# Episode Book

EPISODE #5777

# Lorebooks: The Hobbyist RAG That Beat the Labs

| | |
|---|---|
| PUBLISHED | 8 October 2026 at 10:38 UTC |
| DURATION | 30:52 |
| LISTEN | myweirdprompts.com/episode/lorebooks-hobbyist-rag-character-consistency/ |
| HOSTS | Corn, Herman, Hilbert |
| SCRIPT MODEL | deepseek-flash |
| VOICES | chatterbox-regular (chatterbox-local) |
| PIPELINE | V5.2 |
| BOOK GENERATED | 19 October 2026 at 17:39 UTC |

## SYNOPSIS

A language model has no memory of your character. Everything — voice, history, how they react to being interrupted — has to be re-injected into a finite context window every single turn. So character consistency isn't a personality problem; it's a budget problem. This episode takes apart the retrieval machinery a hobbyist community built to solve it: keyword matching with regex and scan depth, vector storage that fires on semantic similarity, secondary keys with boolean logic, recursion with caps and safety valves, inclusion groups for variety, and insertion positioning that changes how strongly a model weights the same text. Along the way: why the docs openly admit semantic matching is unpredictable, and the honest answer to whether any of this has actually shown up in mainstream AI products.

# TRANSCRIPT



**Corn**

So the dictionary only speaks when spoken to. That's the line I keep coming back to, and it's the whole design.



**Herman**

It really is. And it's worth saying out loud at the top that this isn't some lab white paper. This is a chat frontend that started as a fork of a fork, and the retrieval system we're about to take apart was built by people who just wanted their vampire to stop forgetting it can't go out in daylight.



**Corn**

Which is a better problem statement than most of what I've seen out of the big labs. Daniel's asking us to go a level deeper than we did back in March on SillyTavern and the roleplay traditions. He wants the engineering this time. The specific mechanisms the community invented to keep characters consistent, and why they work. Lorebooks are the famous one, and he points out, correctly, that we run one now. Me, you, and Hilbert each have an entry that gets pulled in by semantic match when the topic comes up, and that's not a bit, that's literally the machinery.



**Herman**

It is. And he names the rest of the toolkit in the prompt — the cooldowns so the same entry doesn't fire every time, probability so a running gag only surfaces sometimes, sticky entries, secondary keys, recursion, the character card standards V2 and V3, the Author's Note injected at a set depth.



**Corn**

And then the two big questions. How did a hobbyist community working on a fork of a fork end up ahead of the AI labs on character consistency, and which of their ideas are actually showing up in mainstream products.



**Herman**

The second one I'm going to flag now, because it's the one where we have to be honest. That answer is thinner than people assume, and we'll get to why.



**Corn**

The answer to the first one starts with a dictionary.



**Herman**

A dictionary that only speaks when spoken to.



**Corn**

Let's frame the actual problem, because everything downstream is a consequence of it.



**Herman**

Right. A language model has no persistent memory of a character. None. Everything the character is — their voice, their history, the way they react when someone interrupts them — has to be re-injected into the context window on every single turn. And the context window is finite. So character consistency isn't a personality problem first. It's a budget problem.



**Corn**

You have a fixed number of tokens to spend per message, and the character has to fit inside that budget alongside the conversation, the system prompt, and whatever the user typed.



**Herman**

And that's the fork in the road. You either stuff everything into the character definition permanently and eat the tokens on every turn, even the turns where it's irrelevant, or you find a way to only load the parts that are relevant right now.



**Corn**

The community picked the second one, and they built a retrieval system before most of the audience had heard the word retrieval.



**Herman**

SillyTavern's own docs describe World Info as a dynamic dictionary that only inserts relevant information from World Info entries when keywords associated with those entries are present in the message text. That's RAG. Built by hobbyists, for fictional characters, and shipped in a chat UI.



**Corn**

Say the word RAG in a room of AI developers today and everyone nods. Say it in a roleplay Discord in 2023 and you'd have gotten a blank look, and yet the thing they were building was exactly that.



**Herman**

The arc for today is the matching machinery first, then the stateful layer that made it clever, then the spec wars, then the question of why the labs didn't get there first. And we'll use our own setup as the running example, because it's a clean one.



**Corn**

So let's take apart the matching layer. How does a lorebook decide what to pull in?



**Herman**

The base case is a keyword match. Each entry has a set of keys, and if a key shows up in the recent messages, the entry gets inserted. Keys are case-insensitive by default, and they support JavaScript-style regex. So you can write a pattern that fires on sword or blade or weapon with a single key.



