Solution Brief

Cross-Industry Architecture
Intel® Optimized Cloud Modules

intel software

Intel® Optimized Cloud Modules for Azure*: Stable Diffusion Distributed Training

Author: Kelli Belcher

December 15, 2023

Fine-tune stable diffusion models in a distributed architecture on Azure with Intel® Xeon® Scalable Processors

The **Intel® Optimized Cloud Modules for Microsoft Azure*: Stable Diffusion Distributed Training** is designed to illustrate the process of fine-tuning a textual inversion stable diffusion model with 3rd or 4th Generation **Intel® Xeon® Scalable Processors** on **Microsoft Azure***. Specifically, we show the process of fine-tuning a **Stable Diffusion v1** model with the **Dicoo image dataset** from the Hugging Face* Hub in a distributed architecture. The objective of this module is to understand how to set up a distributed system so that you can fine-tune the model to your specific workload. The result of this module will be a base stable diffusion model that can generate images that will be suitable for your use case when you modify it to your specific objective and dataset.

Use it as a reference solution for:

- Setting up an Azure cluster for distributed training.
- Fine-tuning a stable diffusion model on a single machine.
- Fine-tuning a stable diffusion model in a distributed system, taking advantage of Intel optimizations.

Who needs it?

- Developers aiming to fine-tune their stable diffusion models on multiple Intel Xeon CPUs, leveraging Intel's accelerated deep learning software libraries, including **Intel® Extension for PyTorch*** and **Intel® oneAPI Collective Communications Library (oneCCL)**.
- Developers interested in learning the process of setting up Azure clusters for distributed training.

What it does

This module demonstrates how to transform a standard single-node PyTorch training scenario into a high-performance distributed training scenario across multiple CPUs. To fully capitalize on Intel hardware and further optimize the fine-tuning process, this module integrates the **Intel® Extension for PyTorch*** and **Intel® oneAPI Collective Communications Library (oneCCL)**. The module serves as a guide to setting up an Azure cluster for distributed training while showcasing a complete project for fine-tuning stable diffusion models.

- It provides step-by-step instructions for configuring an Azure cluster, simplifying the process of establishing a distributed training environment.
- It serves as a guide through the entire lifecycle of fine-tuning stable diffusion models, starting from data preprocessing to model fine-tuning.
- The module capitalizes on Intel® Extension for PyTorch, harnessing the power of **Intel® Advanced Vector Extensions 512 (Intel® AVX-512)** and **Intel® Advanced Matrix Extension (Intel® AMX)** instruction sets. This enables significant acceleration of the fine-tuning process, boosting overall training performance. The use of Intel's optimized communications library, oneCCL, ensures that distributed workflows are streamlined, enhancing efficiency in a multi-node environment.

In summary, this module empowers you to harness the full potential of Intel hardware for distributed fine-tuning a textual inversion stable diffusion model.

Cloud Solution Architecture

To form the cluster, the cloud solution implements **Azure Trusted Virtual Machines**, leveraging instances from the **Dv5 series**. To

enable seamless communication between the instances, each of the machines are connected to the same virtual network and a permissive network security group is established that allows all traffic from other nodes within the cluster. The raw dataset is taken from Hugging Face Hub, and once the model has been trained, the weights are saved to the virtual machines.

