DRIFTRETRIEVAL: LEARNING FROM MANY NEGATIVES WITH LESS BACKPROPAGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large contrastive batches give retrieval models many documents to learn from, but training through every query and document is expensive. We introduce DriftRetrieval, which separates comparing embeddings from updating them. For each query, it constructs a temporary target that moves toward the matching document and away from high-scoring alternatives. The model learns to reach this target with a squared-error loss. Successive steps alternate between updating queries and their matching documents; the remaining embeddings help determine the targets but receive no gradient on that step. Processing small groups of examples at a time keeps GPU working memory bounded as the batch grows. DriftRetrieval reaches 41.66 retrieval on the full 56-task MTEB benchmark, versus 38.81 for a strong InfoNCE baseline that also computes fewer encoder gradients. Across three training seeds on a 14-task subset, the retrieval gain is 3.44 ± 0.78 points (mean and sample standard deviation), while overall scores are similar. In controlled synthetic tests at 32K and 128K pairs, optimized DriftRetrieval achieves about 15% higher throughput than optimized sampled-gradient InfoNCE.

1 INTRODUCTION

Embedding models are commonly trained with contrastive losses such as InfoNCE (van den Oord et al., 2018). In-batch retrieval implementations compare each query with its positive and the other candidate documents in the batch. Queries and documents are embedded as vectors, and backpropagation pulls positives closer together and pushes negatives apart. Larger distributed batches provide more in-batch negatives, but materializing their query–document score matrix can become a substantial memory bottleneck. Encoder activations and backpropagation also contribute to training cost (Gao et al., 2021; Cheng et al., 2025); which component dominates depends on the backbone, input lengths, batch size, and implementation.

Prior work reduces the memory cost of this all-to-all comparison through encoder replay and tiled score computation (Gao et al., 2021; Cheng et al., 2025; Chen et al., 2023), which we discuss in Section 6 and compare against directly in Section 5. These methods preserve full-gradient training, but do not determine whether differentiating every encoding gives the best retrieval quality for its cost.

We propose DriftRetrieval, a highly scalable alternative training method for retrieval models with low memory and backpropagation costs. Our approach adopts drifting models, which were first proposed as a training method in the unrelated field of generative image modeling (Deng et al., 2026), as a low-memory alternative to InfoNCE. In this paradigm, each query and document is again embedded as a vector, but each query is instead assigned a static *target*: a point that is closer to its positive examples and further from the mean of the negative examples. A least-squares loss then measures the distance between the query vector and this target, and a gradient step moves the embedding toward it. Unlike standard contrastive losses, this approach relies on an iid loss computed separately on each vector, without distributed gradient computations in which vectors talk to one another. Information is exchanged between workers when targets are computed, but no gradients need to be traced through these computations. Under matching conditions, this reproduces the query-side InfoNCE parameter gradient up to a positive scalar (Eq. 5).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

The central choice is which embeddings to update. On one step we train queries toward targets computed from document embeddings; on the next, we train the matching documents toward targets computed from queries. Each target is fixed only for the current step and is recomputed as the model learns. The targets depend on the batch, but, once constructed, each squared-error term can be differentiated separately. This lets us process small groups of examples without retaining the entire batch's encoder activations. The method still exchanges information across GPUs to construct targets and train the shared model.

Skipping some encoder gradients is also possible with InfoNCE, so our strongest control does exactly that. This *sampled-gradient InfoNCE* baseline compares against every candidate, but backpropagates through only a subset of query and document encodings. We match its average number of encodings receiving gradients to Drift's. On MEDI2 (Muennighoff et al., 2025), Drift improves retrieval on 13 of 15 MTEB retrieval datasets over this control. The retrieval gain repeats across three training seeds on a fixed 14-task subset.

We evaluate efficiency separately from quality. At the MEDI2 batch size, Drift uses less training-loop time than full-gradient InfoNCE but more than direct sampled-gradient InfoNCE. At much larger synthetic batches, where both methods process examples in small groups to limit memory, optimized Drift achieves about 15% higher throughput than the optimized sampled baseline. These results motivate alternating updates as a useful quality–cost trade-off.

Contributions.

- A retrieval training method that builds temporary embedding targets and alternates query and document updates (Section 3).
- A bounded-GPU-memory implementation that backpropagates only through the selected side of each training batch (Section 4).
- Controlled quality and efficiency comparisons against both full and sampled-gradient InfoNCE, including settings where InfoNCE remains stronger (Section 5).

2 Background and Motivation

2.1 Contrastive retrieval with InfoNCE

An encoder f_θ maps a query or document to a unit-length vector with D coordinates. A batch contains B query embeddings $q_i = f_\theta(x_i)$ and K document embeddings $p_j = f_\theta(y_j)$. For query i, the matching document p_i^+ is the *positive*; other documents in the batch serve as negatives. The dot product $q_i^\top p_j$ measures similarity. Directional InfoNCE (van den Oord et al., 2018) encourages the positive to score highly relative to all K candidates:

$$\mathcal{L}_{\mathrm{NCE}} = -\frac{1}{B}\sum_{i=1}^{B} \log \frac{\exp(q_i^\top p_i^+/T)}{\sum_{j=1}^{K}\exp(q_i^\top p_j/T)}. \tag{1}$$

The temperature T controls how strongly high-scoring documents dominate the comparison. Candidates can come from any GPU in the training job. For example, our main experiment has 2,048 queries and 4,096 documents: one positive and one explicit hard negative per query. Every query also uses the other queries' documents as negatives. Full-gradient InfoNCE backpropagates through all query and document encodings.

2.2 Comparing an example need not mean updating it

A direct implementation stores both encoder activations for backpropagation and a $B \times K$ matrix of similarity scores. The relative cost of these components depends on batch size, input length, and encoder size. Existing methods reduce both memory costs without changing InfoNCE: they process scores in small blocks and recompute encoder activations for small groups of examples (Gao et al., 2021; Cheng et al., 2025). Full-gradient training still computes an encoder gradient for every example.

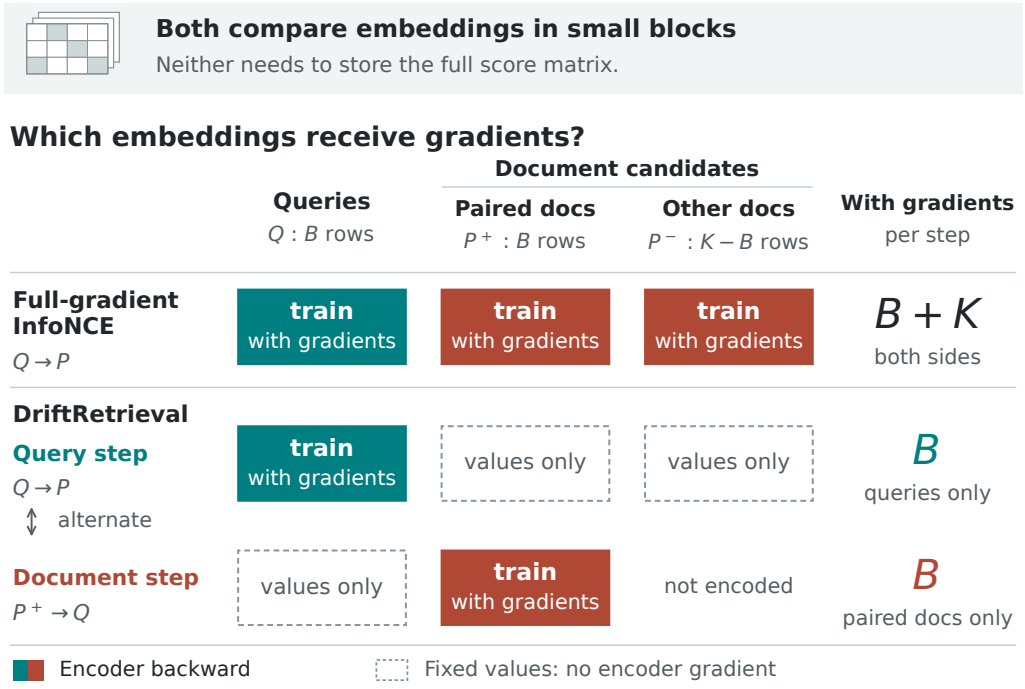


Figure 1: **Use the batch to choose targets; update one side at a time.** Full-gradient InfoNCE backpropagates through every query and document. DriftRetrieval alternates: documents guide query updates on one step, and queries guide matching-document updates on the next. Filled blocks receive encoder gradients; dashed blocks supply embedding values only. With B queries and K documents, Drift differentiates B encodings per step rather than $B + K$. This changes the training update, not just its implementation. Both methods can compute comparisons in small blocks instead of storing a full score matrix.

