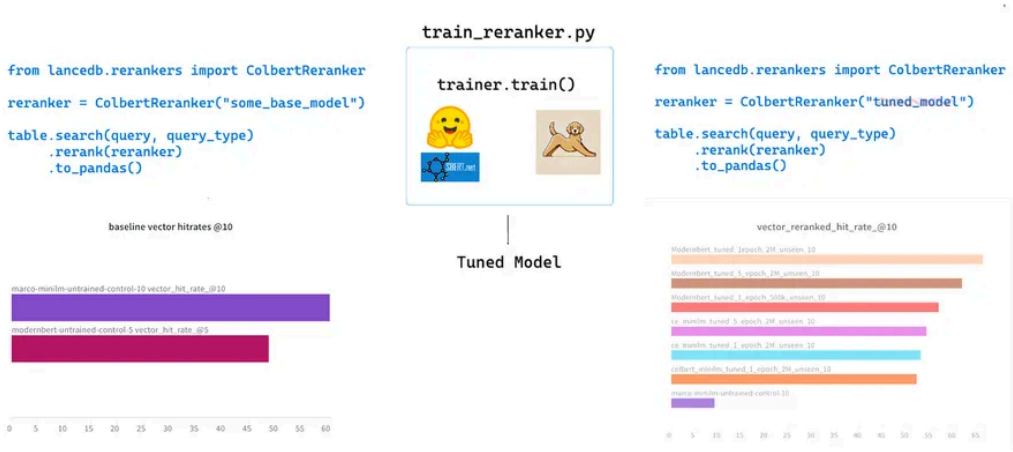


A Practical Guide to Training Custom Rerankers

11 MIN READ APR 10, 2025

A report on reranking, training, & fine-tuning rerankers for retrieval

This report offers practical insights for improving a retriever by reranking results. We'll tackle the important questions, like: *When should you implement a reranker? Should you opt for a pre-trained solution, fine-tune an* *or build one from scratch?*

The retrieval process forms the backbone of RAG or agentic workflows. Chatbots and question-answer systems rely on retrievers to fetch context for every response during a conversation.
Most retrievers default to using vector search. It makes intuitive sense that improved embedding representations should yield better search results.

In this blog series, we'll work backwards from the last component of the retrieval process. This blog deals with the reranking aspect of RAG or retrievers.

So, where does reranking fit in?

To understand the importance of reranking models, let's first look at the difference between embedding and ranking models and try to answer this question:

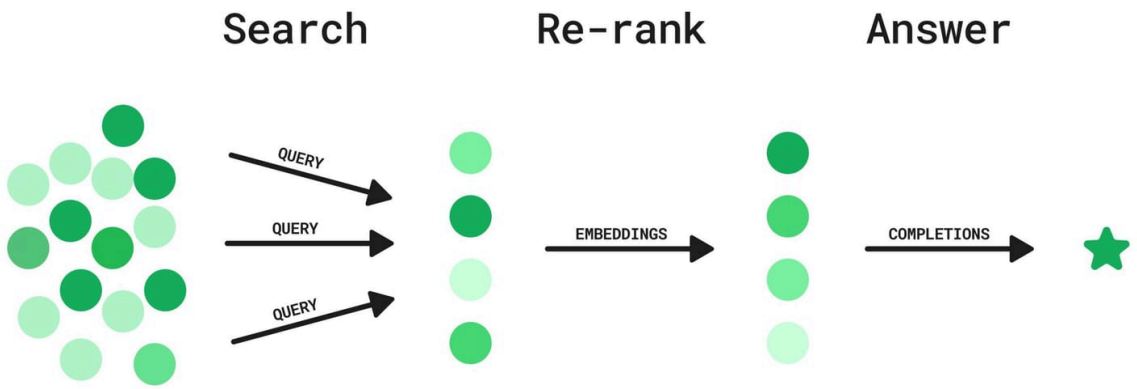
If enhancing embedding models can yield better search results, why not focus on refining (by fine-tuning) existing models or experimenting with new ones?

Embedding vs Ranking models

The fundamental difference between embedding and ranking models is

> Embedding models convert any given data point to a vector representation in a given vector space.

Ranking models rank the data points (or documents in case of text ranking), based on their relevance to the query



source - **https://unfoldai.com/rag-rerankers/**

So the **job of an embedding model is to keep semantically similar docs or queries close in a given vector space**. During vector search these similar docs are fetched as probable responses to the query.

