Qdrant

Fine-Tuning Sparse Neural Retrievers for E-Commerce

Is Not That Scary (And Often Worth It)

Evgeniya Sukhodolskaya
Senior Developer Advocate @ Qdrant

What's in This Talk for You

1 **Sparse Neural Retrieval**

what it is, why use it

2 **Fine-Tuning on Amazon ESCI**

making it work for e-commerce

3 **The Framework**

adapt it to your own catalog

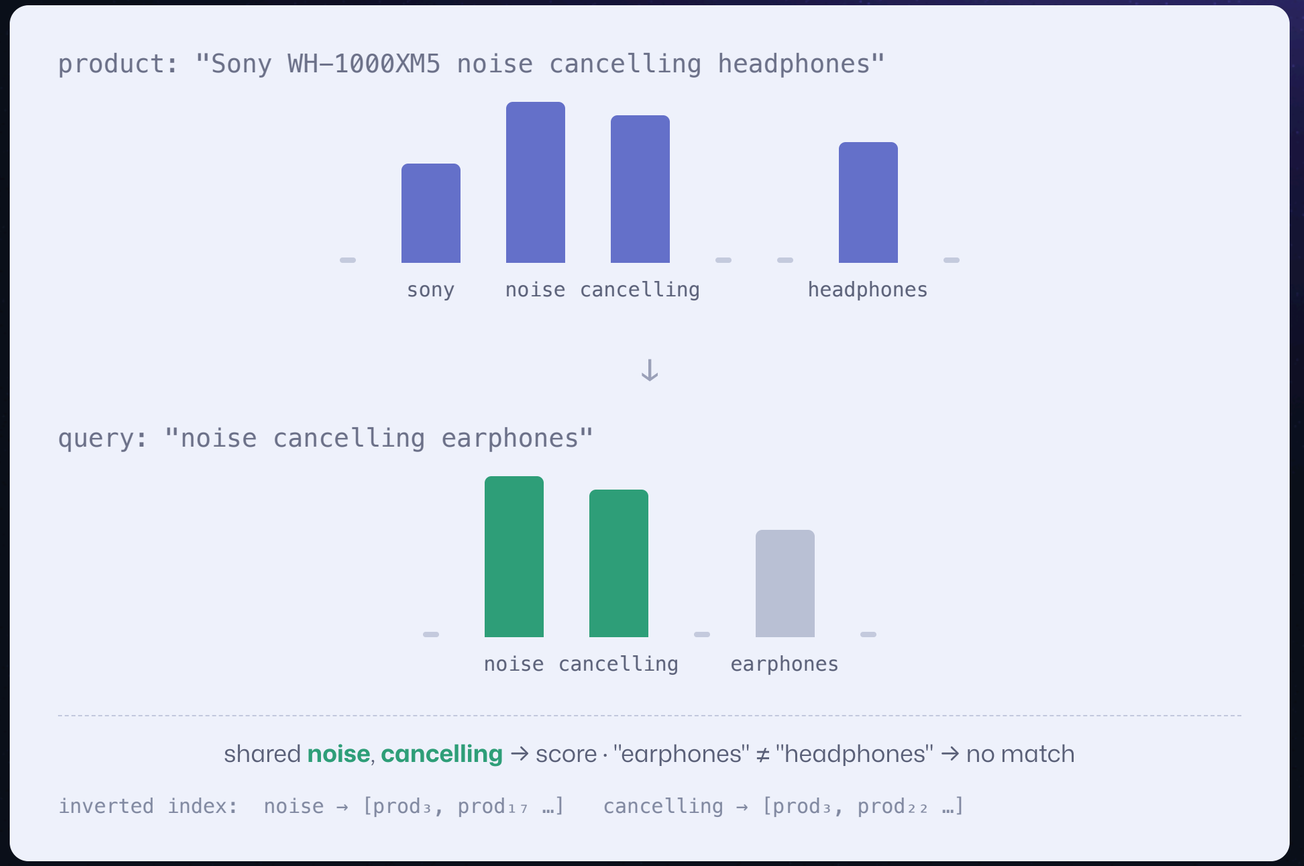
PART 1

Sparse Neural Retrieval

What Is a Sparse Vector, and What Is Sparse Retrieval

Qdrant

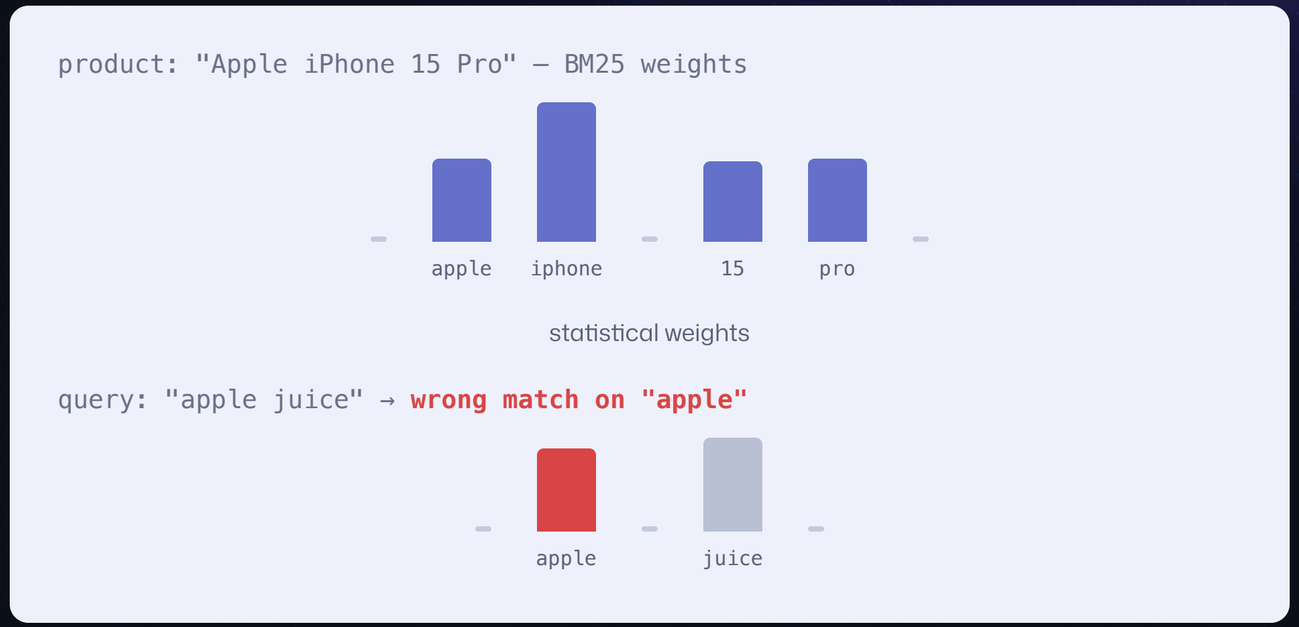
- **Sparse vector:** a large array of numbers, mostly zeros
- For text: each non-zero dimension = a vocabulary term + a weight reflecting its importance
- **Dot product** between query and product vectors → relevance score. Only shared terms contribute
- Rank all products by score → **retrieval**
- **Inverted index:** only products sharing a term with the query get scored → fast at scale