We ask a different question: can an example help define a useful training update without receiving a gradient itself? Drifting Models (Deng et al., 2026) suggest a simple construction. They compute a direction in which a model output should move, place a temporary target in that direction, and train the output toward it. We adapt this idea to retrieval: use the documents to choose a target for each query, then learn to reach that target while treating the documents as fixed values.

3 Drift Contrastive Learning

3.1 Choose a target for each query

Consider a query whose positive document scores lower than several incorrect documents. We want to move its embedding toward the positive and away from those competing documents. A uniform average of the batch would give irrelevant, low-scoring documents just as much influence as the difficult negatives. Instead, we weight candidates by the same softmax probabilities used in InfoNCE:

$$\pi_{ij} = \frac{\exp(q_i^\top p_j/T)}{\sum_k \exp(q_i^\top p_k/T)}, \qquad \mu_i = \sum_j \pi_{ij} p_j, \qquad F_i = p_i^+ - \mu_i. \tag{2}$$

The weighted average μ_i summarizes the documents that currently score highly for query i. The difference $F_i = p_i^+ - \mu_i$ provides the proposed update direction, or *drift field*. The positive is included in this average. If it already dominates the softmax, then μ_i is close to p_i^+ and the update becomes small. If an incorrect document dominates, the update points away from it and toward the positive.

We place a temporary target a short distance along this direction. Because embeddings have unit length, we first remove the component parallel to q_i, which would change length rather than direction:

$\bar{F}_i = F_i - q_i(q_i^\top F_i)$. The target is $z_i = \mathrm{sg}(q_i + \alpha \bar{F}_i)$, where α sets the displacement and sg means *stop gradient*: treat the target as a constant during backpropagation. We then minimize

$$\mathcal{L}_q = \frac{s}{BD} \sum_i \|q_i - z_i\|_2^2. \tag{3}$$

Here s scales the loss, and BD averages it over examples and embedding coordinates. Backpropagation moves the query toward the target; it does not change the target to make the loss smaller. We recompute targets after each model update. The batch therefore determines where to move, but gradients need not pass through that decision.

The query-step code below uses normalized embeddings. Each entry in `positive_idx` selects the matching document for that query. The operation `detach()` implements stop gradient.

```
q = encoder(query_tokens)              # (B, D), with gradients
with no_grad():
    docs = encoder(candidate_tokens)   # (K, D), fixed values
    q_fixed = q.detach()
    weights = softmax(q_fixed @ docs.T / T, dim=1)  # (B, K)
    center = weights @ docs            # (B, D)
    field = docs[positive_idx] - center
    radial = (field * q_fixed).sum(dim=1, keepdim=True)
    field = field - radial * q_fixed
    target = q_fixed + alpha * field
loss = s * ((q - target) ** 2).mean()
loss.backward()
```

This simple version stores all $B \times K$ weights. Section 4 shows how to compute the same update without that storage. Its loss value is a distance to a moving target, not a measure of retrieval accuracy.

Relation to InfoNCE. At the same embeddings, temperature, candidates, and random realization, this query update gives the query-side InfoNCE parameter gradient up to a positive scalar (Eq. 5). It does not include InfoNCE's document-side gradient. We call this formulation *Raw Drift*: it retains the magnitude of the proposed update. *Normalized Drift* instead divides every update by the same batch-level RMS magnitude. Both variants normalize embeddings to unit length; their difference is update scale, not embedding normalization (Appendix E).

3.2 Alternate query and document updates

Queries and documents have complementary roles in retrieval. To train the document side, we swap the roles in the construction above: each positive document moves toward its matching query and away from a weighted average of all queries. For positive document p_j^+, this *reverse* direction is

$$\rho_{ji} = \frac{\exp((p_j^+)^\top q_i / T)}{\sum_k \exp((p_j^+)^\top q_k / T)}, \qquad F_j^{\mathrm{rev}} = q_j - \sum_i \rho_{ji} q_i. \tag{4}$$

We apply the same length correction and squared-error loss, now keeping query embeddings fixed. This is a new document update, not the missing document gradient of directional InfoNCE: it asks which query matches each positive document, rather than accumulating each query's feedback to every candidate document. Appendix A makes the distinction explicit.

Our default schedule uses one direction on each optimizer step:

```
even step: build targets from documents; train queries toward them
odd step:  build targets from queries; train positive documents
```

Thus a batch with 2,048 queries and 4,096 documents requires encoder gradients for 2,048 examples per Drift step, compared with 6,144 for full-gradient InfoNCE. Explicit hard negatives guide query targets but receive no encoder gradient in either direction. Query and document encodings share model parameters, so both can change after every update; we are not freezing a separate document encoder. Alternating steps also differ from computing both losses together: the model and batch can change between steps. We evaluate that scheduling choice empirically.

4 FIXED-MEMORY TRAINING

The simple code above still stores a large score matrix and the activations for every query. We remove these two memory costs separately.

Compute the average without storing every score. For each small group of queries, we visit the documents in small blocks. After a block has contributed to the weighted average in Eq. 2, its scores can be discarded. We retain only a running weighted sum and its total weight, with a rescaling step to keep the exponentials numerically stable. Every candidate still contributes; this is neither negative sampling nor an approximate softmax. Appendix E gives the recurrence and tensor sizes. Exact InfoNCE can use the same technique.

For illustration, 2,048 queries and 4,096 documents give 8,388,608 scores in the full matrix. Blocks of 128 queries and 512 documents need only 65,536 scores at a time. Each query keeps one running maximum, one total weight, and one D-dimensional weighted sum. After all eight document blocks, dividing the sum by the weight gives the full-matrix center, up to rounding. Normalization is over all 4,096 candidates, not independently over each block of 512. More candidates require more blocks, not a wider score tile. All 16 query groups contribute before one optimizer step: the processing-block size limits memory without reducing the training batch or its negative pool.

Backpropagate through small groups of examples. Our first implementation follows Gradient Cache (Gao et al., 2021). It encodes the batch without retaining layer activations, stores embeddings in CPU memory, and computes the target directions. It then re-encodes a small group of queries with gradients, backpropagates their updates, and releases the activations before processing the next group. This second encoding is what we call *replay*. Only the selected side is replayed: queries on one step, matching documents on the next. Parameter gradients accumulate across groups before a single optimizer update.

An optimized version encodes the fixed candidates first. Each selected query can then be encoded once with gradients, assigned its target, and backpropagated immediately. This avoids a separate no-gradient encoding of that query. The same procedure applies with roles reversed on a document step. We call it *candidate-first execution*; it changes the order of computation, not the Drift update. Section 5 compares it with an optimized candidate-first sampled-InfoNCE baseline.

At fixed model, GPU count, and group sizes, these procedures bound the GPU storage for activations and scores as the training batch grows. They do not make total resource use constant. CPU embedding storage and encoding work grow with batch size, and comparing B queries with K documents still requires BK dot products. Distributed training also exchanges candidate summaries and model gradients. Detailed memory accounting, random-state handling, and communication are in Appendix H.

5 EXPERIMENTS

We evaluate retrieval quality, the benefit of additional candidates, and training cost. Real-data experiments measure checkpoint quality; synthetic scaling experiments measure execution cost only.

5.1 EXPERIMENTAL PROTOCOL

We use GritLM's public MEDI2 release of 1,208,406 examples (Muennighoff et al., 2025), with same-source batches. Unless stated otherwise, runs use Qwen2.5-0.5B on 32 MI300A GPUs for 1,253 updates of 2,048 queries: 2,566,144 query presentations. Each query has one positive and one explicit negative, giving 4,096 documents. We denote documents per query by G: $G = 2$ here and $G = 1$ for positive pairs only. Query/document limits are 256/2,048 tokens. InfoNCE uses $T = 0.02$; Raw Drift uses $T = 0.05$ and $s = 1$.

We use bidirectional attention, bfloat16, AdamW at 2×10^{-5}, 3% warmup, clipping at 1, gradient checkpointing, and FSDP (Appendix C). Evaluation uses the English 56-task MTEB suite (Muennighoff et al., 2023) or a fixed 14-task subset for smaller controls. Training seed is 42 unless specified otherwise.

