OpenSeek KTO Alignment – Technical Report

石竞泽 | 今天修改

This document provides a detailed, end-to-end technical description of the KTO (Kahneman–Tversky Optimization style preference / safety / alignment fine-tuning) pipeline implemented under the `final/` directory. The workflow has four major stages:

1. Asset & dataset acquisition (`scripts/download.py`): download the SFT base model + tokenizer and pull the raw dataset.
2. Dataset transformation (`scripts/kto_datasets_process.py`): convert the raw dataset into a KTO-compatible preference format.
3. Alignment training (`trainer/kto.py`) using TRL's `KTOTrainer` with DeepSpeed ZeRO-2 (`recipes/accelerate_configs/zero2.yaml`) and training hyperparameters (`recipes/openseek/config.yaml`), launched by `train.sh` . Checkpoints saved every 1,000 steps.
4. Evaluation (`eval_example/`): contains benchmark outputs and aggregate metrics for the final checkpoint.

Public Checkpoint

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

Typical load snippet:

代码块

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

Directory Overview

代码块

```
final/
  recipes/
    accelerate_configs/
      zero2.yaml          # Accelerate + DeepSpeed ZeRO Stage 2 configuration
    openseek/
      config.yaml         # KTO training hyperparameters (Trainer-compatible YAML)
  scripts/
    download.py           # Download model/tokenizer + raw dataset (NuminaMath-CoT)
    kto_datasets_process.py  # Transform dataset → KTO preference format
  trainer/
    kto.py                # Main training entry (KTOTrainer)
  eval_example/           # Example evaluation results for final checkpoint
      final_result.json   # Aggregated metrics summary
      <benchmark_name>/   # Per-benchmark JSONL + metrics
  README.md               # (This report)
```

Stage 1: Download Base Assets (`scripts/download.py`)

Key actions:

- Downloads tokenizer & model from `BAAI/OpenSeek-Small-v1-SFT` (already SFT-prepared base for alignment).
- Saves them locally under `./models/OpenSeek-Small-v1-SFT` for reproducible offline reuse.
- Loads the raw dataset: `AI-MO/NuminaMath-CoT` from Hugging Face Hub.
- Persists dataset to disk: `./datasets/AI-MO/NuminaMath-CoT` (Arrow + metadata) to avoid repeated network fetches.

Environment variables (optional but recommended):

- `HF_ENDPOINT=https://hf-mirror.com` (for regional mirrors)
- `XDG_CACHE_HOME=./cache` (centralize HF cache)

Execution:

代码块

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

Outputs:

- `./models/OpenSeek-Small-v1-SFT/` (model weights, tokenizer files, config)
- `./datasets/AI-MO/NuminaMath-CoT/` (train/validation splits as provided by source dataset)

Stage 2: Dataset Transformation for KTO (`scripts/kto_datasets_process.py`)

Objective:

Convert the original multi-turn / message-style math reasoning dataset (`NuminaMath-CoT`) into a simplified preference alignment format required by KTO: each example should expose a prompt, a completion, and a binary label.

Implementation specifics:

- Loads the previously saved raw dataset from disk.
- For each sample, extracts the first two entries in `messages` :
  - `messages[0]` → becomes a single-element list assigned to `prompt` .
  - `messages[1]` → becomes a single-element list assigned to `completion` .
- Assigns `label = True` for all entries (i.e., all are treated as preferred / positive examples).
- Selects only the columns `["prompt", "completion", "label"]` .
- Saves the processed dataset to: `./datasets/AI-MO/NuminaMath-CoT-preference` .

Command:

代码块

```
python scripts/kto_datasets_process.py
```

Resulting dataset schema (per split):

代码块

```
{
  "prompt": List[Any]      # list-wrapped message dict(s) or text segment(s)
  "completion": List[Any]  # list-wrapped assistant answer
  "label": bool            # True → preferred sample
```

```
}
```

Example (illustrative, not verbatim):

代码块

```
{
  "prompt": [ {"role": "user", "content": "Solve: 2x + 3 = 7"} ],
  "completion": [ {"role": "assistant", "content": "x = 2"} ],
  "label": true
}
```

Notes & Considerations:

- Current processing creates only positive (True) labels. KTO can also leverage implicit negatives or additional heuristics. If extending, introduce negative variants (e.g., alternative incorrect completions) with `label=False`.
- Left vs right padding: handled later in tokenizer setup (alignment models usually benefit from left padding in generation-oriented training to keep latest tokens aligned in GPU compute).