A **reranking model re-arranges this list such that the most relevant answers appear at the top**. Another scenario where reranking is useful is in cases where you need to combine multiple independent search result sets like from both vector search and full-text search.

Why not use a ranking model for search?

Ranking is typically a much more expensive operation than vector search. You can't run your entire database row-by-row through a reranker model. Unlike embedding models, you can't create vector representations of docs offline, as ranking models need both the query and docs to calculate the rank.

Here are some more differences between embedding and ranking models.

Feature	Embedding Models	Reranking Models
Primary Goal	Generate vector representations capturing semantics.	Calculate precise relevance scores for query-item(s) pairs.
Input	Single item (query *or* document/data point).	* A pair: (query *and* a candidate item).
Output	Dense numerical vector (embedding).	A single relevance score per input pair.
Typical Use	Initial candidate retrieval, semantic search, clustering.	Refining/reordering top candidates from retrieval.
Stage in Pipeline	Early stage (candidate generation/retrieval).	Later stage (refinement/reordering).
Compute Cost	Generally lower per-item (embeddings often pre-computed).	Generally higher per-pair (evaluates interaction).
Scalability	High (efficient for large datasets via vector search).	Lower (applied only to a small subset of candidates).
Architecture	Encodes items independently.	Processes query and item together for interaction.
Quality vs. Speed	Optimized for speed & broad recall on large datasets.	Optimized for precision/quality on a smaller set.
Example	Encoding docs into vectors for a vector DB.	Re-ordering top 50 results based on relevance scores.

* In case of cross-encoders, each query-doc pair's similarity is calculated individually. But late interaction based models, like ColBert, calculates doc representations as a batch and reranks them based on MaxSim operation with query representaions.

Fine-tuning tradeoffs

Embedding models

Upgrading your embedding model, by either switching to a new model or fine-tuning a base model on your data, *can* significantly improve the base retrieval performance. However, fine-tuning the embedding model changes the entire embedding space, which may end up harming the overall performance of the model. In addition, it is disruptive operationally because you need to regenerate all of the embeddings. And with many vector databases that only allows one vector field per record, it means managing many different tables for experimentation. For more info on this, you can read up on catastrophic forgetting in LLMs.

Rerankers

However, because rerankers are used in the later stage of the retrieval pipeline, you can switch them without disrupting the existing workflow. You can switch models, train new ones, or even get rid of them completely without having to re-ingest your data or update query/source embedders. This tends to be a much easier way to squeeze more quality out of your retrieval pipeline.

Training rerankers from scratch

This blog deals with reranker models and how you can start off with base LLMs (not base rerankers) and train them on your private data in a few minutes to get significant retrieval performance gains. Of course you can start off with a trained reranker and fine-tune it on your dataset too.

Dataset

The dataset used is **gooaq** which has over 3M `query and answer` pairs. Here's it is used for training and eval:

First, 2M rows are used for training.

Next 100_000 rows are ingested in LanceDB

From that 100_000, we take 2000 samples and evaluate the hit rate

The baselines

Model	Vector Hitrate
marco-minilm-untrained-control-@10	60.3
modernbert-untrained-control-@5	48.75

Base Model(s) used:

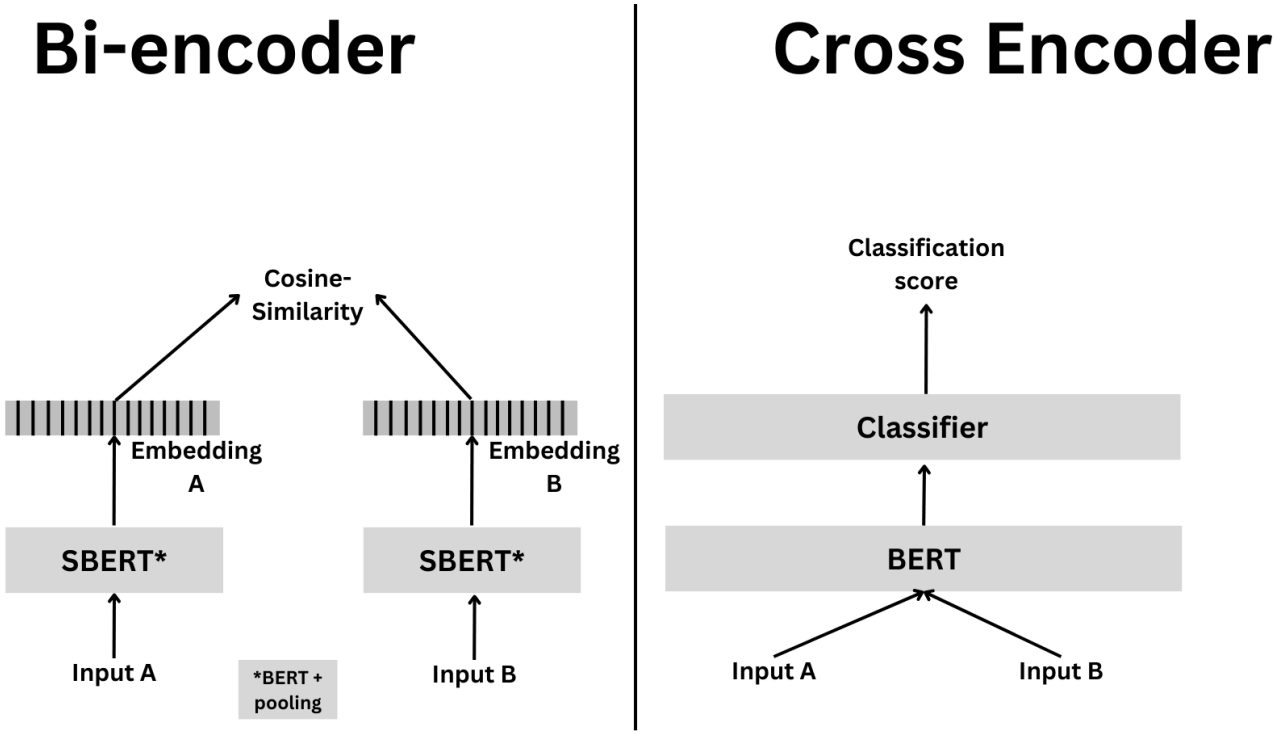
For training cross-encoder and ColBert models, we'll use the same two base models:

1. MiniLM-v6 - A very small base model with only **~6Million params**
2. ModernBert-base - A relatively larger base model with **~150Million params** (yet much smaller than SOTA reranker models)

This allows us to look at the tradeoff between model size (and latency) vs quality improvement.

Training Cross encoders

A cross-encoder is a neural network architecture designed for comparing pairs of text sequences. Unlike bi-encoders, which encode texts separately, cross-encoders process both texts simultaneously for more accurate comparisons.



Joint Processing: Both the query and document/passage are concatenated and fed together into a transformer model (like BERT or RoBERTa).

Cross-Attention: The transformer applies self-attention across the entire concatenated text, allowing each token to attend to all other tokens in both sequences

Dataset preprocessing:

We'll format the train subset of the dataset for training a cross-encoder model using BinaryCrossEntropy loss. To use that loss function, we'll need the dataset is this format: `query/anchor, positive, label`

