OpenSeek Continue Pretraining – Technical Report

石竞泽 | 今天修改

This document describes the three-stage pretraining workflow in this repository, focused on Ubuntu-based HPC clusters:

1. Asset download and dataset building: `scripts/download.py` downloads the tokenizer and base model checkpoints, mixes multiple datasets by ratios, and performs sequence packing.
2. Distributed training: `recipes/accelerate_configs/ddp.yaml` (Accelerate DDP), `recipes/openseek/config.yaml` (training config), and `trainer/pt_dpsk.py` (training entry) are driven by `train.sh` on a cluster. Checkpoints are saved every 1,000 steps.
3. Model soup: `scripts/merge.py` merges the seven checkpoints with the base model using the average strategy.

Public Checkpoint

The continue pretraining checkpoint is released at: `JingzeShi/OpenSeek-1.4B-A0.4B`.

Typical load snippet:

代码块

```
from transformers import AutoModelForCausalLM, AutoTokenizer
model_id = "JingzeShi/OpenSeek-1.4B-A0.4B"
tokenizer = AutoTokenizer.from_pretrained(model_id, trust_remote_code=True)
model = AutoModelForCausalLM.from_pretrained(model_id, trust_remote_code=True)
```

Key Technical Points

- Dataset mixing ratios must sum to 1.0 (validated) to meet the target total sample size after packing/truncation.
- Tokenizer guardrails: right padding; if no `pad_token`, fall back to `eos_token` to avoid shape issues.
- Packing vs truncation:
  - Packing uses TRL `pack_dataset` to concatenate tokenized sequences up to `max_length` (higher throughput, fewer padding tokens).
  - Truncation uses TRL `truncate_dataset` when packing is disabled.
- Reproducibility: set random seeds for dataset shuffling and training; minor variance is possible due to multi-process packing.
- DDP with bf16: use Accelerate `MULTI_GPU` + `mixed_precision: bf16` on GPUs that support BF16 (Ampere+). Adjust `num_processes` to the actual GPU count per node.
- Checkpointing: save every 1,000 steps (`save_steps: 1000`), with `save_total_limit` to bound disk usage.
- Model soup: `average` parameter-wise mean (only matching parameter names are merged).
- IO and performance: persist processed datasets to disk (Arrow) to reduce training-time overhead.

Directory Map and Key Files

- `scripts/download.py`
  - Downloads and saves base model/tokenizer to `./models/OpenSeek_Small_v1`.
  - Mixes many dataset shards by ratios, tokenizes, and packs to `./datasets/OpenSeek-Pretrain-30B`.
- `processor/pt_datasets_process.py`

    - Dataset ratio validation, optional formatting, tokenization, packing/truncation (`trl.pack_dataset` / `trl.truncate_dataset`), and parallel mapping.
- `recipes/accelerate_configs/ddp.yaml`
    - Accelerate DDP configuration (single-node multi-GPU in the template; adapt to your cluster).
- `recipes/openseek/config.yaml`
    - Trainer configuration (logging, save frequency, output dir, dtype, attn implementation, etc.).
- `trainer/pt_dpsk.py`
    - Loads tokenizer/model, builds datasets via `mix_pt_datasets`, initializes `Trainer`, runs training/evaluation.
- `train.sh`
    - Example Slurm batch script that sets CUDA/cuDNN modules/conda, mirrors, and launches training via Accelerate.
- `scripts/merge.py`
    - Builds a model soup from base model and checkpoints using the `average` strategy.

Environment and Dependencies

- OS: Ubuntu 22.04
- GPU: NVIDIA A800
- CUDA/cuDNN: CUDA 12.2, cuDNN 9.8
- Python: 3.10+
- Core packages:
    - torch (CUDA build), transformers, datasets, accelerate, trl
    - sentencepiece, safetensors
- Optional environment variables:
    - `HF_ENDPOINT=https://hf-mirror.com` (if you use a mirror)
    - `XDG_CACHE_HOME=cache` (to consolidate cache location)

Example (adapt to your module system/conda env):

代码块

```
# create and activate a conda env (if not pre-provisioned by your cluster)
conda create -y -n train python=3.10 && conda activate train

# install essentials (pin versions per your infra policies)
pip install --upgrade pip
pip install torch --index-url https://download.pytorch.org/whl/cu124  # example; match your CUDA
pip install transformers datasets accelerate trl sentencepiece safetensors
```

Step 1: Download Base Model/Tokenizer and Build Mixed Packed Dataset

Script: `scripts/download.py`

- Loads `BAAI/OpenSeek-Small-v1`, saves model/tokenizer to `./models/OpenSeek_Small_v1`.
- Defines `datasets_and_ratios` (a list of `{dataset_name: ratio}`) summing to 1.0.
- Sets `total_sample_size=7_500_000`, `max_length=4096`, `packing=True`, `dataset_num_proc=4`, `seed=233`.
- Calls `mix_pt_datasets` (aliased from `pt_datasets_process.py`) to:
  - Load each dataset shard (Hub path or local `load_from_disk` path).
  - Optionally apply a `formatting_func`.
  - Tokenize with right padding (fallback `pad_token=eos_token` if missing).
  - Pack or truncate sequences based on `max_length`.
  - Sample and concatenate by ratio to hit the target size.
- Saves the processed dataset to `./datasets/OpenSeek-Pretrain-30B` (Arrow format).

