** GPU/CPU STEP: Explicitly move data to the device ***
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()

        # *** GPU/CPU STEP: Standard optimizer update (synchronous) ***
        optimizer.step()

        optimizer.zero_grad()
```

Benchmark of CPU, GPU, TPU (Tensor Processing Unit)					
	Dataset	Accelarator	ML LIB	BatchSize	Time (h:m:s)
1	MNIST	**CPU**	PyTorch	512	**00:01:24**
2	MINST	**GPU** T4	Cuda/PyTorch	512	**00:00:41**
3	MNIST	**TPU** v6e1:0	PyTorch	512	**00:00:23**

This implies using a Simple TPU (just a core of TPU is used, ie. No heavy Parallel Processes applied), TPU took only 50% of Computing Time with respect to GPU

Note That
it is For **epochs = 5**; And for more epochs, as it is required for Real Time Training, The Time Difference is Substantial.

And importantly,

Larger Matrix Data can be Effectively Handles in TPU than GPUs
(TPUs are designed to Handle larger Matrix sized data)

BatchSz.	TPU	GPU
1024	00:00:16	00:00:36
1536	00:00:15	00:00:36

Implies larger Batch sizes are effectively handled by TPUs

In this case Batch size of the matrix increased, so that training time in TPU is reduced from 23 sec.