

Tevatron Meets Megatron: Expert-Parallel LLM Reranker Training on an Academic Budget

Zhichao Xu
zhichao.xu@utah.edu
University of Utah
Salt Lake City, UT, USA

Xueguang Ma
x93ma@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Shengyao Zhuang
s.zhuang@uq.edu.au
The University of Queensland
Brisbane, QLD, Australia

Luyu Gao
luyug@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Wenqian Ye
wenqian@virginia.edu
University of Virginia
Charlottesville, VA, USA

Yu Wang
yuki.wang@utah.edu
University of Utah
Salt Lake City, UT, USA

Jamie Callan
callan@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Jimmy Lin
jimmylin@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

Training large dense and mixture-of-experts (MoE) rerankers is constrained by GPU memory and distributed-execution efficiency, which has kept academic reranker research largely on dense models. Tevatron trains rerankers through the Hugging Face Trainer with DeepSpeed and PyTorch FSDP1, a stack that stalls on three fronts: DeepSpeed ZeRO-2 has a known gradient-gathering failure on this path, FSDP1 full sharding is memory-safe but throughput-bound, and neither offers the expert parallelism (EP) that MoE training needs. We present Tevatron 3.0, which integrates Megatron-Core as a reranker training backend while preserving Tevatron’s data processing, evaluation pipeline, and Hugging Face checkpoint format. For full-parameter dense 8B training, Megatron matches FSDP1 on BEIR-15 NDCG@10 with approximately 4% lower epoch time and twice the peak GPU memory under matched data-parallel and recomputation settings, and approximately 22% lower epoch time in its recommended configuration; it supports both LoRA and full-parameter fine-tuning. Expert parallelism makes it possible to train a 30B-parameter MoE reranker (Qwen3-30B-A3B) on two nodes. A controlled study of architecture, LoRA, and distillation on BEIR-15, with three first-stage retrievers and Hugging Face and vLLM serving, shows that the MoE reranker matches dense-8B quality at less than half the activated parameters and at higher inference throughput. We open-source the framework and trained checkpoints.

CCS Concepts

• Information systems → Retrieval models and ranking.

Keywords

Tevatron, MoE Reranker, Megatron.

ACM Reference Format:

Zhichao Xu, Xueguang Ma, Shengyao Zhuang, Luyu Gao, Wenqian Ye, Yu Wang, Jamie Callan, and Jimmy Lin. 2026. Tevatron Meets Megatron: Expert-Parallel LLM Reranker Training on an Academic Budget. In *Proceedings of the 2026 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region (SIGIR-AP 2026)*, December 13–15, 2026, Singapore, Singapore. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3848000.3850192>

🔗 Code: Tevatron v3

🤖 Models: Tevatron v3 Reranker Collection

1 Introduction

The Tevatron toolkit [5] was first developed during the BERT [4] era to give researchers a flexible framework for training and evaluating neural retrievers and rerankers, and Tevatron 2.0 [12] extended it to billion-scale, multilingual, and multimodal dense retrieval. Reranking — re-scoring a first-stage candidate list with a more expensive cross-encoder [28] — has followed the same trajectory: state-of-the-art rerankers are now billion-scale LLMs [31], increasingly trained with knowledge distillation against strong teacher rerankers [6], and recent open-weight releases adopt mixture-of-experts (MoE) backbones that activate only a fraction of their parameters per token [14, 16]. These recipes deliver strong quality, but reproducing them often requires specialized distributed-training engineering, which is unavailable to many academic research groups with limited engineering resources [20].

Concretely, Tevatron’s existing reranker training path — the Hugging Face Trainer, whose memory-efficient training is provided by DeepSpeed [17] and PyTorch FSDP1 [32] — hits three limitations on modern reranking workloads.

- **DeepSpeed ZeRO-2** is less reliable on this path. A known CPU-offloaded gradient accumulation bug makes ZeRO-2 sharding fail in practice, forcing a fallback to PyTorch FSDP1.¹

¹Introduced in <https://github.com/deepspeedai/DeepSpeed/pull/6550> and not fixed until the recent <https://github.com/deepspeedai/DeepSpeed/pull/7967>.



This work is licensed under a Creative Commons Attribution 4.0 International License. *SIGIR-AP 2026, Singapore, Singapore*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-3033-7/2026/12

<https://doi.org/10.1145/3848000.3850192>

- **PyTorch FSDP1 full-sharding** is memory-safe but less efficient in terms of throughput. For an 8B reranker, plain data-parallel (DDP) runs out of memory, while FSDP1 `full_shard` (ZeRO-3) and `shard_grad_op` (ZeRO-2) both fit at near-identical wall-clock times — i.e., the heavier sharding stage brings no speed improvement — and the distributed-optimizer backend delivers higher throughput than either FSDP1 configuration.
- **No expert parallelism.** PyTorch FSDP1 offers no expert parallelism, so MoE rerankers (e.g., Qwen3-30B-A3B [30], 128 experts) cannot be trained efficiently: it can only replicate-then-shard the experts rather than route the active subset.

In this work we present Tevatron 3.0, which adds a Megatron training backend [19] to address these limitations while preserving the rest of the toolkit, together with a reworked evaluation interface. Our contributions are:

- **A Megatron reranker training backend (C1)** that is backward compatible with Tevatron 2.0’s data path, evaluation pipeline, and Hugging Face-loadable checkpoint format, using bridge-based HF↔Megatron weight conversion. We profile the existing FSDP settings on throughput and peak memory, then show that the Megatron backend (ZeRO-1-style distributed optimizer with tensor parallelism) matches FSDP-trained quality and — at matched data-parallel topology — matches its efficiency, while its recommended configuration reduces dense-8B epoch time by ~22%, supporting both LoRA [8] and full-parameter fine-tuning as well as listwise-KL distillation [1, 27] (a drop-in alternate trainer; Section 2.2). The toolkit also provides a unified evaluation interface with two schemas — local scoring of existing ranklists, and rerank-and-score against a remote HF/vLLM scoring-server pool over HTTP — behind one CLI (detailed in Section 2.3).
- **Expert-parallel MoE reranker training (C2):** Megatron’s expert parallelism trains a 30B-parameter MoE reranker on a single-job budget for which PyTorch FSDP1 offers no expert-parallel path, and we provide a named LoRA target-group registry so that low-rank adaptation can selectively target the appropriate expert layers in MoE models.
- **A demonstration and released artifacts (C3):** to exercise the system end-to-end, we evaluate combinations of model architecture (MoE 30B-A3B vs. dense 8B), parameter efficiency (LoRA vs. full-parameter), and loss formulation (listwise-KL distillation vs. contrastive), evaluated on BEIR-15 [22] with three first-stage retrievers, plus a serving-throughput matrix over Hugging Face and vLLM backends [9]. The study is a capability demonstration rather than the paper’s central claim; we release all trained checkpoints as reusable artifacts.

The emphasis of this work is the system: a backend that makes these recipes trainable and servable under an academic compute budget, with the data path, evaluation, and checkpoint format unchanged.² We use a controlled setting to exercise the proposed system end-to-end — and, as one illustrative result, it shows *the MoE reranker matching dense-8B quality at less than half the activated parameters*

²Tevatron 3.0 also adds other model classes to the toolkit — most notably decoder-LM learned sparse retrieval (LACONIC) [26, 29], which turns a causal LM into a SPLADE-style first-stage retriever. We scope this paper to the reranker training backend and refer the reader to that work for the learned-sparse retriever.