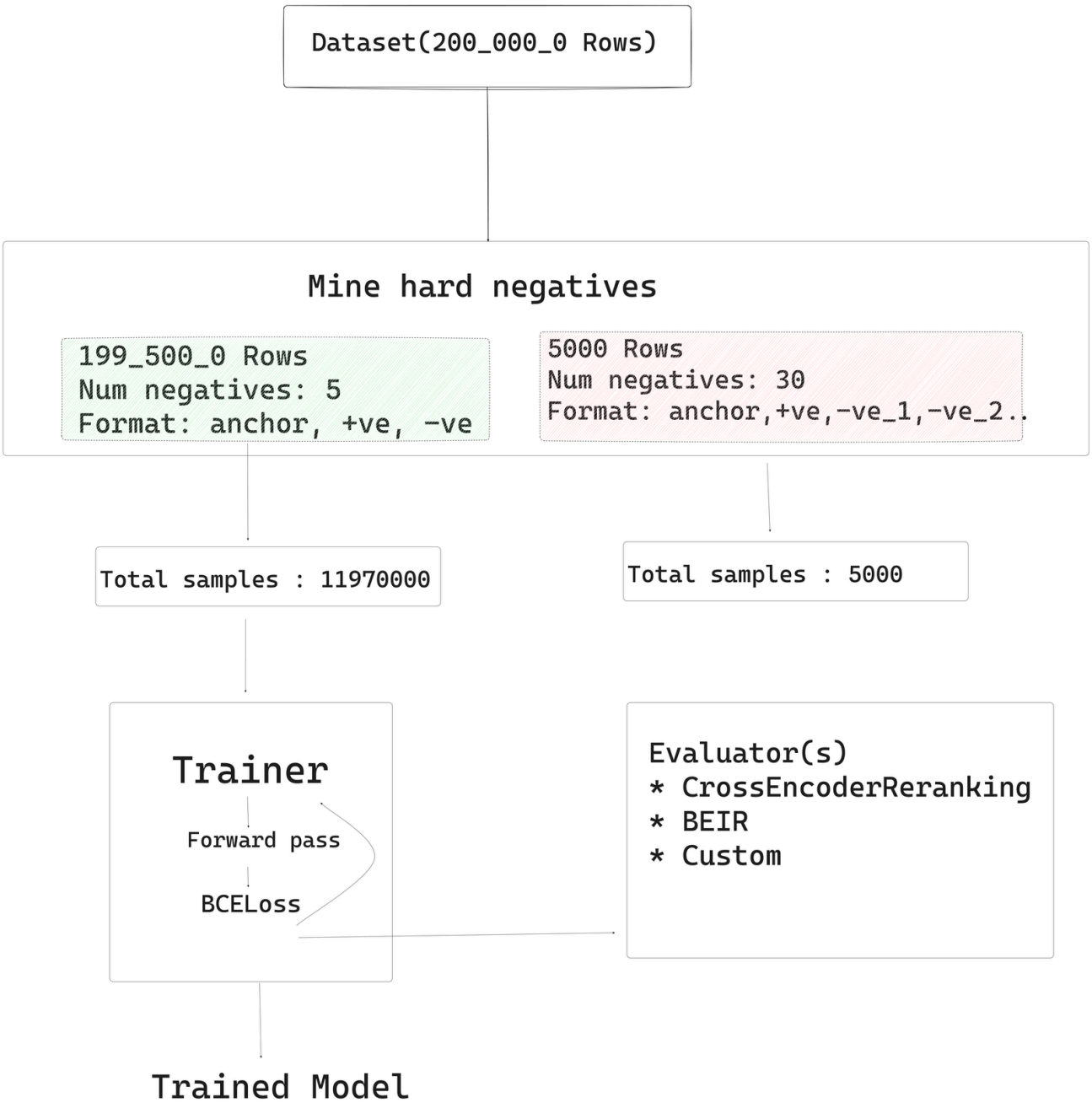
Similarly, for evaluating, we'll create the eval such such that it has many negatives like `anchor, positive, neg_1, neg_2 ..`

Finally, to create the train and test split from the dataset, **we'll mine "hard negatives" from the given dataset**.

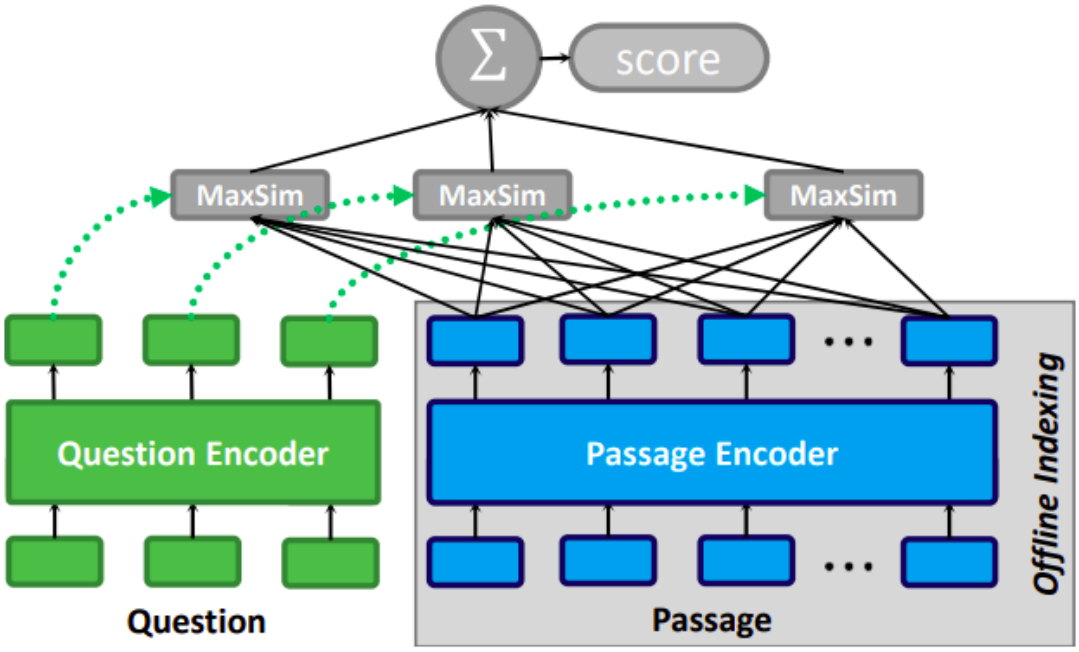
Hard negatives are negatives that have very similar semantic meaning to the query's positive answer. **We use an independent embedding model to find negatives are semantically close to the query**, also known as anchors.

