

[>_] RECONCODA

37.7749N 122.4194W · SCIFACT · NFCORPUS · FINANCEBENCH · MS-MARCO-500K

// RESEARCH BRIEFING

Binary Retrieval as Infrastructure Repurposing

How compact indexes turn existing CPU capacity into usable AI retrieval

THE FINDING, IN ONE SENTENCE

Compact indexing — binary, and just as well, int8 — lets a large evidence-search index fit inside CPU and memory budgets that a float32 index can't touch. It doesn't make search meaningfully better. It doesn't make requests meaningfully faster. And binary specifically isn't the part doing the work — int8 gets you almost all of it.

AUTHOR & TESTED BY

Pawel Zaloga & Zach Alkharusy

PREPARED FOR

ReconCoda BNN / Low-Bit RAG Program

SCOPE

Retrieval only — not a true-BNN or general generation claim

EVIDENCE BASE

Paper A archive — claims register, methods, per-query results, limitations register

// EXECUTIVE THESIS

Reuse what you already have. A compact vector index lets a search service run on modest, existing CPU machines that a conventional float32 index simply can't fit on. That's the real value here — not speed, not a quality breakthrough, just infrastructure you don't have to buy.

1. WHY THIS MATTERS

Most AI systems that answer questions first search a pile of documents, then hand the best passages to a model as evidence. To search by meaning rather than exact wording, the documents get converted into embeddings — long lists of numbers. A large collection of ordinary 32-bit embeddings eats memory fast. Binary embeddings keep only the sign of each number, positive or negative, which shrinks the index dramatically.

The pitch writes itself: go binary, get the same answers, spend a fraction of the memory. That's the claim we set out to test. What we found is narrower and more useful than the pitch: the savings are real, but they come from memory and deployment flexibility, not from binary storage making search better on its own. Quality comes from the whole pipeline around it.

2. WHAT WE TESTED

Three questions:

1. Can a binary index match a float32 index on search quality, when both get the same supporting pipeline?
2. If it can, what's actually doing the work — the binary storage, or the pipeline stages that clean up its rough first pass?
3. What do we really gain, in memory and in response time, once we measure the whole request instead of one fast sub-step?

Then we tried to break it: noisy queries, a much larger corpus, five embedding models, four rerankers, and increasingly aggressive compression.

3. THE CORRECTION THAT CHANGED EVERYTHING

Our first comparison wasn't fair. The binary system had keyword search, a fusion step, a precision recheck, and a reranker running on top of it. The float32 baseline had none of that. Unsurprisingly, binary looked better for reasons that had nothing to do with binary storage — on NFCorpus, the early numbers even showed binary beating float32 outright (0.3715 vs 0.3435, a 108% reading). It wasn't a win. It was two systems doing different amounts of work.

We fixed it by giving float32 the identical pipeline — same embedder, same keyword search, same recheck, same fusion, same reranker. The only remaining variable was how vectors were stored. That single fix moved our headline speed number from a 2.38× end-to-end win down to 1.04× — a statistical tie.

WORTH SITTING WITH

That's a large reversal — the entire quantitative headline flipping from a clear win to a statistical tie. It's a reminder that pipeline asymmetries can hide inside plausible-looking comparisons, and no independent audit of the corrected pipeline has been run yet. Treat that absence as real, not a formality.

The keyword-search stage uses an inverted-index implementation, chosen because a naive per-query scorer doesn't scale to the largest corpus tested here. That choice matters for reading §8: the reranker's measured 93% share of total request time is a property of this measurement setup, not a claim about binary vs. float32 specifically.

General lesson for anyone reading this as a methods example: when two pipelines differ in more than the one variable being tested, the result can't tell you which difference caused the outcome.

4. WHAT WE BUILT

Four steps: cast a wide net with binary or float32 vector search; recheck the top candidates at higher precision; run keyword search in parallel and merge the two result lists; rerank the merged candidates.

We compared four index formats — binary, float32, int8, and product quantization (included specifically to check whether the result is unique to binary). Quality was tested on three corpora — SciFact (300 queries), NFCorpus (323 queries), FinanceBench (150 queries) — plus a 500,000-

passage slice of MS MARCO with 6,980 judged queries, to see whether the result held at real scale. Every comparison used paired statistical tests so small differences weren't mistaken for real ones.

5. THE MAIN RESULT: QUALITY HOLDS

With everything else held constant, binary matched float32 on every corpus tested:

CORPUS	BINARY	MATCHED FLOAT32	READING
SciFact	0.7643	0.7643	Bit-identical rankings
NFCorpus	0.3699	0.3699	100.02% retention — no meaningful difference
FinanceBench	0.7279	0.7153	Binary read higher, but n=150 is too small to call it (CI crosses zero)

The strongest number in the whole study lives outside that table. On the 500K-passage MS MARCO slice — 6,980 judged queries, roughly 100× the query volume of anything else here — binary hit nDCG@10 0.7962 against float32's 0.7968: 99.92% retention, with a confidence interval that includes zero. This is also the only place quality and scale were tested on the *same* embedding model rather than on two separately-measured setups.

