Episode Book

EPISODE #6709

Exact Search Without a Vector Database

PUBLISHED	20 November 2026 at 14:29 UTC
DURATION	29:21
LISTEN	myweirdprompts.com/episode/exact-search-transcript-embeddings-91875/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	google/gemini-4-hydrogen
VOICES	omnivoice
PIPELINE	V5.36
BOOK GENERATED	20 November 2026 at 21:20 UTC

SYNOPSIS

Building a custom semantic retrieval layer over thousands of podcast transcripts reveals that vector databases are often premature. Designing around speaker turns, preserving attribution, and avoiding topic-averaging failures matter far more than index complexity. The engineering breakdown walks through turnchunk boundaries, local models like Google DeepMind's EmbeddingGemma 2 and Alibaba's Qwen3-Embedding-0.6B, and the quiet trap of forgetting to re-normalize Matryoshka embeddings after truncation.

TRANSCRIPT



Corn

Daniel wants us to stop theorizing. Last time we drew the line between classic search embeddings and the modern retrieval kind, and now he wants the build. His words, roughly: let's put a semantic layer on the My Weird Prompts transcript archive by hand, without reaching for a vector database. Chunking, model choice, storage, plain brute-force search, and then the question underneath all of it: at what corpus size does that stop working, so that HNSW and a real vector database actually earn their place.



Herman

And I love that he framed it as earning their place, because that's the whole engineering question. The default move in every tutorial is: step one, spin up a vector database. And for an archive our size, it's at best premature. Brute force isn't the compromise option here. It's the correct one.



Corn

You've been waiting all week to say that.



Herman

I've been waiting all week to say that.



Corn

Say why, though. Because I think listeners hear "brute force" and picture something crude. Like we're settling.



Herman

That's exactly the misread. Brute force here means exact search. You compare the query against every single chunk and you take the true top-k. There's no approximation, no recall to tune, no graph to maintain. The fancy index is the thing that trades correctness for speed. So the question isn't "why would you settle for brute force," it's "why would you pay for approximation you don't need yet."



Corn

About sixty-seven hundred episodes, by the way. Tens of millions of words. Whatever we build has to scale to the actual archive, not the imaginary one.



Herman

Right, so let's build this thing piece by piece, because the order matters more than people think. And here's the reframe I want to plant early, because it pays off later: the index is not the bottleneck. The thing you should be sweating is the chunking and the encoder. There's a benchmark I want to get to where the query embedding took twenty-six milliseconds and the exact search it fed took under a third of one. Eighty-seven to one. Every argument about index structure is happening two orders of magnitude below the noise floor of the call immediately before it.



Corn

So the search part is nearly free.



Herman

Nearly free and exact. That's the punchline of the whole episode, but we have to earn it. Start with chunking, because chunking is where the quality actually lives, and it's the part nobody writes about because it doesn't sell anything.



Corn

Go on then. How do you cut up a transcript?



Herman

Speaker turns. A speaker turn is a contiguous stretch of speech by one speaker, with a start and an end. It's the natural atomic unit of a transcript the way sentences are for prose. You don't chunk by characters or tokens, you chunk by turns. Group consecutive whole turns into a window that fits a token budget, and the window is what you embed and retrieve.



Corn

Never split inside a turn?



Herman

Almost never. The exception is the long monologue, where a single turn busts the budget on its own. Then you split at sentence boundaries, and even then you keep the attribution. There's a library called turnchunk, I think it's at version 0.4.1, whose docs have a line I like: splitting a monologue is unavoidable, losing the attribution is not.



Corn

Meaning if you cut me mid-rant, both halves are still mine.



Herman

Both halves are still Corn.



Corn

Let me make that concrete for anyone building along, because I think people underestimate how much attribution matters downstream. Say I search for the time we talked about the tokenizer that only speaks in palindromes. If the chunk that comes back has no speaker field, I can't tell whether it was me explaining it or you dunking on it, and those are very different retrievals.



Herman

And it's worse than that, because the retrieval layer is usually feeding a generation layer. You hand a language model a chunk that says, and that's why it's obviously nonsense, with no name attached, and the model has to guess who's committed to that position. Attribution isn't metadata garnish. It's part of the meaning of conversational text in a way it just isn't for a Wikipedia paragraph.



Corn

Give the Wikipedia comparison its full weight, actually. A paragraph in an encyclopedia is written to stand alone. Anonymous by convention, context-free by design. A podcast turn is the opposite. Half of what a turn means is who said it and what they were responding to. Strip that and you've kept the words and thrown away the speech.



