FMEngine

Xiaozhe Yao

Jan 21, 2024

GETTING STARTED

FMEngine is a utility library for training very large foundation models. The goal of fmengine is to provide the following:

- **Ergonomic** interface for training foundation models. It is sufficient easy for a beginner to use, but also provides enough flexibility for advanced users to customize their training.
- **Efficient** optimizations built in. FMEngine is equipped with Flash Attention and various fused ops to accelerate training.
- **HPC-friendly** installation with pre-built docker and singularity/apptainer containers. FMEngine is mainly designed and tested on Slurm clusters. We provide starter scripts for running FMEngine on Slurm clusters.
- **Compatible** with existing frameworks and tools, particularly with HuggingFace. Since FMEngine is built with DeepSpeed, it is also compatible with all DeepSpeed features.

CHAPTER

ONE

ACKNOWLEDGEMENT

FMEngine is primarily implemented and maintained by the Efficient Architecture and Systems Labs @ ETH Zurich.

We thank our friends for their generous support:

1.1 Contributors

- @xzyaoi
- @LorrinWWW
- @fishiu: Scaling up Low Rank Adapters + Longer Contexts
- @Taishi-N324: Benchmarking and Scaling Experiments
- @chiennv2000: Mistral Integration

1.1.1 Quick Start

Installation

We provide a docker image and singularity image for the dependencies of FMEngine. You can download the docker image by running:

```
docker pull xzyaoi/fmsys:0.0.7
```

or download the singularity image here.

Please note that the images are built **without** fmengine installed. You are advised to clone the fmengine and bind the directory to the container.

Training preparation

- *Prepare checkpoints*. As the first step, you will need to split a large model checkpoint into smaller pieces for each layer. This can be done by running the following command:

```
python scripts/conversions/llama/from_hf.py \
--model_name_or_path meta-llama/Llama-2-7b-hf  \
--output_dir path_to_outdir/llama2-7b \
--mp_world_size 1
```

You can download pre-configured checkpoints here: Google Drive.

- *Prepare datasets*. We now only supports `.jsonl` format, which is a list of json objects, each of which contains a `text` field. For example, a sample of the dataset can be:

```
{"text": "I love this movie!"}
{"text": "I hate this movie!"}
{"text": "I don't know."}
```

Training

In `/scripts`, we show some examples of training scripts, for example, to finetune a pythia-2.8b model, you can run the following command:

```
deepspeed --num_gpus 4 --num_nodes 1 starter.py \
    --output_dir .cache/models \
    --init_ckpt /pretrained_weights/pythia-160m-deduped \
    --data_path /datasets/quantitative_natural_instructions/train/all.train.jsonl \
    --max_seq_len 1024 \
    --train_steps 1000 \
    --eval_steps 10 \
    --save_steps 100 \
    --log_steps 1 \
    --pipe_parallel_size 1 \
    --model_parallel_size 1 \
    --use_flash_attn true \
    --deepspeed_config ./configs/pythia.json
```

You are also advised to read `./configs/pythia.json` for the deepspeed configuration, which convers the learning rate, batch size, etc.

1.1.2 Parameter-Efficient Fine-Tuning

1.1.3 Distributed Training

Multi-GPU training

Running multi-GPU training is as simple as running the following command:

```
deepspeed --num_gpus 2 --num_nodes 1 cli/train.py \
    --output_dir /workspace/.cache/models \
    --init_ckpt /pretrained/tinyllama-2-1b \
```

(continues on next page)

(continued from previous page)

```
    --data_path /datasets/dataset.train.jsonl \
    --max_seq_len 128 \
    --train_steps 1000 \
    --eval_steps 10 \
    --save_steps 1000 \
    --log_steps 10 \
    --pipe_parallel_size 2 \
    --model_parallel_size 1 \
    --use_flash_attn true \
    --use_fused_ops false \
    --deepspeed_config ./configs/llama.json
```

Multi-host training

FMEngine support multi-host training with deepspeed. To run multi-host training, you need to install `pdsh` first, by running the following command:

```
git clone https://github.com/chaos/pdsh.git
cd pdsh
./configure --enable-static-modules --without-rsh --with-ssh --without-ssh-connect-
↪timeout-option --prefix=/your/preferred/path
make
make install
```

Slurm and HPC

It is strongly recommended to use singlarity/apptainer to run FMEngine on HPC clusters. However, the `deepspeed` launcher cannot trigger slurm command within singularity container (It's probably doable, but quite complicated). Here we suggest to start the training script with `torchrun` instead.

Before running training scripts, you need to login to your wandb account first, by running the following command:

```
$ singularity shell fmsys.sif
> wandb login
> # Enter your API key
```

The wandb configuration will be stored in your home directory (it is supposed to be the same directory on your HPC and within the singularity container), therefore you will need to mount your home directory to the singularity container.

