veScale

Open Source

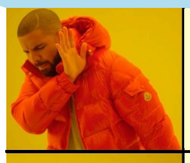
A PyTorch Native LLM Training Framework

An Industrial-Level Framework for "Ease of Use"

ByteDance

Company: 100s~1000s New Models Each Week

Industrial Training Framework



Only Performance

Ease of Use

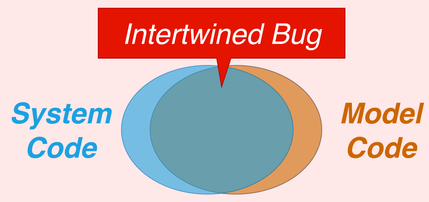
But Current Frameworks are Hard to Use

Not PyTorch

Only 8% non-PyTorch

HuggingFace Models

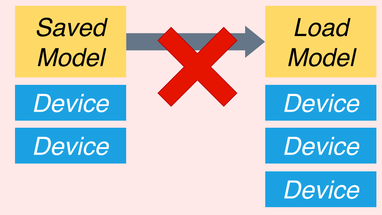
Intertwined System and Model Design



Not Automated Enough

Hard to Debug

No Distributed Checkpoint



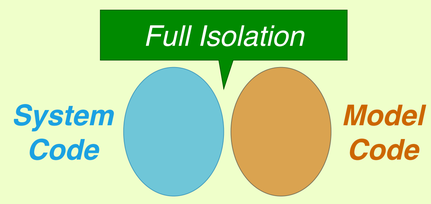
veScale

PyTorch Native

92% PyTorch

HuggingFace Models

Decoupled System and Model Design

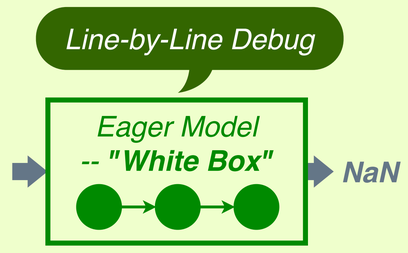


Automatic Parallelism

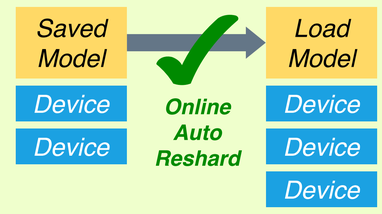
Tensor, Sequence, Data, ZeRO, Pipeline Parallelism

Minimal Manual Effort

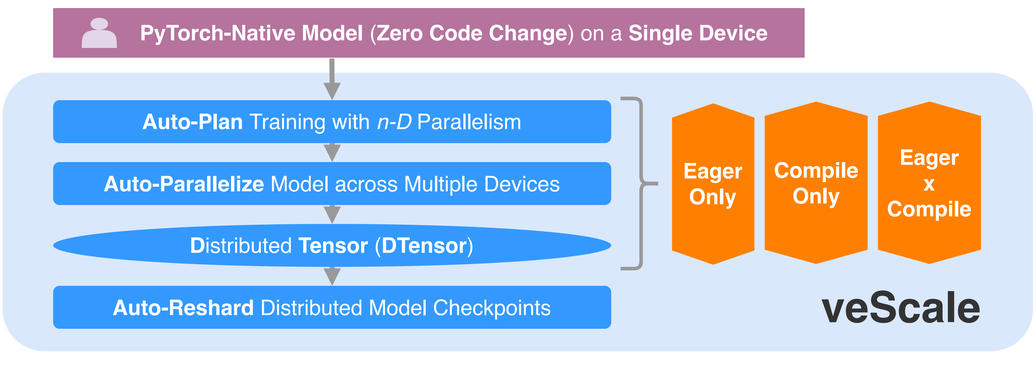
Easy to Debug



Auto Distributed Checkpoint



Overview



Preliminary Results

Simple API

4D Parallel Training

```python
# zero model code change
from <HuggingFace> import <ModelCls>, <ModelArgs>

# create fake model without actual memory usage (optional)
fake_model = deferred_init(<ModelCls>, <ModelArgs>)

# initialize 4D device mesh
mesh = init_device_mesh("cuda", (dp_zero_size, tp_sp_size), mesh_dim_names=["DP_ZERO", "TP_SP"])

# parallelize model in tp & sp
from <PredefinedPlan> import sharding_plan
real_tp_sp_model = parallelize_module(fake_model, mesh["TP_SP"], sharding_plan)

# parallelize model in dp
ddp_model = DDP(real_tp_sp_model, mesh["DP_ZERO"])

# parallelize model with zero2
doptimizer = DistributedOptimizer(torch.optim.AdamW, models=[ddp_model])

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

Save Checkpoint

```python
# 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)
```

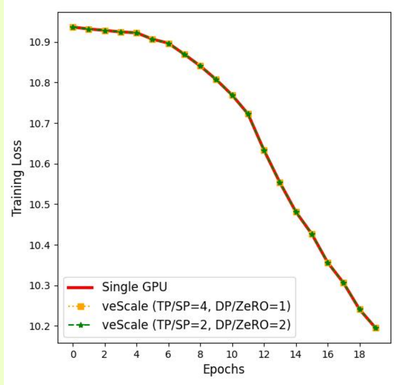
Load Checkpoint (under different world size or 4D degrees)

```python
# 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)
```

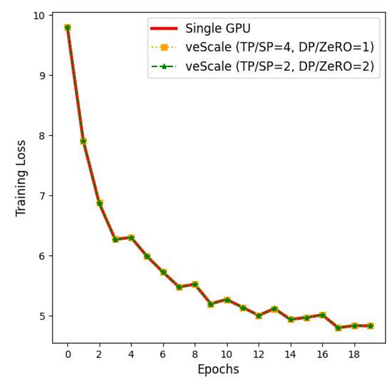
Bitwise Correctness

More Details

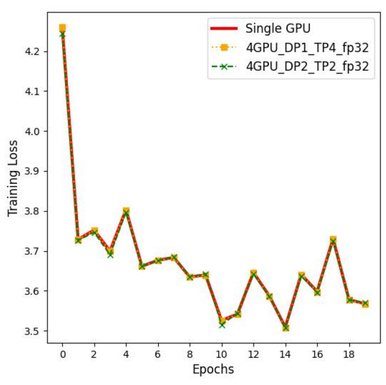
Mixtral 4D Training



LLaMa2 4D Fine-tuning



NanoGPT 4D Fine-Tuning



NanoGPT without veScale