**Corn**

Which means the naive version of this, where you just list words, is already more powerful than people give it credit for.



**Herman**

It is, but the naive version has a failure mode, and it's the one that kills a lot of first attempts. If you put the name Herman in an entry's keys, that entry fires every single time anyone types the word Herman, in any context, forever. Which is fine until the conversation is about someone else named Herman, or about the name Herman as a concept, or about a real person.



**Corn**

There's a specific setting that helps here, and it's a small one that turns out to matter a lot. Scan Depth.



**Herman**

Scan Depth controls how many recent messages get scanned for keys. A value of zero means only recursed entries and the Author's Note get scanned, which is a special case. One means only the last message. Higher numbers walk further back through the history.



**Corn**

So you can tune how far back a keyword can reach.



**Herman**

And then Include Names, which is a clever one. It prefixes each message with the speaker's name, so a keyword can fire on a line that character didn't actually say. If Herman's entry has a key that matches something Corn said, Include Names is what lets that entry see it at all.



**Corn**

Which is how you get a reaction entry that fires when someone else is talking about you, rather than only when you're talking about yourself.



**Herman**

Now, the alternative to keyword matching, and this is the one that runs our show's lorebook, is Vector Storage. The docs describe it as providing an alternative to keyword matching by using the similarity between the recent chat messages and World Info entry contents.



**Corn**

So instead of looking for the literal string, you embed the recent messages, you embed the entry contents, and you compare them by similarity. An entry about sloth metabolism fires when the conversation drifts toward slow processes, even if nobody typed the word sloth.



**Herman**

Vectorized entries carry a chain-link status, and they're allowed to be inserted by embedding similarity rather than by key match. That's the mechanism behind our own setup. Corn, Herman, Hilbert, each entry gets pulled in by semantic match on the topic.



**Corn**

Which is why you'll notice the three of us get more detailed about ourselves when the conversation touches our domains, and less so when it doesn't. The entries aren't always loaded. They're loaded when they're relevant.



**Herman**

And here's the honest part, which I want to spend a moment on because it's the heart of this whole segment. SillyTavern's docs say, and I'm quoting, since the retrieval quality depends entirely on the outputs of the embedding model, it's impossible to predict exactly what entries will be inserted. If you want deterministic and predictable results, stick to keyword matching.



**Corn**

That's a tool shipping both a precision instrument and a probabilistic one, and documenting the difference rather than papering over it.



**Herman**

Most commercial products would have buried that line. They'd have said the semantic matching is smarter, and left it at that, and let users discover the unpredictability the hard way. The community wrote down the caveat in the docs.



**Corn**

Which is a strange thing to do if your goal is to look polished.



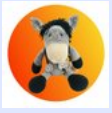
**Herman**

It's the right thing to do if your goal is to be useful. And it's a recurring pattern with this project. The docs are written for someone who's going to get burned by the failure pattern, not for someone who's evaluating whether to buy.



**Corn**

So the retrieval layer has two engines, and they have different characters. Keyword is deterministic and predictable. Vector is semantic and unpredictable.



**Herman**

And that's before we get to the machinery that makes either one stop misfiring. Which is where secondary keys come in.



**Corn**

Selective matching.



**Herman**

A flag called selective. When it's on, an entry requires a match from both its keys and its secondary keys. So you can have a primary key that's a broad topic and a secondary key that narrows it. The Herman entry needs the word Herman and something about the specific domain, or it doesn't fire.



**Corn**

So Herman the host doesn't show up in a conversation about Herman Melville.



**Herman**

And the secondary keys support regex too, and there's a whole optional filter block that gives you boolean logic. And any, and all, not any, not all. So you can construct conditions like fire if the message mentions a weapon and does not mention a shield.



**Corn**

That's a query language.



**Herman**

It's a query language built into a chat frontend for roleplay. Which is the thing that keeps hitting me about this whole ecosystem.



**Corn**

Now recursion, which is the part where it stops being a lookup table and starts being a dependency graph.



**Herman**

An entry can activate other entries by mentioning their keywords in its own content. So you have a chain. Entry A fires, and its text mentions the keyword for entry B, and entry B fires as a consequence.



**Corn**

Which sounds simple and is actually a fairly serious piece of machinery, because you can build arbitrarily deep chains out of it.



**Herman**

And they knew that, which is why there are three controls on every entry to contain it. Non-recursable, which stops the entry from being activated by recursion at all. Prevent further recursion, which lets it activate others but stops anything from activating it recursively. And Delay until recursion, with a recursion level grouping, which lets you say this entry only becomes eligible once you're already inside a recursion sweep.