BM25 On Sparse Vectors. And What BM25 Sparse Vectors Can't Do

Qdrant

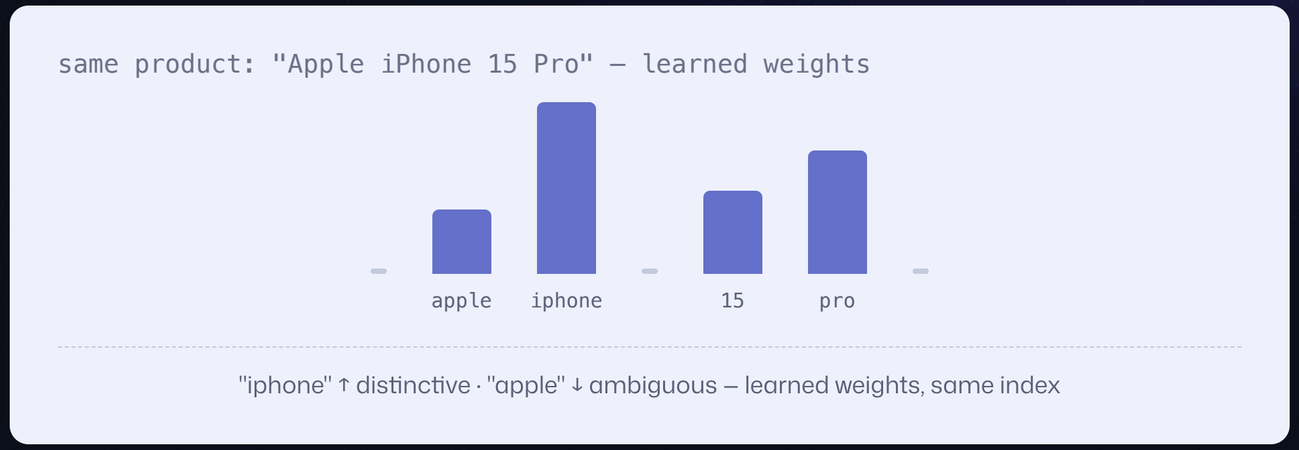
- BM25 per-term weights (TF, IDF, length norm) go into the sparse vector non-zero dimensions
- Same inverted index, same dot-product scoring
- Limitation: weights are **statistical**, not semantic
- Additionally, no synonymy: "earphones" and "headphones" → no overlap, no match



Sparse Embeddings: Same Index, Learned Weights

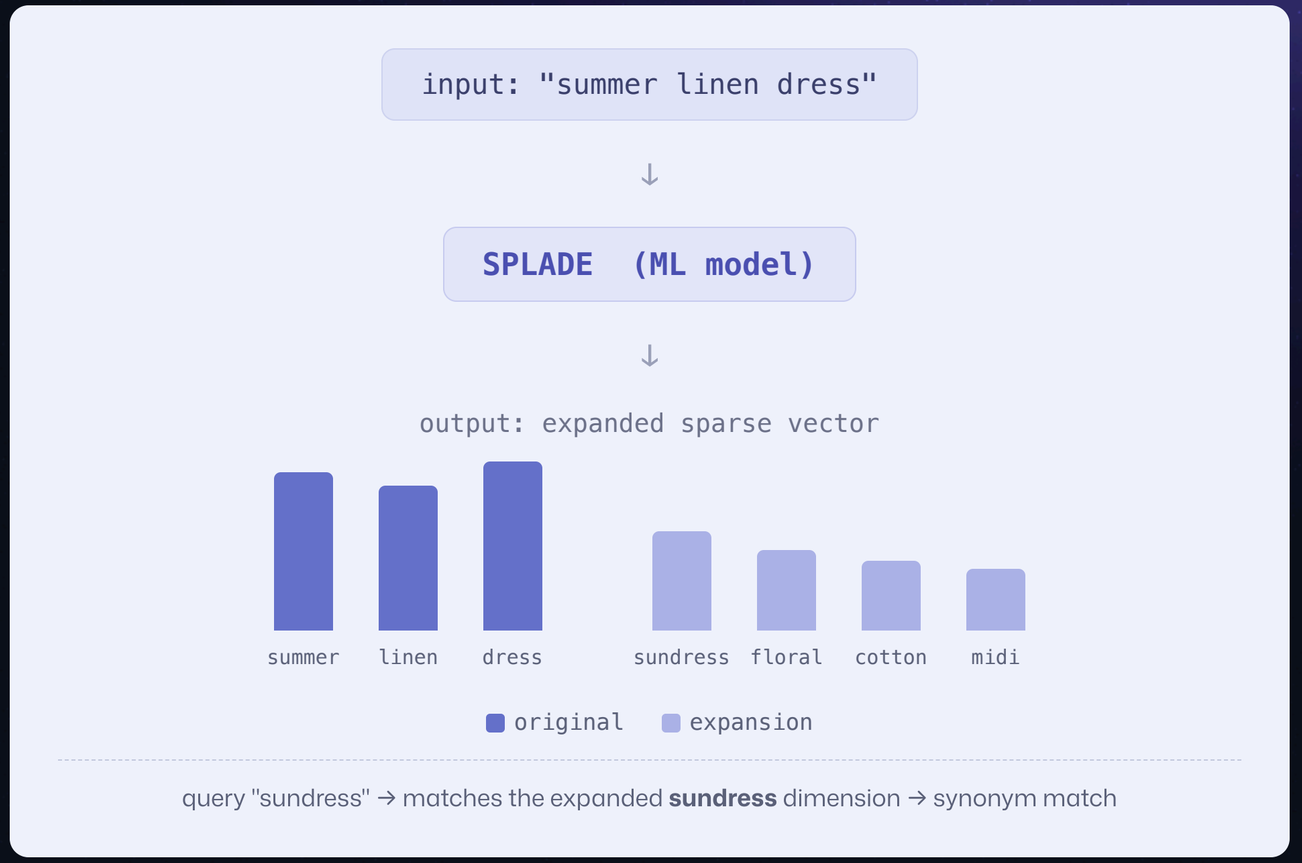
Qdrant

- A sparse **embedding** is the same vocabulary-sized vector, but the weights are **learned from data**, not computed from corpus statistics
- Same inverted index, same exact matching
- But the ML model decides which terms matter and how much for sparse (neural) retrieval



SPLADE: Sparse Lexical and Expansion Model

- The best-known sparse neural retrieval model
- Learned weights per term from MSMARCO
- Also **expands:** adds related terms that the text never contained
- So both document and query get expanded
- Result: **synonym matching** through the same inverted index

