PRACTITIONER REPORT · RETRIEVAL-AUGMENTED GENERATION

Optimizing a Hybrid Retrieval System for Accuracy, Speed & Cost, All at Once

A few cheap, structural changes made a production RAG stack about 32 times smaller, several times faster, and more accurate, all at once. Most of the obvious heavy fixes did nothing.

Dezso Mezo DField Solutions · **dfieldsolutions.com**

Domain: long, hierarchically-structured documents **Method:** measured, single-variable A/Bs

Framing: concept-only, ratios not absolutes

ABSTRACT

This is a writeup of how we tuned a production retrieval-augmented generation (RAG) system. It searches a big corpus of long, tree-structured documents by combining two signals, a dense embedding one and a lexical BM25-style one, then has an LLM write cited answers. The goal was unusual. We wanted better accuracy, faster queries, and lower memory cost all at once, when normally you trade one for another.

It worked because the biggest wins were cheap and structural, not compute-heavy. One-bit vector quantization with an exact float rescore made the index about 32 times smaller. An int8 encoder made the model 4 times smaller and 5 times faster, with identical output. And a handful of lexical and structural ranking priors cost nothing at query time. The lesson that mattered most: the heavy fixes we expected to help, a cross-encoder reranker, a second fine-tune, query rewriting, did nothing or hurt once we measured them properly. The figures below are live, so play with them.

Results at a glance

Nine levers, three axes. The table below is the short version; the rest of the paper makes each row concrete and lets you poke at the mechanism behind it. The gist: the stack went from too large to serve without a dedicated vector database to something that fits in RAM on an ordinary box. Latency dropped several times over, and fairly graded answer quality stayed high. All at once.

32×
smaller vector index, so the separate vector database is gone

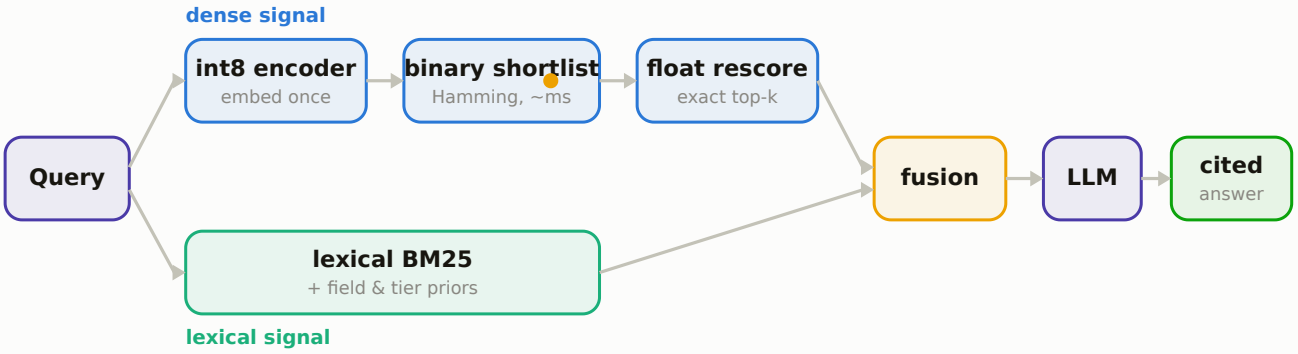
4×
smaller embedding model via int8, with byte-identical output

5×
faster query embedding, the old per-query bottleneck

~1ms
binary shortlist search once the index fits in RAM

Figure 1 · The query path, end to end

Two retrieval signals run in parallel, then fuse. The dense side is the part this report shrank and sped up.



A query is embedded once by the int8 encoder, which feeds the binary shortlist and exact rescore. In parallel, a lexical BM25 pass with the field and tier priors scores the same corpus. The two are fused, and the LLM writes a cited answer from the top results.

Figure 2 · The optimization ledger: every lever, by axis and outcome

Hover a row for the mechanism. Green means a clear win; violet means it helps on both axes at once.

COST / RAM	**1-bit vector quantization + rescore** index ~32× smaller; exact ranking kept; vector DB removed	●
COST / RAM	**int8 encoder** model ~4× smaller	●
SPEED	**int8 encoder** query embedding ~5× faster; identical accuracy	●
SPEED	**binary vector search** shortlist in ~ms	●
ACCURACY	**lexical field weighting** large lift on “query ≈ section title” cases	●
ACCURACY	**home / sub-domain tier routing** large lift; kills cross-variant bleed	●
ACCURACY	**boilerplate demotion** small, consistent lift	●
ACCURACY	**domain fine-tune (once)** clear lift; a second round was neutral	●
BOTH	**drop the cross-encoder reranker** cheaper AND more accurate	●

Reading it: *the two levers that hit* **both** *axes at once, binary quantization and dropping the reranker, are the ones that pay off most. They make the system cheaper and better at the same time, the combination you're usually told you can't have.*

§1

Cost & footprint

The full-precision embedding matrix ate most of the memory. At scale it didn't fit in an ordinary machine's RAM, which forced a dedicated vector database, a cost centre of its own. The fix wasn't a bigger box. It was a change in how the vectors are stored and searched.