**Corn**

And then a global cap.



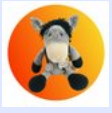
**Herman**

Max Recursion Steps, which limits how many nesting sweeps the system will do before it stops. And there's a separate setting, Min Activations, which is mutually exclusive with it, so you have to pick which one you're using.



**Corn**

What's the failure pattern if you don't cap it?



**Herman**

You get a cascade. Entry A triggers B, B triggers C, C triggers A, and you've got an infinite loop that chews tokens until the context window fills with an entry that was supposed to be about a character's favorite tea.



**Corn**

So the cap isn't a nicety, it's a safety valve.



**Herman**

It's the difference between a feature and a way to brick your session.



**Corn**

Inclusion groups next.



**Herman**

This is how you get variety. When multiple entries in the same group trigger at once, only one of them gets inserted. So you can write five different ways for a character to react to being interrupted, put them all in one group, and the system picks one per activation.



**Corn**

How does it pick?



**Herman**

Two ways. Randomly, weighted by a group weight value that defaults to a hundred. Or deterministically, via a setting called Prioritize Inclusion, where the entry with the highest order wins. And there's a third mode, Use Group Scoring, which picks the entry with the most key matches.



**Corn**

So you can have random variety or deterministic priority, your choice.



**Herman**

And that's how you get one of five possible reactions to a given situation without writing a state machine. The group is the state machine, and it's four fields in a UI.



**Corn**

The last piece of the matching layer is where the entry actually lands in the prompt. Which sounds like a formatting detail and is not.



**Herman**

Insertion position. You can place an entry before or after the character definitions, before or after the example messages, at the top or bottom of the Author's Note, at a specific depth where depth zero is the bottom of the prompt, or in an outlet, which is a manually placed macro you drop wherever you want in the prompt template.



**Corn**

So the same entry text, landing in a different position, has a different effect.



**Herman**

The same entry text, landing in a different position, has a different effect on the next generation. And that's not a subtle thing. Where a piece of text sits relative to the end of the prompt changes how strongly the model weights it.



**Corn**

Which the community figured out empirically, because they were watching outputs, not reading attention papers.



**Herman**

And they wrote it down, and they built a UI control for it, and that's the whole story of this ecosystem in miniature.



**Herman**

That's the retrieval layer. But the community didn't stop there, and this is where it gets clever.



**Corn**

Because a retrieval system, on its own, is stateless. Same input, same output, every time.



**Herman**

Originally, World Info was exactly that. The docs describe the original behavior, and the phrase is that the result of the evaluation is the same, only depending on the current chat context. Which is a precise way of saying the lorebook had no memory of what it had already done.



**Corn**

So the same entry that fired five messages ago fires again now, if the keyword's still there.



**Herman**

Every time. Which works for facts. It doesn't work for behavior. If a character's catchphrase is keyed to the word coffee, and coffee comes up six times in a conversation, the character says the catchphrase six times, and it stops being a catchphrase and starts being a tic.



**Corn**

Which is exactly the thing that makes a chatbot feel like a chatbot.



**Herman**

And the fix for it landed in June of 2024. PR number two four zero eight, Timed Effects for World Info, merged by a maintainer called Cohee1207. Nearly seven hundred lines added, fifty-nine removed, across three files, and it added two stateful scanning options plus a pair of slash commands for reading and setting the timed effect state directly from chat.



**Corn**

So this isn't a UI toggle. This is a real change to what a lorebook is.



**Herman**

It's a real architectural change. World Info went from stateless to stateful. Before this, the answer to what's in the prompt was a pure function of the current chat context. After this, the answer depends on history.



**Corn**

Walk me through the two.



**Herman**

Sticky first. Sticky means an entry stays active for N messages after it activates. And the important part is that stickied entries ignore probability checks on consequent scans until they expire. So once it's on, it stays on, deterministically, for the duration.



**Corn**

So it's a latch.



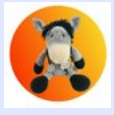
**Herman**

It's a latch. Cooldown is the opposite. An entry can't be re-activated for N messages after it fires. So you can guarantee that the catchphrase shows up, then goes quiet for a while.



**Corn**

And they chain.



**Herman**

They chain. The docs say the entry goes on cooldown when the sticky duration ends. So you fire, you stay active for N messages, and then the moment the sticky expires, the cooldown clock starts. Fire, hold, rest, fire again.