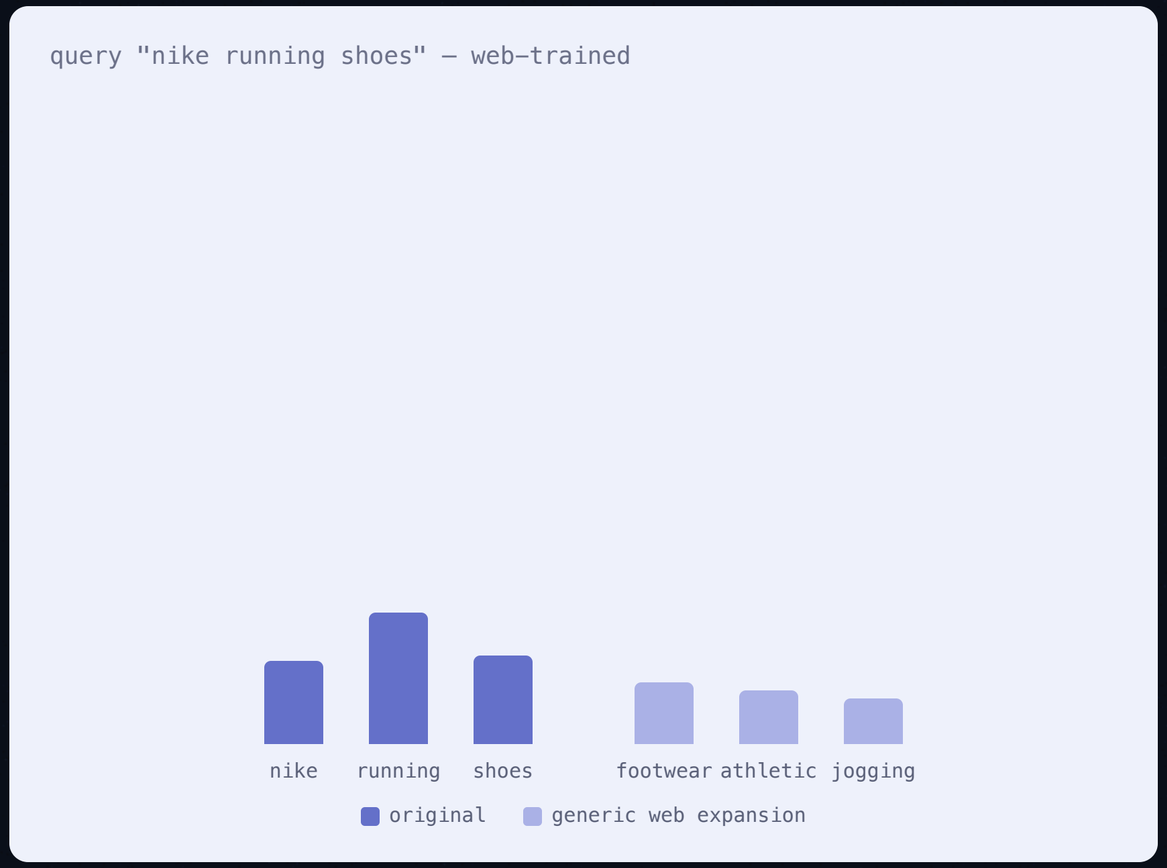


SPLADE for E-Commerce

Qdrant

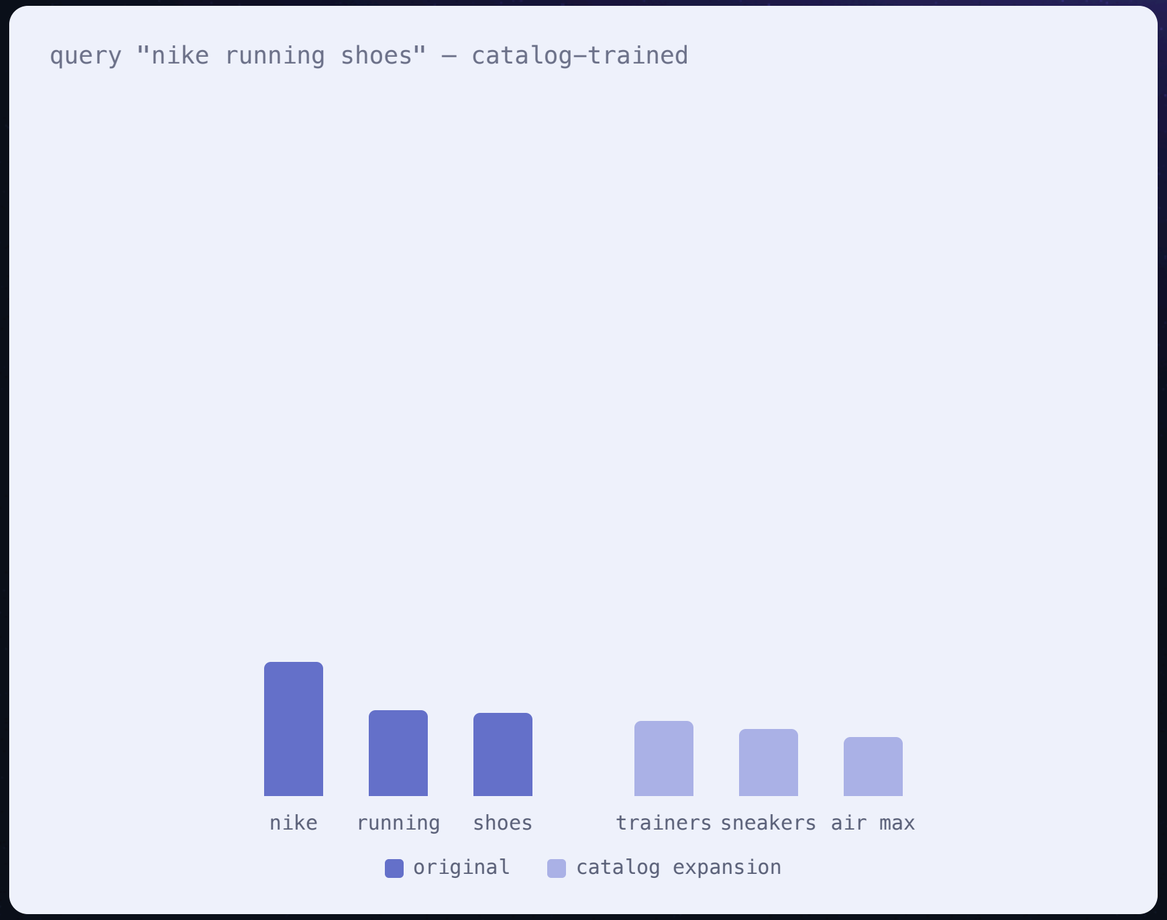
Off-the-shelf

From SentenceTransformers, Qdrant FastEmbed, Cloud Inference. Often trained on **MS MARCO**: web queries, Wikipedia passages. Expansion and weights reflect the **web**, not your **catalogs**.



Fine-tuned

Adjust to **your data** through training. Expansion and weights learn **your catalog's** language. But fine-tuning...? Is it worth it?