Here's an example script of running FMEngine on slurm clusters.

```
#!/bin/bash -x
#SBATCH --account=<YOUR_ACCOUNT>
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --gres=gpu:4
#SBATCH --partition=<YOUR_PARTITION>
#SBATCH --output=/<YOUR_LOG_PATH>/%j_%N_log.out

# Network Configuration, cluster-specific
export NCCL_IB_TIMEOUT=50
```

(continues on next page)

(continued from previous page)

```
export UCX_RC_TIMEOUT=4s
export NCCL_IB_RETRY_CNT=10
export NCCL_ASYNC_ERROR_HANDLING=1

echo $SLURM_JOB_GPUS
echo $SLURM_NTASKS
echo $SLURM_NODELIST

# Convert SLURM_JOB_GPUS to an array
IFS=',' read -ra GPU_ARRAY <<< "$SLURM_JOB_GPUS"

# Get the number of GPUs from the length of the array
NUM_GPUS=${#GPU_ARRAY[@]}

export TOTAL_GPUS=$(($NUM_GPUS * $SLURM_NTASKS))
echo $TOTAL_GPUS

# master_addr=$(scontrol show hostnames "$SLURM_JOB_NODELIST" | head -n 1)
master_addr="$(scontrol show hostnames "$SLURM_JOB_NODELIST" | head -n 1)i"

export MASTER_ADDR=$master_addr

export HOSTNAMES=`scontrol show hostnames "$SLURM_JOB_NODELIST"`
export MASTER_ADDR="$(scontrol show hostnames "$SLURM_JOB_NODELIST" | head -n 1)"
export MASTER_PORT=12802
export COUNT_NODE=`scontrol show hostnames "$SLURM_JOB_NODELIST" | wc -l`

# Print System Information
echo "GPUs available to job: $SLURM_JOB_GPUS"
echo "Total tasks: $SLURM_NTASKS"

# Loop over all nodes
for ((i=0; i<$COUNT_NODE; i++))
do
    srun --nodes=1 --ntasks=1 -w "$(scontrol show hostnames "$SLURM_JOB_NODELIST" | sed -
↪n "$((i+1))p")" \
    singularity run --nv \
    --home <YOUR_HOME_DIRECTORY>:/home/<YOUR_USERNAME> \
    --bind <YOUR_CACHE_DIRECTORY>/transformers_cache:/.hf_cache \
    --env HF_HOME=/.hf_cache \
    --env PYTHONPATH=/workspace \
    --bind <YOUR_CACHE_DIRECTORY>/pretrained_weights:/pretrained \
    --bind <YOUR_CACHE_DIRECTORY>/datasets:/datasets \
    --bind $PWD:/workspace \
    --pwd /workspace \
    fmsys_0.0.4.sif \
    torchrun \
    --master_addr "$MASTER_ADDR" \
    --master_port 12802 \
    --node_rank $i \
    --nnodes $SLURM_NTASKS \
    --nproc-per-node=$NUM_GPUS \
```

(continues on next page)

(continued from previous page)

```
        cli/train.py \
        --output_dir /workspace/.cache/models \
        --init_ckpt /pretrained/llama-2-7b-hf \
        --data_path /datasets/prompt.jsonl \
        --max_seq_len 2048 \
        --train_steps 10 \
        --eval_steps 10 \
        --save_steps 1000 \
        --log_steps 1 \
        --pipe_parallel_size 8 \
        --model_parallel_size 1 \
        --use_flash_attn true \
        --use_fused_ops false \
        --deepspeed_config ./configs/llama.json &
done

wait
```

1.1.4 References

Trainers

Dataset

Model Parallelism

1.1.5 Benchmark and Scaling

There are many factors that affect the training performance. We provide some benchmark results here.

- Training llama-3b on 4x RTX 3090: ~6300 tokens per second. Training Report Configuration and Monitoring.

Scaling to Clusters of GPUs

We conduct scaling experiments on up to 256 NVIDIA A100 40G GPUs. We thank Juelich Supercomputing Center and Ontocord for their generous support in providing the computing resources.

Benchmark 1: Fine-tuning Llama-7B

Configuration

Train batch size (GBS): 512 for # GPUs <= 64; Otherwise, GBS = # GPUs * 8. For example, when # GPUs=128, GBS=1024.

Micro batch size (MBS): 8

Sequence Length: 2048

Total Tokens/second

Tokens/GPU/second

Benchmark 2: Fine-tuning Llama-70B

Train Batch Size	Micro Batch Size	Sequence Length
1024	4	4096

Total Tokens/second

Tokens/GPU/second

1.1.6 Frequently Asked Questions

- Hangs during 'deepspeed.initialize()'

  Try clearing the cache with `rm -rf ~/.cache/torch_extensions/py310_cu*`. See this issue for more details.

1.1.7 Add New Model

Caution: The process of adding new model is not stable yet, please be aware of possible changes.

Adding new model is more complicated than just training with existing implemented models. This tutorial is aimed for advanced users who want to add new models to the library. The process is made to be as compatible as possible with existing libraries