Episode Book

EPISODE #6390

Why Editing Vector Databases Silently Re-Embeds Text

PUBLISHED	31 October 2026 at 18:24 UTC
DURATION	18:33
LISTEN	myweirdprompts.com/episode/editing-vector-database-text-57834/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	anthropic/claude-sonnet-5.5
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	13 cited by the research (listed at the end)
BOOK GENERATED	1 November 2026 at 07:41 UTC

SYNOPSIS

Vector databases do not reverse numbers into words when an interface displays editable text. Plain sentences sit as metadata payloads or separate records matched by foreign keys. Changing text triggers a hidden re-embedding pipeline, risking stale searches where deleted paragraphs continue answering queries. An examination of APIs from Ragie and Qdrant illustrates how deterministic chunk hashing prevents full-document re-embedding, why whitespace cleanup avoids accidental model billing, and how unstable chunk IDs can break citation systems.

TRANSCRIPT



Corn

Daniel's been poking at tools like Ragie that let you open a note sitting in a vector database, change the words, or delete it. His instinct is that once text is embedded it's just numbers, and numbers don't have editable sentences in them. So what is he looking at when he sees readable text? And is an edit quietly re-embedding behind his back?



Herman

Herman Poppleberry here, and the short answer is that nothing is being decoded. Nobody is turning numbers back into words. But the tools are surprisingly open about what they do instead.



Corn

So the numbers never contained the words in the first place.



Herman

Right, and the thing to drop is the idea that a vector database is one thing. A point in one of these systems is three parts: an ID, a vector, and an optional payload of metadata. Most production setups keep the original text either in that payload or in a separate store, keyed by the same ID.



Corn

So the readable text is a copy someone saved.



Herman

A stored copy, yes. Not a reconstruction. The magic is a foreign key. You search with the vector, you get back an ID, and the ID points at the text. When you edit, you're editing the text and then regenerating the vector from it.



Corn

Which is the stuff we covered when Pinecone didn't update the embedding after a text edit.



Herman

Same coupling, but today it's about how the interface hides it. Let's look at what Ragie's API exposes, because the documentation is remarkably honest.



Corn

Go on then.



Herman

The documents SDK has list, create, create raw, create from URL, get, delete, update file, update raw, update from URL, patch metadata, get chunks, get chunk, get chunk content, get content. Read that list for a vector endpoint.



Corn

There isn't one.



Herman

None. Nothing takes a vector and nothing returns one. It's entirely document-centric. And the chunk object is the tell. You ask for a document's chunks and each one comes back with an id, an index, metadata, and a plain text field. Their own example literally says "The text of the chunk."



Corn

Very informative example.



Herman

Very. Plus links with hrefs pointing at a content endpoint that serves plain text. So the readable string is a field in a record.



Corn

Then Daniel's first question is answered. Is the original text stored alongside the vectors? Yes.



Herman

In one of three patterns, and they're worth separating. One, vectors only. The database returns IDs and you look the text up in Postgres or object storage. The index is smaller and faster, but you pay a second round trip and you have to keep the IDs in sync.



Corn

That sync is where I'd expect it to go wrong.



Herman

It does, and we'll get there. Two, vectors plus metadata, which lets you do filtered search, like only documents from this customer. Three, vectors plus the raw text in the same point, so search results can hand back snippets directly. That costs you storage and fatter responses. The common RAG pattern is minimal metadata in the vector store and full documents elsewhere.



Corn

How does the filtered search work in practice? If I say only this customer's documents, is it searching everything and then throwing most of it away?



Herman

It depends on the engine. Some filter after the nearest-neighbour search, which can leave you with too few results. Imagine asking for the ten closest matches, and eight of them belong to other customers. You're left with two. Better engines apply the filter during the



Herman (continued)

search, so they keep walking outward until they've found ten allowed results. Either way, the filter runs against the metadata, which is ordinary structured data. The vector only answers "what's similar," and the metadata answers "what's allowed."



Corn

So a permissions bug would live in the metadata, not in the numbers.



Herman

Usually, yes. If a customer ID is wrong on a point, the vector will happily surface it to the wrong person. The embedding has no idea who owns it.



