

torchtitan

`torchtitan` is currently in a pre-release state and under extensive development.

`torchtitan` is a proof-of-concept for Large-scale LLM training using native PyTorch. It is (and will continue to be) a repo to showcase PyTorch's latest distributed training features in a clean, minimal codebase. `torchtitan` is complementary to and not a replacement for any of the great large-scale LLM training codebases such as Megatron, Megablocks, LLM Foundry, Deepspeed, etc. Instead, we hope that the features showcased in `torchtitan` will be adopted by these codebases quickly. `torchtitan` is unlikely to ever grow a large community around it.

Our guiding principles when building `torchtitan`:

- Designed to be easy to understand, use and extend for different training purposes.
- Minimal changes to the model code when applying 1D, 2D, or (soon) 3D Parallel.
- Modular components instead of a monolithic codebase.
- Get started in minutes, not hours!

Intro video - learn more about `torchtitan` in under 4 mins:

The screenshot shows a VS Code editor interface with the `torchtitan` project open. The Explorer sidebar on the left shows the project structure, including `train_configs` and `debug_model.toml`. The main editor area displays the `debug_model.toml` file, which contains configuration for training a Llama 7B model. The terminal window at the bottom shows the output of the training process, including warnings, information about the device mesh, dataset preparation, and training progress metrics.

```
train_configs > debug_model.toml
30 batch_size = 8
31 seq_len = 2048
32 warmup_steps = 2 # lr scheduler warm up, normally 20% of the train steps
33 max_norm = 1.0 # grad norm clipping

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
[rank0]:2024-04-15 19:32:44,960 - root - WARNING - ENV [TORCH_NCCL_ASYNC_ERROR_HANDLING] = 1 will be overridden to 3 based on j
ob config
[rank0]:2024-04-15 19:32:46,983 - root - INFO - Building 1-D device mesh with ['dp'], [0]
[rank0]:2024-04-15 19:32:46,986 - root - INFO - Building sentencepiece tokenizer locally from ./torchtitan/datasets/tokenizer/
tokenizer.model
[rank0]:2024-04-15 19:32:46,992 - root - INFO - SentencePieceTokenizer built: #words 32000, BOS ID 1, EOS ID 2
[rank0]:2024-04-15 19:32:46,992 - root - INFO - Preparing alpaca dataset from HuggingFace
[rank0]:2024-04-15 19:32:49,128 - root - INFO - Building llama debugmodel with ModelArgs(dim=256, n_layers=2, n_heads=16, n_kv
_heads=None, vocab_size=32000, multiple_of=256, ffn_dim_multiplier=None, norm_eps=1e-05, max_batch_size=32, max_seq_len=32768,
depth_init=True, norm_type='fused_rmsnorm')
[rank0]:2024-04-15 19:32:49,140 - root - INFO - Model llama debugmodel size: 10,009,216 total parameters
[rank0]:2024-04-15 19:32:49,140 - root - INFO - GPU capacity: NVIDIA H100 (0) with 95.04GiB memory
[rank0]:2024-04-15 19:32:49,149 - root - INFO - Applied selective activation checkpointing to the model
[rank0]:2024-04-15 19:32:49,150 - root - INFO - Applied FSDP to the model
[rank0]:NCCL version 2.20.5+cuda12.4
[rank0]:2024-04-15 19:32:58,401 - root - INFO - GPU memory usage for model: 0.02GiB(0.02%)
[rank0]:2024-04-15 19:32:58,402 - root - INFO - Metrics logging active. Tensorboard logs will be saved at ./outputs/tb/2024041
5-1932
[rank0]:2024-04-15 19:32:58,411 - root - INFO - Training starts at step 1
[rank0]:2024-04-15 19:32:58,411 - root - INFO - Profiling active. Traces will be saved at ./outputs/profiling/traces
[rank0]:2024-04-15 19:33:00,667 - root - INFO - step: 1 loss: 10.8315 memory: 9.31GiB(9.80%) wps: 7,264 mfu: 0.09%
[rank0]:2024-04-15 19:33:00,668 - root - INFO - Synchronizing and adjusting timeout for all ProcessGroups to 0:01:40
[rank0]:2024-04-15 19:33:00,727 - root - INFO - step: 2 loss: 10.7670 memory: 9.32GiB(9.80%) wps: 277,348 mfu: 3.40%
[rank0]:2024-04-15 19:33:00,782 - root - INFO - step: 3 loss: 10.6376 memory: 9.32GiB(9.80%) wps: 302,887 mfu: 3.71%
[rank0]:2024-04-15 19:33:00,837 - root - INFO - step: 4 loss: 10.4035 memory: 9.32GiB(9.80%) wps: 299,199 mfu: 3.66%
[rank0]:2024-04-15 19:33:00,893 - root - INFO - step: 5 loss: 10.1725 memory: 9.32GiB(9.80%) wps: 293,654 mfu: 3.60%
[rank0]:2024-04-15 19:33:00,954 - root - INFO - step: 6 loss: 9.9228 memory: 9.32GiB(9.80%) wps: 274,198 mfu: 3.36%
[rank0]:2024-04-15 19:33:01,010 - root - INFO - step: 7 loss: 9.6817 memory: 9.32GiB(9.80%) wps: 291,934 mfu: 3.58%
[rank0]:2024-04-15 19:33:01,064 - root - INFO - step: 8 loss: 9.4662 memory: 9.32GiB(9.80%) wps: 305,761 mfu: 3.74%
[rank0]:2024-04-15 19:33:01,343 - root - INFO - step: 9 loss: 9.2911 memory: 9.32GiB(9.80%) wps: 58,945 mfu: 0.72%
[rank0]:[rank0]:[W415 19:33:01.565330886 CPUAllocator.cpp:249] Memory block of unknown size was allocated before the profiling
started, profiler results will not include the deallocation event
[rank0]:2024-04-15 19:33:01,400 - root - INFO - step: 10 loss: 9.1348 memory: 9.32GiB(9.80%) wps: 289,965 mfu: 3.55%
[rank0]:2024-04-15 19:33:01,567 - root - INFO - Sleeping for 2 seconds for others ranks to complete
[rank0]:2024-04-15 19:33:03,569 - root - INFO - Training completed
```