PART 2

Fine-Tuning SPLADE on Amazon ESCI

Kudos to Thierry Damiba

Fine-Tuning Sparse Embeddings
for E-Commerce

Qdrant

1 Why Sparse Neural

→

2 Setup & Training

→

3 Evaluation & Hard Negatives

→

4 Specialization vs Generalization

→

5 The Framework

qdrant.tech/articles/sparse-embeddings-ecommerce – scan a part to read it

The Stack

Qdrant

TRAINING

SentenceTransformers v5

`SparseEncoder` support was added in v5

INDEXING

Qdrant

SIMD-optimized inverted indexes for retrieval & mining training data

COMPUTE

Modal

Serverless GPU. Pay per second on an A100, run jobs in parallel

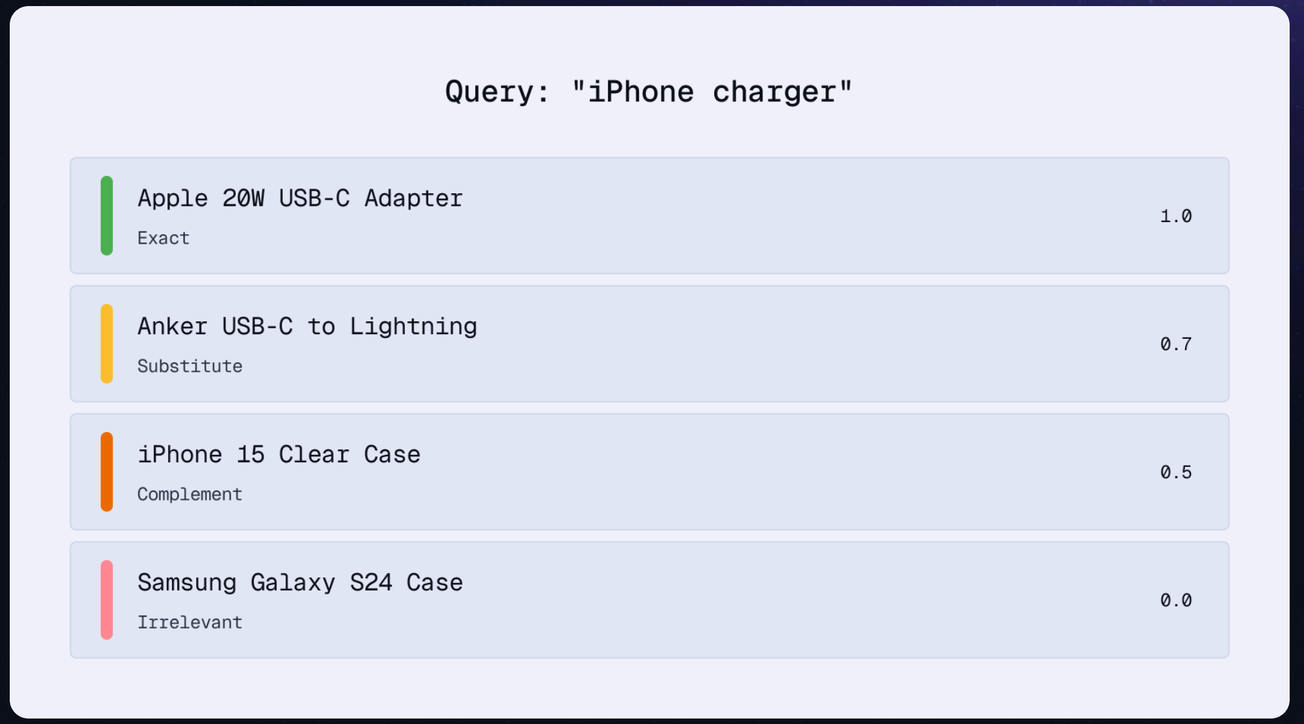
DATA

Amazon ESCI

The fourth pillar – next slide →

Amazon ESCI

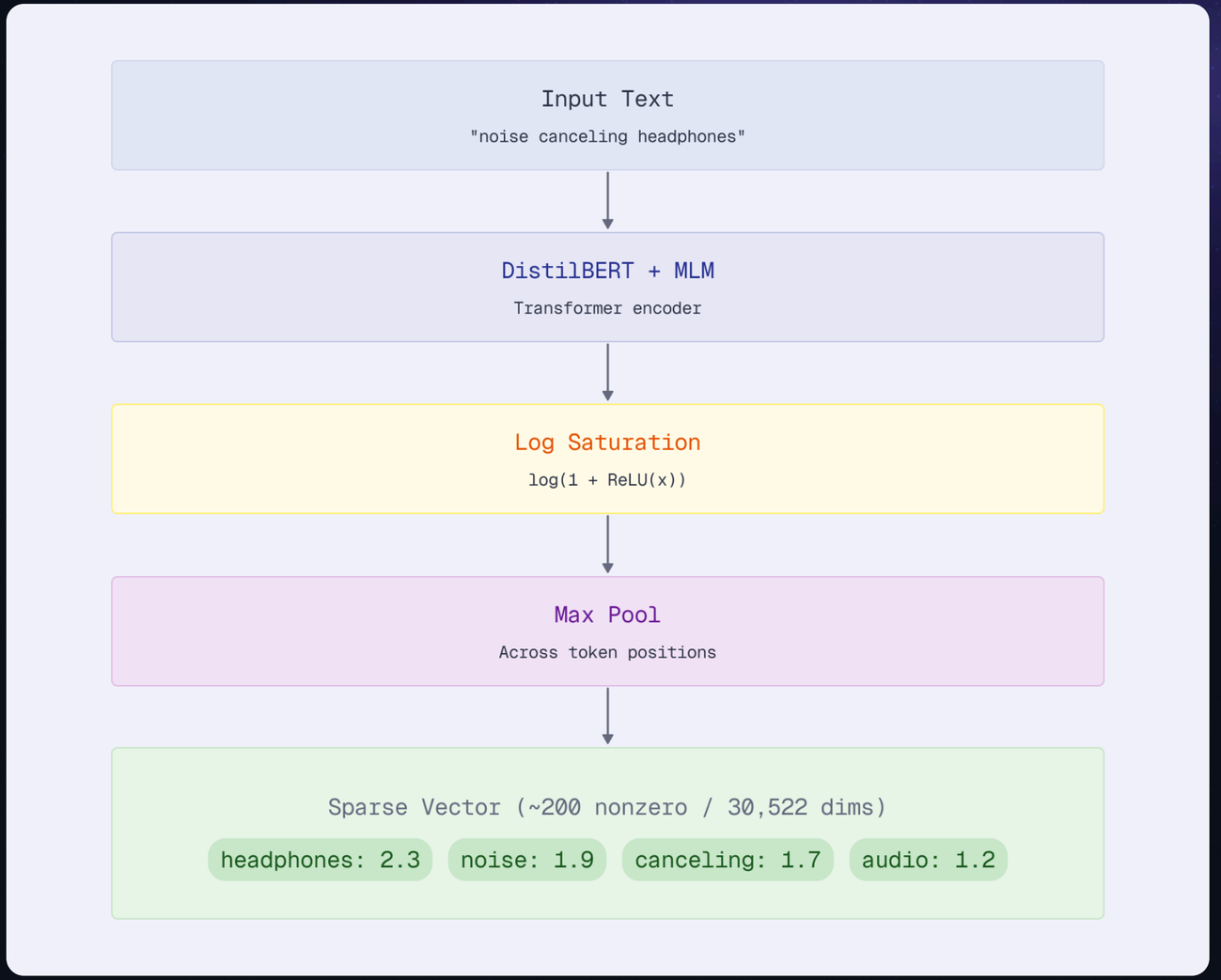
- Amazon **ESCI** (Shopping Queries Dataset, KDD Cup 2022). 1.2M+ query-product pairs, human-labeled
- Four grades: **E**xact, **S**ubstitute, **C**omplement, **I**rrelevant
- For experiments: capped at **100K pairs** → ~6 min on an A100, under $1
- **E + S → relevant examples.** On irrelevant examples later.