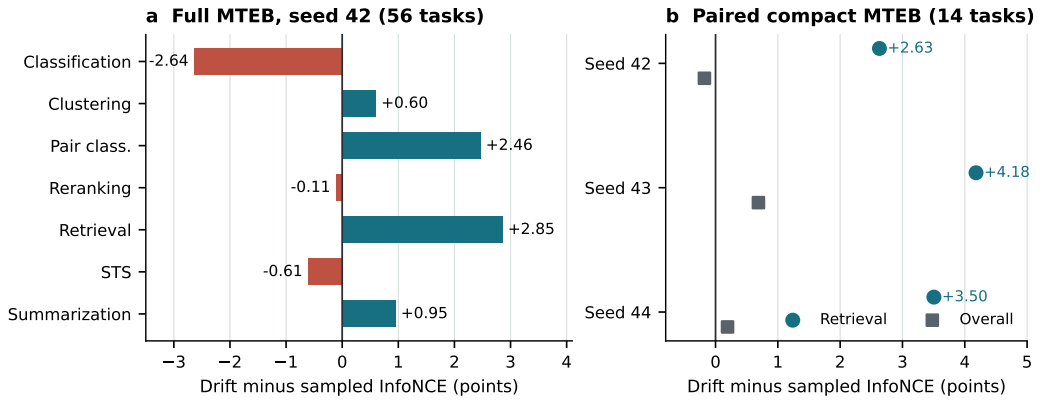


Figure 2: **Drift minus sampled InfoNCE. a:** Full 56-task MTEB differences at seed 42: retrieval improves, but classification and STS regress. **b:** Independent paired training seeds on the 14-task subset: retrieval improves in every seed, with overall quality near a tie. The panels use different task suites.

5.2 Retrieval quality and training cost

Table 1 matches data, example count, model, hardware, and sequence limits. Full-gradient InfoNCE differentiates every encoding. Sampled InfoNCE compares all candidates but differentiates half the queries and a quarter of the documents: 2,048 encodings, matching Drift. Equal counts do not imply equal compute when input lengths differ.

Training-loop time is the Trainer measurement; *application time* also includes initialization, saving, and exit, but excludes queue wait. HBM is peak PyTorch allocation, maximized across GPUs. *Direct* execution retains selected encoder activations; *streamed* execution uses small blocks and CPU-resident embeddings.

Table 1: **MEDI2 retrieval quality and training cost.** Qwen2.5-0.5B, 32 MI300A GPUs, seed 42, 1,253 updates, and full 56-task MTEB. Training examples are matched; temperatures and execution differ. Application time includes startup, saving, and exit.

Objective	Backpropagation through	Execution	HBM (GiB/GPU)	Loop (min)	Application (min)	MTEB overall	Retrieval
Full-gradient InfoNCE	All queries + all documents	Streamed	4.049	65.9	78.1	53.15	33.77
Sampled InfoNCE	50% queries + 25% documents	Direct	4.828	**16.1**	**30.8**	55.19	38.81
Raw Drift	Queries or paired documents	Streamed replay	**4.035**	30.5	47.5	**55.54**	**41.66**

Drift improves retrieval over full-gradient InfoNCE with 54% less loop time at similar memory. Sampled InfoNCE is stronger and faster: Drift gains 2.85 retrieval points (13 of 15 datasets) and 0.35 overall but loses 2.64 classification points (Figure 2). This is a retrieval-quality gain, not a speed win over every baseline.

Seed replication and pair-only control. On the 14-task subset, paired seed-42/43/44 differences average $+0.23 \pm 0.44$ overall and $+3.44 \pm 0.78$ retrieval (sample standard deviations). Retrieval improves in every seed, unlike overall quality. Without explicit hard negatives ($G = 1$), Drift still improves seed-42 retrieval over sampled directional InfoNCE, 42.47 versus 41.46, at approximately tied overall scores, 56.62 versus 56.71.

Does access to more candidates help? We keep 2,048 positive pairs ($G = 1$) per batch but expose each query to 256 or all 2,048 documents. Within each method, examples, steps, and gradient-sampling rates stay fixed; only candidate visibility changes, without speeding up the smaller-pool computation.

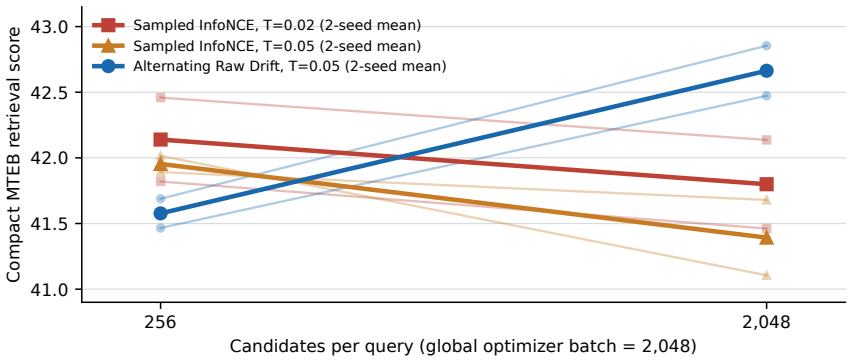


Figure 3: **Candidate coverage at fixed optimizer batch.** Retrieval averages four MTEB-14 tasks. Thin lines show independent training seeds; thick lines show their means. Only candidate visibility changes, not encoded examples. InfoNCE uses $T = 0.02$ or 0.05; Drift uses $T = 0.05$.

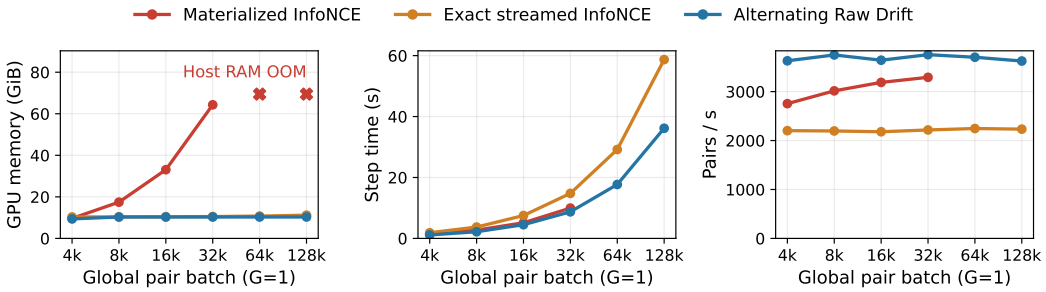


Figure 4: **Larger batches without retaining a larger score matrix.** Synthetic positive pairs, Qwen2.5-0.5B, 32 MI300A GPUs. Memory is peak PyTorch allocation per GPU. Both streamed methods reach 128K pairs; Drift does less backward work than full-gradient InfoNCE. Red crosses mark *host-memory* failures of materialized InfoNCE at 64K and 128K; their height is not a measured GPU allocation. These short runs measure systems capacity, not downstream quality.

Drift gains about one retrieval point in both seeds, while sampled InfoNCE declines at both temperatures (Figure 3; Appendix Table 7). InfoNCE leads at 256 candidates, Drift at 2,048. All four tasks improve for Drift in both seeds; excluding ArguAna, the largest contributor, leaves gains of 0.42 and 0.67 points (Appendix F).

Other model families. Direct Raw Drift and no-grad-gather InfoNCE (the GritLM setting; Muennighoff et al., 2025) also have full 56-task results on Mistral-7B, Qwen2.5-0.5B, and Gemma-3-1B. Drift is close on Mistral and Gemma and stronger on Qwen. These supporting comparisons do not include sampled InfoNCE or establish fixed-memory training on every model (Appendix D.1).

5.3 Memory and throughput scaling

We scale from 4,096 to 131,072 positive pairs per update on 32 MI300A GPUs, fixing model, query/document lengths (64/512), and block sizes. Each synthetic run takes nine steps; timing excludes the first two. Figure 4 compares materialized InfoNCE, exact streamed InfoNCE, and alternating Drift with replay. Materialized InfoNCE uses no-grad gather: remote embeddings have no autograd connection through the gather. This is distinct from sampling local encoder gradients.

Materialized InfoNCE reaches 64.31 GiB at 32K pairs and fails at 64K. At 128K, streamed InfoNCE and Drift use 11.20 and 10.31 GiB and take 58.70 and 36.16 seconds per step, respectively. Drift's $1.62\times$ throughput reflects less backward work, not exclusive access to blockwise scores. This InfoNCE implementation also retains an $O(BD)$ query cache (Appendix H; Table 6).

Table 2: **Optimized paired execution.** Qwen2.5-0.5B, 32 MI300A GPUs, synthetic $G = 1$, q64/p512. Each allocation runs two fresh-process repeats per method in reversed order. Times average six steady steps; HBM is maximum rank allocation.