To read more in-depth about the training recipe, you can refer to sentence-transformer's cross encoder training **guide**.

In brief, this is how the training pipeline looks



Training ColBERT reranker



For training late interaction models like Colbert, we'll use Pylate, a sentence transformer-based library that implements late interaction utils. The only difference in the training script would be using

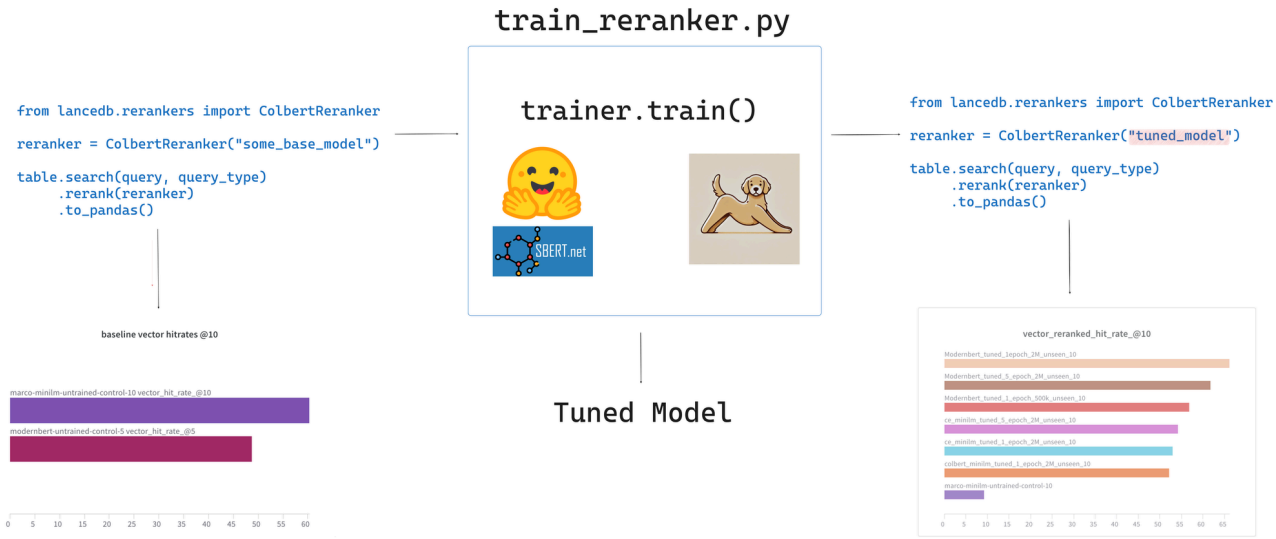
> **Contrastive loss** is used for training late interaction models. Here's how it works
>
> > **Token Embeddings**: Extract and normalize token-level representations for query and documents
> >
> > **MaxSim Scoring**: For each query token, find maximum similarity with any document token, then sum
> >
> > **Label Assignment**: Generate labels identifying which documents match each query. We'll get 2 scores, one for -ve sample and the other for +ve. Now this becomes a classification problem.
> >
> > **Cross-Entropy**: Apply cross-entropy loss between similarity scores and labels.
>
> **ColbertTripletEvaluator** similar to cross-encoder reranker, but using late interaction multi-vector embeddings

We'll use `PyLate` python package for fine-tuning late interaction ColBert models.