What's Inside SPLADE

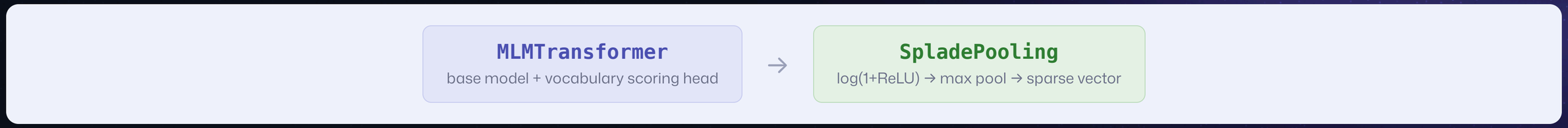
Four building blocks you configure for finetuning:

- **Base model** – a language model (DistilBERT, BERT). Plus a **vocabulary scoring (MLM) head**: per token, scores all ~30K vocab words. Source of expansion
- **Log saturation** `log(1 + ReLU(·))` per token – keeps vectors sparse
- **Max pooling** across tokens to produce one sparse vector



SPLADE Architecture Choices in SentenceTransformers v5

Qdrant



What you choose

- **Base model:** any model with an MLM head (BERT, DistilBERT, a pretrained SPLADE…)
- **Architecture:** full `splade` (query&document encoded by the model) vs `inference_free_splade` (query static lookup)
- **Pooling:** `max` (standard) or `sum`

What Thierry chose

- Base: plain **DistilBERT** – to measure how much domain fine-tuning alone buys you, starting from a general model
- Architecture: **full `splade`**
- Pooling: **`max`**

How (SentenceTransformers v5) SPLADE Models Learn

Qdrant

One loss, two jobs at once:

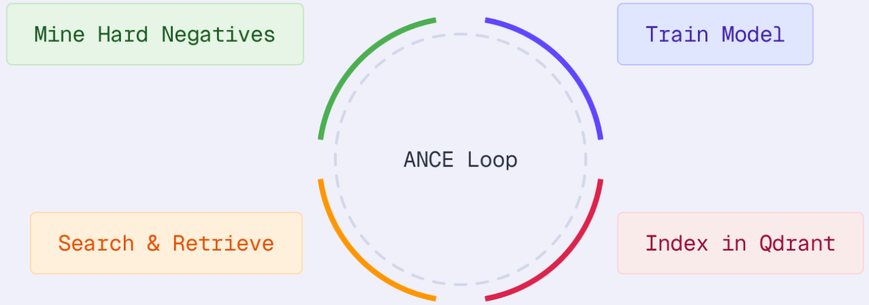
- **Rank well**: pull a query and its relevant product together, push irrelevant products away (but where to get "good" irrelevant ones?)
- **Stay sparse**: penalize lighting up too many tokens, making vectors dense

Why a penalty for sparsity? Ranking alone doesn't care: a denser vector ranks just as well, so the model would turn on every dimension and produce a dense vector.

SpladeLoss = ranking loss (SparseMultipleNegativesRankingLoss) + sparsity regularizer (FlopsLoss), with separate weights for queries and documents — a query should be sparser than a product description.

Iterative Hard (Non-Trivial) Negative (Irrelevant Products) Mining with Qdrant

ANCE-inspired



- Training on random product pairs teaches the model easy lessons – obviously unrelated products
- What helps more: **hard negatives** – products that look right to the current model but aren't
- **Qdrant's role:** after each round, index the full catalog with the current model. For each training query, run a fast sparse search. Wrong results near the top → hard negatives for the next round

The Result

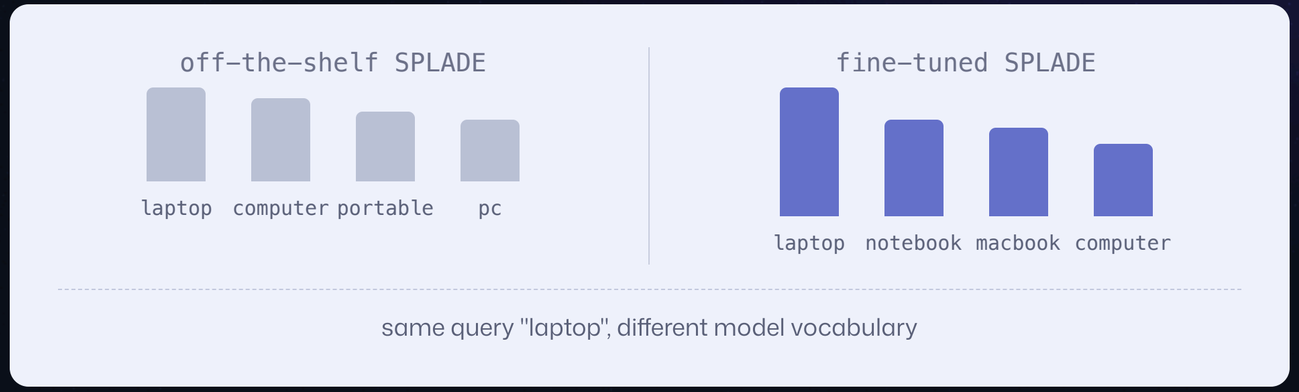
Thierry's run — ~100K ESCI training subset, 10K test queries. A signal, not a benchmark.

