



# A PyTorch Native LLM Training Framework

## An Industrial-Level Framework for "Ease of Use"



Company: **100s~1000s** New Models Each Week

### Industrial Training Framework



**Only Performance**

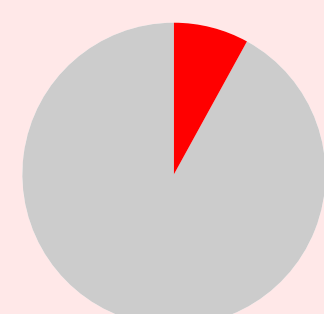


**Ease of Use**

But Current Frameworks are **Hard to Use**

#### Not PyTorch

**Only 8% non-PyTorch**

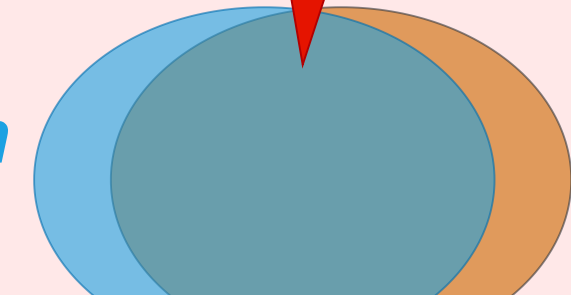


**HuggingFace Models**

#### Intertwined System and Model Design

**Intertwined Bug**

**System Code**



**Model Code**

#### Not Automated Enough

Tensor Parallel?  
Pipeline Parallel?  
nD Parallel?



**Manual Effort**

#### Hard to Debug

Where is Bug?

**Compiled Model -- "Black Box"**

NaN

#### No Distributed Checkpoint

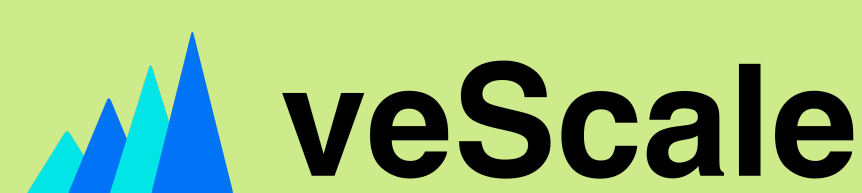
Saved Model

Device  
Device



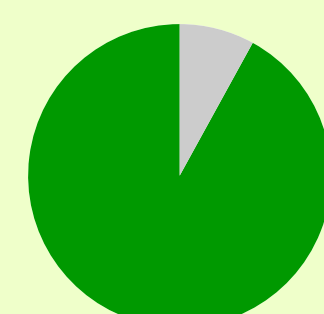
Load Model

Device  
Device  
Device



#### PyTorch Native

**92% PyTorch**

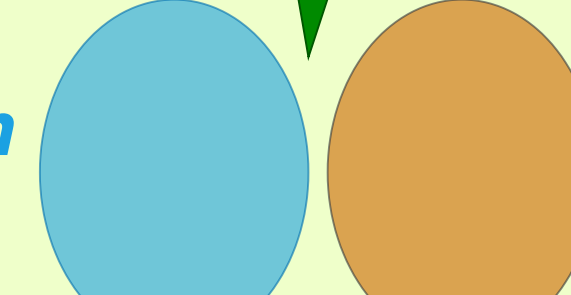


**HuggingFace Models**

#### Decoupled System and Model Design

**Full Isolation**

**System Code**



**Model Code**

#### Automatic Parallelism

Tensor, Sequence, Data, ZeRO,  
Pipeline Parallelism



**Minimal Manual Effort**

#### Easy to Debug

Line-by-Line Debug

**Eager Model -- "White Box"**

NaN

#### Auto Distributed Checkpoint

Saved Model

Device  
Device



Load Model

Device  
Device  
Device

### Overview

PyTorch-Native Model (Zero Code Change) on a Single Device

Auto-Plan Training with  $n$ -D Parallelism

Auto-Parallelize Model across Multiple Devices

Distributed Tensor (DTensor)