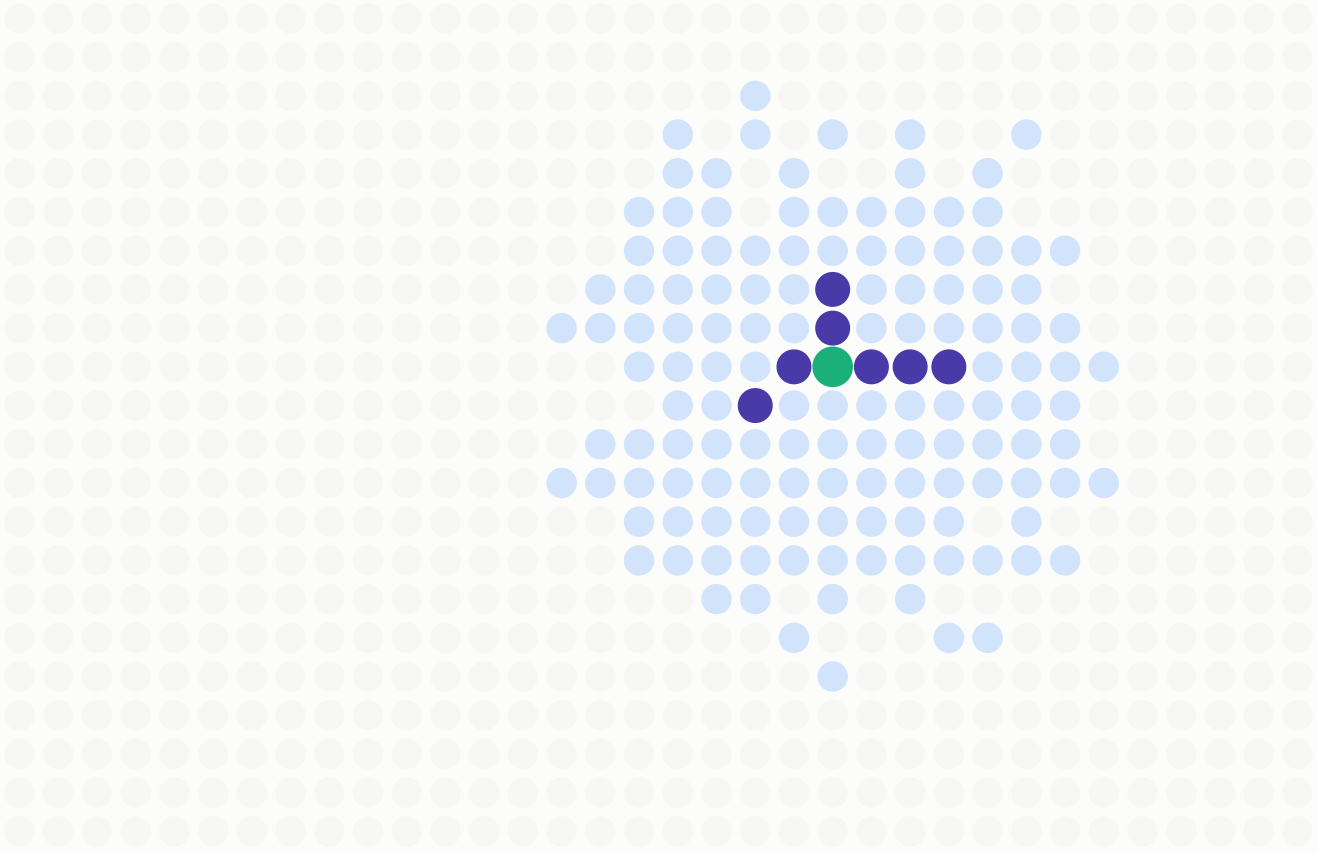
1.1 One-bit vector quantization + two-stage rescore 32× smaller

Keep just **one bit per embedding dimension**, the sign of each component. Then search in two stages. First, rank every record by **Hamming distance** to the binarized query. That's a bitwise XOR and a hardware population count, so it produces a wide candidate pool in milliseconds. Second, read the full-precision vectors for that pool only. They're memory-mapped, so just the rows you touch load, and you rescore them with exact cosine for the final top-k.