What actually changed after fine-tuning:

- **Expansion learned the catalog's vocabulary:**
  "laptop" → also "notebook", "computer", "macbook"
  "wireless earbuds" → "bluetooth", "airpods", "tws"
- **Term weights shifted:** brand names heavier (search "Sony" → want Sony); generic terms ("best", "cheap") lighter

nDCG@10: BM25 **0.305** · off-the-shelf **0.326** · fine-tuned **0.389**

`model on HF: Qdrant/splade-ecommerce-esci`



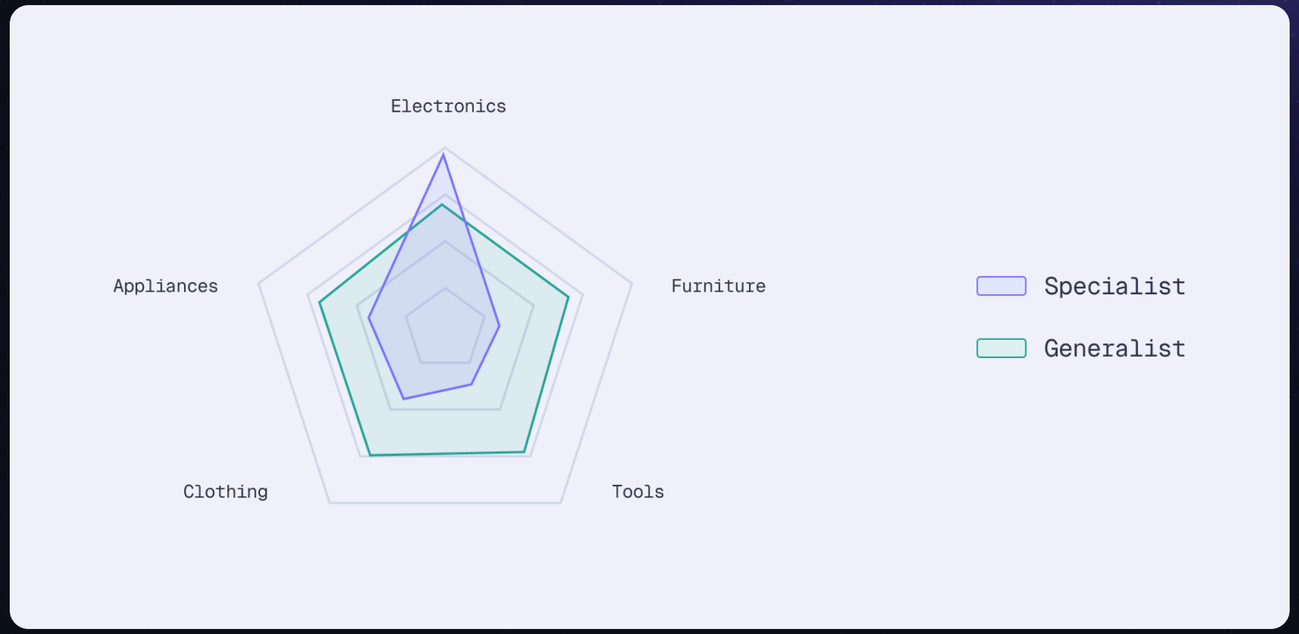
Tradeoffs: Specialization vs Generalization

Qdrant

Catalog	BM25	Fine-tuned (on ESCI)
Amazon ESCI (trained on)	baseline	**+27.5%**
WANDS (Wayfair)	baseline	**+7.9%**
Home Depot	baseline	**+10%**
General web search	baseline	**gets worse ↓**

Fix: **multi-domain training** – fine-tune on several e-commerce catalogs at once.

`model on HF: Qdrant/splade-ecommerce-multidomain`



There are more tips & tradeoffs in the series: *data balancing, query generation, training parameters, and Modal setup.*

Should I Fine-Tune?

- **Single retailer, real click* data** → fine-tune on your catalog
- **Marketplace / many catalogs** → multi-domain training
- **Little or no labeled data** → start off-the-shelf, collect clicks, fine-tune later
- **BM25 / hybrid already does the job** → don't bother

** Fine-tuning on synthetic queries adapts catalog vocabulary, but won't learn real user intent from behavior. You get part of the win.*

PART 3

qdrant-sparse-finetune

qdrant-sparse-finetune

Qdrant

These 3 lines parse your products, generate queries if you don't have them, build and train the SPLADE model with ANCE-inspired Qdrant-based mining, and save the model.

OPEN SOURCE
github.com/qdrant/
sparse-finetune

```
$ pip install qdrant-sparse-finetune
```

```
from qdrant_finetune import finetune
model_path = finetune("products.csv")
```

What's Inside

Qdrant

1

Parse your catalog

`CSV / JSON / JSONL / HF`
`→ {product_id, text}`

→

2

Find or generate queries

`your labels, or LLM writes`
`3 / product · optional`

→

3

Create SPLADE encoder

`SparseEncoder`
`from ST v5`

→

4

Train with Qdrant-based mining

`train → index → mine`
`→ repeat 3×`

→

5

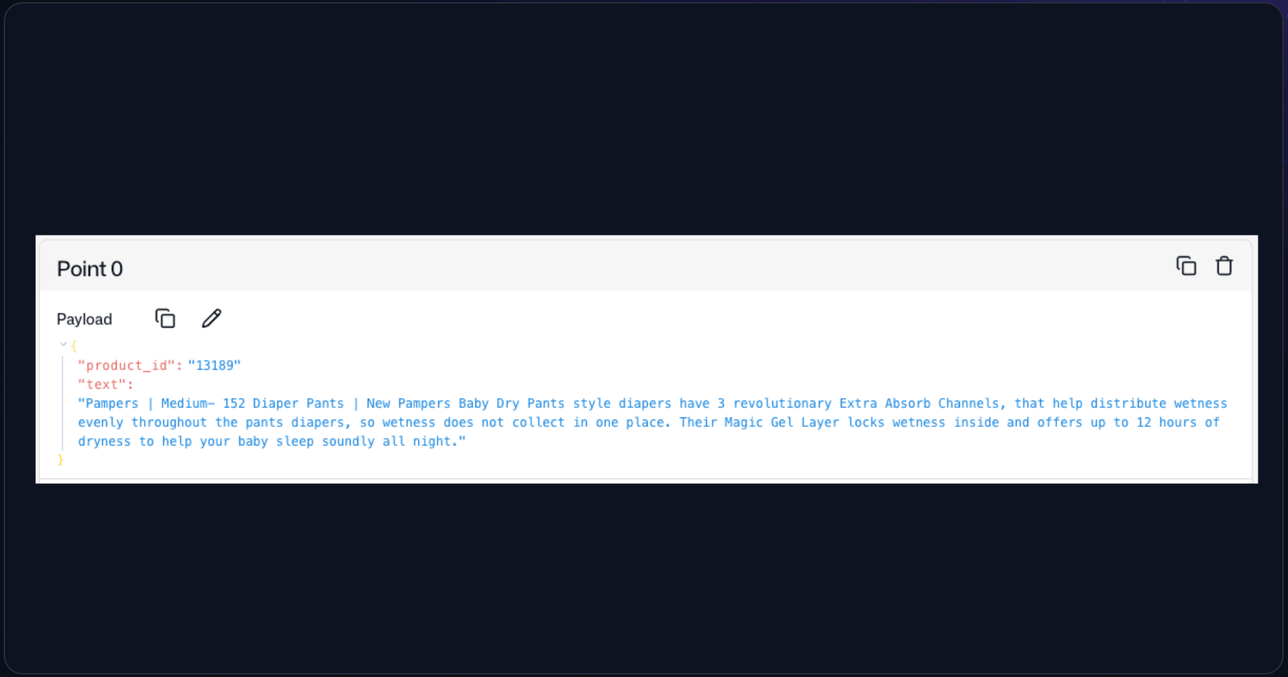
Save the model

`output_dir/`
`run_name/final/`

data model & training output

Step 1: Your Products

- Step 1 flattens each product into **one string** – what the model encodes and what the LLM sees when generating queries
- Columns are auto-detected by name, but the list is narrow. For safety, pass **`text_fields`** explicitly. Print one product before training
- **ID column must be named** `product_id`, `id`, `asin`, `sku`, or `item_id`



Step 2: Your Queries

Qdrant

Do you have labeled query-product pairs?