Corn

I'd pick the dull one where the text lives in a normal database. I trust things I can open with a text editor.



Herman

Predictable. And at the storage layer it's honestly dull. A typical pgvector table has an id, the embedding model name, a content hash, the embedding column, the original text, metadata as JSONB, created and updated timestamps. Text and vector are columns in the same row.



Corn

A spreadsheet with one weird column.



Herman

A column of fifteen hundred numbers, in the case of OpenAI's small embedding model. Some others are a thousand or so. That's what's stored. Notice the model name sits in the row too. If you switch models later, old and new vectors aren't comparable, so you need to know which model made which one.



Corn

So the row is also a label saying how to read the numbers.



Herman

Now here's what happens when you hit update.



Corn

This is the part I'm suspicious of.



Herman

You should be. Ragie's chunk reference says plainly that updating through update file or update raw will regenerate document chunks, including their index. And update raw takes the data as text or JSON. You hand back the source text. Not a diff, nothing aimed at an embedding.



Corn

So I fix a typo and the whole document goes round again.



Herman

It re-enters the pipeline. The status field walks a ladder: pending, partitioning, partitioned, refined, chunked, indexed, summary indexed, keyword indexed, ready, and failed if something breaks. The document is usable for retrieval at indexed, and the summary lands a few seconds later.



Corn

Nine rungs for a typo.



Herman

The stages after chunked are the embedding and index-writing. Those re-run on update. So to Daniel's question: yes, quietly, asynchronously, and by design.



Corn

What I like is that the vendor agrees with him. Usually you have to catch them at it.



Herman

Deletion is the same shape. Delete takes a document ID and an async flag, and you never address a vector. Deleting the document takes its chunks and their vectors with it.



Corn

And I'd guess the cheap path is the metadata.



Herman

Patch metadata is its own endpoint, and along with delete it's one of only two async operations Ragie documents. Changing a label doesn't touch the embedding. Changing the text is a full re-ingest. That cost difference is why the endpoint exists at all.



Corn

So if I retag a document from draft to approved, nothing gets re-embedded.



Herman

Nothing. The vector describes what the text says, and a status label isn't part of what the text says. That's a useful test for any field. If changing it wouldn't change what the model reads, it belongs in metadata.



Corn

Then any editor that feels instant when I change a sentence is doing one of three things.



Herman

Debouncing, batching, or being vague about when the index catches up. Debouncing means waiting until you stop typing before it sends anything. Batching means collecting edits and processing them together. The text field updates immediately, since it's just a stored string. The search index can lag behind it.



Corn

So for a few seconds the note says one thing and search answers about another.



Herman

Which is the stale vector problem, and it's why you can't simply patch the vector. Let's do that properly.



Corn

Go ahead. Why not just nudge the numbers?



Herman

Because there's nothing to nudge toward. A vector is the output of a model reading the whole chunk. Change one word and the right new vector is whatever the model produces on the new text. You can't derive it by editing the old numbers. Qdrant's operations tutorial says it flatly: vectors are a transformation of raw data, and the transformation doesn't happen by itself when the raw data changes.



Corn

But a one-word change should move the vector only a little. Why isn't a small nudge close enough?



Herman

Sometimes the move is small, but you can't know how small without running the model. Swap "is" for "is not" and the meaning flips while the text barely changes. Nothing in the old numbers tells you which direction the new meaning lies. The only way to find out is to ask the model again.



Corn

Their phrase for what happens if you forget?



Herman

Search runs against embeddings of text that no longer exists. A deleted paragraph keeps answering queries like a ghost.



Corn

I'd rather the ghost didn't have my old salary in it.



Herman

Fair. So underneath every editing interface is an upsert by ID. Insert a point with an ID that already exists and it's replaced with the new vector and payload, no error. The UI wraps that in a document object so you never see the vector.



Corn

But re-embedding a whole two-hundred-page PDF per typo sounds ruinous.



Herman

It would be, and this is the part I like. Qdrant's recommended loop gives every chunk a deterministic ID, its position in the source, plus a fingerprint, a hash of the exact text that was embedded. On each run you compare. Unchanged chunks are left alone. Changed text gets re-embedded. New text is added. Text missing from the source is deleted.