Auto-Reshard Distributed Model Checkpoints

Eager Only

Compile Only

Eager x Compile

**veScale**

### Preliminary Results

#### Simple API

##### 4D Parallel Training

```
# zero model code change
from huggingface import ModelClass, ModelArgs

# create fake model without actual memory usage (optional)
fake_model = DeferredInitModelClass("modelargs")

# initialize 4D device mesh
mesh = init_device_mesh("cuda", (dp_size, tp_size, mesh_dim_names["tp_2D"], "tp_3D"))

# parallelize model in tp 4D
real_tp_model = parallelize_model(fake_model, mesh["tp_3D"], sharding_plan)

# parallelize model in dp 2D
ddp_model = DDP(real_tp_model, mesh["dp_2D"])

# parallelize model with zero2
optimizer = DistributedOptimizer(torch.optim.AdamW, model=ddp_model)

# train model as if on a single device
for x in range(dataset):
    loss = ddp_model(x)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

##### Save Checkpoint

```
# prepare checkpoint state for the model and optimizer
checkpoint_state = {"model": distributed_model, "optimizer": distributed_optimizer}
# save the checkpoint
vescale.checkpoint.save("user/vescale/gpt", checkpoint_state)
```

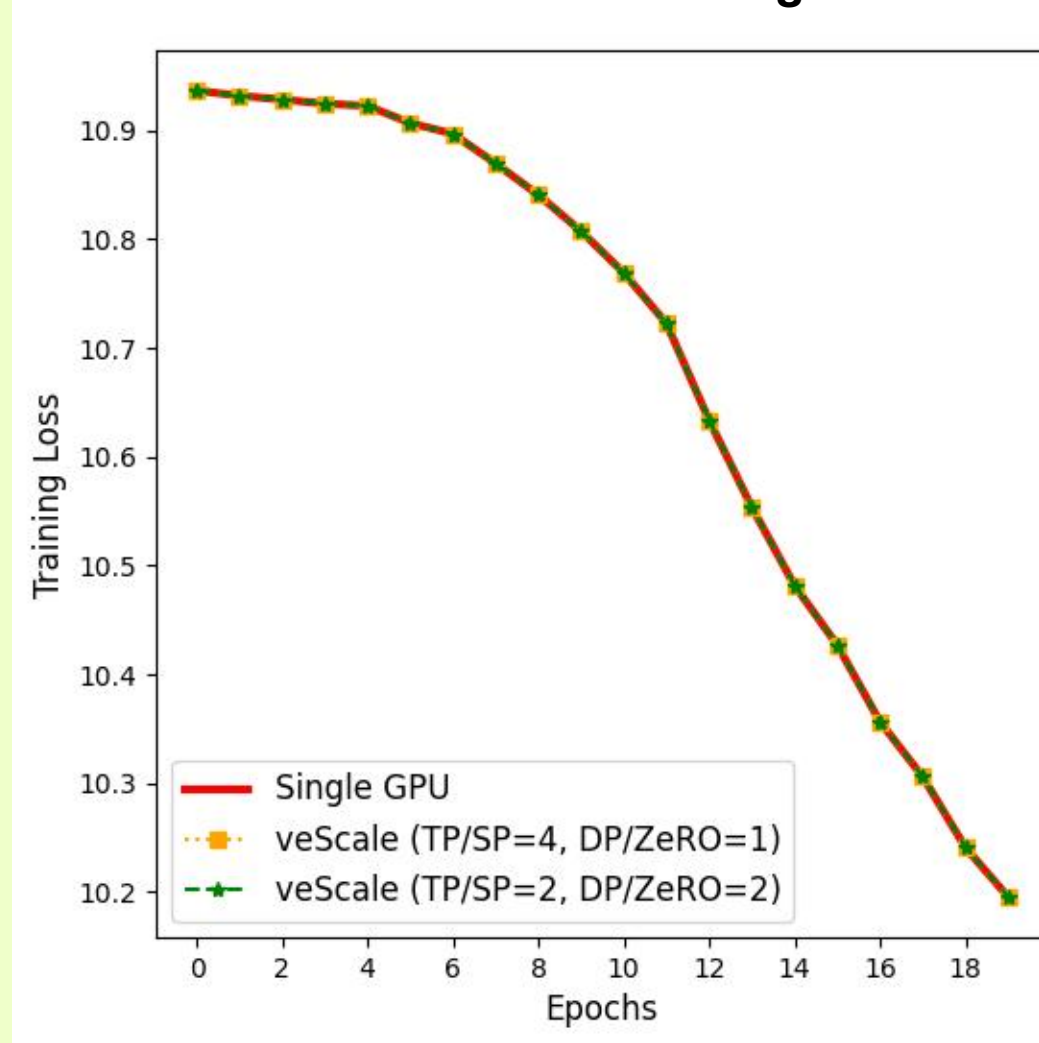
##### Load Checkpoint (under different world size or 4D degrees)