The index comes out about **32 times smaller**, and the ranking is **identical** to exact cosine: all of the exact top-k survives a modest candidate pool. Press play to watch the two stages.

Figure 3 · Two-stage retrieval: binary shortlist → exact float rescore

1024 corpus vectors. Stage 1 keeps a wide pool by cheap Hamming distance. Stage 2 rescores only that pool in full precision.



Why it works: *for the shortlist you only need the right order, and the gap between the top candidates is wide, so sign quantization keeps it. The float rescore of a few hundred candidates then restores exactness. This only breaks down for queries far off the data manifold, which real queries aren't.*

GOTCHA

A naive popcount is slow, and at small scale it can be *slower* than a BLAS float matmul. Use a hardware popcount. And keep in mind the real win here is **memory**, not raw speed. The speed follows once the whole corpus fits in RAM.

1.2 int8 encoder, and why *not* 1-bit for the model

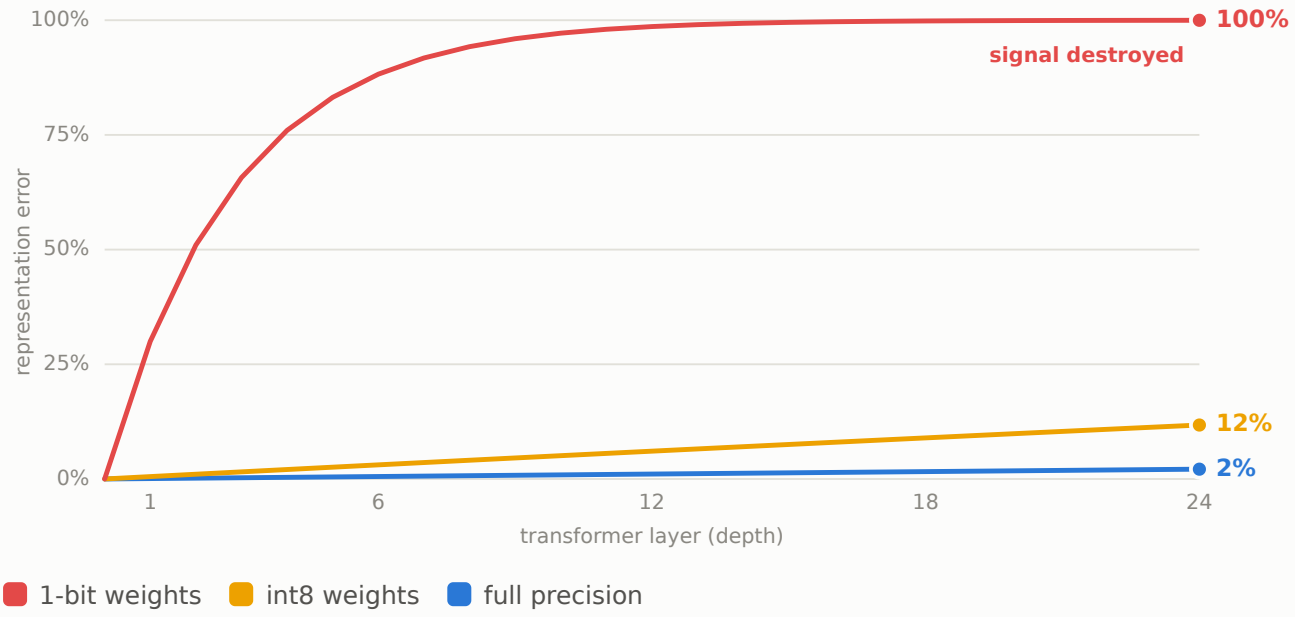
4× smaller · 5× faster

We ran dynamic int8 quantization on the model's linear layers, which hold most of the parameters and compute, and left the tokenizer and pooling at full precision. The model came out **about 4 times smaller** and **about 5 times faster** on CPU, and retrieval was **byte-for-byte identical**. The int8 and full-precision query cosines matched at roughly 1.0000.

So why binarize the *vectors* but not the *model*? A trained transformer pushes a signal through two dozen layers, and quantization error **compounds** as it goes. One-bit weights wipe the signal out, unless the model was *trained* to be 1-bit from the start. int8 is the safe floor after the fact. The figure below shows how the error grows layer by layer at each weight precision.

Figure 4 · Why weights need int8 but vectors tolerate 1-bit

Illustrative. Relative representation error building up across transformer layers. Drag to change depth.