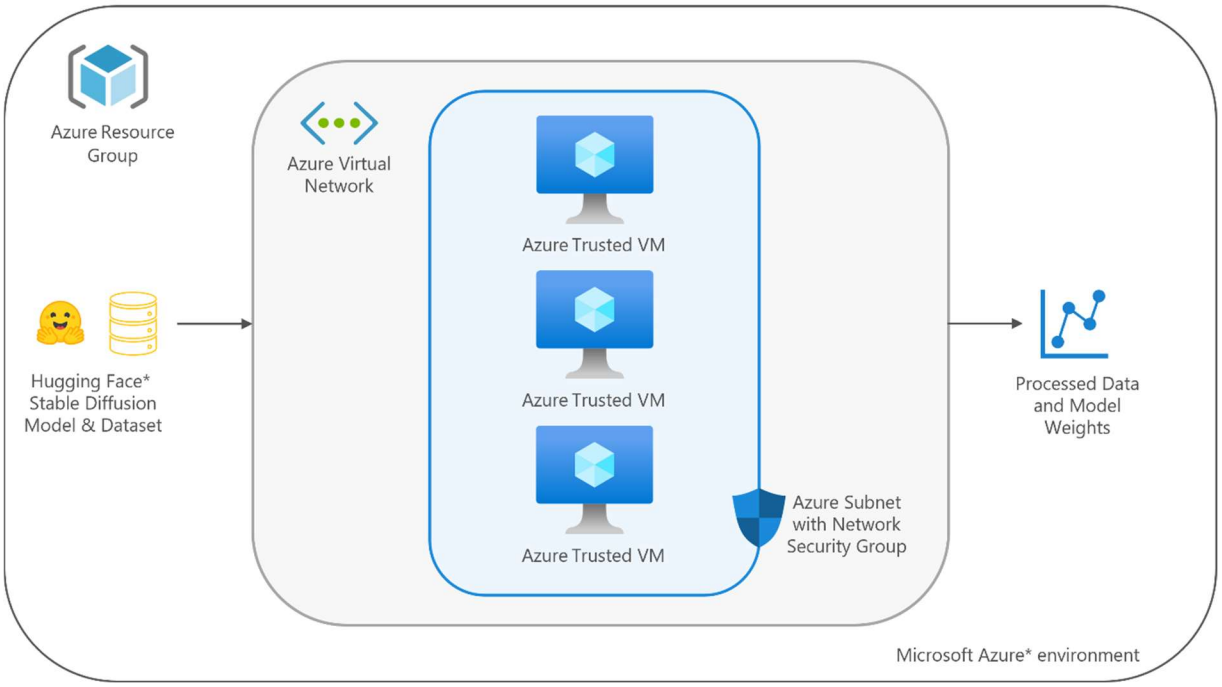

Figure 1: Architectural Diagram of Azure Stable Diffusion Distributed Training module. Image by author.

Solution Component Overview

This solution is derived from the Stable Diffusion v1 implementation. The code has been enhanced through refactoring to achieve better modularity and suitability for distributed fine-tuning on 3rd or 4th Generation Xeon CPUs. For data, we use the Dicoo image dataset obtained from Hugging Face. To fully leverage Intel hardware capabilities and enable distributed training, we incorporated the Intel Extension for PyTorch and oneCCL.

Code Highlights

Enable the Intel Extension for PyTorch

The Intel Extension for PyTorch elevates PyTorch performance on Intel hardware with the integration of the newest features and optimizations that have not yet been incorporated into open source PyTorch. This extension efficiently utilizes Intel hardware capabilities, such as Intel AVX-512 and Intel AMX on Intel Xeon CPUs. Unleashing this power is straightforward – just wrap your model and optimizer objects with `ipex.optimize`.

```
# Set weight_dtype to bfloat16
weight_dtype = torch.bfloat16

# Move vae and unet to device and cast to weight_dtype
unet.to(accelerator.device, dtype=weight_dtype)
vae.to(accelerator.device, dtype=weight_dtype)

# Optimize unet and vae models
import intel_extension_for_pytorch as ipex
unet = ipex.optimize(unet, dtype=weight_dtype)
vae = ipex.optimize(vae, dtype=weight_dtype)
```

Gradient Accumulation with Hugging Face Accelerate

The Accelerate library by Hugging Face streamlines the gradient accumulation process. This package helps to abstract away the complexity of supporting multi-CPUs/GPUs and provides an intuitive, user-friendly API, making gradient accumulation and clipping hassle-free during the training process.

```
# Initializing Accelerator object
self.accelerator = Accelerator(
        gradient_accumulation_steps=gradient_accumulation_steps,
        cpu=True,
)

# Gradient Accumulation
with self.accelerator.accumulate(self.model):
       with self.autocast_ctx_manager:
             _, loss = self.model(X, Y)
       self.accelerator.backward(loss)
       loss = loss.detach() / gradient_accumulation_steps

# Gradient Clipping
self.accelerator.clip_grad_norm_(
       self.model.parameters(), self.trainer_config.grad_clip
)
```

Distributed Training

For distributed training, we utilized oneCCL. With optimized communication patterns, oneCCL enables developers and researchers to train newer and deeper models more quickly across multiple nodes. It offers a tool called `mpirun`, which allows you to seamlessly launch distributed training workloads.

```
# Generate multi-CPU configuration file
accelerate config

# Launch distributed fine-tuning job
mpirun -f ~/hosts -n 3 -ppn 1 accelerate launch textual_inversion.py --
pretrained_model_name_or_path=$MODEL_NAME --train_data_dir=$DATA_DIR --learnable_property="object"   --
placeholder_token="<dicoo>" --initializer_token="toy" --resolution=512  --train_batch_size=1  --seed=7  --
gradient_accumulation_steps=1 --max_train_steps=30 --learning_rate=2.0e-03 --scale_lr --
lr_scheduler="constant" --lr_warmup_steps=0 --output_dir=./textual_inversion_output --mixed_precision bf16
--save_as_full_pipeline
```

Next Steps

Download the module from GitHub ›

Check out the full suite of Intel® Cloud Optimization Modules ›

Register for office hours for implementation support from Intel engineers ›

Come chat with us on our DevHub Discord server to keep interacting with other developers ›

1121/SS/CMD/PDF