Pairs	InfoNCE s	Drift s	Throughput gain	HBM GiB (I/D)
32,768	8.573	7.472	14.7%	10.423 / 10.306
131,072	35.547	30.954	14.8%	10.772 / 10.306

Comparison with optimized sampled-gradient InfoNCE. Sampled InfoNCE differentiates half the queries and documents, matching Drift's average count for $G = 1$. Initial replay implementations tie in speed. We optimize both with candidate-first execution and run each twice on the same eight nodes, reversing method order. Table 2 shows about 15% higher Drift throughput at similar memory; each window covers three query and three document updates. This matches hardware and gradient counts, not learning rules: sampled InfoNCE uses selected gradients of its complete loss, whereas Drift uses two target-based updates. Large-batch downstream quality remains untested. Appendix H.3 reports implementation variants and synchronization checks.

5.4 Ablations and correctness

Tests compare full-matrix and blockwise fields, direct and replayed gradients, and single- and multi-GPU updates. They verify the query-side InfoNCE identity and the distinct reverse update. Direct and streamed training give nearly identical MTEB-14 scores: 55.38 versus 55.31 for InfoNCE and 56.40 versus 56.42 for alternating Drift (Table 12).

Alternation is a separate choice. On the same 32-worker setup, it takes 704 loop seconds versus 1,143 for updating both directions, at 8.20 versus 8.40 GiB. Their overall/retrieval scores are 56.40/42.32 and 56.68/42.46. Startup delays reverse the application-time ordering, so this supports lower loop cost, not a shorter total job. Another matched sampled-InfoNCE control ties sparse Drift at comparable cost (56.88 versus 56.85 overall): partial backpropagation alone does not establish a Drift advantage.

6 Related Work

Dense retrieval and language-model embeddings. DPR (Karpukhin et al., 2020) and Sentence-BERT (Reimers & Gurevych, 2019) established dual-encoder retrieval and sentence representations. Contriever, E5, BGE, and Nomic Embed advance contrastive pretraining through data scale and supervision (Izacard et al., 2022; Wang et al., 2022; Xiao et al., 2024; Nussbaum et al., 2024). Instructor introduces instruction conditioning (Su et al., 2023); E5-Mistral, NV-Embed, and GritLM adapt decoder language models (Wang et al., 2024; Lee et al., 2025; Muennighoff et al., 2025). Our bidirectional backbone follows LLM2Vec and NV-Embed (BehnamGhader et al., 2024; Lee et al., 2025). DriftRetrieval instead focuses on which encodings receive gradients and their execution cost.

Memory-efficient contrastive training. Gradient Cache (Gao et al., 2021) separates representation gradients from encoder backpropagation, bounding activation memory by replaying small groups. Inf-CL and DisCo-CLIP reduce loss memory through tiling and distributed gradient decomposition (Cheng et al., 2025; Chen et al., 2023). Our implementation uses replay and online softmax reduction (Milakov & Gimelshein, 2018; Dao et al., 2022). These techniques also permit exact InfoNCE without a full score matrix. Our distinction is the update rule and selective encoder work, not tiling alone; we therefore compare against both exact streamed and sampled-gradient InfoNCE.

Contrastive gradients and detached targets. InfoNCE's attraction–repulsion structure motivates hardness weighting and decoupled contrastive learning (Wang & Liu, 2021; Yeh et al., 2022). Tao et al. (2022) analyze contrastive, asymmetric, and decorrelation methods through representation gradients; their UniGrad method uses a moving-average correlation matrix. Raw Drift retains the query-side InfoNCE derivative up to scale (Eq. 5), but pairs it with a distinct reverse field normalized over queries.

BYOL and SimSiam use detached targets (Grill et al., 2020; Chen & He, 2021); mean-shift self-supervision uses neighboring representations (Koohpayegani et al., 2021), and retrieval distillation uses teacher scores or margins (Hofstätter et al., 2020; Ren et al., 2021). Drift constructs supervised displacements from the current batch, without a separate teacher, predictor, or momentum encoder. Its stopped-branch cross-entropy form (Eq. 6) yields the same Raw Drift directions: squared-error regression is not essential to the update.

Drifting fields. Drifting Models (Deng et al., 2026) motivate our temporary targets; attraction and repulsion also appear in particle methods such as Stein variational gradient descent (Liu & Wang, 2016). We specialize this construction to retrieval: the positive supplies attraction and the softmax-weighted candidate average supplies repulsion. This gives both an interpretable update and the summaries used in blockwise execution.

Alternating updates and partial gradients. DAFT initially freezes the document encoder and also tests alternating encoder updates per batch, finding that they work but train more slowly (Leonhardt et al., 2022). Sciavolino (2021) studies a fixed DPR tower. ConRetroBert tests alternating freezing in reaction-template retrieval, where it underperforms the local EMA-stabilized configuration (Basher et al., 2026). Drift instead shares encoder parameters: all representations can change after each update, but only query or paired-document encodings receive gradients. Its reverse field differs from InfoNCE's document derivative, and unpaired negatives receive no gradient. Block-coordinate guarantees (Beck & Tetruashvili, 2013) do not directly apply to these shared parameters or alternating Adam steps. Our sampled-InfoNCE controls test whether gains extend beyond reducing gradient coverage alone.

Detached negatives and batch size. Momentum queues, cross-batch memory, refreshed indices, and negative caches expand candidate sets through stored representations (He et al., 2020; Wang et al., 2020; Xiong et al., 2021; Lindgren et al., 2021). Drift recomputes candidates at current parameters, avoiding cross-step staleness. RocketQA and TAS-B emphasize negative quality and batch composition (Qu et al., 2021; Hofstätter et al., 2021); analyses of noisy contrastive learning explain why more negatives need not help (Wu et al., 2022). We accordingly distinguish systems capacity at 128K pairs from quality on MEDI2 and weakly supervised data.

7 Limitations

CPU storage, encoding work, and pairwise comparisons grow with batch size; communication depends on sharding and block sizes. Our synthetic 128K-pair tests establish capacity, not downstream quality. Direct sampled InfoNCE remains faster at the measured MEDI2 batch size.

Our Nomic subset experiments favor InfoNCE, with no demonstrated Drift quality or time-to-quality win (Appendix G); their held-out pairs guided selection and constitute validation data. The appendix's CUDA study uses official Inf-CL score computation but changes encoder-gradient coverage. It neither establishes image–text quality for detached Drift nor reproduces Inf-CL's LAION training. We make no general speed or memory superiority claim over Inf-CL.

8 Conclusion

DriftRetrieval uses all candidates to choose embedding targets, then updates queries and matching documents on alternating steps. Blockwise execution bounds GPU working memory as batches grow. MEDI2 experiments show retrieval gains over full and sampled-gradient InfoNCE, while optimized large-batch tests show a throughput benefit. Nomic and direct sampled-InfoNCE comparisons show that these advantages depend on the training data and execution setting. The central practical question is which query and document updates are worth their cost, and how that choice depends on the training data.

AI USE STATEMENT

We used generative AI tools to improve the clarity and readability of the manuscript and to assist with debugging distributed training code. We reviewed all AI-assisted work and take responsibility for the final content of this paper, including its text, claims, and associated code.

REFERENCES

Mohammad Jahid Ibna Basher, Ali Khodabandeh Yalabadi, Ivan Garibay, and Ozlem Ozmen Garibay. ConRetroBert: EMA stabilized dual encoders for template-based single-step retrosynthesis. *arXiv preprint arXiv:2605.12736*, 2026.

Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013. doi: 10.1137/120887679.

Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. Llm2vec: Large language models are secretly powerful text encoders. In *First Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2404.05961.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15750–15758, 2021.

Yihao Chen, Xianbiao Qi, Jianan Wang, and Lei Zhang. DisCo-CLIP: A distributed contrastive loss for memory efficient CLIP training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22648–22657, 2023.

Zesen Cheng, Hang Zhang, Kehan Li, Sicong Leng, Zhiqiang Hu, Fei Wu, Deli Zhao, Xin Li, and Lidong Bing. Breaking the memory barrier of contrastive loss via tile-based strategy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10036–10045, 2025.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Mingyang Deng, He Li, Tianhong Li, Yilun Du, and Kaiming He. Generative modeling via drifting. *arXiv preprint arXiv:2602.04770*, 2026.

Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. Scaling deep contrastive learning batch size under memory limited setup. In *Proceedings of the 6th Workshop on Representation Learning for NLP*, pp. 316–321. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.repl4nlp-1.31.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21271–21284, 2020.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. Improving efficient neural ranking models with cross-architecture knowledge distillation. *arXiv preprint arXiv:2010.02666*, 2020.

Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. Efficiently teaching an effective dense retriever with balanced topic aware sampling. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2021. doi: 10.1145/3404835.3462891. URL https://arxiv.org/abs/2104.06967.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022. URL https://openreview.net/forum?id=jKN1pXi7b0.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 6769–6781. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.550.

Soroush Abbasi Koohpayegani, Ajinkya Tejankar, and Hamed Pirsiavash. Mean shift for self-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10326–10335, 2021.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nv-embed: Improved techniques for training llms as generalist embedding models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2405.17428.

Jurek Leonhardt, Marcel Jahnke, and Avishek Anand. Distribution-aligned fine-tuning for efficient neural retrieval. *arXiv preprint arXiv:2211.04942*, 2022.

Erik Lindgren, Sashank Reddi, Ruiqi Guo, and Sanjiv Kumar. Efficient training of retrieval models using negative cache. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/2175f8c5cd9604f6b1e576b252d4c86e-Abstract.html.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances in Neural Information Processing Systems*, volume 29, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/b3ba8f1bee1238a2f37603d90b58898d-Abstract.html.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. MTEB: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 2014–2037. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.eacl-main.148. URL https://aclanthology.org/2023.eacl-main.148/.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. In *International Conference on Learning Representations*, 2025.

Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic embed: Training a reproducible long context text embedder. *arXiv preprint arXiv:2402.01613*, 2024.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5835–5847. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.naacl-main.466. URL https://aclanthology.org/2021.naacl-main.466/.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019. doi: 10.18653/v1/D19-1410.

Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, QiaoQiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. RocketQAv2: A joint training method for dense passage retrieval and passage re-ranking. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2825–2835. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.224. URL https://aclanthology.org/2021.emnlp-main.224/.

Christopher Sciavolino. Towards universal dense retrieval for open-domain question answering. Master's thesis, Princeton University, 2021. arXiv preprint arXiv:2109.11085.

Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. One embedder, any task: Instruction-finetuned text embeddings. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 1102–1121. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.71. URL https://aclanthology.org/2023.findings-acl.71/.

Chenxin Tao, Honghui Wang, Xizhou Zhu, Jiahua Dong, Shiji Song, Gao Huang, and Jifeng Dai. Exploring the equivalence of siamese self-supervised learning via a unified gradient framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Feng Wang and Huaping Liu. Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2495–2504, 2021.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11897–11916. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.642. URL https://aclanthology.org/2024.acl-long.642/.

Xun Wang, Haozhi Zhang, Weilin Huang, and Matthew R. Scott. Cross-batch memory for embedding learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6388–6397, 2020.

Chuhan Wu, Fangzhao Wu, and Yongfeng Huang. Rethinking infonce: How many negative samples do you need? In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 2509–2515. International Joint Conferences on Artificial Intelligence Organization, 2022. doi: 10.24963/ijcai.2022/348. URL https://www.ijcai.org/proceedings/2022/0348.pdf.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. URL https://arxiv.org/abs/2309.07597.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=zeFrfgyZln.

Chun-Hsiao Yeh, Cheng-Yao Hong, Yen-Chi Hsu, Tyng-Luh Liu, Yubei Chen, and Yann LeCun. Decoupled contrastive learning. In *Computer Vision – ECCV 2022*, pp. 668–684. Springer, 2022.

A Derivations and Correctness

This appendix distinguishes two statements that are easy to conflate: the query update agrees with a query-only InfoNCE gradient up to scale, but our reverse document update is not the document gradient of that same loss. The identities below concern gradients before clipping or an optimizer step. They do not imply that two alternating Adam updates equal one simultaneous update.

For unit-normalized query q_i and candidate p_j, let $\pi_{ij} = \mathrm{softmax}_j(q_i^\top p_j/T)$. The Raw Drift query field is the tangent projection of $p_i^+ - \sum_j \pi_{ij} p_j$. With target step α, surrogate multiplier s, embedding dimension D, and mean-squared regression over all coordinates, its parameter gradient is

$$\nabla_\theta L_{\text{query-Drift}} = \frac{2s\alpha T}{D} \nabla_\theta L_{\text{InfoNCE}}(Q_\theta, \mathrm{sg}(P_\theta)). \tag{5}$$

This identity holds for the query direction only. Directional InfoNCE key descent uses $q_j - \sum_i \pi_{ij} q_i$, whereas reverse Drift uses $q_j - \sum_i \rho_{ji} q_i$ with ρ normalized across queries for each document. The fields generally differ. With equal temperature, target-step scale, surrogate multiplier, and directional averaging, simultaneous Raw Drift has the parameter-gradient direction of

$$\frac{1}{2}\left[\mathrm{CE}(Q_\theta, \mathrm{sg}(P_\theta)) + \mathrm{CE}(P_\theta^+, \mathrm{sg}(Q_\theta))\right]. \tag{6}$$

The second term treats paired positives as anchors and normalizes over queries. It is not the document-side derivative of the first term. The identity concerns two gradients at the same model state, not two successive optimizer updates.

Normalized Drift changes only the field scale: for the complete distributed directional batch, let

$$r = \sqrt{\max\left(\frac{1}{BD}\sum_{i,d} \bar{F}_{id}^2, \epsilon\right)} \quad \text{and use} \quad \bar{F}_i/r.$$

Every anchor receives the same batch-level multiplier $1/r$; this is not per-anchor unit normalization. The main systems method uses Raw Drift.

The repository tests both statements in float64 at the parameter-gradient level. It also compares dense and online-softmax fields, direct and replayed representation gradients, clipped AdamW updates, counts of query and document rows receiving gradients, and checkpoint-resume direction parity. A two-process Gloo regression checks distributed sufficient-statistic merging and the production global-coordinate gradient denominator. All checks used for this draft pass.

B Implementation Details

We use *anchor* for an embedding receiving a gradient on the current step and *candidate* for an embedding used to construct its target. These are roles, not separate models: queries are anchors on query steps and candidates on document steps. *Discovery* means the initial encoding without retained activations; *replay* means encoding the selected examples again with gradients.

The fixed-memory implementation records RNG state during detached discovery so dropout is reproduced when those encodings are replayed with gradients. Candidate embeddings are stored in CPU memory and returned to the accelerator in score tiles. Each tile updates the running row maximum m, partition function Z, and weighted value sum R; distributed ranks rescale their local Z, R to a common maximum before summing. Query fields are reduce-scattered to their owners. During replay, all but the last chunk use FSDP `no_sync`; this suppresses repeated gradient synchronization but not all model-shard communication.

The query/document update schedule and memory implementation can be changed independently. `both` constructs both directional fields in one update; `alternating` selects query and reverse fields on successive optimizer steps. Direct execution retains autograd activations for the encodings being updated. Fixed-memory execution first encodes without gradients, then recomputes only those encodings during backpropagation. Reverse alternating steps encode only aligned positives, not explicit negatives. Unsupported combinations fail validation rather than silently changing semantics.

C EXPERIMENTAL DETAILS

MEDI2. We pin the public MEDI2 reconstruction to Hugging Face revision `b0c9ed0`. Same-source batches contain 2,048 queries and 4,096 documents globally. Runs use 1,253 optimizer updates, query/document limits 256/2,048, AdamW at 2×10^{-5}, a linear schedule with 3% warmup, seed 42, bfloat16, mean pooling, bidirectional SDPA, gradient checkpointing, and FSDP. The Qwen comparison uses 32 GPUs. Earlier model-family runs use 64 GPUs for Qwen, 128 for Mistral, and 32 for Gemma. InfoNCE uses $T = .02$ and Raw Drift uses field temperature .05 unless stated otherwise.

Systems runs. The four-cell end-to-end study fixes Qwen2.5-0.5B, 32 GPU workers, the MEDI2 order, global query batch 2,048, $G = 2$, 1,253 updates, and seed 42. Streamed runs construct each local batch from four 16-query replay chunks; direct runs use the equivalent live batch of 64. Complete application time begins at process allocation and ends after final checkpoint save and clean exit. The pair-batch scaling test uses synthetic maximum-length q64/p512 pairs, $G = 1$, nine finite updates, and reports the mean after excluding the first two timed steps.

Platform and evaluation. Tuolumne nodes provide four MI300A GPU workers. Measurements use PyTorch 2.9.1, ROCm 6.4.3, and RCCL on 8 nodes unless specified otherwise. Full quality uses the English MTEB 56-task suite. Aggregation fails on missing or malformed task files; the 12 CQADupstack subsets merge into one retrieval dataset before the dataset-level mean. The fixed 14-task MTEB subset covers all seven reported task categories.