The rule: *quantize outputs (embeddings) hard, quantize weights gently. Eight bits keep the compounded error tiny, and the hardware has fast int8 matmul.* **Deployment note:** *you have to pick the quantized backend explicitly per architecture (x86 vs ARM), or it quietly fails to initialize.*

1.3 fp16 vector storage

lossless

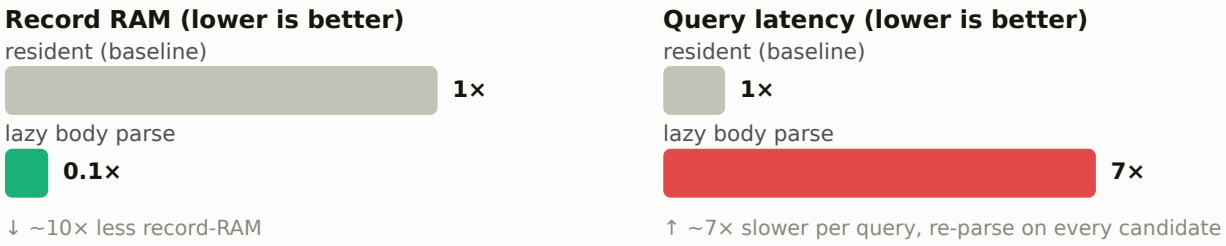
Storing the full vectors in half precision is **lossless for ranking**. The cosine order doesn't change, and it halves the on-disk and rescore footprint. One caveat: cast the query to the matrix dtype so the mixed-precision math doesn't throw.

1.4 Lazy record loading, a trade not a free win conditional

Keeping only light metadata in memory (identifiers and byte offsets) and parsing each record body on demand cut record RAM by about 10 times. But it made queries **about 7 times slower**, because ranking touches *a lot* of candidate records, and every one triggers a re-parse. The figure shows the trade directly.

Figure 5 · Lazy loading: RAM saved vs latency paid

Two measures, two panels, never one dual-axis chart. Both are ratios against the resident-record baseline (=1×).



Verdict: *keep it as an opt-in knob for tight-RAM situations, not the default. There's a cleaner version, resident metadata with only the body text loaded lazily, that avoids the penalty because ranking uses the metadata and only the final results need bodies. It just needs a fetch hook at the results boundary.*

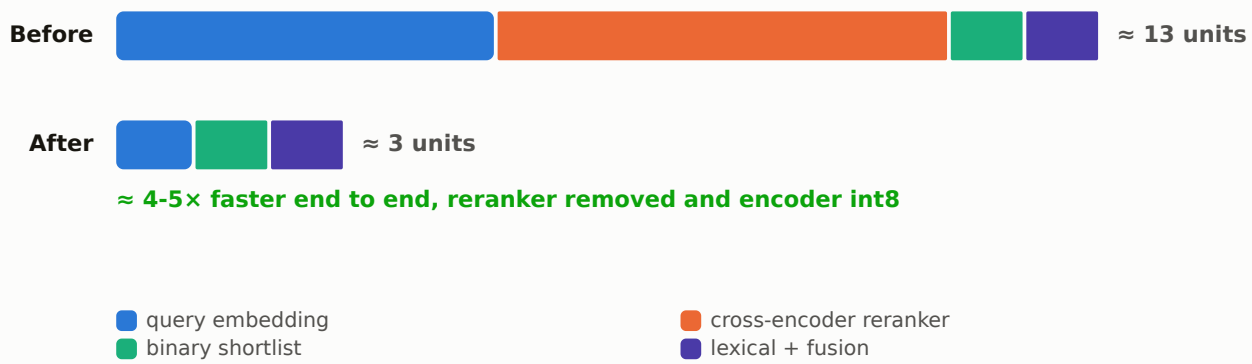
§2

Speed

Once the index is binarized and in RAM, the shortlist search takes about a millisecond. The bottleneck then moves to **embedding the query**, which is exactly what the int8 encoder speeds up, by roughly 5 times. After that, the cheap lexical side and the fusion step are what's left. The figure breaks down where a query's time goes, before and after.

Figure 6 · Where the query millisecond goes, before vs after

Illustrative split of relative per-query cost. The reranker and the float encoder were the two fat slices, and both got removed or shrunk.