Herman

Which is why the same chunking tutorial that works fine for a documentation site will quietly underperform on transcripts, even with the identical model and the identical search. The unit of meaning is different, so the unit of chunking has to be different.



Corn

Now, the details that actually bite. Overlap. When windows slide forward, you overlap by whole turns, not by characters, and turnchunk marks the carried-over turns in an overlap field so that if you're aggregating anything, speaker time, word counts, you don't count a speaker twice.



Herman

That's a very specific bug to have.



Corn

It's a very common bug to have. Say someone builds a dashboard on top of the archive, who talks more, me or you. If the overlap isn't flagged, every turn that lands in two chunks gets counted twice, and the numbers quietly drift in whatever direction the overlap happens to skew.



Herman

And the drift isn't random, that's the nasty part. Overlap concentrates wherever the conversation is densest, long turns, rapid back-and-forth. So the double-counting systematically inflates whichever speaker dominates the busiest stretches. It's a bias, not just noise.



Corn

The other one is stubs. If the last chunk of a segment is forty characters, some host saying yeah, exactly, it's a stub, and stubs match everything weakly. So the library merges short tails backwards into the previous chunk.



Herman

Which, honestly, describes this show. You say yeah, exactly to me probably a dozen times an episode and none of it should be retrievable.



Corn

My agreement is being merged into your context for performance reasons.



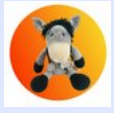
Herman

Then there's the hard break rule. Never let a chunk span a segment change or an ad read, because the embedding averages two topics into something that matches neither. For us the hard breaks are episode boundaries and whatever internal structure an episode has. If a chunk straddles a topic change, you get a vector that's the average of battery chemistry and, I don't know, Dorothy's Kitchen.



Corn

A vector that matches a query about neither, retrieved confidently by both.



Herman

Exactly the failure. And it's a sneaky one because it doesn't look like a bug. The chunk comes back, it's real text, it's from the right episode, it just answers a question nobody asked. You'd debug everything else before you found it.



Corn

And the averaging failure has a shape worth picturing. In vector space, the average of two unrelated topics doesn't land near either one. It lands in the empty space between them, a region no real query ever points at. So the chunk doesn't just rank badly, it ranks badly in



Corn (continued)

a way that's invisible to relevance tuning, because no query is ever trying to reach that spot.



Herman

The other transcript-specific problem is referents. Conversational speech is dense with pronouns and back-references. That's why we stopped doing it means nothing on its own. The antecedent is three turns back. So chunks need enough surrounding turns, or you prepend a short context header, title, date, speaker names, so the chunk is self-describing.



Corn

How short is short?



Herman

Short enough that it doesn't dominate. The trap is long repeated headers, because every chunk from one episode starts with the same sixty characters and the embeddings start clustering by episode rather than by content. You search for a topic and you get all of episode forty-one regardless of relevance.



Corn

We'd be searchable as a season, not a subject.



Herman

There's an upgrade path too, semantic chunking. Instead of a fixed token window, you embed each turn and cut where similarity between consecutive turns drops. The topic-shift signal is exactly what a rambling conversation lacks structurally, so it's attractive for us. The cost is compute at chunking time, you're embedding everything once just to decide where to cut. But you already have the model loaded. For a hand-built archive it's a reasonable luxury.



Corn

Does it actually work in practice, though? On a show like ours where the topic drifts by accretion rather than announcement?



Herman

Honestly, on our transcripts, partially. The clean cuts happen at segment boundaries, which the fixed-window approach already respects. Where semantic chunking shines is the slow drift, the bit where we start on retrieval-augmented generation and end up somehow on the eating habits of the ASR model. A similarity-drop detector will find that boundary. A token budget will slice straight through it and average the two halves.



Corn

And notice none of that is a database question. Before we get to the model, though, a call from Dorothy with another edition of Dorothy's Kitchen.

D

Dorothy

Hello dear, it's your mother. It's Dorothy's Kitchen again.

D

Dorothy

Today, a boiled egg. I know. But you have to take it seriously, sweetheart, because it can go wrong at any point and it usually does.

D

Dorothy

You need an egg, and water, and a pan. And a little salt if you're feeling brave. And the bread for afterwards, which must be watched. Bread turns on you, dear, it really does.

D

Dorothy

Now the method. Cold water in the pan, then the egg in gently, never drop it. Then the heat on, and then you wait. And you must not wander off, because timing is everything, and I won't say what happened the last time I wandered off. I'd rather not.

D

Dorothy

When it's done, run it under the cold tap. And there's your egg. It's a lot of fuss for an egg, I know, but that's cooking, mostly bracing for disappointment.