READ CAREFULLY

Worth saying plainly: "no meaningful difference in our tests" is not the same claim as "binary is lossless." Don't let one become the other.

6. WHERE THE QUALITY ACTUALLY COMES FROM

The headline finding here is that binary storage isn't carrying this alone.

We reproduced the setup from a 2025 published study (Huerga-Pérez et al.) — binary search, no repair stages — and got their result: quality fell 9.3–18.0% across four tests, a real and statistically clear cliff. Then we added the repair stages back one at a time. Full-precision queries recovered a big

share of the loss. An 8-bit precision recheck recovered most of the rest. Keyword fusion plus reranking closed nearly all of what remained.

Binary search is a fast, cheap way to gather plausible candidates. The pipeline stages after it are what make those candidates trustworthy.

Three stress tests on that claim:

- **Fewer dimensions.** At 128 dimensions, the unrepaired signal collapses to ~51% — exactly what the published literature predicts. With the full repair pipeline running, it holds at 96.8–97.6%. Repair isn't a high-dimension artifact; it works even where there's nothing to spare.
- **A weaker reranker.** We swapped in four rerankers, including a deliberately weak, untrained keyword-only rescorer with no learned model behind it at all. Every one of them reached full recovery. Repair doesn't need a sophisticated reranker — just *a* reranker, applied identically to both sides.
- **Past binary.** At a gentler 4× compression, standard reduced-precision float formats already retain 97–100% before any reranking, confirming a competing paper's own headline claim. The one format that broke was an aggressive 4-bit float, recovering only to 77–85% even with full repair — the one real floor we found, and it sits on a different, shallower compression axis than the 32× result this brief is about.

The one embedder that didn't fully recover — e5-base-v2, at 94.59% retention on NFCorpus — gets its own explanation in §10.

This section's mechanism claim — reranking drives recovery — rests on §8's direct 93% latency measurement, not on any ablation ordering that could bias attribution.

7. WHAT COMPACT INDEXING BUYS — AND WHY "BINARY" OVERSELLS IT

The memory result is simple. A roughly one-million-vector, 3072-dimension index, tested under an 8 GB CPU limit:

FORMAT	MEMORY	FITS IN 8 GB?
Float32	11,707 MB	NO
Int8	2,927 MB	YES
Binary	366 MB	YES
Product quantization (same 32×)	366 MB	YES

Float32 couldn't run at all. Binary and PQ leave 7.8+ GB free — enough to change which existing machines can host the service without a new high-memory purchase.

BUT THIS OVERSELLS "BINARY" SPECIFICALLY

This is a benefit of *compact indexing broadly*, not of binary specifically. Int8 lands bit-identical to the fair float32 comparator with a simpler, more established technique, and it also clears the 8 GB bar; PQ matches binary's footprint outright. Binary's only uncontested edge over int8 is a further 8× memory cut on top of what int8 already delivers — not better correctness. If a deployment doesn't need that extra 8×, int8 is the more conservative way to get nearly all of this section's benefit.

8. WHAT DIDN'T HOLD: END-TO-END SPEED

Timed in isolation, binary search was clearly faster — 559.2 searches/sec vs. 234.7 for float32, a 2.38× win at similar recall.

But nobody uses an isolated lookup. Once candidate rechecking, keyword search, fusion, and reranking are counted, the gap was 65.55ms for float32 vs. 63.08ms for binary — a 1.04× improvement, about 4%, and it missed our own 1.2× bar for a meaningful win. The reranker alone eats roughly 93% of request time; shaving the remaining 7% barely moves the total.

NOT A DISAPPOINTMENT TO HIDE

It sharpens the decision. Reach for low-bit indexing to solve a memory or infrastructure problem, not to make requests faster. If latency is the actual problem, look at reranker choice, rerank depth, batching, or hardware first — in our tests, sending more candidates to the reranker often cost latency without buying quality.

A STRUCTURAL TENSION, NOT A FOOTNOTE

The reranker is both the thing that makes compact retrieval safe to use (§6) and the thing that erases the speed win (this section). We found no configuration that kept one without the other.

9. WHAT THIS MEANS FOR THE BROADER BNN PROGRAM

This is evidence about **compact retrieval** — not evidence that a true binary neural network can replace a general language model. The broader program keeps three separate questions apart: can compact vectors store and search enterprise knowledge efficiently (yes, under these conditions); can low-bit or ternary language models handle bounded tasks more cheaply than a strong hosted model (current tests show no win on both cost and quality at once); and can a true BNN — binary weights *and* activations, packed-bit runtime — run an enterprise workflow safely (untested here).

A CEILING WE DELIBERATELY DIDN'T CHASE

A GPU-only reranker hit 0.8753 on FinanceBench against 0.7279 for the CPU-eligible arm — a 0.147 absolute gain — but at roughly 1,450–1,660× the CPU latency, on a reduced 15-query sample (disclosed because the gap is orders of magnitude and the qualitative verdict doesn't hinge on sample size, not because the sample size doesn't matter). Don't cite this brief as evidence that CPU-only retrieval is close to GPU-reranked quality. It isn't. It's evidence that CPU-only retrieval is close to CPU-only float32 quality — a narrower and more defensible claim.