The other speed lever is ***not doing work you don't need****. Dropping the cross-encoder reranker (§3.1) removed the biggest single per-query cost with no accuracy loss, and a semantic cache (§6) skips repeated questions entirely.*

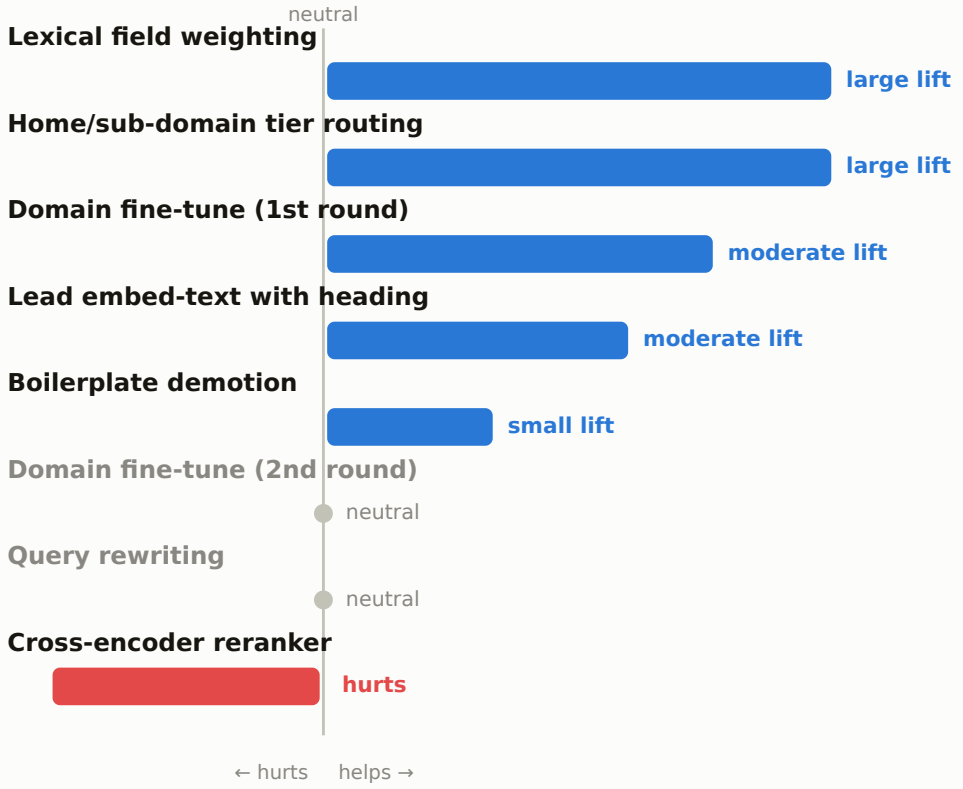
§3

Ranking quality

This is where the surprising results are. The cheapest, most structural signals moved accuracy more than any model-heavy lever did, and the fix people reach for first, a reranker, actually hurt.

Figure 7 · Accuracy levers on a directional scale

Rough size and direction as reported, not invented percentages. Bars right of zero help, left of zero hurt.



The pattern is clear. The **lexical and structural priors***, title weighting, tier routing, boilerplate demotion, all of them nearly free at query time, do the most, while the heavy* **cross-encoder reranker** *is the only thing that measured negative.*

3.1 A cross-encoder reranker is not free accuracy neutral→negative

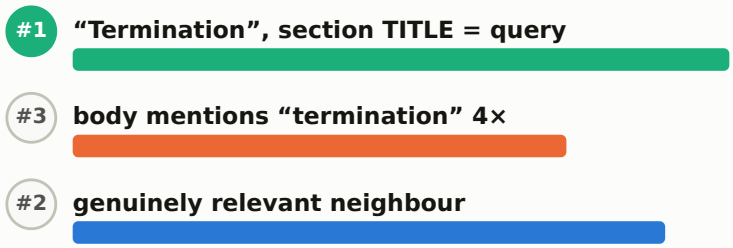
The instinct is to add a reranker to fix precision. On a **well-tuned structural pipeline**, the cross-encoder came out **neutral to negative**. It reshuffled an already-good pool and *pushed correct results down*, and it cost several times the latency. The structural signals, routing, priors, and fusion, had already done the ranking. So benchmark the reranker against rerank-off on your own pipeline. Don't ship it on faith.

3.2 Field weighting in the lexical index large lift

Lexical (BM25-style) scoring ranks by term frequency, so a record that just *mentions* a phrase in its body can outrank the one whose **title is that phrase**. The record you actually want can drop out of the lexical top-N. The fix is to repeat the section title a few times in the indexed text, so a record titled like the query wins. The figure shows that record climbing as the boost goes up.

Figure 8 · Title field-weighting rescues the on-point record

Drag the title boost. The lexical rank of the title-match record (title = query) against a body-mention rival.