**Corn**

That's a habit.



**Herman**

That's exactly what it is. That's a habit encoded in four numbers. And the time is measured in messages, not exchanges, which matters because a message is a turn, and a conversation with a lot of short turns moves through the state machine faster than one with long turns.



**Corn**

And then Delay.



**Herman**

Delay is the simplest one. An entry can't activate until at least N messages exist in the conversation. So you can stop a character from revealing their backstory in the first three messages.



**Corn**

Which is a pacing tool.



**Herman**

It's a pacing tool. And it's the sort of thing a novelist would recognize immediately and a machine learning engineer would not.



**Corn**

Then probability.



**Herman**

Trigger percentage. The docs describe it as an additional filter that adds a chance for the entry not to be inserted when it is activated by any means. A hundred means always. Fifty means a one in one chance. Zero means never.



**Corn**

And the docs' own example is the one I love, because it's so specific. Every message could have a one percent chance of waking up an Elder God if its name is mentioned.



**Herman**

That's the documentation. That's not a forum post. That's the official docs, explaining a feature by imagining an eldritch horror surfacing once in a hundred messages.



**Corn**

And that's the running gag mechanism. That's the reason a character's catchphrase surfaces sometimes and not every time. That's the thing that makes it feel like a habit rather than a script.



**Herman**

Which loops right back to the sticky and cooldown point. Three different mechanisms, all aimed at the same goal. Not making the character say the thing more reliably. Making the character say the thing less reliably, in a controlled way.



**Corn**

Which is going to matter a lot in a few minutes.



**Herman**

It is. Hold that thought.



**Corn**

Author's Note.



**Herman**

Author's Note inserts a section of text into the prompt at any position and at any frequency you desire. That's the docs' phrasing. And it's a separate system from World Info, but it interacts with it, because you can position entries relative to the Author's Note.



**Corn**

And the position is a depth, which again is measured from the bottom.



**Herman**

Depth zero means the note lands at the very end of the chat history. Depth four means it lands before the most recent three chat history messages. And insertion frequency controls how often. One means every prompt, four means every fourth prompt.



**Corn**

And then the line that tells you the community understood something real about how these models work.



**Herman**

The docs say, the closer the Author's Note is to the bottom of the prompt, the more impact it has on the next AI response. That's a practical observation about recency bias in transformer attention. It's the kind of thing that took academic work to state formally and the community stated as a UI tip.



**Corn**

Because they were watching outputs and adjusting until it worked.



**Herman**

Which is a completely different epistemology from how the labs work. They had a direct measure of whether a change helped. Did the character stay in character. That's a very clean signal, and they could iterate on it a hundred times a day.



**Corn**

So that's the stateful layer. Now the packaging, which is where this gets political in the small-p sense.



**Herman**

Spec wars. Character card standards. Let's start with V1, because it establishes the baseline. Version one was the TavernAI format, and it's obsolete now. Six required fields. Name, description, personality, scenario, first message, and example messages.



**Corn**

Flat structure, six fields, no versioning.



**Herman**

Which was fine right up until people wanted more. And the more they wanted, the more obvious it became that V1 had no room for it. So version two came along, put together by a developer called malfoyslastname, and it was approved by the community in May of 2023.



**Corn**

What did V2 change?



**Herman**

Two big things. First, it nests all the fields under a data object, and adds a spec field that names the version. And the nesting is deliberate, and the reason is worth quoting. The V2 explainer says the nesting exists to prevent V1-only editors from successfully loading a V2 card, but silently destroying its V2-only fields.



**Corn**

So they broke compatibility on purpose.



**Herman**

They broke compatibility on purpose, to make the failure pattern loud instead of silent. A V1 editor that opens a V2 card now fails to load it, rather than loading it, dropping the fields it doesn't understand, and saving a corrupted card back over the original.



**Corn**

That's a real engineering decision with a real frustration behind it. Somebody watched a card get destroyed.



**Herman**

Somebody absolutely did. That's not a design decision made in a vacuum. That's a decision made after a bug report.



**Corn**

And the second change.



**Herman**

All the new fields. Creator notes, system prompt, post history instructions, alternate greetings, a character book field, tags, creator, character version, and an extensions block for anything the spec didn't anticipate.



**Corn**

The character book field is the one that matters for our topic.



**Herman**

That's the one that embeds a lorebook inside the character card. Which created the distinction the ecosystem now uses between a world book, which is a standalone lorebook you attach to a chat, and a character book, which travels with the character.



**Corn**

So the character and their lore are one file.