PROGRAM DIRECTION, STATED PLAINLY

Binary-first, not binary-only. Reach for compact retrieval when the constraint is genuinely memory or infrastructure fit. If a deployment needs both the quality-recovery benefit *and* low end-to-end latency at once, this study doesn't resolve that — the reranker is the one mechanism responsible for both, and nothing here decouples them. Treat quality-recovery and latency as one coupled design problem when that's the case, not two separately tunable knobs.

10. LIMITS OF THE STUDY

This is a careful retrieval study, not a universal proof. Latency numbers come from one vector engine (Qdrant) on an emulated four-vCPU/8GB shape — not a real cloud instance, not cross-validated on a second engine. FinanceBench has only 150 queries and uses extracted passages rather than full filings.

On the e5-base-v2 shortfall: we found a measured correlate, not a guess. Roughly 62% of its output dimensions are strongly one-signed across the corpus — about double the next-worst model tested, consistent on both corpora checked. A competing explanation (values simply collapsing toward zero) was tested directly and ruled out. That's a mechanism awaiting a fix, not an open mystery.

TWO NUMBERS WORTH STATING WITHOUT SOFTENING

Injecting ~10% noise drops nDCG@10 by about 9 points ($p=0.001$) on both SciFact and NFCorpus — binary doesn't degrade faster than float32 under noise, but the system as a whole is not noise-proof. And the 366 MB memory figure was, for most of the project, a theoretical projection; the first real build attempt hit an HNSW misconfiguration that made disk usage swing between 6 GB and 195+ GB before it was fixed. The number in §7 is the corrected, measured figure.

Before any formal submission: pin model versions, refresh the dependency lockfile, export figures in the venue's preferred format, and anonymize the repository if blind review is required.

11. CONCLUSION

Under the conditions actually tested — one vector engine, emulated four-vCPU/8GB hardware, the corpora in §4, and a fair pipeline on both sides — compact indexing (binary, and comparably int8)

keeps a large index inside a memory budget float32 can't touch, without a measurable quality penalty on most tests. That's real, and it's scoped to what was measured. It is not yet a claim about a second engine, real cloud hardware, a bigger financial-domain set, or a fully-fixed e5-base-v2 — those stay open questions, not fine print.

The mechanism is the whole story. Quality belongs to the complete pipeline — broad retrieval, precision recheck, keyword evidence, reranking. Speed belongs to the same pipeline, and that's exactly why the gain is modest: reranking dominates the time budget, and reranking is also what makes the quality result trustworthy in the first place.

Compact retrieval is a practical way to repurpose infrastructure you already have. It's not a substitute for good ranking, good systems design, or careful evaluation — and it shouldn't be sold as one.

READER'S CHECKLIST — IF YOU'RE CITING THIS WITHOUT READING IT END TO END

1. Binary's real edge over int8 is a further 8× memory cut, not correctness (§7).
2. A GPU reranker beats everything here by 0.147 absolute, at ~1,500× the CPU latency, on N=15 (§9) — a ceiling, not a deployment option.
3. "No worse than float32 under noise" ≠ noise-proof; absolute sensitivity is ~9 points (§10).
4. Quality-recovery and latency share a root cause (the reranker); no configuration here decoupled them (§8–9).
5. The pipeline-fairness correction was substantial ($2.38\times \rightarrow 1.04\times$, §3) — no second, independent audit of the corrected pipeline has been run yet.

SOURCES

Yamada, I., Asai, A., & Hajishirzi, H. (2021). Efficient Passage Retrieval with Hashing for Open-domain Question Answering. *ACL-IJCNLP 2021*. doi:10.18653/v1/2021.acl-short.123

Jégou, H., Douze, M., & Schmid, C. (2011). Product Quantization for Nearest Neighbor Search. *IEEE TPAMI*, 33(1), 117–128.

Karpukhin, V. et al. (2020). Dense Passage Retrieval for Open-Domain Question

Answering. *EMNLP 2020*, 6769–6781.

Robertson, S., & Zaragoza, H. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in IR*, 3(4), 333–389.

Thakur, N. et al. (2021). BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of IR Models. *NeurIPS 2021*.

Huerga-Pérez, J. et al. (2025). Optimization of embeddings storage for RAG systems using quantization and dimensionality reduction techniques. arXiv:2505.00105.

Reimers, N. (2024). Int8 & Binary Embeddings. *Cohere blog*.

Shakir, A., Reimers, N., Aarsen, T. et al. (2024). Binary and Scalar Embedding Quantization. *Hugging Face blog*.

Qdrant (2023). Binary Quantization – Vector Search, 40x Faster.
Full citations in references.bib.

[>_] RECONCODA

LOW-BIT RAG PROGRAM · EVIDENCE-LED