Title record promoted to #1. The common "typed the section title" case is rescued.

Big, cheap gain on the common case where someone types the section's title. It's build-time only, so queries stay untouched. But **over-boosting hurts***: past a point the title record buries genuinely more relevant body matches, so tune the repeat count.*

3.3 Lead the embedded text with the strongest signal pure-quality

Where the title sits in the text you embed matters. Moving the section heading to the **front**, instead of burying it after a long breadcrumb, measurably raised the right record's dense rank. It means re-embedding, but there's no inference cost, just better quality.

3.4 Guard against silent “body-only” embedding invariant

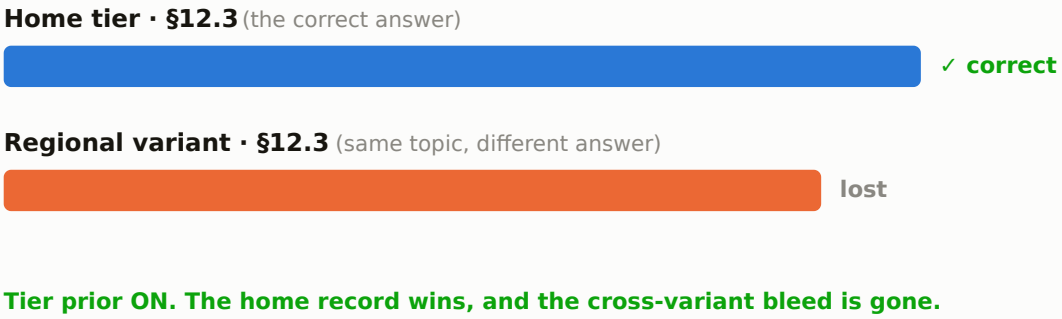
An offline embedding job once indexed only record *bodies* and dropped the header, so the strongest match signal never reached the vectors and the system quietly underperformed. Always assert that the text you embed is the text you meant to embed, and that the query side and the document side use the *same* text builder. A mismatch here is invisible in unit tests and deadly in production.

3.5 Home / sub-domain tier routing large lift

When two overlapping sub-corpora share a namespace, say a top-level set and a regional variant covering the same topics, the retriever blends them and the correct home record loses to its near-identical variant. A simple **tier prior**, preferring the home tier unless the query names the variant, was a big, cheap gain. A plain term-frequency signal can't fix this, because both records match just as well.

Figure 9 · Tier routing kills cross-variant bleed

Toggle the tier prior. Without it, a topically-identical variant outranks the correct home record.



The home and variant records match a generic query equally well, so no lexical signal can tell them apart. Only a structural prior can. And when the query does name the variant, the prior steps aside.

3.6 Demote boilerplate magnets small, consistent

Boilerplate sections like Definitions, Short title, Effective date, and Applicability match *generic* queries and crowd the top of the results. Sinking them below the substantive sections, unless the query actually asks for a definition or scope, is a small but consistent lift.

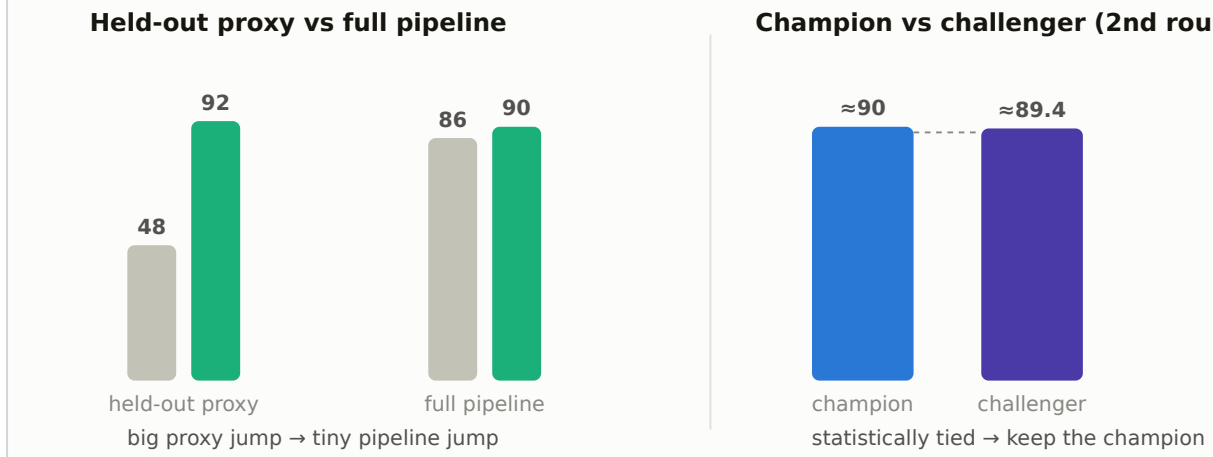
§4

Domain fine-tuning of the embedder

The first round of domain adaptation was a clear win. We built synthetic *query to record* pairs with **hard negatives pulled from the same parent document**, which teaches the model to map a plain question to *this* sub-section rather than its siblings. A **second** round, this time on hard negatives mined from real retrievals (the records that actually beat the right answer in production), came out **tied** with the champion.