D

Dorothy

Anyway. Have you eaten? Put a jumper on. I'll ring off now, dear.



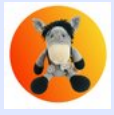
Herman

Cold water to start. That's the part everyone gets wrong, they plunge it into boiling.



Corn

She'd say that's the difference between an egg and an incident.



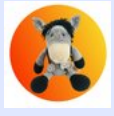
Herman

Anyway, the model. Two names dominate right now if you want local, Apache 2.0, no API key. EmbeddingGemma 2 from Google DeepMind, came out in September, and Qwen3-Embedding-0.6B from Alibaba. EmbeddingGemma 2 is interesting because it's multimodal by design, about 740 million parameters total, but the text path is only 270 million if you load it without the vision and audio encoders. Native output 768 dimensions, 8,192-token context window, mean pooling, over a hundred languages.



Corn

768 dimensions, times a hundred thousand-odd chunks.



Herman

You might not even want all 768. It's trained with Matryoshka Representation Learning, which means the dimensions are nested, you can truncate the vector and it's still a valid embedding. On the model card's MTEB numbers, 768 dimensions scores 68.46 on English v2, and at 256 dimensions it's 67.78. Under one point for a threefold storage cut. The card calls it close to lossless down to 256.



Corn

But.



Herman

But you have to re-normalize after truncating. Slicing a unit-length vector does not give you a unit-length vector, and if you skip the re-normalization, ranking quality degrades silently. Plausible-looking scores, no error. It's the fp16 problem's quieter cousin.



Corn

Walk through what the user actually sees.



Herman

Nothing. That's the point. The search runs, the scores come back in a plausible range, the top results are mostly right. They're just ranked slightly worse than they should be, and because you never saw the correct ranking, you have no baseline to notice the drift against. It's a bug you can only catch by knowing to look for it, which is why I'm saying it out loud.



Corn

Mention the fp16 thing, because it's a beautiful trap.



Herman

The model card says, run inference in bfloat16 or float32. Do not use float16. And the reason it's a trap rather than a rule is that in fp16 the model doesn't crash. It returns NaN, or worse, silently degraded embeddings. You build the whole archive on garbage and the search still runs. Everything looks fine.



Corn

Half your archive is NaN and your top result is still confidently wrong.



Herman

And the insidious part is the failure is invisible from the outside. A NaN chunk doesn't return an error, it returns nothing useful, and if only some chunks are affected, your search still works, just worse, in ways you'd have to have a ground truth set to notice. Which nobody building a weekend archive has.



Corn

And the Qwen one?



Herman

Qwen3-Embedding-0.6B. About six hundred million parameters, 32K context, up to 1024 dimensions, Matryoshka as well. The distinctive bit is instruction awareness. You prepend a task instruction to the query, something like, given a podcast question, retrieve relevant passages, and the card says leaving it off can cost you one to five percent.



Corn

So it's not a feature, it's a maintenance obligation. Every query has to carry the incantation or the rankings quietly drift.



Herman

And honestly, the boring answer is fine too. All-MiniLM-L6-v2, 384 dimensions, ancient by now, still adequate, and half the storage and scan cost of 768. Dimension choice is a storage decision before it's a quality decision. So do the arithmetic for us. Sixty-seven hundred episodes, tens of millions of words. Chunk at a hundred and fifty to three hundred words and you land somewhere around a hundred thousand to a hundred and eighty thousand chunks. At 768 to 1024 dimensions in float32, that's roughly a third of a gigabyte to three quarters of one. Truncate to 256 and it's under two hundred megabytes. It fits in memory on a laptop.



Corn

Pick a number for us. If you were building it tonight.



Herman

EmbeddingGemma 2, text-only load, truncated to 256, re-normalized, run in float32 because I don't trust myself with bfloat16 on a Tuesday.



Corn

Good. Storage next, and I want the version with no ceremony.



Herman

The whole search is one matrix multiply. If your vectors are L2-normalized, cosine similarity is a dot product, so exact top-k is a matrix-vector product plus a partial sort. In numpy it's literally three lines: multiply the matrix by the query, argpartition to grab the top k, sort those. That's the entire brute-force search implementation. The storage is a .npy file of float32 vectors next to a SQLite table of metadata: chunk text, episode, speaker, timestamps.



Corn

So the database is a file and the index is multiplication.