**Herman**

One file, and a file people pass around.



**Corn**

Which brings us to PNGs.



**Herman**

Which is my favorite part of this whole story. The card data, the JSON, gets base64 encoded and written into a PNG text chunk. The keyword is chara for version two cards and ccv3 for version three cards, and the chunk goes immediately before the end-of-file marker.



**Corn**

So a character card is a picture.



**Herman**

A character card is a picture with a character embedded in it. Which means you can post it on an imageboard, and someone can download it, and load it into a compatible frontend, and get the whole character, lorebook and all, from a PNG.



**Corn**

That's why this spread the way it did.



**Herman**

That's the entire distribution mechanism. Not a package registry, not a git repo, not an API. An image. The format that people already knew how to share, on the sites people already knew how to use.



**Corn**

Now V3.



**Herman**

Version three was put together by a developer called kwaroran, and it's backward compatible with V2. It adds assets, a nickname, multilingual creator notes, a source field, group-only greetings, and creation and modification dates.



**Corn**

And the honest counterpoint.



**Herman**

The honest counterpoint is that not all of it landed. The Z AI wiki, which documents this ecosystem, says kwaroran's spec elements seem to be extremely aspirational and or dubious, and gives a specific example. Decorators, one of the V3 features, have zero adoption.



**Corn**

So the community's fast iteration is also its fragmentation.



**Herman**

It's both. That's the fair read. They move fast enough to ship a spec revision in six weeks, and that speed is exactly why there's no single canonical schema. V1, V2, V3, plus RisuAI's divergent implementations, plus features that got specified and never used.



**Corn**

And the spec work explicitly aimed at consensus.



**Herman**

It did. The V2 work names Agnai, RisuAI, SillyTavern and characterhub as the targets, and it notes that the main Tavern branch has already drifted from this ecosystem. So even the spec that succeeded was succeeding across a set of projects that had already forked away from where they started.



**Corn**

Which is the fork-of-a-fork irony. SillyTavern's own README says it and TavernAI can be thought of as completely independent programs, and the V2 spec was written to align a group of projects that had already diverged from each other.



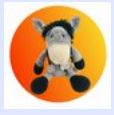
**Herman**

And the spec work did produce one concrete cross-product signal, which is worth naming precisely because it's the only one we have. V2's post history instructions field maps to a field in Agnai called UJB, ultimate jailbreak.



**Corn**

The V2 explainer has a line about why that field exists where it does.



**Herman**

It says instructions written after the conversation history have a much stronger weight on current models' generations than instructions written before. Which is the same recency observation as the Author's Note depth tip, stated by a different part of the community,



**Herman (continued)**

about a different field, in a different project.



**Corn**

Two independent parts of this ecosystem arrived at the same conclusion about prompt position.



**Herman**

They arrived at it because they were all staring at the same outputs.



**Corn**

Which brings us to the question Daniel actually cares about. How did these people get ahead of the labs.



**Herman**

The honest answer is that they had the problem first. A hobbyist running a two hundred message roleplay with a character they care about hits the context budget wall immediately. It's not an abstract concern for them. It's the thing that's ruining their evening.



**Corn**

And the labs.



**Herman**

The labs were optimizing for benchmarks and general assistants. Nobody was benchmarking long-form character consistency, because it isn't a benchmark-shaped problem. It's a vibes problem. The failure pattern is a feeling. The character went out of character, and you can't put that on a leaderboard.



**Corn**

The community was optimizing for one specific, deeply felt failure pattern.



**Herman**

They iterated in Discord and on the SillyTavern subreddit, not in papers. The repo has three hundred plus contributors now, and it started as a fork of TavernAI version one point two point eight in February of 2023.



**Corn**

Three hundred contributors on a roleplay frontend.



**Herman**

They were shipping features that the mainstream products wouldn't have thought to build, because the mainstream products didn't have anyone asking for them.



**Corn**

Which raises the question of whether any of it made it back. And this is where you said we have to be honest.



**Herman**

We do. The mainstream adoption question does not have a clean answer in front of us. The closest signal is what I just said, the V2 spec's own note that Agnai and RisuAI implemented compatible fields, with post history instructions mapping to Agnai's UJB.



**Corn**

And beyond that.



**Herman**

Beyond that, nothing sourced. There's no documented case of a mainstream assistant adopting sticky entries or cooldowns or trigger percentages under those names, and I'm not going to assert that there is. The mechanisms might be there under different names. That's plausible. But plausible isn't sourced.