Figure 10 · Adapt once; the second round rarely helps

Two lessons in one figure. Left: the held-out proxy against the full-pipeline metric. Right: champion against challenger.



The proxy isn't the truth. *A fine-tune that won big on a small "rank the gold above your own few negatives" set barely moved the end-to-end number, because the fusion and priors downstream soak up most of the embedder's gain. So decide on the end-to-end metric, and* **keep a champion***: once an embedder is adapted, more fine-tuning tends to be flat and can even regress.*

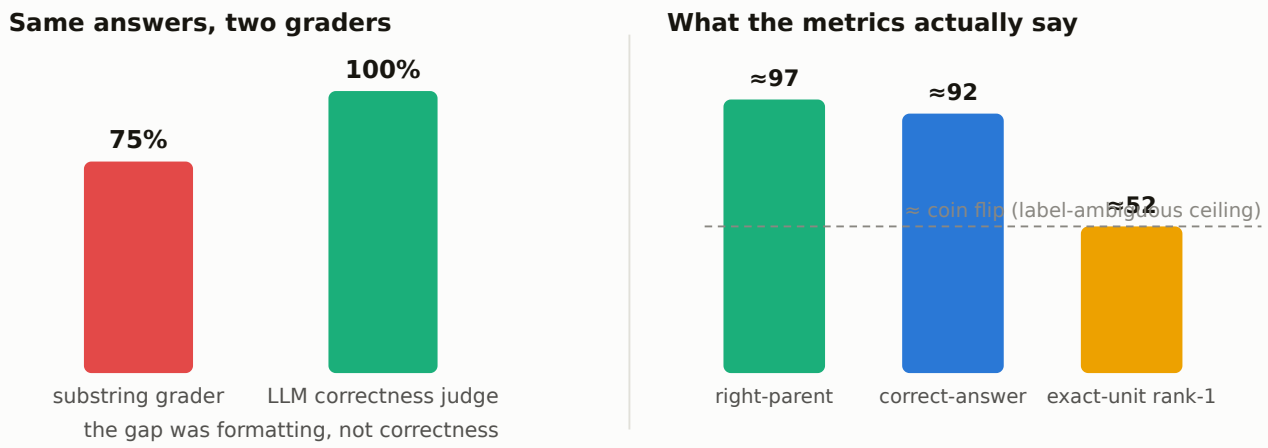
§5

Evaluation methodology

How you grade decides what you optimize, and a brittle grader can hide a system that's actually right. A substring exact-match grader failed **correct** answers that cited an equally valid reference or phrased a value differently. It read the system at about 75%. An **LLM judge of correctness** on the *same* outputs read about 100%. The brittle grader was scoring formatting, not correctness.

Figure 11 · Same outputs, two graders, and the exact-unit ceiling

Left: brittle substring grader against a semantic LLM judge on the same answers. Right: the three metrics that actually drive the product.



Keep deterministic checks for behavior and safety *(refusals, scope, no fabrication), where a regex is the right tool.* **Use an LLM judge for correctness***, where equivalence is a matter of meaning. And note the* **exact-unit ceiling***: mapping a plain question to one exact sub-unit at rank 1, out of millions, is beyond the state of the art and often ambiguous even to label. Right-parent-document and correct-answer are the metrics that reflect what a user actually feels.*

ALWAYS A/B UNDER IDENTICAL SETTINGS

Every comparison here changed one thing and held everything else fixed: same corpus slice, same flags, same candidate pool. A few early wins and losses vanished once the settings matched. Cache your baselines and re-run them next to the challenger.