D COMPLETE RESULTS

D.1 MODEL-FAMILY CATEGORIES

Table 3 expands the compact scope rows from the main paper. The InfoNCE rows use no-grad gather and the Raw Drift rows use direct autograd. These model-family comparisons test retrieval quality, not fixed-memory execution.

Table 3: Full English MTEB by category. Every row has 56/56 coverage.

Backbone	Objective	Overall	Class.	Cluster	Pair	Rerank	Retrieval	STS	Summ.
Mistral-7B	InfoNCE no-grad (GritLM)	58.83	64.78	45.87	81.55	60.04	48.27	77.38	29.79
Mistral-7B	Raw Drift	**59.09**	63.42	48.91	80.40	60.81	48.28	77.18	29.28
Qwen2.5-0.5B	InfoNCE no-grad	52.85	64.13	42.13	75.33	55.81	33.67	74.31	29.19
Qwen2.5-0.5B	Raw Drift	**55.34**	62.99	43.93	79.30	56.72	40.35	76.21	28.03
Gemma-3-1B-IT	InfoNCE no-grad	52.63	65.58	41.97	73.34	56.88	32.21	73.70	31.16
Gemma-3-1B-IT	Raw Drift	**52.88**	63.43	42.89	76.15	57.30	32.60	75.26	28.71

D.2 SCHEDULE EVIDENCE

The earlier schedule comparison in Table 4 matches the semantic training protocol but not the distributed topology. It is included for transparency and is not used as a fully controlled causal estimate. The production Drift implementation also preserves an extra $1/W$ coefficient, so changing worker count changes the effective pre-optimizer gradient scale before clipping and Adam.

Table 4: Supporting Qwen Raw Drift schedule evidence on full MTEB.

Schedule	GPUs	Local batch	Overall	Retrieval
Both directions	64	32	55.56	41.04
Alternating	32	64	55.53	41.66

D.3 SAMPLED-INFONCE TRAINING-TIME SENSITIVITY

Table 5 tests whether more sampled-InfoNCE updates close the compact retrieval gap at approximately the same measured Trainer time as alternating Drift. The 2,370-step runs are independent trainings,

not continuations of the 1,253-step checkpoint. All use the same seed-42 MEDI2 protocol and 32 GPU workers. More updates also mean more pair presentations and a different absolute learning-rate trajectory. Halving the peak rate probes, but does not exhaust, schedule sensitivity.

Table 5: Independent G=2 Qwen runs on strict MTEB-14. The sampled-InfoNCE query/document gradient-sampling fractions remain 0.5/0.25 in all three sampled rows. Trainer time excludes initialization and final checkpoint save. The strongest observed sampled baseline is the shorter 1,253-step run; this is not an equal-example or complete-wall-clock comparison.

Objective	Steps	Peak LR	Trainer (s)	Overall	Retrieval
Sampled InfoNCE	1,253	2×10^{-5}	969	56.60	39.83
Sampled InfoNCE	2,370	2×10^{-5}	1,804	55.06	37.67
Sampled InfoNCE	2,370	1×10^{-5}	1,785	54.96	35.47
Alternating Raw Drift	1,253	2×10^{-5}	1,829	56.42	42.46

D.4 Pair-batch scaling results

Table 6 gives every point in Figure 4. The materialized baseline fails through host exhaustion on MI300A's shared physical memory pool; a dash denotes that no finite step measurement exists.

Table 6: All 18 runs in the synthetic pair-batch scaling comparison.

Method	Global batch	Alloc. GiB	Step s	Pairs/s
Materialized InfoNCE, no-grad	4K	9.643	1.487	2,754
	8K	17.454	2.717	3,016
	16K	33.072	5.139	3,188
	32K	64.312	9.954	3,292
	64K	–	–	host OOM
	128K	–	–	host OOM
Exact streamed InfoNCE	4K	10.334	1.860	2,202
	8K	10.361	3.731	2,196
	16K	10.417	7.516	2,180
	32K	10.527	14.784	2,216
	64K	10.750	29.181	2,246
	128K	11.195	58.699	2,233
Fixed-memory alternating Drift	4K	9.385	1.128	3,630
	8K	10.306	2.185	3,749
	16K	10.306	4.499	3,642
	32K	10.306	8.727	3,755
	64K	10.306	17.700	3,703
	128K	10.306	36.158	3,625

E Method and Tensor Details

Query and document gradient counts. One controlled run uses 32 GPUs, 64 queries per GPU, two documents per query, and $D = 896$. Its local query and document representations have shapes $(64, 896)$ and $(128, 896)$; the global representations have shapes $(2048, 896)$ and $(4096, 896)$. A direct score matrix is $(2048, 4096)$. The first document in each two-document group is positive. All remaining documents, including other queries' positives, act as negatives. The direct-form encoder-graph counts per GPU are:

```
InfoNCE:     64 query graphs + 128 document graphs
Drift even:  64 query graphs +   0 document graphs
Drift odd:    0 query graphs +  64 positive graphs
```

All 128 local documents still contribute values on an even Drift step. On an odd step, paired positives are anchors and queries are detached values; explicit hard negatives need not be encoded. Fixed-memory replay retains activations for one chunk at a time, but the counts describe which rows receive encoder gradients over the complete step.

Table 7: **Retrieval response to candidate coverage.** Equal-weight four-task MTEB-14 retrieval means over two training seeds. Seed changes are shown separately; each method keeps the same global optimizer batch and gradient-sampling rates at both pool sizes.

Method	T	Pool 256	Pool 2,048	Δ_{42}	Δ_{43}
Sampled InfoNCE	0.02	42.14	41.80	-0.36	-0.32
Sampled InfoNCE	0.05	41.96	41.40	-0.91	-0.21
Alternating Raw Drift	0.05	41.58	42.66	$+1.01$	$+1.17$

Raw and normalized fields. Both variants unit-normalize the *embeddings*. Raw Drift keeps each tangent field $\bar{F}_i$ at its measured magnitude, so a query whose positive dominates the softmax receives a smaller update. Normalized Drift measures one RMS field magnitude across the distributed batch and divides *every* update by that scalar. It does not normalize each example individually. This changes the effective step scale from batch to batch; the primary fixed-memory method uses Raw Drift.

Computing the weighted average in blocks. Equation 2 needs one average document vector per query, not all of the weights that produced it. To see why, temporarily write each document's unnormalized weight as $w_j = \exp(q^\top p_j/T)$. The center is $(\sum_j w_j p_j)/(\sum_j w_j)$. Each block contributes to these two sums, so its scores can be discarded afterward. Crucially, we sum weighted vectors and weights separately: simply averaging the centers of different blocks would be wrong.

For numerical stability, we subtract the largest score seen so far before taking exponentials. Each query therefore keeps three quantities: m, that largest score; Z, the sum of shifted weights; and R, the sum of shifted weighted document vectors. The center is still R/Z because the common exponential scale cancels. When a new block has a larger score, we first rescale the old sums to the new maximum. For block scores $S = QP^\top/T$, the update is

$$m' = \max(m, \max_j S_{:j}), \tag{7}$$

$$A = \exp(m - m'), \qquad E = \exp(S - m'), \tag{8}$$

$$Z' = A \odot Z + E\mathbf{1}, \tag{9}$$

$$R' = A \odot R + EP. \tag{10}$$

Here maxima and rescaling are row-wise; P contains the current block of document embeddings. For example, a full $(2048, 4096)$ score matrix can be processed using only $(128, 128)$ temporary score blocks. For the 128 active queries, m and Z each have 128 entries and R has shape $(128, D)$. Adding more document blocks does not enlarge this state. The result is exact up to floating-point reduction order, without sampling or dropping candidates. Exact streamed InfoNCE can use the same calculation. Its full-gradient version must additionally compute the document gradients and backpropagate through document encodings.

F Candidate-Coverage Control

The main-text candidate-pool experiment keeps the global optimizer batch at 2,048, 1,253 updates, model, input lengths, data order, and the number of encoded query/document examples fixed within each seed and objective. It masks the softmax candidate pool to 256 or 2,048 entries. This does not implement a lower-cost small-pool trainer. The two objectives retain their respective temperature and gradient-sampling settings. The comparison measures the effect of candidate coverage within each method; it does not isolate the loss function.

Excluding ArguAna, Drift's remaining three-task retrieval changes are $+0.42$ and $+0.67$ in seeds 42 and 43. Comparing the changes between methods gives $+1.36$ and $+1.48$ points on the four-task retrieval mean. These two training seeds do not establish monotonicity, statistical significance, or transfer to the 56-task suite.