Table 1: Functionality comparison between the existing PyTorch FSDP1 backend (via the Hugging Face Trainer) and the proposed Megatron backend. [†]Parameter/gradient sharding is currently achieved through tensor parallelism; direct FSDP-style ZeRO-2/3 data-parallel sharding is left to future work. The PyTorch FSDP1 column describes the reranker training backend used in Tevatron 2.0.

Capability	PyTorch FSDP1	Megatron
Data parallelism (DP)	✓	✓
Optimizer-state sharding (ZeRO-1)	✓	✓
Grad/param sharding (ZeRO-2/3)	✓	via TP [†]
Tensor parallelism (TP)	✗	✓
Pipeline parallelism (PP)	✗	✓
Expert parallelism (EP, MoE)	✗	✓
Activation recompute	✓	✓
Full-parameter fine-tuning	✓	✓
LoRA fine-tuning	✓	✓
MoE-aware LoRA target groups	✗	✓
Reuses Tevatron data + eval	✓	✓
HF-format checkpoints (vLLM-ready)	✓	✓

and higher inference throughput, a favorable operating point that the framework now makes straightforward to reproduce.

2 Tevatron 3.0 System Design

Instead of an overhaul, Tevatron 3.0 is a set of *additive* features on top of Tevatron 2.0. The guiding principle is *backward compatibility*: every new capability below — the Megatron training backend [19], distillation, and the dual-schema evaluation interface — plugs into the existing framework’s data path, collator, and checkpoint conventions, reusing them unchanged. A practitioner can easily launch new experiments by changing a launch script or adding a flag, instead of migrating their data or re-learning the toolkit.

We first describe the design principle (§2.1), the implementation of the core training backend (§2.2), then the evaluation layer (§2.3), the extension points exposed to toolkit users (§2.4), and finally the characterization of backends (§2.5).

2.1 Design Principle

The most significant addition of Tevatron 3.0 is a Megatron-Core training backend alongside the existing Hugging Face Trainer path, exposed through the same command-line interface and consuming the same training data. The design goal is *substitutability*: a user switches backends by changing a launch script, instead of their data, their evaluation, or the delivery format of the checkpoint. Three properties make this possible.

First, the backend reuses the toolkit’s existing data path and evaluation pipeline without modification. The dataset, collator, and BEIR [22] evaluation modules are shared with the Hugging Face path; only the trainer object differs. A reranker is scored as a standard causal language model — the log-odds of the “yes”/“no” answer tokens at the final prompt position — so no architecture-specific value head is introduced and the same scoring code runs under any parallelism configuration (Section 2.2).

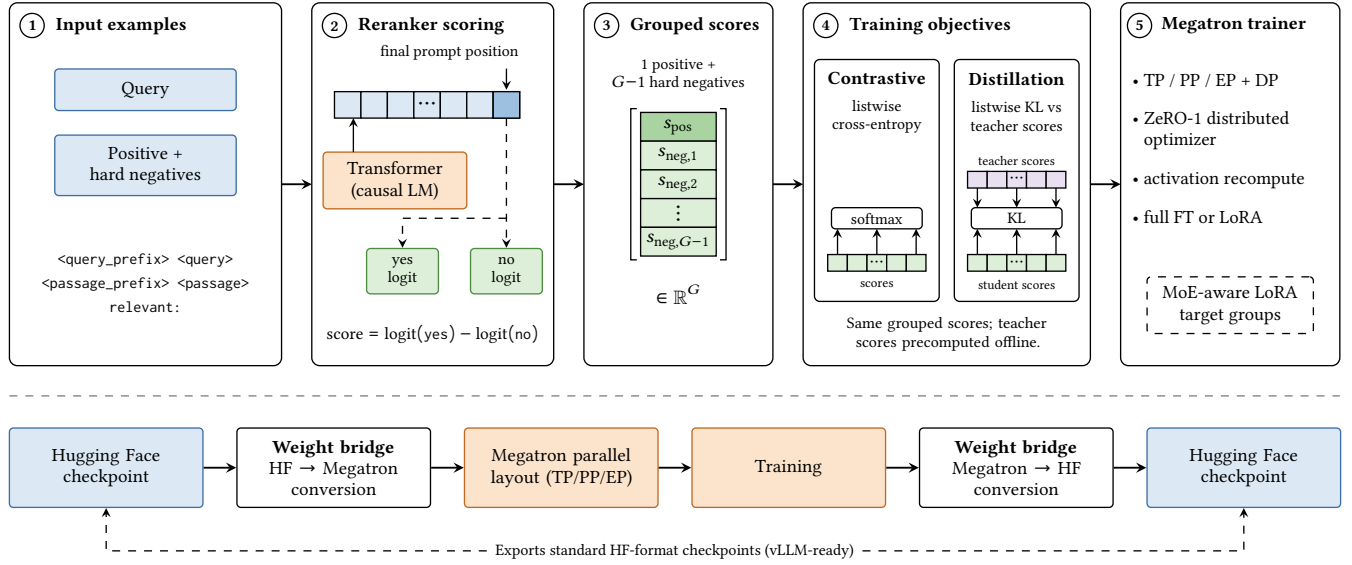


Figure 1: Overview of the Megatron training backend. Query–passage examples are scored by a standard causal language model and organized into listwise groups for contrastive or distillation training. The Megatron backend provides tensor, pipeline, and expert parallelism while preserving Hugging Face checkpoint compatibility through a bidirectional weight bridge.

Second, weights cross the Hugging Face $\leftrightarrow$ Megatron boundary in both directions through a weight bridge:³ a pretrained Hugging Face checkpoint is sharded into the Megatron parallel layout at load time, and the trained model is exported back to Hugging Face format at save time. Full-parameter fine-tuning and LoRA use the corresponding bridge save paths, while the latter writes pre-merged weights. The result is that every Tevatron 3.0 checkpoint loads through the ordinary Hugging Face and vLLM paths with no conversion step, preserving the rest of the ecosystem.

Third, the backend exposes the parallelism dimensions that PyTorch FSDP1 lacks: tensor (TP), pipeline (PP), and expert (EP) model parallelism. Data parallelism (DP) spans the remaining ranks after TP and PP are configured, while EP partitions the experts of each MoE layer across the ranks within a data-parallel group; in the configurations we use, EP divides DP rather than adding a further multiplicative dimension (Section 3). Megatron’s distributed optimizer shards optimizer states across DP ranks while replicating parameters and gradients, corresponding to a ZeRO-1-style scheme. This design minimizes communication overhead and, as shown in Section 2.5, is faster than the FSDP1 configurations used by the Hugging Face Trainer in its recommended topology and within a few percent at matched topology, in exchange for higher per-rank memory at the same data-parallel degree, because ZeRO-1 replicates parameters and gradients where FSDP1 shards them. The backend’s memory levers beyond ZeRO-1 optimizer-state sharding are therefore tensor parallelism for dense layers and expert parallelism for MoE experts, not FSDP-style parameter and gradient sharding. Critically, EP unblocks MoE reranker training. We treat the precise TP/PP/EP configuration as a hardware-dependent

deployment choice; the framework documentation provides recommended configurations for different model scales.

Table 1 summarizes the functionality gap between the two backends. The Megatron backend adds tensor, pipeline, and expert parallelism alongside support for activation recomputation. It retains backward compatibility with Tevatron’s existing data formats, evaluation pipeline, and Hugging Face checkpoints.

2.2 Implementation

Figure 1 summarizes the training path. The Megatron backend reuses Tevatron’s existing dataset, collator, and evaluation components; the trainer and distributed model execution are the only backend-specific components. The implementation follows the same sequence under both contrastive and distillation training: query–passage examples are converted into generative relevance scores, those scores are arranged into listwise groups, an objective is applied to each group, and gradients are updated under the configured parallel topology. The resulting checkpoint is then exported back to Hugging Face format through the weight bridge.