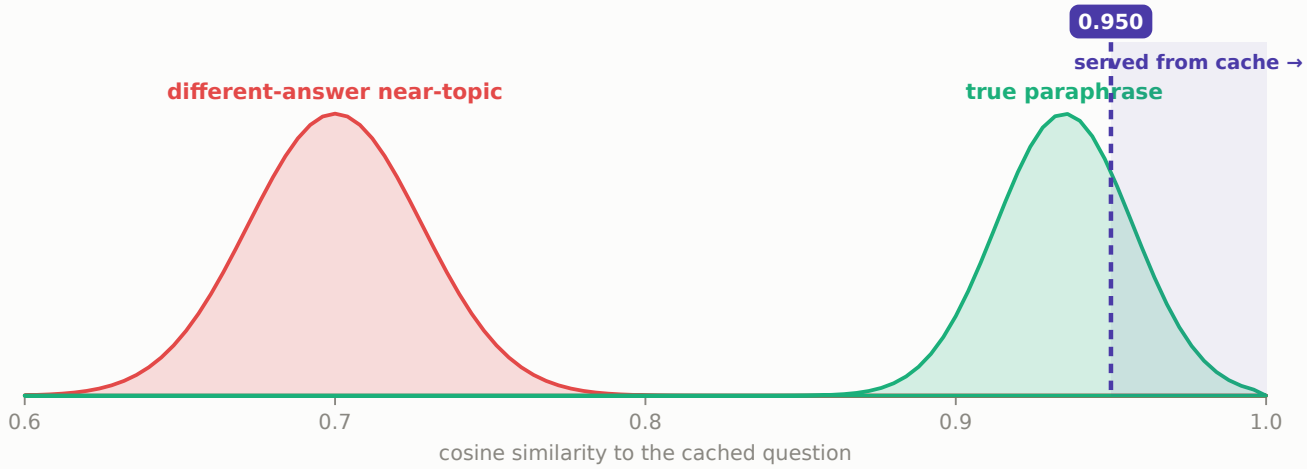
§6

Caching & safety

A cache that serves a stored answer for a **paraphrase** is only safe if paraphrases and *near-topics with different answers* sit far apart in embedding space. They do, but you have to measure it. True paraphrases of the same question land around cosine 0.90 to 0.97. A near-topic with a genuinely different answer (same subject, different jurisdiction or variant, so a *different correct answer*) sits down around 0.70. That gap is what makes a high threshold safe. Drag the threshold and watch the margin.

Figure 12 · Semantic cache: safety by measured margin

Two cosine distributions. A threshold set in the gap serves only near-identical phrasings, with plenty of room before any wrong-answer collision.



Safe. The threshold sits squarely in the gap: paraphrases hit, and there is a wide margin to any different-answer collision (~0.70). This is the belt-and-suspenders sweet spot.

Never ship a paraphrase cache without measuring that gap for your own model. *Then add a couple of safety nets:* **partition keys** *(jurisdiction or variant), and* **context-free turns only**, *so you never serve a follow-up like "what about the other variant?" from a global entry. And* **cache the sources, not just the text**: *if the value of your system is cited answers, store and re-emit the citations along with the cached answer.*

§7

Principles: the lessons that transfer

Strip away the specifics and seven rules are left. If you take nothing else from this, take these.

1

Measure every lever end to end

The reranker, the second fine-tune, and query rewriting all *sounded* like wins, and were flat or worse in practice. The real wins were cheap and structural.

2

Quantize outputs hard, weights gently

1-bit vectors are fine, because you only need the order plus a rescore. Models need int8, because per-layer error compounds.

3

The cheapest signal is often the strongest

Field weighting, tier priors, and boilerplate demotion, all lexical or structural and all nearly free, moved accuracy more than any model-heavy lever.

4

Grade meaning with a judge, behavior with a regex

Use an LLM judge for correctness, where two answers can be equivalent. Use regexes for refusals, scope, and fabrication, where exactness is the point.

5

Right answer beats exact unit

Optimize the metric that reflects what a user feels, not the one that's easy to compute. Exact-unit rank-1 can sit at a coin flip while the product is excellent.

6

Assert your data invariants

The most damaging bug was silent: we were embedding the wrong text. A one-line assertion (embed-text == intended text) would have caught it.

7

Keep a champion, replace it only on a clear win

Once something works, a challenger has to beat it on the end-to-end number under identical settings. Not on a proxy, not on a hunch.

★

The one-line takeaway

You can improve accuracy, speed, and cost at the same time, because the biggest wins are **cheap and structural**. But only measurement tells you which ones are real.

About this document. An interactive take on a practitioner writeup about hybrid-retrieval optimization. Every number is a **ratio or a relative change**, kept separate from any particular corpus, and the qualitative levers are shown by direction and rough size, not invented percentages. The figures render live in your browser, and no data leaves the page.

Charts follow a colorblind-validated palette · light & dark themes · built for repeated reference.

Dezso Mezo · DField Solutions · dfieldsolutions.com