Table 8: Change from 256 to 2,048 candidates/query on the complete fixed 14-task MTEB subset. Retrieval columns are per-task score changes; ArguAna is the largest Drift contributor in both seeds.

Seed	Objective	Overall	ArguAna	NFCorpus	SciFact	SCIDOCS	Retrieval mean
42	Sampled directional InfoNCE	-0.50	-1.28	$+0.35$	-0.07	-0.44	-0.36
42	Alternating Raw Drift	$+0.37$	$+2.78$	$+0.39$	$+0.55$	$+0.32$	$+1.00$
43	Sampled directional InfoNCE	-0.56	-0.33	$+0.01$	-1.02	$+0.05$	-0.32
43	Alternating Raw Drift	$+0.36$	$+2.65$	$+0.40$	$+1.11$	$+0.52$	$+1.16$

G Weak-Pair Boundary Details

The controlled Nomic experiment uses a revision-pinned 18-source subset of 5.17 million query–positive pairs, not the full released data. Qwen2.5-0.5B receives a global pair batch of 768, $G = 1$, query/passage limits 256/512, seed 42, and 1,200 updates on eight RTX A4000 GPUs. Thus each run presents 921,600 pairs (17.8% of this subset). The content-disjoint cleaned held-out pair set has 66,900 examples. All four checkpoints saved and all four compact MTEB evaluations completed 14/14 tasks.

Table 9: Matched Nomic weak-pair pilot. Scores are percentages; Trainer time is measured within each run. Different A4000 nodes and, for three runs, preemptible scavenger allocations limit runtime comparisons.

Objective	Pair macro R@1	MTEB-14	Retrieval	Trainer (s)
Directional InfoNCE	98.55	59.51	43.98	6,160
Symmetric InfoNCE	98.63	59.02	43.08	6,039
Balanced Raw Drift	98.29	58.59	43.17	4,320
Source-aware Drift	97.70	57.36	41.47	4,408

The source-aware rule, designed to increase reverse updates for scarce QA sources, loses to balanced Drift on each of its six targeted sources and is not the default method. Pair Recall@1 is near saturation in every run, so the 14-task MTEB results better distinguish the methods. Runs used different A4000 nodes, so shorter Drift Trainer times here are not a controlled throughput claim. None of the scavenger allocations was preempted.

Earlier 300-step pilot selection used the held-out pair set repeatedly and therefore measures validation, not untouched test performance. Directional InfoNCE reached 98.39 pair R@1 (98.46 with another seed), while no Drift pilot exceeded 97.71. The strongest pilot Drift pair-recall checkpoint used a balanced query/document schedule, temperature .05, and surrogate scale 22,400. It scored 58.03 overall and 42.02 retrieval on MTEB-14 versus 59.45 and 43.36 for InfoNCE. Longer 4.92-million-presentation runs preserved the ordering: balanced Drift reached 98.82 pair R@1 in 2,282 Trainer seconds versus InfoNCE at 99.13 in 3,092 seconds, but their final MTEB-14 overall scores were 59.17 versus 60.70. These results do not establish a quality-preserving increase in negatives or a time-to-quality advantage for detached Drift on weakly supervised pairs.

H Additional Systems Details

H.1 Discover, summarize, and replay

The replay implementation has three stages. First, a representation- discovery pass encodes local query and candidate chunks under `no_grad`, moves their embeddings to CPU, and releases layer activations. It records RNG state only for query or document chunks that will receive gradients. Second, it moves small CPU embedding tiles back to the accelerator to compute Eq. 10 and the signed output gradient for active anchors. Third, it restores RNG state, rebuilds one active encoder chunk at a time, and injects that cached gradient at the embedding output, following the discovery/replay structure of Gradient Cache (Gao et al., 2021):

```
with no_grad():
    q_cpu, q_rng = encode_chunks_to_cpu(query_tokens)
```

```
    p_cpu, p_rng = encode_chunks_to_cpu(candidate_tokens)

for active_cpu in anchor_tiles:
    anchors = all_gather(active_cpu.cuda())
    m, Z, R = -inf, 0, 0
    for values_cpu in p_cpu.split(candidate_tile):
        m, Z, R = online_merge(m, Z, R,
                               anchors, values_cpu.cuda())
    m, Z, R = distributed_merge(m, Z, R)
    field = tangent(aligned_positive - R / Z, anchors)
    embedding_grad = signed_scaled_gradient(field)

for anchor_tokens, rng, grad in active_replay_chunks:
    restore(rng)
    live = encoder(anchor_tokens.cuda())
    dot(live, grad.cuda()).backward()
```

For an alternating query step with global query count B, embedding dimension D, and world size W, the CPU representation cache is $O((B+K)D/W)$ per rank, in addition to token inputs and active replay gradients. The accelerator working set is controlled by replay and score tiles rather than the logical batch. Host storage and total encoder-forward work still grow with the number of fresh examples; bounded accelerator allocation is not constant total-memory or constant-compute training. Tiled score arithmetic also remains $O(BKD)$; streaming changes the live score storage, not the number of pairwise dot products.

For Raw Drift's active directional field, the cached query output gradient is $-2s\alpha\bar{F}_i/(BD)$; the reverse direction additionally applies its configured reverse-field multiplier. In the reported Drift runs, the local surrogate is scaled by $1/W$ before FSDP averages parameter gradients. This leaves an additional $1/W$ factor relative to a centralized gradient. The full-gradient InfoNCE control compensates for distributed averaging. Because the factor affects clipping and Adam's numerical behavior, we retain it in the reported runs and state it explicitly rather than rescaling trained checkpoints.

H.2 Distributed alternating Drift

Each rank owns a shard of candidates. A local anchor tile of c_q rows is all-gathered, so each rank scores Wc_q anchors against a local candidate tile of c_p rows. Temporary score storage is $O(Wc_qc_p)$ and field state is $O(Wc_qD)$, both bounded in B and K at fixed W, c_q, c_p. Local (m, Z, R) states are merged across ranks by first rescaling Z and R to a common row maximum and then summing; completed field rows are reduce-scattered to their owners. The complete candidate table is never a persistent GPU tensor, although its local shard is cached on CPU.

FSDP parameter communication can occur during representation discovery and replay. During replay, all but the final active chunk use `no_sync`, so an optimizer step has one synchronized backward phase; this suppresses gradient synchronization, not every FSDP collective. Deferred synchronization also retains a model-sized parameter-gradient term in the live working set.

On query steps, every query is replayed while all documents contribute detached values. On reverse steps, only paired-positive documents are replayed and explicit negatives need not be encoded. The direction is derived from the optimizer step stored by the Trainer, so an odd-step checkpoint resumes with the correct phase. Telemetry records the active direction beside gradient norm and memory measurements to prevent cadence-dependent summaries of only one half of the schedule.

H.3 Counterbalanced optimized text-retrieval execution

Implementation comparisons. With replay on both methods, deterministic packed sampled InfoNCE takes 8.72/35.53 seconds at 32K/128K pairs, versus Drift's 8.80/35.58 seconds. There is no meaningful speed advantage in that comparison. A dropout-safe sampled implementation uses less allocated memory (7.63/8.35 GiB) but takes longer. The main-text paired comparison instead uses candidate-first execution for both methods. In a separate MEDI2 Drift comparison, candidate-first

Table 10: **Individual optimized paired-execution repeats.** I denotes sampled-gradient InfoNCE, D alternating Raw Drift. Peak allocated and reserved HBM are maxima across 32 ranks; peak host RSS is process-local.

Pairs	Order	Method	Step (s)	Alloc. GiB	Reserved GiB	Host RSS GiB
32,768	1	I	8.539	10.423	12.408	7.716
	2	D	7.455	10.306	12.344	7.715
	3	D	7.489	10.306	12.369	7.715
	4	I	8.607	10.423	12.408	7.726
131,072	1	D	31.015	10.306	12.363	7.825
	2	I	35.502	10.772	13.145	7.899
	3	I	35.591	10.772	13.125	7.899
	4	D	30.892	10.306	12.383	7.829

Table 11: **Controlled CUDA systems comparison at 16K global pairs.** ViT-B/32, two RTX A4000 GPUs, local replay chunk 128, shared FP16 CPU replay storage, and the same fixed checkpoint. Values average two reverse-order repetitions after discarding two warmup steps. Encoder FLOPs use PyTorch's operator counter; encoder FLOPs, score FLOPs, and candidate communication are normalized to exact Inf-CL. Process wall time covers launch, eight training steps, and clean exit, without checkpoint saving.