Pre-Release Updates:

(4/25/2024): torchtitan is now public but in a pre-release state and under development. Currently we showcase pre-training **Llama 3 and Llama 2** LLMs of various sizes from scratch. `torchtitan` is tested and verified with the PyTorch nightly version `torch-2.4.0.dev20240412`. (We recommend latest PyTorch nightly).

Key features available

1. FSDP2 with per param sharding
2. Tensor Parallel
3. Selective layer and operator activation checkpointing
4. Distributed checkpointing
5. 2 datasets pre-configured (45K - 144M)
6. GPU usage, MFU, tokens per second and more displayed via TensorBoard
7. Learning rate scheduler, meta init, Optional Fused RMSNorm
8. All options easily configured via toml files
9. Interoperable checkpoints which can be loaded directly into `torch tune` for fine tuning

We report our Performance verified on 64 A100 GPUs

Coming soon

1. Async checkpointing
2. FP8 support
3. Context Parallel
4. 3D Pipeline Parallel
5. `torch.compile` support
6. Scalable data loading solution

Installation

```
1 git clone https://github.com/pytorch/torchtitan
2 cd torchtitan
3 pip install -r requirements.txt
4 pip3 install --pre torch --index-url https://download.pytorch.org/whl/
  nightly/cu121 # or cu118
5 pip install .
```

Downloading a tokenizer

`torch titan` currently supports training Llama 3 (8B, 70B), and Llama 2 (7B, 13B, 70B) out of the box. To get started training these models, we need to download a `tokenizer.model`. Follow the instructions on the official meta-llama repository to ensure you have access to the Llama model weights.

Once you have confirmed access, you can run the following command to download the Llama 3 / Llama 2 tokenizer to your local machine.

```
1 # Get your HF token from https://huggingface.co/settings/tokens
2
3 # llama3 tokenizer.model
4 python torch titan/datasets/download_tokenizer.py --repo_id meta-llama/
   Meta-Llama-3-8B --tokenizer_path "original" --hf_token=...
5
6 # llama2 tokenizer.model
7 python torch titan/datasets/download_tokenizer.py --repo_id meta-llama/
   Llama-2-13b-hf --hf_token=...
```

Start a training run

Llama 3 8B model locally on 8 GPUs

```
1 CONFIG_FILE="./train_configs/llama3_8b.toml" ./run_llama_train.sh
```

TensorBoard

To visualize TensorBoard metrics of models trained on a remote server via a local web browser:

1. Make sure `metrics.enable_tensorboard` option is set to true in model training (either from a `.toml` file or from CLI).
2. Set up SSH tunneling, by running the following from local CLI

```
1 ssh -L 6006:127.0.0.1:6006 [username]@[hostname]
```

3. Inside the SSH tunnel that logged into the remote server, go to the `torch titan` repo, and start the TensorBoard backend

```
1 tensorboard --logdir=./outputs/tb
```

4. In the local web browser, go to the URL it provides OR to `http://localhost:6006/`.

Multi-Node Training

For training on ParallelCluster/Slurm type configurations, you can use the `multinode_trainer.slurm` file to submit your sbatch job.

To get started adjust the number of nodes and GPUs

```
1 #SBATCH --ntasks=2
2 #SBATCH --nodes=2
```

Then start a run where `nnodes` is your total node count, matching the sbatch node count above.

```
1 srun torchrun --nnodes 2
```

If your gpu count per node is not 8, adjust:

`--nproc_per_node`

in the torchrun command and

`#SBATCH --gpus-per-task`

in the SBATCH command section.

License

This code is made available under BSD 3 license. However you may have other legal obligations that govern your use of other content, such as the terms of service for third-party models, data, etc.