The same pipeline will be used for training models from scratch and fine-tuning

Using fine-tuned Reranker models with LanceDB

LanceDB has integrations with all popular reranking models. Switching to a new model is as simple as changing a parameter.



For example here's how you can use ColBERT model as a reranker in LanceDB, with hybrid search query type.

```
from lancedb.rerankers import ColbertReranker

reranker = ColbertReranker("my_trained_model/") # local or hf hub

rs = tbl.search(query_type="hybrid")
        .vector([...])
        .text(query)
        .rerank(reranker)
```

```
        .to_pandas()
```

LanceDB also integrated with many other rerankers including Cohere, Cross-encoder, AnswerdotaiRerankers etc. Answerdotai reranker integration allows you to use multiple various types of reranker architectures. Alternatively, Here's how you would use ColberReranker using Answerdotai integration.

```
from lancedb.rerankers import AnswerdotaiRerankers

reranker = ColbertReranker(
            "colbert",    # Model type
            "trained_colbert_model/" # local of HF hub
            )

rs = tbl.search(query_type="hybrid")
        .vector([...])
        .text(query)
        .rerank(reranker)
        .to_pandas()
```

Performance comparison

The experiment dashboard contains all combinations of model architectures, base models, and **top_k** combinations. Here, I'll present some summary numbers making it simpler to compare the performance and judge the speed-accuracy tradeoffs.

Control experiments - These are baseline runs with no reranking. Same as covered above

Experiment	Vector@10	Vector@5	FTS@10	FTS@5
modernbert-untrained-control-5	N/A	48.75	N/A	33.1
marco-minilm-untrained-control-10	60.3	N/A	N/A	39.25

Base model reranking performance

This shows that any base model without being trained as reranker model will output rubbish.

All reranking results in this experiment are first overfetched by a factor of 4, then reranked and finally top_k results are chosen. This allows reranker to do its job. Without overfetching there will be no performance difference.

Experiment	Vector Reranked@10	Vector Reranked@5	FTS Reranked@10	FTS Reranked@5
modernbert-untrained-control-5	N/A	9.70	N/A	4.30
marco-minilm-untrained-control-10	9.20	N/A	4.75	N/A

Comparison with a dedicated reranker model

Similarly, a dedicated reranker model, trained with many optimization tricks on a huge corpus of diverse data improves search performance significantly.

Experiment	Vector Reranked@10	Vector Reranked@5	FTS Reranked@10	FTS Reranked@5
answerai-colbert-small-v1-control_5	N/A	59.70	N/A	50.45
answerai_colbert_control_10	68.25	N/A	58.15	N/A

Training models from scratch

All models except the first rows in both the tables are trained from scratch. The naming convention used is `{reranker_arch}{base _ arch}{num_ negatives}_{num epochs}`

Experiment	Vector Reranked@5	FTS Reranked@5	Hybrid@5
colbert-ModernBERT-base-1-neg-5-epoch_5	55.10	46.80	55.30
colbert-ModernBERT-base-5-neg-1-epoch-5	55.50	47.30	55.50
colbert-ModernBERT-base-2-neg-4-epoch-5	**60.85**	50.50	**61.10**
colbert-ModernBERT-base-1-neg-1-epoch-5	43.35	37.40	38.55
colbert-ModernBERT-base-2-neg-1-epoch-gooaq-1995000	54.10	46.95	53.25
colbert-MiniLM-L6-H384-uncased-1-neg-1-epoch_5	44.40	36.50	38.00
cross-encoder_minilm_tuned_1_epoch_2M_unseen_5	50.95	50.10	51.50
cross-encoder_minilm_tuned_5_epoch_2M_unseen_5	51.40	**51.55**	51.40
colbert_minilm_tuned_1_epoch_2M_unseen_5	43.25	36.35	37.50
cross-encoder_Modernbert_tuned_5_epoch_2M_unseen_5	56.40	53.75	57.40
cross-encoder_Modernbert_tuned_1_epoch_2M_unseen_5	**61.05**	**53.95**	**62.30**