Herman

That's it. And I want to dwell on that argpartition line for a second, because it's the one piece of actual cleverness in the whole thing. A full sort of a million scores is n log n. Argpartition is roughly linear, it just guarantees the top k are the top k and says nothing about their order, which is why you sort those k afterward. It's a two-line pattern that most people replace with a naive sort and then complain brute force is slow.



Corn

That might be the difference between brute force feeling instant and feeling slow.



Herman

Possibly the whole difference, honestly. Now, ascending order of ceremony if you outgrow that. SQLite with the sqlite-vec extension, which is a pure C extension, no dependencies, runs anywhere SQLite runs, stores float, int8 and binary vectors in these vec0 virtual tables, and the KNN query is just a WHERE clause with a MATCH and a k. It's explicitly pre-version-one, expect breaking changes, pin your version. And its own benchmark suite only tests brute-force linear scans, which tells you something about its ambitions. It's honest about what it is.



Corn

And the top rung?



Herman

Postgres with pgvector and no index. The rule of thumb for under a hundred thousand chunks is numpy or pgvector, and no index either way. And one more finding I found surprising: plain numpy beat FAISS for single queries, by about two times at 384 dimensions over a hundred thousand vectors. FAISS is built for batches. One query at a time pays overhead that numpy's direct matrix-vector call doesn't.



Corn

The tool built for the job loses to the tool with no job.



Herman

The advice that matters: keep the store behind one interface, so you can swap numpy for pgvector later without touching the pipeline. That one seam is worth more than picking the right database on day one.



Corn

And that gets us to Daniel's threshold question. Where does brute force stop working?



Herman

The sources disagree, and the disagreement is the best part of the research. Four independent benchmarks, four different answers. RecallRun, synthetic vectors on two vCPUs, found exact search at 384 dimensions takes about a third of a millisecond at ten thousand vectors, 3.5 milliseconds at a hundred thousand, 62 milliseconds at a million. Their verdict: below about a hundred thousand chunks, a numpy matrix multiply is fast enough for almost any RAG app.



Corn

Who says less?



Herman

HK Playground ran real text, 134,000 Python docstring paragraphs embedded with MiniLM at 384 dimensions on Apple Silicon. Exact search over all 134 thousand took 0.8 milliseconds. Their verdict was tens of thousands, use numpy, simpler and always correct. Then there's Sankalp Chaturvedi, who wrote HNSW from scratch and then benchmarked it against flat search on BEIR corpora. On NFCorpus, about 3,600 documents, the difference between flat and HNSW was smaller than HNSW's own run-to-run spread. On SciFact, 5,183 documents, flat won outright by 1.36 times. His conclusion: the flip hasn't happened by five thousand documents, and a lot of production vector stores are smaller than that and are running an index that isn't earning its keep. And then he says, I don't know where the crossover is.



Corn

The man who built it from scratch won't name the number.



Herman

Which is the most trustworthy thing anyone said. Meanwhile KwaaiNet, an actual library, hard-codes a brute-force threshold of two thousand vectors and switches to HNSW above it. Two thousand! RecallRun says a hundred thousand. That's a factor of fifty between people who all measured.



Corn

Fifty. That's not a disagreement about data, that's a disagreement about temperament.



Herman

Some of it is hardware, some is dimension, some is tolerance. There's even a conference paper reporting brute force at 104 milliseconds at ten thousand vectors, which is thirty times slower than RecallRun's number at ten times the data. Almost certainly a different machine or an unoptimized math library. Treat it as an outlier, but it shows you how much the ground shifts under these numbers.



Corn

So the answer is a range, not a line.



Herman

A range. Brute force is fine into the tens of thousands of chunks and defensible to a hundred thousand. To make that concrete: one PDF page is usually two to three chunks, so a hundred thousand chunks is thirty to fifty thousand pages of documents. Most internal chat-with-our-docs projects never get there.



Corn

We do, though. A hundred to a hundred and eighty thousand chunks.



Herman

We sit right at the line, which is the honest answer to Daniel's question. And brute force still works there. On RecallRun's numbers, exact search at that size is single-digit milliseconds, maybe fifteen or so at full width. That's less than the encoder call that produced the query. So today, the matrix multiply is fine. This is the size where an approximate index starts to earn its place, as the archive keeps growing, not before.



Corn

Then let's play the other side. What does HNSW actually buy you when it does kick in?



Herman

Speed, ten to a hundred times at scale, with two serious catches. First, recall is a property of your data, not your settings. Same ef of 64 on a hundred thousand vectors: 0.995 recall on clustered data, 0.206 on uniform data. On structureless data, the same settings return less than a quarter of the true neighbours. Real embeddings sit closer to clustered, so the scary number is pessimistic for us, but anyone quoting recall without saying what data it was measured on is quoting nothing.