Method	Encodings with gradients	Encoder rows/rank	Encoder FLOPs	Score FLOPs	Cand. comm.	Step (s)	Pairs/s	HBM (GiB)	Process wall (s)
Exact Inf-CL	2	32,768	1.00×	1.0×	1.0×	32.53	503.6	7.59	282.3
Dense replay control ($\rho = 1$)	2	32,768	1.00×	1.0×	1.0×	33.42	490.2	7.53	293.0
Single-side replay ($\rho = .5$)	1	24,576	**0.625**×	1.0×	1.0×	**26.03**	**629.5**	**5.42**	**237.6**

execution reduces training-loop time by 15.1% relative to replay at unchanged allocated memory; this is an implementation improvement, not a new loss.

Table 10 gives every fresh-process measurement behind Table 2. The 32K allocation was `f3aGTn3hoNuM` (InfoNCE–Drift–Drift–InfoNCE); the 128K allocation was `f3aGTuRYe7o5` (Drift–InfoNCE–InfoNCE–Drift). Both Flux allocations completed successfully. Each process executed nine optimizer updates; timings average steps 3–8, so the alternating Drift window contains three complete query/reverse cycles. Both methods see the same fresh global pair batch and use the same Qwen model, sequence limits, replay tile, and allocation at each size. The sampled InfoNCE control backpropagates through 50% of queries and 50% of documents per step; Drift backpropagates through all queries or all paired documents on alternating steps. The average number of encodings receiving gradients is the same, but the estimators differ.

The callback timestamps rank-zero logs. In the installed Trainer, its loss gather ends with a GPU-to-host `.item()` before the callback, forcing completion of the logged work. As a cross-rank check, summing the same six step intervals using the maximum of all 32 ranks differs from the rank-zero sum by at most 0.003 seconds over 44–214-second windows in all eight repeats. The table reports steady training, not initialization, final checkpoint saving, or time to a target quality score.

H.4 Single-side gradient replay with Inf-CL

This image–text experiment measures the cost of backpropagating through both encoders. It keeps Inf-CL's exact score computation and is *not* Drift training or evidence of image–text Drift quality. Inf-CL tiles the contrastive computation rather than storing the full score matrix, but computes gradients for both image and text encodings. Our variant scores all image–text pairs, then uses each pair's stable sample ID to select one side for gradient replay. The selected side flips in the next epoch. Its gradient is doubled. At fixed weights and batch, averaging the gradients from complementary selections gives the full gradient. Across training steps the assignment is deterministic and both model and batch change, so we do not claim a conditionally unbiased gradient or the exact symmetric gradient on any individual step.

In the eight-step 16K-pair measurement, single-side replay is $1.25\times$ faster at steady state, uses 29% less allocated HBM, and shortens the measured eight-step process by $1.19\times$. The dense replay control tracks exact Inf-CL, isolating the gain to fewer encoder-gradient computations rather than

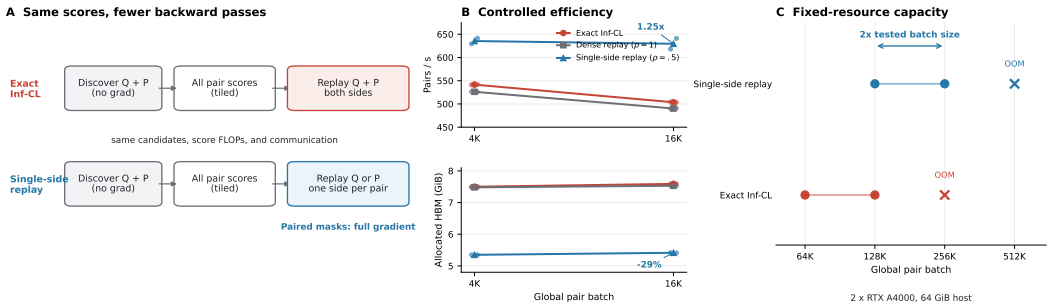


Figure 5: **Full InfoNCE scores with single-side gradient replay. A:** Exact Inf-CL and our method discover both towers and compute the same full score computation, but our $\rho = .5$ schedule backpropagates through either the image or text encoding for each pair. **B:** Controlled fixed-state measurements on two RTX A4000 GPUs; small points are two reverse-order repetitions and lines join their means. Dense $\rho = 1$ replay controls for implementation overhead. **C:** Largest passing power-of-two batch under the same two-GPU, 64-GiB-host allocation and shared FP16 replay storage; crosses are measured cgroup OOMs. LR-zero runs execute the complete training path but measure systems behavior rather than convergence.

a faster score kernel. Both methods retain identical candidate sets, modeled score FLOPs, and candidate-ring traffic. A separate 50-step counterbalanced replication at the same checkpoint and storage policy measured 36.58 s for exact Inf-CL versus 31.73/29.44 s for single-side replay across its two repetitions (a 1.15–$1.24\times$ speedup); complete process wall followed a similar range, with HBM unchanged. The encoder accounting changes from 32,768 to 24,576 row evaluations per rank: both methods encode images and text without gradients first, while the gradient-carrying replay falls from 16,384 to 8,192 rows. PyTorch's operator counter refines this proxy: profiler-covered discovery-and-replay encoder work falls from 59.0 to 36.9 GFLOPs per pair, a 37.5% reduction. This excludes the unchanged score-field kernels and all communication.

Under the same 64-GiB host allocation, exact Inf-CL passes at 128K pairs and host-OOMs at 256K; single-side replay passes at 256K and host-OOMs at 512K. The measured maximum batch size doubles at these tested powers of two; this is not a general $2\times$ capacity ratio. Accelerator allocation is nearly bounded by replay tiles, but CPU token/image storage still grows linearly. Complete-field replay also leaves candidate-ring and parameter-gradient communication unchanged.

Text-retrieval execution. Both objectives can avoid materializing the full score matrix. Exact streamed InfoNCE, however, must replay every query and candidate row whose encoder gradient contributes to the full contrastive update. Alternating Drift treats the streamed field as a detached target and recomputes only queries or paired documents with gradients on each update. Table 12 measures the resulting trade-off end to end, including startup, final checkpoint save, and process exit.

Table 12: **End-to-end MEDI2 efficiency and compact transfer.** Qwen2.5-0.5B on 32 MI300A GPUs; all runs use 1,253 optimizer updates and 2,566,144 query presentations, with $G = 2$ and query/passage limits of 256/2,048 tokens. Application time includes initialization, training, final checkpoint saving, and clean exit, excluding queue wait. Memory is the maximum training-time PyTorch allocated memory across GPUs. Scores use the same predeclared 14-task evaluation suite. Results are single-seed observations; the methods use different gradient schedules.

Objective	Execution	PyTorch alloc. (GiB/GPU)	Train loop (min)	Application (min)	MTEB-14	Retrieval
InfoNCE	Direct differentiable gather	15.286	22.0	48.1	55.38	35.57
InfoNCE	Exact streamed + offloaded GC	4.049	65.9	78.1	55.31	35.32
Raw Drift	Direct alternating	8.199	11.7	40.0	56.40	42.32
Raw Drift	Fixed-memory alternating	4.035	30.5	47.4	56.42	42.46

At a matched PyTorch-allocated working set, alternating Drift is $2.16\times$ faster in the Trainer loop and $1.65\times$ faster over the complete application than exact streamed InfoNCE. The direct and fixed-

memory Drift checkpoints differ by only 0.02 MTEB-14 point; direct and exact streamed InfoNCE differ by 0.07 point. The runs process the same number of queries, but are not matched in FLOPs or gradients: Drift alternates query and document updates, while InfoNCE computes gradients for both. Backward work and candidate participation therefore differ. These runs use temperature 0.02 for InfoNCE and temperature 0.05 with surrogate scale 1 for Drift; all optimizer settings are reported in Appendix C.

Encoder-work proxy. The speed difference has a simple first-order explanation for pair-only training. Let F denote one full-batch forward pass through queries and documents, and approximate backward through both as $2F$. Full two-tower Gradient Cache performs one detached discovery forward, then one replay forward and backward, for roughly $F + F + 2F = 4F$. Assuming equal query and document forward cost, alternating replay backpropagates through one side per update, for approximately $F + \frac{1}{2}F + F = 2.5F$ averaged across query and document steps. The resulting $4/2.5 = 1.6\times$ ratio is an illustrative accounting estimate, not measured FLOPs or a validation from numerical agreement. The observed pair-batch throughput ratio happens to be $1.62\times$, but unequal sequence lengths, activation recomputation, kernels, omitted explicit negatives on reverse steps, score computation, communication, and host transfers can all change the realized ratio.