```
# prepare checkpoint state for the model and optimizer
checkpoint_state = {"model": distributed_model, "optimizer": distributed_optimizer}
# load the checkpoint
vescale.checkpoint.load("user/vescale/gpt", checkpoint_state)
```

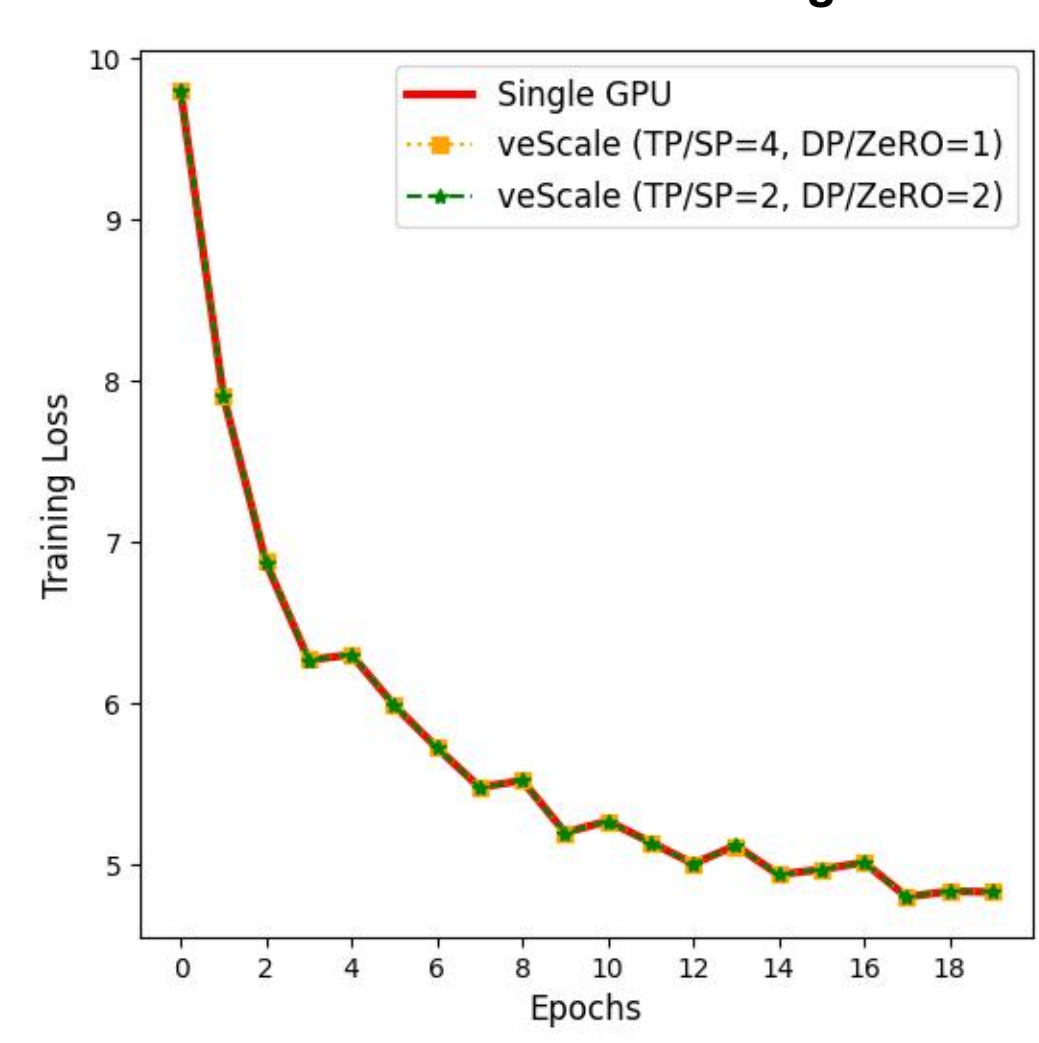
#### Bitwise Correctness



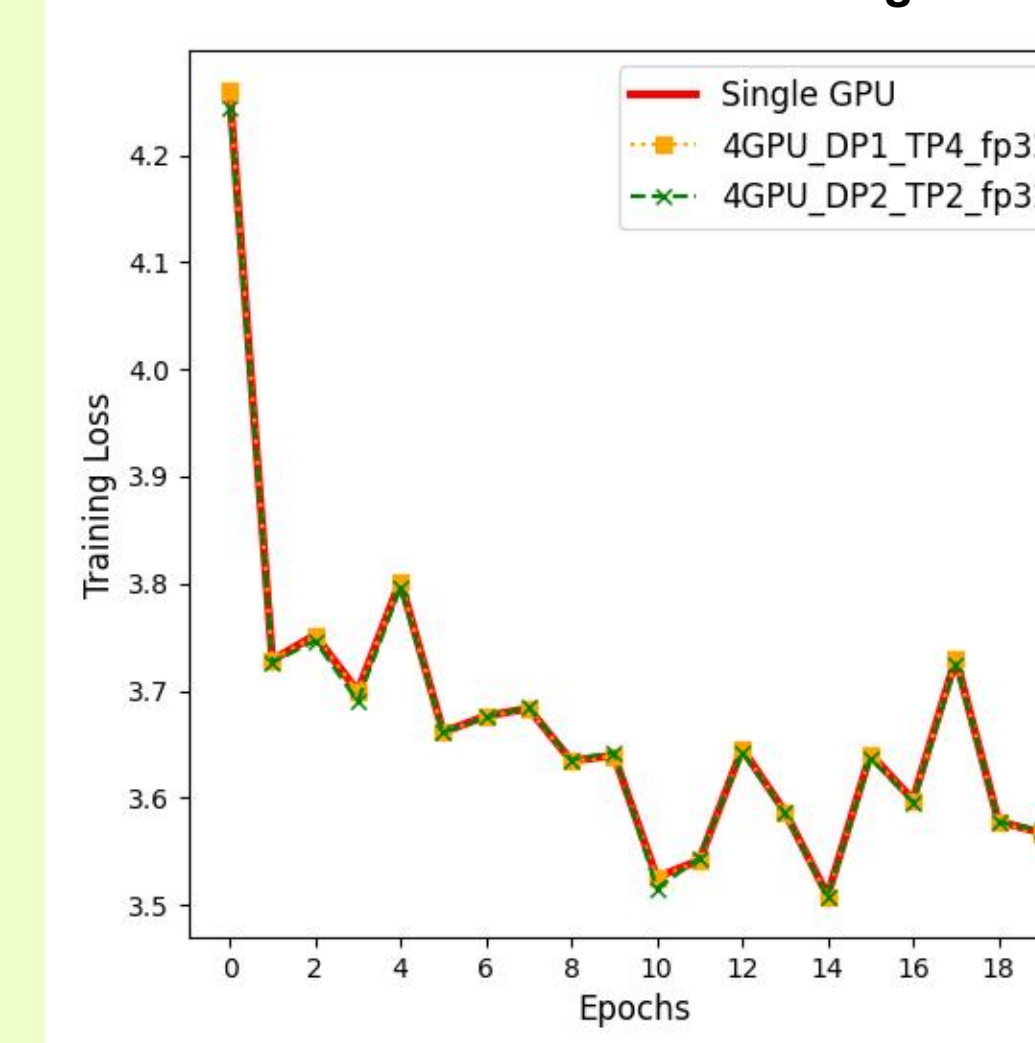
##### Mixtral 4D Training



##### LLaMa2 4D Fine-tuning



##### NanoGPT 4D Fine-Tuning



##### NanoGPT without veScale