Corn

So one edited paragraph costs one embedding call.



Herman

One, not two hundred. And there's a fifth case that's subtle. If the text only moved, same content in a new position, you reuse the vector. The fingerprint matches, so no model call.



Corn

Wait, that means the vector belongs to the text, not the place it sits.



Herman

Right. Reorder your whole document and most vectors survive. It cuts against the idea that any edit means re-embedding everything, and the editable-database pitch tends to skip it.



Corn

What stops a stray trailing space from invalidating the hash?



Herman

Normalization, and Qdrant warns about exactly that. Noise costs you re-embedding you didn't need. You trim and standardize before hashing, or someone's whitespace fix bills you for a model call.



Corn

A trailing space with a price tag. Nice.



Herman

Multiply it out. Someone runs a find-and-replace that strips trailing spaces across ten thousand chunks, and without normalization every hash changes. You've just paid to re-embed a library to learn nothing new.



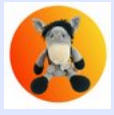
Corn

And the vectors come back essentially identical.



Herman

Near enough, yes. Pure waste.



Herman

Also the long sections. That example uses a small model with a two hundred fifty-six token window and three hundred eighty-four dimensions, and anything over the window is silently truncated. That's why chunking exists at all.



Corn

Silently truncated is a phrase I never want near my notes.



Herman

It's worse than it sounds. If a chunk runs long, the tail never reaches the model, so the vector represents only the first part. You could search for something that's plainly in your document and never find it, because the part that mentions it was never read.



Corn

And nothing tells you.



Herman

Nothing. That's why the chunk size is a setting worth checking against the model's window, not a detail to leave on default.



Herman

Now the gotcha I think is freshest. Chunk IDs aren't stable. Ragie regenerates chunk indices on update. And documents created before the eighteenth of September, twenty twenty-four and never touched since have chunks with no index at all.



Corn

Hold on. I think I know whose job it is to announce this.



Herman

Ha. Mum?



Corn

Mum. It's time for Dorothy's Grim Fact.

D

Dorothy

Hello sweetheart, it's your mother. Time for my grim fact, dear.

D

Dorothy

Ragie's documents that were made before the eighteenth of September, two thousand and twenty four, and never touched since, have chunks with no index at all. They just come back as minus one.

D

Dorothy

Isn't that lovely. Anyway, I've made a casserole and there's rather a lot of it, so do come by Sunday, and bring your laundry.



Herman

Delighted she called.



Corn

Laundry, apparently.



Herman

So the practical consequence is this. If your app cached chunk IDs, or built citations on them, an edit silently invalidates them. The citation still looks fine and points at nothing, or at the wrong text.



Corn

That's the one I'd get bitten by. I'd cite chunk seven and never know it moved.



Herman

Say an edit near the top of a document splits one paragraph into two. Every chunk after it shifts down a slot. Your saved "see chunk seven" now lands on what used to be chunk six. The link resolves, the text is real, and it's the wrong passage.



Corn

Which is worse than a broken link, because a broken link at least admits it's broken.



Herman

A wrong citation looks like a correct one, and nobody questions it.



Corn

So what do you anchor citations to instead?



Herman

Something that travels with the content. The document ID plus a content hash, or a quoted snippet you can match against. Position is the least reliable thing you could pick.



Herman

And the storage choice has a privacy side. Raw text in the vector payload makes editing and snippets trivial but puts plaintext in the index. A side store reduces that exposure but adds the drift risk, where the vector and the text stop matching.



Corn

So neither is free. Which would you pick?



Herman

For anything sensitive, the side store, and I'd write a check that compares hashes on a schedule. You?



Corn

Same store, plain text, and I'd never touch it. That's my whole security model.



Herman

Not editing it is a legitimate strategy.



Corn

Does anything in this mean Daniel's old claim was wrong? That you can't edit vectors.



Herman

It was right, and the tools agree. You're not editing a vector database. You're editing a document store that owns a vector index.



Hilbert

Does the card catalogue count? Because mine is a shoebox of index cards, and each card points at a folder in the cabinet by a number in the corner.



Herman

It counts. Same structure, number in the corner is your foreign key.