Reranker scoring. We format each query–passage pair as

<query_prefix><query> <passage_prefix><passage> relevant:

and score it with a generative reranker as the difference between the “yes” and “no” answer-token logits at the final non-padding prompt position. We define a scalar relevance score:

$$s_{\theta}(q, p) = z_{\theta}(q, p)_{\text{yes}} - z_{\theta}(q, p)_{\text{no}}.$$

For each query, one positive passage and $G-1$ hard negatives are scored independently and then reshaped into a grouped score vector $\mathbf{s}_i \in \mathbb{R}^G$. Because scoring uses only the ordinary causal LM output

³<https://docs.nvidia.com/nemo/megatron-bridge/latest/>

tensor, the same formulation applies without modification under data, tensor, pipeline, and expert parallelism.

```
# output_tensor: (B, S, V) LM logits; attention_mask: (B, S)
seq_lengths = attention_mask.sum(dim=-1) - 1 # last real token
last_logits = output_tensor[torch.arange(B), seq_lengths] # (B, V)
scores = last_logits[:, yes_token_id] - last_logits[:, no_token_id]
grouped_scores = scores.view(-1, G) # (n_groups, G); positive: 0
```

Contrastive and distillation losses. Both training objectives operate on the same grouped scores s . For contrastive training, the positive passage is placed at a fixed position in each group and optimized with listwise cross-entropy. For distillation, the student distribution over the same G passages is matched to a softened teacher distribution using listwise KL divergence. Teacher scores are computed offline and stored with the training examples, so distillation introduces no teacher model or additional forward pass during training. It therefore changes only the loss applied to the grouped scores s ; the data path, scoring logic, parallel configuration, and checkpoint procedure remain unchanged.

```
if loss_kind == "contrastive":
    labels = torch.zeros(grouped_scores.size(0), dtype=torch.long)
    loss = F.cross_entropy(grouped_scores, labels) # positive: 0
elif loss_kind == "distill":
    student = F.log_softmax(grouped_scores / s_temp, dim=-1)
    teacher = F.softmax(teacher_scores.view(-1, G) / t_temp, dim=-1)
    loss = F.kl_div(
        student, teacher, reduction="batchmean") * (s_temp ** 2)
```

Parallelism and LoRA. Parallelism is critical in scaling up the model size in a multi-GPU, multi-node distributed training setup. In Tevatron 3.0, parallelism is configured by three flags: tensor (TP), pipeline (PP), and expert (EP) model-parallel sizes, with data parallelism derived as $DP = \text{world_size} / (TP \cdot PP)$. Expert parallelism is carved out of the data-parallel group, with EP dividing DP in the configurations we use (Section 3). The data-parallel scheme uses Megatron’s distributed optimizer (ZeRO-1-style: optimizer state sharded across DP ranks, parameters and gradients replicated). LoRA is injected through a pre-wrap hook so adapter parameters land in the DDP gradient buffers and the distributed optimizer sees only the trainable adapters. For MoE models, a named target-group registry maps roles to module patterns, preventing naive dense patterns from inadvertently matching all the expert layers.

```
LORA_TARGET_GROUPS = {
    "attn": ("linear_qkv", "linear_proj"),
    "mlp": ("linear_fc1", "linear_fc2"), # dense MLP only
    "moe_experts": (
        "*.experts.*.linear_fc1", "*.experts.*.linear_fc2"),
    "moe_shared": ("*.shared_experts.linear_fc1",
        "*.shared_experts.linear_fc2"),
    "moe_router": ("*.router.weight",), # usually frozen
}
```

Weight bridge. A pretrained Hugging Face model is sharded into the Megatron parallel layout at the start of the distributed training. When checkpointing during or after the training run, the model in Megatron parallel layout is written to disk in Hugging Face format via the weight bridge (full parameters for full fine-tuning; pre-merged weights for LoRA), so they load through the standard Hugging Face and vLLM paths with no conversion step.

Distillation training. Distillation uses a sibling training driver with the same parallelism flags. Teacher scores are annotated *offline* by `tevatron.utils.annotate_with_teacher`, then loaded by the

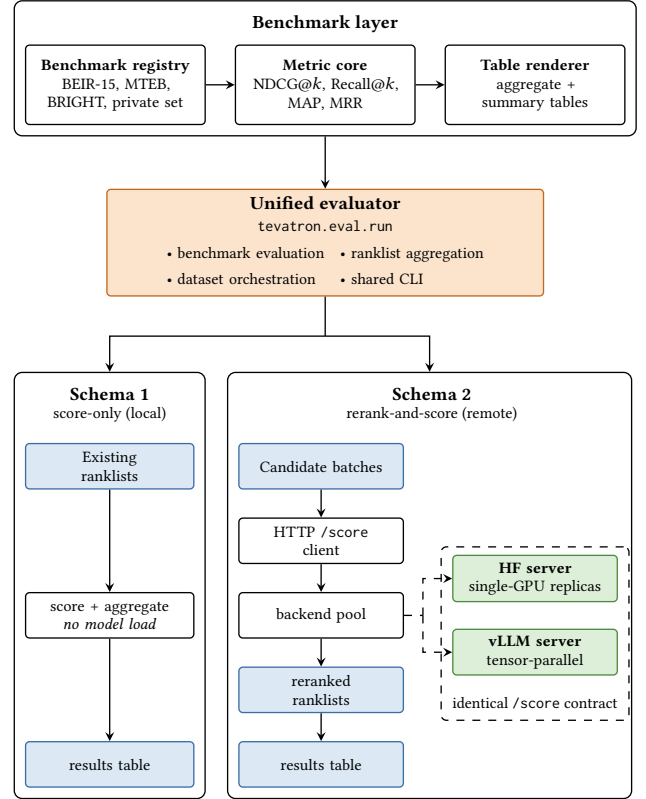


Figure 2: Overview of the unified evaluation and serving pipeline. The same evaluator supports local scoring of existing ranklists and remote rerank-and-score evaluation through interchangeable Hugging Face and vLLM serving.

trainer, which exposes the teacher and student temperatures. The scoring and checkpoint paths are shared with contrastive training.

```
# Contrastive (listwise CE):
torchrun ... -m tevatron.megatron.driver.train \
    --model_name_or_path Qwen3-8B-Base --dataset_name rlhn/rlhn-680K \
    --train_group_size 8 ...

# Distillation (listwise KL; offline teacher scores).
# Same parallelism flags; add dataset and temperatures.
torchrun ... -m tevatron.megatron.driver.distill_train \
    --model_name_or_path Qwen3-8B-Base \
    --distill_dataset_path rlhn-680K-qwen3-reranker-8b-top200 \
    --train_group_size 8 --teacher_temp 2.0 --student_temp 1.0 ...
```

2.3 A Unified Evaluation and Serving Pipeline

Evaluation is likewise an additive layer over the existing reranking-eval code rather than a replacement. It separates three concerns — an invariant metric core (score(qrels, results, k) → NDCG@k / Recall@k / MAP / MRR), a benchmark registry (which datasets, where qrels come from, how they aggregate), and a generic table renderer — so a benchmark beyond BEIR-15 is a registry entry rather than a code change (Section 2.4). As shown in Figure 2, a unified entry point (`tevatron.eval.run`) runs benchmark evaluation — aggregating per-dataset scores into the BEIR-15 mean — and supports two usage schemas behind the same CLI:

- (1) **Score-only (local).** Given existing ranklists (first-stage runs or already-reranked outputs), it scores and aggregates directly without loading a model:

```
python -m tevatron.eval.run \
  --ranklist_pattern ../../{dataset}/rank.text \
  --results_dir results/e5-base --name e5-base-v2
```

- (2) **Rerank-and-score via a remote HTTP pool.**

The reranker is hosted as a persistent scoring server — vllm or hf backend — behind a small HTTP API (POST /score); the evaluator becomes a thin client that dispatches candidate batches to one or more backend URLs and reassembles the ranklists. The model is loaded once per pool and reused throughout the evaluation, and the pool can run on a remote (multi-GPU) node while the evaluation is driven from elsewhere:

```
# Host on a remote GPU node; load the model once.
python -m tevatron.eval.server --backend vllm \
  --model <ckpt> --tensor_parallel_size 2 --port 8100

# Client (anywhere): rerank+score against one or more backend URLs.
python -m tevatron.eval.run \
  --backends http://node:8100 http://node:8101 ... \
  --rerank_input_pattern ../../{dataset}/rerank.jsonl \
  --results_dir results/<ckpt>
```

Both backends expose the identical /score contract, so the same client load-balances across a pool of single-GPU hf replicas or tensor-parallel vLLM servers interchangeably; this is what makes the {HF, vLLM} throughput matrix (Table 6) a one-line backend swap. Our inference measurements show that the vLLM serving path provides substantially higher throughput than the Hugging Face path.

2.4 Extensibility

New loss formulation. The training objective is an isolated function of the grouped scores; contrastive CE and listwise-KL distillation are two branches over the same (n_groups, G) tensor (Section 2.2). A new objective — MarginMSE [7], RankNet [2], or listwise loss formulations [3] — is one more branch; the scoring, parallelism, and data machinery are untouched:

```
if loss_kind == "contrastive":
    loss = F.cross_entropy(grouped_scores, labels) # positive: 0
elif loss_kind == "distill":
    loss = F.kl_div(
        student_lp, teacher_p, reduction="batchmean") * (s_temp ** 2)
# elif loss_kind == "my_loss":
#     loss = my_objective(grouped_scores, ...) # new loss
```

New backbone / architecture. Models are loaded through the HF-config-driven weight bridge, so any bridge-supported architecture trains with no reranker-side change. The one place a new family may need attention is LoRA targeting, which is a named registry mapping roles to module patterns — the same mechanism we added to make LoRA work on MoE expert layers (Section 3). Each new model family adds a corresponding registry entry:

```
LORA_TARGET_GROUPS = {
    "attn": ("linear_qkv", "linear_proj"),
    "moe_experts": (
        "*.experts.*.linear_fc1", "*.experts.*.linear_fc2"),
    # "my_family_block": ("...module.name.patterns...",) # add
}
```

New evaluation benchmark. Scoring is split from benchmark definition: the metric core scores a ranklist against qrels with no benchmark knowledge, while a Benchmark registry entry carries the dataset list, qrels loader, split convention, and table layout.

Adding a suite beyond BEIR-15 (MTEB [15], BRIGHT [21], a private set) is one entry; the metric core, orchestrator, renderer, and backends are reused unchanged:

```
BENCHMARKS = {
    "beir15": Benchmark(
        datasets=..., load_qrels=...,
        split_for=..., summary_layout=...),
    # "my_bench": Benchmark(...), # run --benchmark my_bench
}
```

New inference / serving backend. The scoring server is defined by one interface: a backend loads a model and implements score() returning per-candidate scores; the HTTP layer, client, load-balancing, and evaluator are backend-agnostic. Supporting a new engine (e.g., SGLang [33], TensorRT-LLM)⁴ is a new subclass — exactly how the vllm and hf backends already coexist behind the same contract:

```
class _BaseBackend:
    def score(self, req: ScoreRequest) -> list[ScoreItem]: ...

class VLLMBackend(_BaseBackend): ... # implemented
class HFBackend(_BaseBackend): ... # implemented
class MyEngineBackend(_BaseBackend): # new backend
    def score(self, req): ...
```

2.5 Backend Characterization

Before introducing the Megatron backend, we characterize what the existing PyTorch FSDP1 backend can and cannot do, and then compare with our new Megatron backend. We fine-tune the same dense 8B reranker (Qwen3-8B [30], full parameters, contrastive loss, RLHN-680K [23], one epoch, 64 queries per step, single 8×H200 node) under three data-parallel schemes and measure wall-clock and peak GPU memory per rank (Table 2). Peak memory is read from the training run’s logged GPU statistics.

Profiling the existing FSDP backend. We make two observations. First, full-parameter 8B training requires sharding: plain DDP, which replicates the optimizer state on every rank, exhausts all 140 GiB and fails at the first optimizer step — an 8B model under mixed-precision AdamW needs roughly 128 GiB of model states per rank (parameters, gradients, fp32 master copy, and two moment buffers) before activations. Second, among the training configurations that can fit in one 8×H200 node, the heavier sharding option does not bring throughput improvement: full_shard (ZeRO-3), which additionally shards parameters and all-gathers them per layer, matches shard_grad_op (ZeRO-2) in both wall-clock (11.4 vs ~11.6 h) and peak memory (51.2 vs 48.8 GiB), because at 8B on one node the optimizer state — sharded in both — dominates the footprint and the extra parameter gather is hidden by prefetch. The existing backend is therefore *memory-safe but throughput-bound*: it leaves no obvious headroom to recover by tuning the sharding stage, which motivates a backend with a faster data-parallel scheme.

Megatron matches FSDP quality and efficiency. A faster backend is only useful if it does not sacrifice task performance. We verify the training quality by training the dense 8B reranker with both backends under matched data, recipe, and global batch, and evaluating on BEIR-15 (mean NDCG@10 over 15 tasks; Section 4.1) across all three first-stage retrievers. Table 3 shows the two backends are indistinguishable on the full-parameter path: every first stage agrees within 0.002 NDCG@10. The LoRA path likewise tracks closely,

⁴<https://github.com/NVIDIA/TensorRT-LLM>

Table 2: Profiling the PyTorch FSDP1 backend on dense 8B full-parameter fine-tuning, single 8×H200 node, matched 64-query batch. Plain DDP replicates all state; ZeRO-2 shards gradients and optimizer state; ZeRO-3 also shards parameters. Plain DDP is infeasible; the two sharded schemes fit but differ negligibly.

DP scheme	Sharding	Wall-clock	Peak mem/GPU	Fits 1 node
Plain DDP	none	—	OOM (>140 GiB)	✗
FSDP shard_grad_op	ZeRO-2	~11.6 h	48.8 GiB	✓
FSDP full_shard	ZeRO-3	11.4 h	51.2 GiB	✓

Table 3: Reranker quality (BEIR-15 NDCG@10) is backend-invariant on the full-parameter path (agreement within 0.002 on every first stage). Dense 8B, contrastive loss.

Backend	Param-eff.	BM25	E5	SPLADE-v3
<i>First stage (no reranker)</i>	—	<i>0.415</i>	<i>0.503</i>	<i>0.504</i>
PyTorch FSDP1	full-FT	0.595	0.623	0.617
Megatron	full-FT	0.596	0.624	0.619
PyTorch DDP	LoRA	0.598	0.626	0.620
Megatron	LoRA	0.590	0.614	0.610

with the Hugging Face variant marginally ahead — by 0.008 to 0.012 NDCG@10, a gap comparable to the small differences in the loss comparison (Section 4.3) that we cannot reliably distinguish from the variation between independent training runs.