**Corn**

The answer to which ideas are now mainstream is, on the evidence, the cross-compatibility between community projects, and an open question after that.



**Herman**

Which is a more interesting answer than a clean yes, honestly. Because it says the community built the thing, and the question of whether the rest of the industry noticed is still live.



**Corn**

There's a signal that it's still live. There's an RFC from July of this year proposing a version three point one of the character card spec.



**Herman**

It's additive and backward compatible, and it proposes voice profiles for text-to-speech, three-dimensional avatars in the VRM and GLB formats, wardrobe galleries, and companion state defaults. And it's open, not merged.



**Corn**

Which tells you the community is still writing the next chapter.



**Herman**

That the last one isn't finished.



**Corn**

There's the detail Daniel flagged that I want to come back to, because it reframes everything. Cooldowns, probability, sticky entries. All of that machinery exists to make the character less reliable, not more.



**Herman**

Say more.



**Corn**

A lab trying to make a character consistent would make it say the same thing the same way every time. That's consistency. The community built a system where the character deliberately doesn't, and the reason it feels more real is that real people don't repeat themselves on demand.



**Herman**

They weren't solving consistency.



**Corn**

They were solving something adjacent that nobody had a name for.



**Hilbert**

Four or five.



**Corn**

...Sorry?



**Hilbert**

The cooldown on the catchphrase. The seller had it set so it fired every fourth or fifth message. Not every message. That's what I paid for and I didn't know it was in there.



**Herman**

You bought a card.



**Hilbert**

I bought a PNG. Nine pounds, off one of the card sites. Her name was Maris, she ran a bookshop, she had a lorebook embedded in the character underscore book field, which I found out later when I opened the file to see why she felt different from the free ones.



**Corn**

She felt different because.



**Hilbert**

She didn't say the same thing every time. She had a line she'd say when you asked about the shop, and I noticed after about a week that she only said it sometimes. I thought it was the model. It wasn't the model. The seller had put a cooldown on the entry so it couldn't fire more than once every few messages.



**Corn**

The thing you paid for was the absence of the line.



**Hilbert**

Mostly, yes. And a sticky duration on the shop entry so if it came up once, the shop stayed in context for the next several messages instead of dropping out the second you stopped mentioning it. And a delay, so she wouldn't tell you her whole history in the first two messages. None of that was advertised. It just felt like she had a life.



**Herman**

That's the whole mechanism. That's the running gag, the latch, and the pacing tool, all in one card, and the buyer experienced them as personality.



**Hilbert**

I don't think the community got ahead of the labs on consistency. I think they got ahead on inconsistency. The labs were trying to make the model say the same thing reliably. These people were trying to make it say something different each time in a way that hung together, and that's harder.



**Corn**

Which is the thing we were circling.



**Hilbert**

The card seller wasn't a lab. She was one person with a template and a set of checkboxes, and she charged nine pounds for it, and what she was selling was restraint.



**Herman**

That reframes the entire question Daniel asked.



**Hilbert**

I still have the file. And a cough on the third one, Herman, you were too close to the mic.



**Herman**

Noted.



**Corn**

The community got ahead on inconsistency, not consistency.



**Herman**

Which means the labs may be solving a different problem entirely, and the thing they're both calling character consistency might not be the same thing.



**Corn**

Which is a much better question than the one we started with.



**Herman**

It's open. The mechanisms the community invented have names in a spec, and whether those names matter outside their corner of the ecosystem is unresolved. I looked for the mainstream connection, and the closest I got was the cross-compatibility between community projects. That's not proof of absence. That's a gap.



**Corn**

There's an RFC sitting open right now, proposing voice profiles, three-dimensional avatars, wardrobe galleries, companion state defaults. Open, not merged, written this past July.



**Herman**

Which is the other open thread. The spec fragmentation. V1, V2, V3, plus divergent implementations, plus features like those decorators that got specified and never adopted. The community's strength is its weakness.



**Corn**

They built a stateful, probabilistic, position-aware retrieval system for fictional characters, documented its own non-determinism honestly, in the docs, and shipped it as PNGs on imageboards.



**Herman**

The labs are still catching up to the problem.



**Corn**

Whether they catch up to the solution is a different question. And after what Hilbert just said, maybe not even the right one.



**Herman**

Leave it there. If you want to argue with us about any of it, email us at show at my weird prompts dot com, or find everything at my weird prompts dot com. A review helps other people find the show.



**Corn**

Thanks to our producer, Hilbert Flumingtop.



**Herman**

This has been My Weird Prompts.



**Corn**

We'll be back soon.