Experiment	Vector Reranked@10	FTS Reranked@10	Hybrid@10
colbert-ModernBERT-base-1-neg-5-epoch_10	62.65	54.10	62.40
colbert-ModernBERT-base-5-neg-5-epoch-10	65.20	55.40	64.95
colbert-ModernBERT-base-5-neg-1-epoch-10	62.45	54.50	62.30
colbert-ModernBERT-base-2-neg-4-epoch-10	**67.90**	56.85	**67.70**
colbert-ModernBERT-base-1-neg-1-epoch-10	49.10	43.00	44.20
colbert-ModernBERT-base-2-neg-1-epoch-gooaq-1995000	61.20	53.90	60.55
colbert-MiniLM-L6-H384-uncased-1-neg-1-epoch-10	50.45	41.15	42.30
cross-encoder_minilm_tuned_1_epoch_2M_unseen_10	53.00	55.95	53.05
cross-encoder_minilm_tuned_5_epoch_2M_unseen_10	54.15	57.20	53.05
colbert_minilm_tuned_1_epoch_2M_unseen_10	52.15	44.40	46.00
cross-encoder_Modernbert_tuned_5_epoch_2M_unseen_10	61.75	**60.85**	62.10
cross-encoder_Modernbert_tuned_1epoch_2M_unseen_10	**66.15**	**60.70**	**67.20**

Key Findings

1. **Reranking Impact Analysis**:

   Vector search performance improved significantly with reranking, showing up to **12.3%** improvement at top-5 and **6.6%** at top-10

   FTS results saw mixed improvements from reranking, with some models showing up to **20.65%** gains and others suffering performance degradation

2. **Model Architecture Comparison**:

   ModernBERT-based models demonstrated the strongest performance, particularly the 2-neg-4-epoch variants which achieved near-baseline performance at top-10 (-0.51%)

   MiniLM-based models consistently underperformed compared to their ModernBERT counterparts. This is expected as these models are tiny. They still end up improving base performance by a lot—**which highlights that reranking is a low-hanging fruit.**

   Cross-encoder models showed competitive performance in FTS reranking, outperforming colbert-based approaches

3. **Training Parameter Effects**:

   Increasing negative samples generally improved model performance across architectures

   The 2-neg-4-epoch combination provided the best balance of performance to training cost

   Epoch count showed diminishing returns beyond 2 epochs for most models. This could also highlight overfitting or catastrophic forgetting.

💡

Note: This isn't a primer on "how to train best models". With slight tuning using standard random/grid search, the same models with the same configurations should show much better performance.

But that's beyond the scope of this blog. This blog primarily deals with how reranking works, and if it is right for your use-case

1. **Best Non-Finetuned Models**:

   For top-5: cross-encoder_Modernbert_tuned_1_epoch_2M_unseen_5 (Vector: 61.05, FTS: 53.95, Hybrid: **62.30**)

   For top-10: cross-encoder_Modernbert_tuned_1epoch_2M_unseen_10 (Vector: 66.15, FTS: 60.70, Hybrid: **67.20**)

This shows models trained only on a few million data points (including augmentation) can perform similar to a dedicated reranker model trained on a much larger corpus and also tuned carefully.

Hybrid Search Variability:

Hybrid search performance varied based on individual result quality from component searches.

In cases where an individual result perform poorly, hybrid seach performs even worse than vector search. **But in cases where both FTS and vector search result sets are decent, hybrid search performs better than any of the individual search types**.

Fine-tuning existing rerankers

With enough high-quality data, you can also improve the performance of an existing reranker model by fine-tuning as demonstrated by the finetuned answerai-colbert models which consistently outperformed all other configurations

Baseline performance of dedicated reranker model

Experiment	Vector Reranked@10	Vector Reranked@5	FTS Reranked@10	FTS Reranked@5
answerai-colbert-small-v1-control_5	N/A	59.70	N/A	50.45
answerai_colbert_control_10	68.25	N/A	58.15	N/A

Finetuned performance

Experiment	Vector Reranked@10	Vector Reranked@5	FTS Reranked@10	FTS Reranked@5
answerai-colbert-small-v1-1-neg-1-epoch_5	N/A	62.35	N/A	51.30
answerai-colbert-small-v1-1-neg-1-epoch_1	70.00	N/A	58.35	N/A

Should you train a reranker from scratch or fine-tune them?

It depends on the size and quality of the dataset. Just like other tasks, the rules of thumb for training from scratch vs fine-tuning apply to rerankers as well.

You should train from scratch if:

If there are no rerankers available for the base model of your choice. For example, `ModernBert` itself is not a reranker model, we use it as a base for

creating cross-encoder and colbert architecture reranker models. Hence, we train these architectures from scratch.