Run (on Ubuntu cluster login/compute node):

代码块

```
python scripts/download.py
```

Notes:

- With `packing=True`, the "sample size" refers to packed sequences, not raw documents.
- Adjust `max_length` and downstream batch size/grad-accumulation for available VRAM.
- Ensure sufficient disk space for cached and processed datasets.

Step 2: Distributed Training (DDP + Accelerate + Trainer)

Core configs/scripts:

- `recipes/accelerate_configs/ddp.yaml`
  - `distributed_type: MULTI_GPU`, `mixed_precision: bf16`, `gpu_ids: all`.
  - Set `num_processes` to the actual GPU count per node (e.g., 4 or 8).
- `recipes/openseek/config.yaml`
  - Logging: `logging_steps: 1`, `report_to: [tensorboard]`.
  - Saving: `save_strategy: steps`, `save_steps: 1000`, `save_total_limit: 10`.
  - Output: `output_dir: data/OpenSeek-1.4B-A0.4B`, `overwrite_output_dir: true`.
  - Model: `model_name_or_path: ./models/OpenSeek_Small_v1`, `torch_dtype: bfloat16`, `trust_remote_code: true`.
  - Attention: `attn_implementation: sdpa` (ensure compatibility with your CUDA/cuDNN stack).
- `trainer/pt_dpsk.py`
  - Loads tokenizer, builds dataset via `mix_pt_datasets`, sets up `Trainer`, trains and evaluates.
- `train.sh`
  - Example Slurm script that configures CUDA modules/conda and launches Accelerate.

Submit training job (Slurm):

```
代码块
1   sbatch train.sh
```

Outputs:

- Checkpoints at `data/OpenSeek-1.4B-A0.4B/checkpoint-1000`, `...-2000`, ..., `...-7000`.
- TensorBoard logs (if enabled) for monitoring.

Recommendations:

- Make `num_processes` in `ddp.yaml` match `--gpus` allocated by Slurm.
- Ensure `output_dir` has sufficient free space; prune old checkpoints with `save_total_limit`.
- Tune effective batch size (global batch = per-device batch × num GPUs × grad-accum) to stabilize training.

Step 3: Model Soup (Checkpoint Merging)

Script: `scripts/merge.py`

- Merges base model `./models/OpenSeek_Small_v1` with checkpoints `data/OpenSeek-1.4B-A0.4B/checkpoint-1000..7000`.
- Strategy:
  - `average`: unweighted parameter-wise mean across all checkpoints.
- Only matching parameter keys are merged; others are skipped.

- Saves merged model, tokenizer, config, and `merge_info.json` to output directory (default `./models/OpenSeek_Small_v1-merged-average`).

Run:

```
代码块
1   python scripts/merge.py
```

Notes:

- BF16 saves reduce disk footprint; convert to FP32 if you need full precision checkpoints for downstream tasks.
- Inspect merge logs for matched key counts to verify architecture compatibility.

End-to-End Repro (Commands)

1. Build datasets and cache base assets:

```
代码块
1   python scripts/download.py
```

2. Launch distributed training (Slurm):

```
代码块
1   sbatch train.sh
```

3. Merge checkpoints into a model soup:

```
代码块
```

```
python scripts/merge.py
```

Troubleshooting

- Ratios do not sum to 1.0:
  - Fix `datasets_and_ratios`; `download.py` prints total ratio for verification.
- OOM (GPU memory):
  - Reduce `max_length` / `per_device_train_batch_size`, increase `gradient_accumulation_steps`, enable gradient checkpointing.
- DDP process count mismatch:
  - Align `ddp.yaml:num_processes` with the actual GPU count allocated by Slurm.
- BF16 not supported:
  - Switch to FP16 or FP32 and adjust configs accordingly (expect perf/throughput changes).
- Slow data loading:
  - Persist datasets to disk (Arrow), increase `dataset_num_proc`, and ensure fast storage.

Reproducibility

- Seeds are set for both dataset preparation and training. Minor nondeterminism may remain due to multi-processing and packing.
- Pin package versions and keep hardware/software constant to improve reproducibility.