Stage 3: Alignment Training (KTO) – `trainer/kto.py`

3.1 Launch Mechanism

Training is launched via Accelerate + DeepSpeed ZeRO Stage 2 for memory efficiency and multi-GPU scaling. The Slurm / shell entry is encapsulated in `train.sh`:

代码块

```
ACCELERATE_LOG_LEVEL=info accelerate launch \
  --config_file recipes/accelerate_configs/zero2.yaml \
  ./trainer/kto.py --config recipes/openseek/config.yaml
```

3.2 Accelerate + DeepSpeed Config (`zero2.yaml`)

Key parameters:

- `distributed_type: DEEPSPEED` & `zero_stage: 2` → ZeRO-2 sharding optimizer states + gradients (parameter partition not as full as ZeRO-3, but lower overhead).

- `mixed_precision: bf16` → uses BF16 if supported (Ampere+); stable vs FP16 on many math-heavy workloads.
- `num_processes: 8` → should match the number of visible GPUs (adjust to your cluster allocation).
- No optimizer or parameter CPU offload (`offload_*: none`) to reduce PCIe pressure (requires enough GPU RAM).

3.3 Training Hyperparameters (`config.yaml`)

Extracted key fields:

- Logging & checkpointing: `logging_steps: 1` , `save_steps: 1000` , `save_total_limit: 1` (keeps only the latest checkpoint to save disk).
- Model source: `model_name_or_path: ./models/OpenSeek-Small-v1-SFT` (the SFT base from Stage 1).
- Attention backend: `attn_implementation: flash_attention_2` (ensure FlashAttention v2 build compatibility).
- Data: `dataset_name: /workspace/datasets/AI-MO/NuminaMath-CoT-preference` (adjust to your actual path if different); `max_length: 4096` .
- Optimization:
  - `learning_rate: 2e-5`
  - Scheduler: `cosine_with_min_lr` + `min_lr_rate: 0.1` (final LR = base_lr * 0.1 at tail).
  - `warmup_ratio: 0.1`
  - `weight_decay: 0.01`
  - `gradient_accumulation_steps: 2` (effective batch = per_device * GPUs * accum).
  - `gradient_checkpointing: true` + `use_reentrant: false` (saves memory at cost of extra compute).
  - `max_grad_norm: 1.0` (gradient clipping).
  - `bf16: True` (reinforces BF16 usage in Trainer config).
  - Custom flags: `use_liger_kernel` , `use_liger_loss` (implies specialized fused ops or custom objective—ensure installed extensions if required).
- Epoch count: `num_train_epochs: 4` .

3.4 Tokenizer & Padding (`kto.py`)

- Tokenizer uses left padding (`tokenizer.padding_side = "left"`), typical for generation-focused alignment so most recent tokens align along the right edge in attention windows, improving efficiency for some kernels.
- If tokenizer lacks a `pad_token` , it falls back to `eos_token` .

3.5 Model & Reference Model

- Both `model` and `ref_model` are loaded from the same base. KTO uses the reference model to compute relative preference signals / calibration. Keeping them identical at initialization is standard.
- Quantization hooks (via `get_quantization_config`) are available but not explicitly set in the provided configs (would allow 4/8-bit experiments if desired).

3.6 Trainer Initialization

- Uses TRL `KTOTrainer` with:
  - `train_dataset=dataset[script_args.dataset_train_split]` (defaults typically `train`)
  - Optional eval dataset only if `eval_strategy != "no"` (currently disabled for speed).
  - `peft_config=get_peft_config(model_args)` (enables LoRA/other parameter-efficient fine-tuning if configured in `ModelConfig`). If PEFT is not explicitly configured, it may default to full fine-tuning.
- `use_cache` is disabled during training if gradient checkpointing is on.

3.7 Checkpoint Artifacts

At each save step (every 1,000 steps):

- Model weights (BF16)
- Trainer state (optimizer, scheduler unless limited by DeepSpeed stage boundary)
- RNG states for reproducibility

Because `save_total_limit: 1`, only the latest checkpoint directory is retained (rolling deletion of older ones). If you intend to run model soup or regression comparisons, increase this limit.