Yes → Path B

CSV `query, product_id, label`
E/S/C/I or 2/1/0 auto-detected

No, just run → Path A

omit `--queries`, set `OPENAI_API_KEY`
(or any litellm provider) → LLM writes 3/product

No, want to review → Path C

`generate-queries` → open JSONL → delete bad rows → pass as `--queries`

Step 3: Architecture Choices

Qdrant

Defaults

- **`base_model: "distilbert/distilbert-base-uncased"`**
  lightweight general language model
- **`architecture: "inference_free_splade"`**
  query = vocab table lookup, no GPU
- **`pooling_strategy: "max"`**
  standard SPLADE pooling
- **`query_regularizer_weight: 5e-5`**
  `document_regularizer_weight: 3e-5`

Alternatives / advice

- **Base model:** any model with an MLM head. `base_model` & `architecture` go together
- For e-commerce, it's better to pick the strategy **"splade"** aka full neural query encoder. A lookup table can't tell "apple iphone" from "apple fruit"
- pooling_strategy: "sum" could be an option, as in the original SPLADE 2021 (but it worked worse than "max").
- **Sparsity controls:** higher → fewer terms, faster index. Lower → more recall

Step 4: The ANCE-Inspired Loop: mining "hard" irrelevant products with Qdrant

Qdrant

Each **iteration** (from round 2):

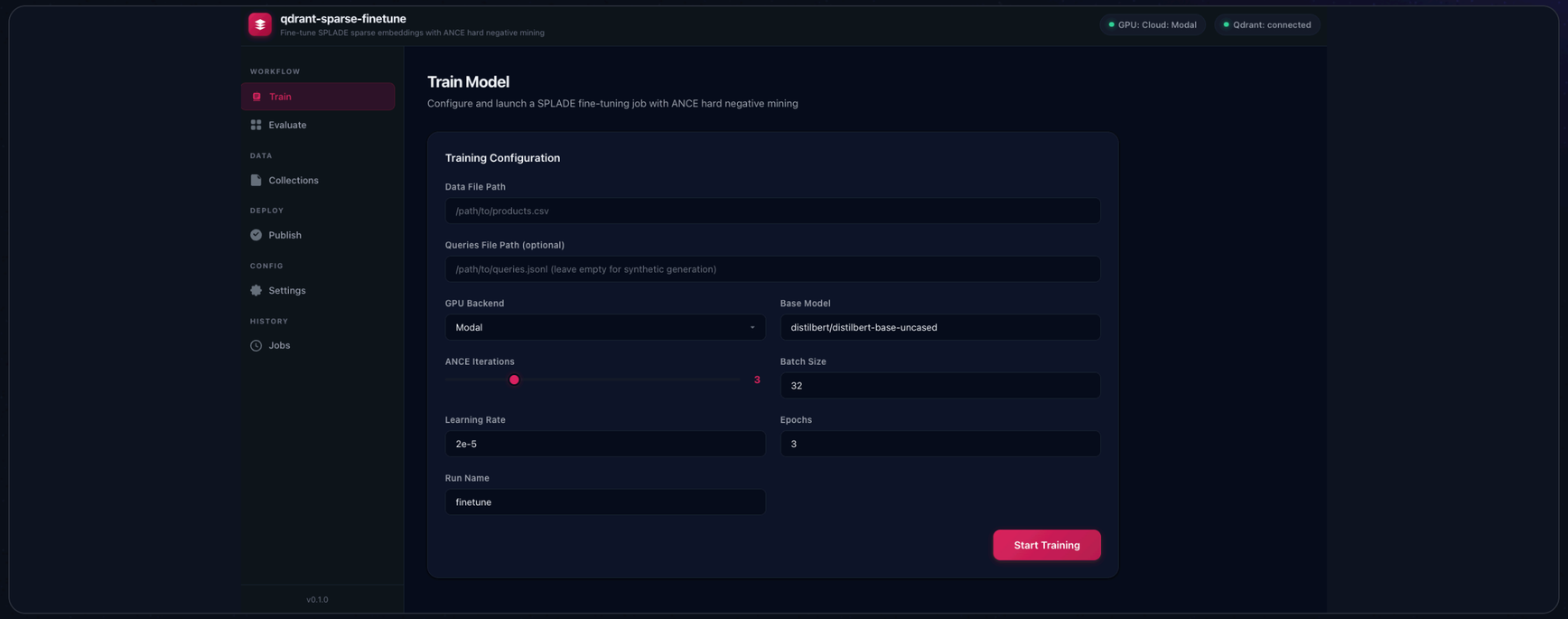
1. Temp Qdrant collection `{name}_ance_N`
2. Index full catalog with current model
3. Search each training query, return top-k
4. Filter out known positives
5. Sample `num_negatives` from top-k
6. Delete temp collection
7. Train on anchor (query) + positive (relevant) + hard negatives

sample_strategy	What	When
`"top"` (default)	Hardest: rank 1, 2, 3	Strongest signal; false-neg risk
`"mixed"`	Top half + middle	Near-dup catalogs, sparse labels
`"random"`	Random from top-k	Safest; weakest signal

Storage: *`n_products × avg_nonzero_dims × 8 bytes`* – ~4 MB for 5K products, ~80 MB for 100K.