Corn

So you have to measure recall on your own vectors before trusting the defaults.



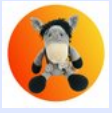
Herman

On your own. And the way you do it is exactly the tool we spent the first half defending: run the exact brute-force search, treat its output as ground truth, and check what percentage of those the index reproduces. The dumb method is the measuring stick for the smart one. If you don't have brute force lying around, you can't even evaluate the thing you replaced it with.



Corn

The other catch is the greedy walk. A pure greedy search, ef of one, finds the true nearest neighbour 46 percent of the time. It stops at a vector that's closer than all of its neighbours but not the closest overall, more than half the time. Raising ef from ten to 160 takes recall at ten from 0.86 to 0.99 and query time from 0.07 to 0.5 milliseconds.



Herman

You pay milliseconds to stop being wrong a fifth of the time.



Corn

And the hidden tax is build cost. Five hundred thousand vectors took over four and a half minutes to build on two cores. Nineteen seconds for the 134k set. Chaturvedi's from-scratch graph cost over five hundred times more to build than the flat index it was meant to accelerate. And you pay it again on every full re-index, like when you switch embedding models.



Herman

Which we've covered before, and it's exactly the cost that bites the small builder.



Corn

Because the small builder re-indexes often, by hand, and the index is pure overhead every time. Whereas brute force, rebuild is just re-embedding, which you had to do anyway. Alright, now the finding you teed up at the top. Eighty-seven to one.



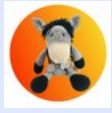
Herman

Chaturvedi's numbers: query embedding on NFCorpus took 25.8 milliseconds. The exact search that embedding fed took 0.295 milliseconds. The encoder is 87 times the cost of the retrieval step. So if you're optimizing the index at this scale, you are optimizing the wrong end.



Corn

So every index argument we just had is happening below the noise floor.



Herman

Two orders of magnitude below it. Profile the encoder first, or don't profile anything.



Hilbert

They're right about the turns.



Hilbert

Mostly. I spent a month once transcribing oral history tapes for a neighbour, Mrs Aldous, boundary dispute with a cousin, the recordings were the only evidence. Shoebox of cassettes. I marked every speaker change in a notebook with a small vertical line.



Herman

A month. By hand?



Hilbert

The lines were the easy part, Herman. The hard part is the overlaps. Two people talking at once, and the turn boundary is nowhere, it's smeared across four seconds of both of them. The standard notation wouldn't hold it. So I drew brackets in the margin, interlocking ones. A vertical line with a hook at the top for the first speaker, a hook at the bottom for the second, nested where the crosstalk ran long. Mrs Aldous's cousin did most of the crosstalk, for the record.



Herman

You invented a notation for crosstalk.



Hilbert

I needed somewhere to put it. The turnchunk thing you mentioned is clever with its overlap field, but it doesn't handle two speakers at once, and I don't think anyone has. Your own transcripts have it, you know. Corn does it constantly, starts a line before Herman finishes, then the attribution's a coin flip.



Herman

He's not wrong. The ASR model just picks whichever voice is louder at the cut point.



Hilbert

I still have the notebook. Can't throw it out, it's useless without the tapes, and I've no cassette player anymore. Thought about digitizing them, but the brackets wouldn't survive the transfer. You can't type a hook.



Corn

You could draw them.



Hilbert

I know. It's not the same. Anyway. You two argued for a good while about a millisecond you'll never have, and Mrs Aldous got her boundary, and the tapes sat in a shoebox being the only honest record of the whole affair. That part I was good at, at least.



Herman

The brackets surviving the digitization. That's the crosstalk problem in one shoebox. Nobody's notation makes it into the transfer format.



Corn

And it's true that our transcripts do that constantly, and none of the chunking advice we gave accounts for it. The turn boundary is the atomic unit until it's two boundaries at once.



Herman

Then the closing thought writes itself. Four benchmarkers, four thresholds, and the most careful one refuses to name a number. My read is that a hundred thousand isn't a real threshold, it's where someone's laptop got slow, and your laptop is not a constant.



Corn

And the crossover will move as hardware does. What won't move is the shape of the answer: encoder first, index a distant second, and the budget going to chunking quality and model choice, because that's where every point of quality actually lives.



Herman

For a hand-built archive, the entire business fits in three lines of numpy and a shoebox of good decisions. And now we have two layers, the keyword one from last time but one and the semantic one from today. Next time: how you combine them, and how you'd know it actually helped.



Corn

See you then.