With quality controlled, we now turn to efficiency, and report it at two levels: a *configuration* comparison (each backend in its recommended setup) and a *matched-topology* comparison (same DP degree and activation recompute) that isolates framework efficiency from the chosen topology.

Recommended-configuration comparison. Under each backend’s recommended single-node configuration, the Megatron backend (TP = 2, DP = 4, without activation recomputation) completes one dense-8B epoch in 8h 55m compared to 11h 22m for PyTorch FSDP1 full_shard, a 22% reduction in wall-clock time. A similar gap appears under LoRA (8h 47m vs. 10h 50m, ~19%). Because these default setups differ in both parallel topology and recomputation policies, this comparison reflects practical out-of-the-box runtimes rather than an isolated framework speedup.

Matched-topology comparison. To isolate the framework, we re-run both backends at *data parallelism-matched* DP=8 with activation recompute enabled on both: PyTorch FSDP1 shard_grad_op (ZeRO-2) vs. Megatron TP=1/DP=8 (ZeRO-1) (Table 4). Here the gap nearly closes — Megatron is only ~4% faster with ~2× the per-rank memory, because ZeRO-1 replicates parameters and gradients while FSDP shard_grad_op shards gradients. These results demonstrate that the execution efficiency of both backends is comparable at matched topology. The observed speedup of the recommended Megatron setup primarily stems from operating in a lower-communication topology (TP=2, DP=4) without activation recomputation, rather than low-level runtime overheads. These results identify expert parallelism for MoE training (Section 3),

Table 4: Dense-8B full-parameter training efficiency, single 8×H200 node, matched 64-query batch. *Recommended* compares each backend’s out-of-the-box config (different topology + recompute); *Matched DP=8* holds DP and activation recompute fixed to isolate the framework, where the two are within ~4% and Megatron’s ZeRO-1 trades memory for the small speed edge. Megatron’s real differentiator is expert parallelism (Section 3), not dense throughput.

Comparison	Backend (topology)	Shard	Wall-clock	Mem/GPU
Recommended	FSDP1 (DP=8)	ZeRO-3	11h 22m	51 GiB
	Megatron (TP=2/DP=4)	ZeRO-1	8h 55m	106 GiB
Matched DP=8	FSDP1 (DP=8)	ZeRO-2	11h 00m	49 GiB
	Megatron (TP=1/DP=8)	ZeRO-1	10h 36m	106 GiB

rather than dense throughput alone, as the main advantage of the Megatron backend. PyTorch FSDP1 provides no corresponding expert-parallel training path.

The backend supports LoRA and full-parameter fine-tuning identically. We turn to expert parallelism next.

3 Expert-Parallel MoE Training

The key advantage of the Megatron backend is that it enables mixture-of-experts training. A modern MoE reranker like Qwen3-30B-A3B [30] carries 30B total parameters across 128 experts but activates only ~3B per token (top-8 routing). The memory bottleneck is from the *experts*, not the attention/feed-forward: holding all 128 experts’ weights and optimizer state on every rank is inefficient or infeasible. PyTorch FSDP1 can only replicate-then-shard them as ordinary parameters, with no mechanism for routing computation to the experts on a given rank.

Expert parallelism and topology. Megatron introduces expert parallelism (EP) as an additional parallel dimension alongside the tensor- and data-parallel axes described in Section 2.2. Whereas TP shards the dense transformer layers and DP replicates the model across data-parallel ranks, EP partitions only the MoE experts. In practice, the EP dimension folds into the existing 3D Megatron topology by subdividing each data-parallel group into expert-parallel subgroups: dense layers continue to execute under the same TP×DP configuration, while MoE layers dispatch routed tokens only among the ranks within an EP group. As a result, the overall world size is still TP×DP (with pipeline parallelism omitted here), with EP dividing DP in our configuration: the experts of each MoE layer are partitioned EP ways and replicated DP/EP times, and the distributed optimizer shards their optimizer state only across those DP/EP replicas, while the dense layers and their optimizer-state shards span all DP ranks, and only the MoE layers incur the additional expert communication. Per-rank expert memory therefore scales with experts/EP rather than the full expert count. Training Qwen3-30B-A3B across two 8×H200 nodes, i.e., sixteen GPUs with TP=1, DP=16, and EP=16, so that each GPU holds one sixteenth of the experts, with an expert replication factor DP/EP of one and hence the full optimizer state of its own experts, peaks at 78.9 GiB per GPU because the 128 experts are partitioned sixteen ways, a capability unavailable in PyTorch FSDP1.

Table 5: NDCG@10 reranking results on different first-stage retrievers (top-200 candidates). Task abbreviations: ARG arguana, CFE climate-fever, DBP dbpedia-entity, FEV fever, FIQ fiqa, HOP hotpotqa, NFC nfcopus, NQ nq, QUO quora, SCD scidocs, SCF scifact, TRC trec-covid, TOU webis-touche2020, MSM MS MARCO dev, CQA CQADupstack (avg of 12). Row labels: Dense 8B = Qwen3-8B, MoE 30B-A3B = Qwen3-30B-A3B; C = contrastive, D = listwise-KL distillation; F = full-parameter, L = LoRA.