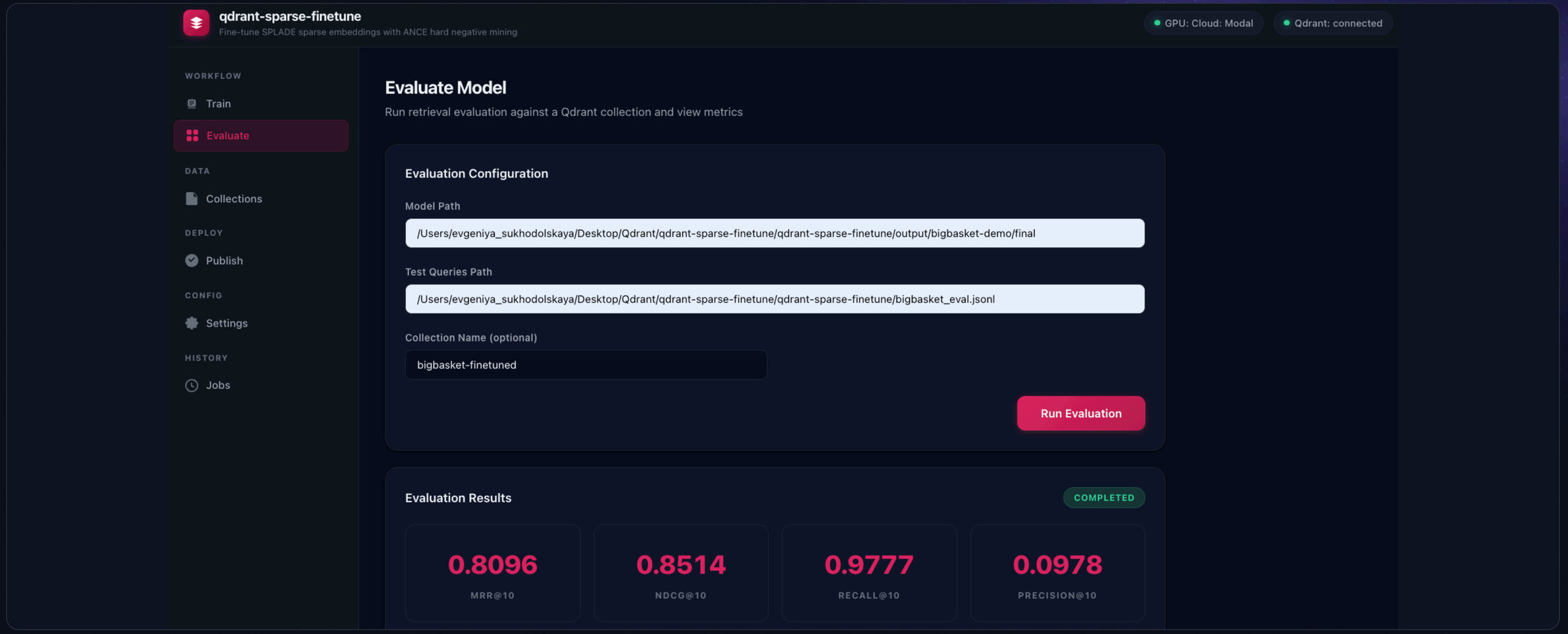
Three Ways to Run It

	Python API	CLI	Dashboard
How	`from qdrant_finetune import Trainer`	`qdrant-finetune pipeline`	`qdrant-finetune studio`
Runs on	Local machine	Modal cloud GPU	Modal cloud GPU
Hardware	CPU or CUDA	A10G serverless	A10G serverless



Full 3-iteration run on 5K products ≈ **$0.80** with Modal & synthetic query generation.

Evaluation



Auto-detected labels: ESCI (E/S), WANDS (2/1), binary (1/0). ⚠ Built-in post-training eval runs on **training** queries by default – run a **held-out** split for an honest number.

What I Tried: BigBasket Dataset

5,000 grocery products. GPT-4o-mini queries. $1. A smoke test.

Metric	BM25	SPLADE off-shelf	SPLADE fine-tuned
`recall@10`	0.9575	0.9750	**0.9777**

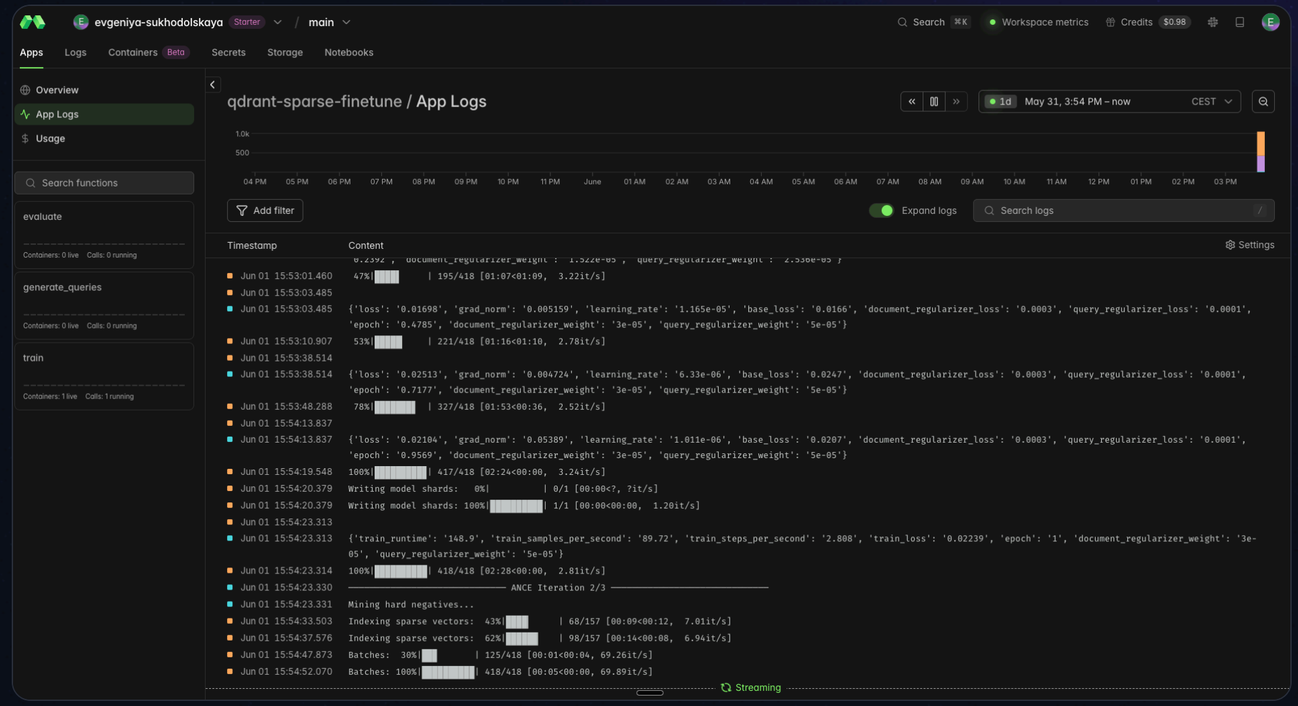
Held-out: 1,482 queries, 5K corpus. Nearly flat — baseline already ~96% recall (synthetic queries built from product text). The case where it's worth investing in non-synthetic data.

Hard negatives Qdrant mining found

query: "pampers medium diapers" → Pampers, wrong size · competitor, right size · adult diaper, wrong audience

Random sampling of irrelevant products rarely produces these.

Fine-tuning gained catalog domain terms in the documents, but **synthetic queries didn't close the "wash → soap" gap.** For that, you need **real click logs.**



Before You Run qdrant-sparse-finetune: Mind, It's Raw-ish:)

Qdrant

1. **ID column** must be `product_id` / `id` / `asin` / `sku` / `item_id`, or it breaks silently
2. **Pass `text_fields`** explicitly. Print a sample product first
3. **Post-training eval is on training queries** by default. Run a held-out eval
4. **...?**

Qdrant

Thank you.

Evgeniya Sukhodolskaya · Senior Developer Advocate @ Qdrant

My LinkedIn

Thierry's LinkedIn

Workshop · 3–5 pm

Meetup · 6–9 pm

Tomorrow, June 11 · Merantix AI Campus

Vector Space Workshop 3–5 pm (multimodal search with Qdrant, DeepMind, AskNews & Benzinga)

Vector Space Meetup 6–9 pm (meetup with cognee, n8n, deepset & LlamaIndex + Qdrant CTO on a future of vector search)