You should fine-tune a reranker if:

A reranker of your choice already exists, but you suspect your dataset might have become specialised enough, or the distribution might have drifted from the original datasets used to train these rerankers.

A general rule to remember is that fine-tuning an existing architecture converges much faster. Another thing to be careful about is Catastrophic forgetting in Large Language Models (LLMs). It refers to the phenomenon where a model, after being trained on a new task, forgets or significantly degrades its performance on previously learned tasks

Tradeoffs

As the 'no free lunch theorem' states, no single optimization algorithm is universally superior across all possible problems; there are tradeoffs involved when reranking retrieval results.

Latency vs Performance

For reference, here's **the performance improvements (from above tables)**

Reranker	Vector Reranked@10	FTS Reranked@10	Latency
No Reranker (Baseline)	60.30	39.25	28.05
ColBERT ModernBERT	67.90	56.85	60.88
ColBERT MiniLM	50.45	41.15	31.03
ColBERT Answer V1	68.25	58.15	37.83
Cross-Encoder ModernBERT	66.15	60.70	68.22
Cross-Encoder MiniLM	54.15	57.20	24.79

This chart shows abalation table comparing latency at various top_k

Reranker	Model Size	Limit 20 (ms)	Limit 50 (ms)	Limit 100 (ms)	Slowdown Factor (Limit 20)
No Reranker (Baseline)	N/A	28.05	18.32	31.89	1.00x
ColBERT ModernBERT	150M	60.88	114.70	243.85	2.17x
ColBERT MiniLM	6M	31.03	55.07	105.00	1.11x
ColBERT Answer V1	33M	37.83	67.53	135.76	1.35x
Cross-Encoder ModernBERT	150M	68.22	144.56	269.70	2.43x
Cross-Encoder MiniLM	6M	24.79	48.99	82.56	0.88x

These numbers have been calculated on **L4 GPU, which is comparable to a T4**. The reasoning is that these are the cheapest and most readily available GPUs.

> 💡 These numbers can be further optimized using various techniques, which can cause major gains in performance. Some simple optimizations:
> * torch.compile
> * Quantization
> * Using better GPUs

Who should and should NOT use rerankers?

> As expected, rerankers end up adding *some* latency to the query time. Depending on the use-case this latency might be significant.
>
> If your goal is to **always have latencies less than, say, 100ms, you'll need to carefully choose if a reranker is worth the penalty**.
>
> All other more **general cases where an added latency of at most ~50ms doesn't do much damage, should definitely exploit reranking** before thinking about more disruptive techniques like fine-tuning or switching to a new embedding model.

If your use-case does not have a hard latency limit, you can stick to using CPUs, as even without acceleration, reranking models only add a few 100 ms.

Reranker	Vector Reranked@10	FTS Reranked@10	Latency
No Reranker (Baseline)	60.30	39.25	53.85
ColBERT ModernBERT	67.90	56.85	246.44
ColBERT MiniLM	50.45	41.15	107.90
ColBERT Answer V1	68.25	58.15	137.59
Cross-Encoder ModernBERT	66.15	60.70	276.01
Cross-Encoder MiniLM	54.15	57.20	83.02

API based rerankers

This blog covers only OSS models, but you can also choose to use an API based reranker. The best way to access them is by evaluating manually for accuracy and speed. There are various unknows in API-based approach that won't allow direct comparison in this experiment:

1. The latency varies a lot.
2. The architecure of these models is unknown
3. You generally can't train or fine-tune them on your own data

In some of our experiments we've seen very high accuracies using API-based rerankers, but with unstable latencies. To find more details you can check out this **example** on our docs

Reproducibility

> The code and instructions to reproduce this can be found **here**.
>
> All of the trained artifacts can be found **here**.

SHARE WITH FRIENDS

AYUSH CHAURASIA

NO COMMENTS

Start the conversation

Become a member of **LanceDB Blog** to start commenting.

Sign up now

Already a member? Sign in

FEATURED

Lance File 2.1: Smaller and Simpler

ENGINEERING 10 MIN READ

Designing a Table Format for ML Workloads

ENGINEERING 14 MIN READ

The Lancelot Chronicles: Your Quest in the Realm of Open Source

ANNOUNCEMENT 3 MIN READ

Lance v2: A columnar container format for modern data

ENGINEERING 12 MIN READ

Sign up Documentation Join Discord Community YouTube

LanceDB