Method	ARG	CFE	DBP	FEV	FIQ	HOP	NFC	NQ	QUO	SCD	SCF	TRC	TOU	MSM	CQA	BEIR-15
<i>BM25</i>																
First	0.397	0.165	0.318	0.651	0.236	0.633	0.338	0.306	0.789	0.149	0.679	0.595	0.442	0.228	0.302	0.415
Dense 8B C/F	0.797	0.361	0.487	0.911	0.512	0.833	0.410	0.658	0.810	0.261	0.793	0.887	0.353	0.427	0.447	0.596
Dense 8B C/L	0.734	0.318	0.504	0.903	0.499	0.828	0.407	0.656	0.837	0.247	0.798	0.885	0.358	0.424	0.445	0.590
Dense 8B D/F	0.733	0.411	0.481	0.909	0.485	0.810	0.404	0.623	0.780	0.242	0.795	0.865	0.424	0.433	0.391	0.586
Dense 8B D/L	0.686	0.392	0.493	0.907	0.472	0.809	0.396	0.624	0.826	0.236	0.792	0.854	0.410	0.431	0.406	0.582
MoE 30B-A3B C/F	0.809	0.374	0.494	0.908	0.520	0.834	0.410	0.662	0.829	0.271	0.809	0.882	0.323	0.433	0.450	0.600
MoE 30B-A3B C/L	0.747	0.321	0.497	0.896	0.494	0.826	0.411	0.659	0.854	0.261	0.794	0.895	0.288	0.426	0.455	0.588
MoE 30B-A3B D/F	0.739	0.408	0.483	0.910	0.490	0.810	0.406	0.624	0.793	0.247	0.795	0.858	0.400	0.433	0.402	0.587
MoE 30B-A3B D/L	0.661	0.389	0.490	0.905	0.468	0.809	0.404	0.623	0.841	0.240	0.790	0.858	0.403	0.429	0.382	0.579
<i>E5-base-v2</i>																
First	0.446	0.266	0.422	0.850	0.399	0.691	0.354	0.582	0.861	0.187	0.720	0.696	0.264	0.427	0.374	0.503
Dense 8B C/F	0.801	0.406	0.526	0.944	0.581	0.867	0.427	0.744	0.790	0.278	0.809	0.890	0.331	0.474	0.487	0.624
Dense 8B C/L	0.739	0.344	0.542	0.935	0.555	0.860	0.422	0.743	0.825	0.264	0.816	0.888	0.330	0.468	0.482	0.614
Dense 8B D/F	0.743	0.479	0.513	0.941	0.551	0.839	0.419	0.700	0.770	0.256	0.807	0.883	0.408	0.478	0.419	0.614
Dense 8B D/L	0.694	0.453	0.529	0.939	0.531	0.836	0.402	0.699	0.821	0.250	0.804	0.876	0.399	0.479	0.434	0.610
MoE 30B-A3B C/F	0.815	0.430	0.530	0.941	0.599	0.868	0.431	0.751	0.814	0.289	0.827	0.880	0.305	0.476	0.490	0.630
MoE 30B-A3B C/L	0.756	0.348	0.541	0.926	0.562	0.859	0.434	0.746	0.846	0.277	0.808	0.887	0.271	0.470	0.496	0.615
MoE 30B-A3B D/F	0.750	0.475	0.514	0.942	0.553	0.840	0.421	0.698	0.785	0.259	0.806	0.876	0.390	0.478	0.429	0.615
MoE 30B-A3B D/L	0.669	0.452	0.520	0.935	0.528	0.837	0.416	0.697	0.836	0.250	0.804	0.874	0.397	0.475	0.407	0.606
<i>SPLADE-v3</i>																
First	0.488	0.256	0.445	0.810	0.380	0.689	0.363	0.586	0.814	0.156	0.716	0.732	0.312	0.459	0.344	0.504
Dense 8B C/F	0.815	0.391	0.524	0.939	0.561	0.831	0.416	0.742	0.798	0.270	0.807	0.899	0.349	0.471	0.471	0.619
Dense 8B C/L	0.748	0.338	0.546	0.930	0.543	0.826	0.408	0.740	0.829	0.256	0.812	0.892	0.348	0.466	0.467	0.610
Dense 8B D/F	0.749	0.460	0.518	0.936	0.528	0.807	0.410	0.697	0.771	0.249	0.805	0.886	0.427	0.475	0.407	0.608
Dense 8B D/L	0.697	0.434	0.532	0.935	0.508	0.805	0.393	0.695	0.821	0.244	0.802	0.869	0.409	0.476	0.423	0.603
MoE 30B-A3B C/F	0.828	0.406	0.530	0.936	0.581	0.833	0.418	0.749	0.819	0.279	0.823	0.885	0.319	0.473	0.473	0.623
MoE 30B-A3B C/L	0.763	0.342	0.540	0.923	0.537	0.824	0.416	0.744	0.849	0.266	0.804	0.900	0.278	0.467	0.478	0.608
MoE 30B-A3B D/F	0.756	0.456	0.519	0.938	0.532	0.807	0.414	0.696	0.783	0.253	0.804	0.882	0.392	0.476	0.418	0.608
MoE 30B-A3B D/L	0.672	0.431	0.524	0.931	0.505	0.806	0.409	0.695	0.834	0.245	0.800	0.866	0.400	0.474	0.396	0.599

LoRA target groups for MoE. Applying LoRA to an MoE model exposes a subtlety. The conventional dense target specification adapts the attention and MLP projection modules; but on an MoE model the MLP leaf modules (`linear_fc1/linear_fc2`) also name every one of the 128 expert feed-forward networks, so a naive dense pattern either silently adapts all experts (an enormous, unintended parameter count) or, if restricted to attention only, leaves the experts — where the reranking signal concentrates — untouched. We

observe the latter failure directly: an attention-only LoRA on the MoE backbone underperforms its full-parameter counterpart by a wide margin, worse even than dense LoRA at far smaller scale. We therefore expose a named target-group registry (Section 2.2) that maps roles (`attn`, `moe_experts`, `moe_shared`, `moe_router`) to the correct module patterns, and use the expert-aware specification (`attn + moe_experts + moe_shared`, leaving the router frozen for stability) for all MoE LoRA runs in our study.

Table 6: Reranking throughput (pairs/s, summed over eight independent single-GPU workers on H200s; BM25 candidate set; contrastive full-FT checkpoints). The MoE serves faster than dense 8B, and the advantage grows under vLLM.

Backend	8B dense	30B-A3B MoE	MoE speedup
Hugging Face	579	668	1.15×
vLLM	1,834	2,622	1.43×
vLLM speedup	3.17×	3.93×	

4 Demonstration: A Controlled Study of Reranker Recipes

Having established the backend (Section 2) and its MoE capability (Section 3), we use Tevatron 3.0 to run a study that would be awkward or infeasible on the prior stack: a controlled comparison of model architecture, parameter efficiency, and loss formulation, all trained through one backend and evaluated through one pipeline. We present it as a *demonstration of the system* — evidence that the framework supports controlled, apples-to-apples comparison at scale, and release the checkpoints for reproducibility.

4.1 Setup

All rerankers are trained on RLHN-680K [23] for one epoch with a listwise group size of 8 (one positive, seven hard negatives), maximum sequence length 512, bf16, on a single 8×H200 node (two nodes for the MoE runs), holding the global batch at 64 queries per step so that throughput and wall-clock are directly comparable. Full-parameter and LoRA runs share this recipe; LoRA uses a 10× larger learning rate. Distillation runs minimize a listwise KL against per-candidate teacher scores precomputed offline from a Qwen3-Reranker-8B teacher [31]. We report one training run per configuration. Without repeated runs, we cannot determine whether small performance differences reflect systematic effects or random variation across independent runs.

We evaluate on BEIR-15 [22] — the mean NDCG@10 over 15 tasks (BEIR-13, MS MARCO dev, and CQADupstack averaged over its 12 sub-forums) — reranking the top 200 candidates of each first stage. We use three first-stage retrievers chosen to span the lexical / dense / learned-sparse spectrum and a range of first-stage strength: BM25 [18], E5-base-v2 [24], and SPLADE-v3 [10], with BEIR-15 NDCG@10 of 0.415, 0.503, and 0.504 respectively. We deliberately avoid first stages strong enough to leave no reranking headroom; reporting all three lets us read how the reranking gain varies with first-stage quality rather than reading a single operating point.

4.2 MoE Quality and Serving Throughput

The central question for deployment is whether the MoE reranker, activating less than half the parameters per token, gives up quality relative to the dense 8B. As shown in Table 5, the 30B-A3B MoE model matches the retrieval effectiveness of the dense 8B baseline across all first-stage retrievers, despite activating fewer than half the parameters per token. The aggregate occasionally favors the MoE, but the margins are small, with no variance estimate to judge them against, and the macro-average hides per-task structure, so we do not claim the MoE is strictly better performance-wise.

Inference throughput. We next test whether fewer activated parameters translate into higher serving throughput. Table 6 measures reranking throughput (query–passage pairs per second) over the full BM25 candidate set, scoring with eight independent single-GPU workers under two serving backends, Hugging Face and vLLM. Two effects are visible and they compound. Across backends, vLLM is 3.2–3.9× faster than Hugging Face, consistent with its use of paged-KV and continuous batching [9]. Across architectures, the MoE is faster than the dense 8B — 1.15× on Hugging Face, widening to 1.43× on vLLM. The MoE’s smaller active parameter count and vLLM’s batching are plausible contributors to these gains. The matrix measures end-to-end throughput under each configuration; isolating the contributions of batching, kernels, and expert routing would require additional profiling. The fastest configuration (vLLM × MoE, 2,622 pairs/s) delivers 4.5× the throughput of the slowest (Hugging Face × 8B, 579 pairs/s).

The 30B-A3B MoE thus matches dense-8B quality with higher serving throughput in this setting. Training it uses two nodes with expert parallelism, versus one for the dense 8B; total training cost and a serving break-even point remain unmeasured.