Hilbert

I rewrote the entry on a Dymo label gun once, for a folder about the water board, and the old label I'd stuck on the front of a different folder still said the old thing. Eleven cross-references pointed at it. Herman, you'd have liked the mess. Everything still pointed at the number and nothing pointed at what the number meant now.



Corn

The text changed, the pointers didn't.



Hilbert

I keep the old version of every entry in a biscuit tin in the hallway. Folded in four, dated in pencil. The new one goes in the drawer. That tin is how I found which entries had really changed and which I'd only moved to the front. Mrs Abelson at the water board sent me the same letter three times and I only needed to file it once. Her record for the shortest reply was four days. Mine took her eleven.



Herman

That's the moved versus changed problem done with a tin.



Hilbert

It's in the hallway next to the boiler. It runs hot in there, but paper's fine. I just find the whole thing very interesting, honestly.



Corn

Four days from a water board is fast.



Hilbert

It was a Tuesday.



Herman

Which does bring up the category question. If the text is always a stored copy and the vector is always derived, what is a vector database?



Corn

A search index that came with a filing cabinet, maybe. Or a database with a fancy column.



Herman

Either description works. What matters is which part is the source of truth, and it's always the text. The vector can be thrown away and rebuilt, as long as you kept the words and remember which model you used.



Corn

So if the index burns down, you re-embed and carry on.



Herman

At a cost, but yes. If the text burns down, you've got a pile of numbers nobody can read.



Herman

And as the tools get polished, the re-ingest gets more invisible. The chunk ID instability and the stale vector problem get harder to notice, and harder to debug.



Corn

Which means when search gives a strange answer, you can't tell whether the model is wrong or the index is just behind.



Herman

The first thing I'd check is whether the stored text and the vector were made from the same version. That one comparison rules out a lot.



Corn

The better the trick, the more you want to know it's a trick. Same as the biscuit tin, really. Somebody has to keep the old copy.



Herman

Thanks to Hilbert Flumingtop for producing, as ever.



Corn

If you enjoyed this one, try episode fifteen, AI Gets Personal; and episode nineteen, AI Images. This has been My Weird Prompts. If you liked it, a review helps more people find us.



Herman

See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Ragie, Get Document Chunks, updated ~22 days before 2026-10-31 [primary]**
   https://docs.ragie.ai/reference/getdocumentchunks
2. **Ragie, Update Document Raw, updated ~2 months before 2026-10-31 [primary]**
   https://docs.ragie.ai/reference/updatedocumentraw
3. **Ragie, API Walkthrough (chunk `text`, `links`, `modality_data`, media types) [primary]**
   https://docs.ragie.ai/docs/changes-to-the-api
4. **Ragie, Delete Document [primary]**
   https://docs.ragie.ai/reference/deletedocument
5. **Ragie, Asynchronous Operations (async only for Delete Document and Patch Document Metadata) [primary]**
   https://docs.ragie.ai/docs/asynchronous-operations
6. **Ragie TypeScript SDK docs, full operations list [primary]**
   https://github.com/ragieai/ragie-typescript/blob/main/docs/sdks/documents/README.md
7. **Ragie Python SDK docs, status ladder [primary]**
   https://github.com/ragieai/ragie-python/blob/main/docs/sdks/documents/README.md
8. **Qdrant, Incremental Embedding Updates [primary]**
   https://qdrant.tech/documentation/tutorials-operations/incremental-embedding-updates/
9. **Storing raw data vs. storing only vectors**
   https://aboutvectordatabase.com/learn/storing-raw-data-vs-storing-only-vectors/

10. **Actian VectorAI DB FAQ (point structure, upsert semantics)**
https://docs.vectoraidb.actian.com/home/support/faq

11. **pgvector `embeddings` table schema (text + vector + JSONB metadata)**
https://github.com/ESousa97/py-rag-engine/commit/25d8663d7e2ba143a496422bd4dbc42460d097a9

12. **Wikipedia, last edited 2026-09-10**
https://en.wikipedia.org/wiki/Vector_database

13. **Ragie changelog (entity extraction context change, 2026-02-18)**
https://www.ragie.ai/changelog