3.8 Performance & Memory Tips

- If encountering OOM:
  - Lower `per_device_train_batch_size`
  - Increase `gradient_accumulation_steps`
  - Reduce `max_length`
  - Enable quantization (4-bit/8-bit) if latency acceptable
- If throughput is low:
  - Ensure FlashAttention 2 is correctly installed (or switch to `sdpa` fallback)
  - Disable unnecessary logging (though `logging_steps: 1` is useful during early debugging, raise later)

Stage 4: Evaluation (`eval_example/`)

This stage reports benchmark results using a unified Chain-of-Thought prompting configuration.

```
PROMPT_TYPE="cot"
aime24:        seed 1, temperature 0.6, n_sampling 1, max_tokens_per_call 3072
amc23:         seed 1, temperature 0.6, n_sampling 1, max_tokens_per_call 3072
gsm8k:         seed 1, temperature 0.6, n_sampling 1, max_tokens_per_call 3072
math500:       seed 1, temperature 0.6, n_sampling 1, max_tokens_per_call 3072
minerva_math:  seed 1, temperature 0.6, n_sampling 1, max_tokens_per_call 3072
olympiadbench: seed 1, temperature 0.6, n_sampling 1, max_tokens_per_call 3072
```

Directory layout (unchanged):

```
eval_example/
  final_result.json   # Aggregated metrics
  <benchmark>/
    <run_id>_metrics.json   # Summary metrics
    <run_id>_result.jsonl   # Raw generations
```

`final_result.json` consolidates the per-benchmark metrics produced under the above consistent decoding / prompting setup.

End-to-End Execution Summary

```
# 1. Download base model + raw dataset
python scripts/download.py

# 2. Transform dataset into KTO preference format
```

```
python scripts/kto_datasets_process.py

# 3. Launch KTO alignment training (DeepSpeed ZeRO-2)
sbatch train.sh   # or run the accelerate command directly if not using Slurm

# 4. (After training) Evaluate checkpoint(s)
# (Use your evaluation tooling; results stored under eval_example/)
```

Reproducibility

Aspect	Mechanism	Notes
Random Seeds	`seed: 233` + `set_seed()`	Multi-worker data map & packing can still introduce slight nondeterminism.
Checkpointing	Every 1,000 steps	Only last retained unless `save_total_limit` increased.
Determinism	Not fully enforced	For stricter determinism: set CUDA deterministic flags (may degrade performance).

Recommendations:

- Pin versions of `transformers`, `datasets`, `trl`, `accelerate`, `torch`.
- Archive `zero2.yaml` + `config.yaml` with final model for auditability.

Extending / Modifying the Pipeline

Goal	Change

Introduce negative preferences	Modify `kto_datasets_process.py` to generate paired positive/negative samples (set some `label=False`).
Multi-reference completions	Convert single-element lists to multiple alternatives in `completion`; adjust trainer consumption.
Curriculum alignment	Stage multiple processed datasets; fine-tune sequentially.
Longer context	Increase `max_length`; ensure GPU memory headroom and FlashAttention support.
Retain multiple checkpoints	Increase `save_total_limit`; optionally add model averaging or smoothing post-training.
Parameter-efficient tuning	Configure LoRA / prefix tuning in the `ModelConfig` (passed via KTOTrainer).

Troubleshooting

Issue	Symptoms	Mitigation
OOM (GPU)	CUDA out of memory	Reduce batch size / seq length, enable gradient checkpointing (already on), use quantization.
Divergent loss	Loss spikes or NaNs	Lower LR, disable exotic kernels, check BF16 support, verify data labels.
Slow startup	Long dataset load	Ensure dataset is saved locally (disk I/O), increase

		`num_proc` during preprocessing only (not in training).
FlashAttention errors	Kernel build failures	Switch `attn_implementation` to `sdpa` or install matching CUDA toolkit & driver.
Checkpoints not saving	Missing directories	Verify write permissions & disk quota; ensure `output_dir` exists & not readonly.
No evaluation	Metrics absent	Set `do_eval: True` & `eval_strategy: steps` or `epoch`; supply proper eval split.

Security & Integrity Notes

- Trust only vetted model repos when `trust_remote_code=True`.
- Validate dataset integrity (hash or size) before large-scale training.
- For multi-tenant clusters, restrict write paths and use namespace-isolated caches.

License & Attribution

- Code headers indicate Apache 2.0 licensing.
- Base model: `BAAI/OpenSeek-Small-v1-SFT` (refer to its upstream license & usage terms).
- Dataset: `AI-MO/NuminaMath-CoT` (comply with its license and any redistribution constraints).

Summary

This `final/` pipeline layers preference-style alignment (KTO) atop an SFT base using a memory-efficient ZeRO-2 BF16 stack. It emphasizes reproducible dataset transformation, lean checkpoint retention, and modular extensibility for future preference formats, negative sampling, or evaluation automation.

Happy aligning.