4.3 Parameter Efficiency and Distilled Training

LoRA recovers most of full fine-tuning. In Table 5, rank-16 LoRA on the dense 8B trails full-parameter training by ~0.006–0.010 NDCG@10 across first stages while training a small fraction of the parameters; on the MoE, the expert-aware target specification (Section 3) is what makes this recovery possible at all. Combined with the near-identical wall-clock of LoRA and full-parameter runs (the frozen-base forward/backward dominates), LoRA primarily provides memory savings rather than latency reductions, offering a practical trade-off between peak memory and ranking accuracy.

Distillation and contrastive learning specialize on different datasets. Across the evaluated configurations, contrastive training achieves higher mean BEIR-15 NDCG@10 than distillation, although distillation improves performance on some individual datasets. Per-task analysis reveals that the aggregate BEIR-15 score obscures substantial dataset-specific performance divergence. With the teacher temperature fixed at 2, distillation improves NDCG@10 on CFE and TOU but reduces it on ARG and NQ across the evaluated configurations. We believe two structural factors bound the effect further: our 8B teacher is the *same size* as the dense student and, for the 30B-A3B MoE student, smaller in total but larger in active parameters, so neither student has a clearly larger teacher to distill from, and the training data’s hard negatives are already denoised, removing distillation’s usual false-negative-relabeling value. We therefore report the loss comparison as *illustrative under this fixed recipe*; pinning down the exact contrastive-vs-distillation tradeoff — its dependence on temperature, group size, and teacher capacity — is beyond this work’s scope.

4.4 Sanity Check Against Prior Rerankers

We include a comparison against prior reranker baselines (Table 7) as a *sanity check on our training pipeline*, not as a competitive claim. Tevatron 3.0 is a training-and-serving system; the checkpoints in this paper are artifacts produced *by* that system under one fixed recipe, released so that others can reproduce and build on

Table 7: Sanity check of our released checkpoints against prior rerankers (per-dataset NDCG@10 on 15 BEIR tasks, all reranking SPLADE-v3 top-200). Baselines differ in training data and training-stage retrievers. The results place our released checkpoints in the expected range. AVG reports the BEIR-13 mean, excluding CQADupstack and MS MARCO, as commonly reported by prior works. One contrastive and one distilled full-parameter checkpoint per backbone. Best result per column in bold, second best underlined; the first-stage row is excluded.

Model	ARG	CFE	DBP	FEV	FIQ	HOP	NFC	NQ	QUO	SCD	SCF	TRC	TOU	MSM	CQA	AVG
<i>First stage</i>																
SPLADE-v3 (no rerank)	0.488	0.256	0.445	0.810	0.380	0.689	0.363	0.586	0.814	0.156	0.716	0.732	0.312	0.459	0.344	0.519
<i>Prior rerankers</i>																
RankT5 [34]	0.330	0.215	0.442	0.832	0.445	0.710	0.381	0.614	<u>0.831</u>	0.181	0.750	0.807	0.440	-	-	0.537
RankLLaMA [13]	0.560	0.280	0.483	0.839	0.465	0.753	0.303	0.663	0.850	0.178	0.732	0.852	0.401	-	-	0.566
RankQwen [27]	0.791	0.405	0.543	0.940	0.559	0.848	0.424	<u>0.745</u>	0.776	<u>0.271</u>	<u>0.819</u>	0.883	0.329	-	-	0.641
<i>Ours (Tevatron 3.0)</i>																
Dense 8B, contrastive	<u>0.815</u>	0.391	0.524	<u>0.939</u>	<u>0.561</u>	0.831	0.416	0.742	0.798	0.270	0.807	0.899	0.349	0.471	<u>0.471</u>	<u>0.642</u>
Dense 8B, distill	0.749	0.460	0.518	0.936	0.528	0.807	0.410	0.697	0.771	0.249	0.805	<u>0.886</u>	<u>0.427</u>	<u>0.475</u>	0.407	0.634
MoE 30B-A3B, contrastive	0.828	0.406	<u>0.530</u>	0.936	0.581	<u>0.833</u>	<u>0.418</u>	0.749	0.819	0.279	0.823	0.885	0.319	0.473	0.473	0.646
MoE 30B-A3B, distill	0.756	<u>0.456</u>	0.519	0.938	0.532	0.807	0.414	0.696	0.783	0.253	0.804	0.882	0.392	0.476	0.418	0.633

them. Because these checkpoints are reproducible baseline artifacts rather than heavily tuned competitive submissions, small differences should not be over-interpreted. The baselines were also generally trained on different data with different first stages, so the rows are not strictly controlled. All rows rerank the same SPLADE-v3 top-200 candidates; the first row is that first stage alone. Per-dataset NDCG@10 uses the task abbreviations of Table 5, and AVG is the BEIR-13 mean as commonly reported by prior works [13, 27, 34, 35].

The per-dataset results against the strongest baseline, RankQwen, are mixed in both directions and the margins are small. Against our contrastive MoE checkpoint, the row with the highest mean, RankQwen leads on five datasets (DBP, FEV, HOP, NFC, and TOU) by 0.004 to 0.015 NDCG@10 and trails on the other eight by 0.001 to 0.043, and the two BEIR-13 means differ by 0.005. With single runs and no variance estimates (Section 4.1), we do not read this as an advantage in either direction. A system-level comparison would require retraining the baselines under our recipe, which the backend makes possible but which is outside the scope of this paper.

5 Conclusion and Future Work

We present Tevatron 3.0, which integrates a Megatron-Core training backend into the Tevatron reranker toolkit while preserving its existing data processing, evaluation pipeline, and Hugging Face checkpoint interoperability. Profiling the existing PyTorch FSDP1 path demonstrates that it is memory-safe but throughput-bound and lacks expert parallelism. The Megatron backend matches FSDP1 reranker quality and, at matched data-parallel topology, its execution efficiency (while reducing epoch time by ~22% in its recommended configuration). Crucially, expert parallelism makes 30B-scale MoE reranker training feasible on a two-node academic budget. Our controlled study demonstrates that a 30B-A3B MoE reranker matches dense-8B retrieval effectiveness with fewer than half the activated parameters and substantially higher serving throughput,

LoRA recovers most full-parameter quality, and distillation helps on some datasets but lowers the BEIR-15 mean under our fixed recipe. We release all framework code and trained checkpoints as reusable artifacts for the community to continue building on Tevatron 3.0.

In the broader IR ecosystem, Tevatron 3.0 addresses a persistent infrastructure gap. While established neural retrieval toolkits (such as OpenMatch [11] and FlagEmbedding [25]) support standard bi-encoder and cross-encoder pipelines, they rely on trainers that lack native expert parallelism. Concurrently, recent work has begun exploring sparse MoE architectures for dense retrieval embeddings [16, 20]. Tevatron 3.0 complements this line of research by providing a reliable and robust open-source, expert-parallel training and serving pipeline specifically engineered for generative MoE cross-encoders with backward compatibility.

Several directions remain open. The distillation comparison calls for experiments varying temperature, group size, and — most promisingly — teacher capacity, since our 8B teacher leaves no clear capacity headroom to distill into. Broader architecture and scale coverage, and extending the backend’s pipeline and context parallelism to longer-context reranking, are natural next steps. Three limitations of the present study also mark out next steps. First, the compute we call an academic budget, one 8-GPU node for the dense runs and two for the MoE run, is still beyond many groups; tensor and pipeline parallelism, activation recompute, and LoRA are the backend’s levers for smaller or cheaper allocations, but we have not profiled them on GPUs with less memory than an H200, and doing so would give practitioners concrete recipes for 80 GB and smaller cards. Second, the systems comparison covers only the Hugging Face Trainer path of Tevatron 2.0, namely FSDP1 and, for LoRA, DDP; FSDP2, DeepSpeed ZeRO with its recently merged gradient-accumulation fix, and other Megatron-based training frameworks are not compared. Third, every result is a single run, and the comparisons in this paper should be repeated over several seeds before small differences are acted upon.

References

- [1] Sebastian Bruch, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2019. An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval* (Santa Clara, CA, USA) (ICTIR '19). Association for Computing Machinery, New York, NY, USA, 75–78. doi:10.1145/3341981.3344221
- [2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to Rank Using Gradient Descent. In *Proceedings of the 22nd International Conference on Machine Learning*. 89–96.
- [3] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to Rank: From Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning*. 129–136.
- [4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Tamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. doi:10.18653/v1/N19-1423
- [5] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Tevatron: An Efficient and Flexible Toolkit for Neural Retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) (SIGIR '23). Association for Computing Machinery, New York, NY, USA, 3120–3124. doi:10.1145/3539618.3591805
- [6] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531 [stat.ML] <https://arxiv.org/abs/1503.02531>
- [7] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2021. Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation. arXiv:2010.02666 [cs.IR] <https://arxiv.org/abs/2010.02666>
- [8] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL] <https://arxiv.org/abs/2106.09685>
- [9] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) (SOSP '23). Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165
- [10] Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. 2024. SPLADE-v3: New Baselines for SPLADE. arXiv:2403.06789 [cs.IR] <https://arxiv.org/abs/2403.06789>
- [11] Zhenghao Liu, Kaitao Zhang, Chenyan Xiong, Zhiyuan Liu, and Maosong Sun. 2021. OpenMatch: An Open Source Library for Neu-IR Research. arXiv:2102.00166 [cs.IR] <https://arxiv.org/abs/2102.00166>
- [12] Xueguang Ma, Luyu Gao, Shengyao Zhuang, Jiaqi Samantha Zhan, Jamie Callan, and Jimmy Lin. 2025. Tevatron 2.0: Unified Document Retrieval Toolkit across Scale, Language, and Modality. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Padua, Italy) (SIGIR '25). Association for Computing Machinery, New York, NY, USA, 4061–4065. doi:10.1145/3726302.3730135
- [13] Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2024. Fine-Tuning LLaMA for Multi-Stage Text Retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2421–2425.
- [14] Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. 2025. Generative Representational Instruction Tuning. In *International Conference on Learning Representations*. https://proceedings.iclr.cc/paper_files/paper/2025/hash/70cf215430492f7d34830a24e744b3f1-Abstract-Conference.html
- [15] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2023. MTEB: Massive Text Embedding Benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, Andreas Vlachos and Isabelle Augenstein (Eds.). Association for Computational Linguistics, Dubrovnik, Croatia, 2014–2037. doi:10.18653/v1/2023.eacl-main.148
- [16] Zach Nussbaum and Brandon Duderstadt. 2025. Training Sparse Mixture Of Experts Text Embedding Models. arXiv:2502.07972 [cs.CL] <https://arxiv.org/abs/2502.07972>
- [17] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) (SC '20). IEEE Press, Article 20, 16 pages.
- [18] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of the Third Text Retrieval Conference (NIST Special Publication, Vol. 500-225)*, Donna K. Harman (Ed.). National Institute of Standards and Technology, Gaithersburg, MD, USA, 109–126. doi:10.6028/NIST.SP.500-225.routing-city
- [19] Mohammad Shoeibi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL] <https://arxiv.org/abs/1909.08053>
- [20] Anubhav Shrestha, Safal Shrestha, Minwu Kim, Torsten Suel, and Keith Ross. 2026. Mixture-of-Experts Language Models Can Be Strong and Efficient Retrievers. arXiv:2609.13486 [cs.IR] <https://arxiv.org/abs/2609.13486>
- [21] Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han-yu Wang, Haisu Liu, Quan Shi, Zachary S. Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Serkan O. Arik, Danqi Chen, and Tao Yu. 2025. BRIGHT: A Realistic and Challenging Benchmark for Reasoning-Intensive Retrieval. In *The Thirteenth International Conference on Learning Representations*. <https://openreview.net/forum?id=ykuc5q381b>
- [22] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. <https://openreview.net/forum?id=wCu6T5xFje>
- [23] Nandan Thakur, Crystina Zhang, Xueguang Ma, and Jimmy Lin. 2025. Hard Negatives, Hard Lessons: Revisiting Training Data Quality for Robust Information Retrieval with LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 9064–9083. doi:10.18653/v1/2025.findings-emnlp.481
- [24] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text Embeddings by Weakly-Supervised Contrastive Pre-training. arXiv:2212.03533 [cs.CL] <https://arxiv.org/abs/2212.03533>
- [25] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-Pack: Packaged Resources To Advance General Chinese Embedding. arXiv:2309.07597 [cs.CL] <https://arxiv.org/abs/2309.07597>
- [26] Zhichao Xu, Aosong Feng, Yijun Tian, Haibo Ding, and Lin Lee Cheong. 2025. CSPLADE: Learned Sparse Retrieval with Causal Language Models. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, Kentaro Inui, Sakriani Sakti, Haoen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dharendra Pratap Singh (Eds.). The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, Mumbai, India, 99–114. doi:10.18653/v1/2025.ijcnlp-long.7
- [27] Zhichao Xu, Zhiqi Huang, Shengyao Zhuang, and Vivek Srikumar. 2025. Distillation versus Contrastive Learning: How to Train Your Rerankers. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, Kentaro Inui, Sakriani Sakti, Haoen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dharendra Pratap Singh (Eds.). The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, Mumbai, India, 564–578. doi:10.18653/v1/2025.findings-ijcnlp.33
- [28] Zhichao Xu, Fengran Mo, Zhiqi Huang, Crystina Zhang, Puxuan Yu, Bei Wang Phillips, Jimmy Lin, and Vivek Srikumar. 2026. A Survey of Model Architectures in Information Retrieval. *Transactions on Machine Learning Research* (2026). Survey Certification. <https://openreview.net/forum?id=xAlBtBHRrX>
- [29] Zhichao Xu, Shengyao Zhuang, Crystina Zhang, Xueguang Ma, Yijun Tian, Maitrey Mehta, Jimmy Lin, and Vivek Srikumar. 2026. LACONIC: Dense-Level Effectiveness for Scalable Sparse Retrieval via a Two-Phase Training Curriculum. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Australia) (SIGIR '26). Association for Computing Machinery, New York, NY, USA, 4298–4304. doi:10.1145/3805712.3809869
- [30] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Tang, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuyuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 Technical Report. arXiv:2505.09388 [cs.CL] <https://arxiv.org/abs/2505.09388>
- [31] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. arXiv:2506.05176 [cs.CL] <https://arxiv.org/abs/2506.05176>
- [32] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balıoğlu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao,

- Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. arXiv:2304.11277 [cs.DC] <https://arxiv.org/abs/2304.11277>
- [33] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=VqkAKQibpq>
- [34] Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2023. RankT5: Fine-Tuning T5 for Text Ranking with Ranking Losses. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) (SIGIR '23). Association for Computing Machinery, New York, NY, USA, 2308–2313. doi:10.1145/3539618.3592047
- [35] Shengyao Zhuang, Zhichao Xu, and Ivano Lauriola. 2026. Layer-wise Token Compression for Efficient Document Reranking. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Australia) (SIGIR '26). Association for Computing Machinery, New York, NY, USA, 4426–4432. doi:10.1145/3805712.3809871