Episode Book

EPISODE #5768

Who Actually Draws the Follow-Up Chips?

PUBLISHED	7 October 2026 at 22:43 UTC
DURATION	28:39
LISTEN	myweirdprompts.com/episode/follow-up-chips-architecture/
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

When you finish a chat with an AI assistant, the little follow-up chips underneath look like the model thinking ahead. They're not. This episode walks the full pipeline: a separate, cheaper inference call fired after the main answer finishes, a schema-validated JSON array, a synthetic "puppet turn" smuggled into the prompt, and a UI-only data part that's stripped before the model ever sees it again. Then we turn to artifacts — documents, code blocks, spreadsheets — and why that's a completely different problem: a representation question about what the model emits and who turns it into something clickable. Along the way: the hallucination risk of a small model with no tool results, the post-filter that catches invented features, and the production failure where the chips ask you a question the assistant just asked. The through-line: almost none of this is the model deciding anything.

TRANSCRIPT



Corn

Here's what Daniel sent us this time. The short version is: all those little pieces of interface you assume are the model talking are usually not the model at all.



Herman

Right, and he's specific about it. Two examples, then a broad question underneath both.



Corn

The first one is the suggested follow-up chips. You get an answer, and underneath it there are two or three little buttons offering what you might ask next. Daniel wants to know what actually generates those. Is it another pass from the main model, is it a smaller dedicated model, is it a rules-based system, or is it some combination of the three. And then the plumbing on top of that. What gets passed into that system, and what does the request and response flow actually look like between the model, the API layer, and the front end that draws the chips.



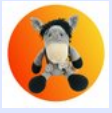
Herman

That's question one through three already.



Corn

Then the second example, artifact recognition. The model produces something and the interface decides it's a document, or an email, or a code block, or a spreadsheet, and gives you special controls for it. Daniel asks how that distinction is represented technically. Is the model returning structured data or metadata alongside the prose, are tool calls or schemas involved, or does some other layer classify the output after the fact.



Herman

And then the umbrella question over the top of both.



Corn

And then the umbrella question. Walk through the architecture behind these small pieces of AI user experience, from inference all the way through the orchestration layer to whatever ends up rendered on screen.



Herman

So the through-line is going to be that almost none of this is the model deciding to do something. These are layers around it.



Corn

Layers with their own costs, their own failure modes, and apparently their own job titles. Let's get into it.



Herman

Start with the core thesis, then, because it makes the rest of this simpler to hold in your head.



Corn

Please.



Herman

The dominant pattern for follow-up suggestions is a second, cheaper inference call. Not the main model. A separate call, fired after the main answer finishes, to a small fast model. And the artifact problem is a completely different kind of problem. It's a representation problem. What does the model emit, and who turns that into something you can click. So one of these is about a second inference, the other is about how output is encoded.



Corn

Same lesson underneath both, though.



Herman

Same lesson. The model doesn't render anything. It doesn't know what a chip is. It doesn't know what an iframe is. Everything you see has been parsed and drawn by something that is not the model.



Corn

Here's the thing that got me, though. Both of these features exist to make the thing feel less like a machine and more like a conversation. The chips feel like the assistant is thinking a step ahead. The artifact pane feels like the assistant understood what you actually wanted. And in both cases, the illusion is produced by the layer around the model, not the model.



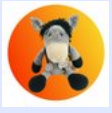
Herman

Which is why the architecture matters more than the surface behavior. Two things worth flagging before we go deep, because they structure the whole episode.



Corn

Go on.



Herman

First, there are two competing ways to do artifacts, and they are different architectures. One is inline tagged text that the client parses. The other is tool calls returning structured data that the host renders. We'll contrast them properly once we've done the mechanism. Second, there's a cost story almost nobody talks about in public, which is that every single suggestion is a second inference call. Fire one after every response, across millions of users, and that is a line item.



Corn

And nowhere in the interface does it say "this is being generated by a second, cheaper model, at your expense, right now."



Herman

Nowhere in the interface. Right, let's take the suggestion pipeline end to end, because that's the one with the nice paper trail.



Corn

The pipeline. What's actually happening after the answer finishes streaming.



Herman

Say the main answer is still streaming token by token into the chat window. Somewhere near the end of that, or right after it, a second request goes out. Vercel's AI SDK cookbook says it in one sentence: after the AI responds, generate contextual follow-up questions using a fast model. That's the whole architecture in a line. A fast model, separate from the main one.



Corn

Named models?



Herman

In the AI Hero tutorial, Matt Pocock wires it up with two variables. A main model that streams the answer, and a suggestions model, which in his demo is Gemini 2.0 Flash. Two models, two streams, one interface. The suggestion stream is written into a part of the same message the user is already looking at.



Corn

So while one model is still writing, the other one is already guessing what you'll click.



Herman

Already guessing. And the pattern shows up in places you wouldn't immediately expect. Claude Code has an internal prompt suggestion service, and according to a community reconstruction of its system prompts, it runs on what's described as a small fast model. Haiku, in their framing. The reconstructed prompt reads: you are predicting what the user will say next to an AI coding assistant. Given the conversation so far, suggest one to three short follow-up prompts the user might naturally say next.



Corn

One to three, not five.



Herman

One to three, each two to eight words, match the user's style, do not suggest things the assistant just completed, return a JSON array of strings. And I want to be clear about the provenance there, because it matters. That repo is explicitly labeled as reconstructed from source analysis. It is not an official Anthropic document. Treat the wording as illustrative, not authoritative.



Corn

Noted. Though reconstructed or not, the shape of it is exactly what you'd design if you were doing this for real.



Herman

The shape is what you'd design. And the DIY version predates all of it. There's an OpenAI developer community thread from February of twenty twenty-four where someone asks how to do follow-up questions, and the answer is basically: use another AI API call to the cheapest model, at high temperature. The reasoning in that post is the interesting part. It is best not to burden the original, and perhaps more expensive, AI on ancillary tasks, the outputs of which would also confuse chat history.



Herman

It's doing all the work. Because if you asked the main model to generate the suggestions inline, the suggestions become part of the transcript. And then on the next turn, the model reads back its own suggestions as though the user had said them. You've contaminated the conversation with your own UI furniture.



Corn

So the second call isn't only about cost.



Herman

It's about context hygiene. The suggestions live outside the transcript. They're not part of what gets replayed to the model next turn.



Corn

How do you actually pull that off, mechanically?



Herman

This is the part I find elegant. The suggestions come back as what the SDK calls data parts. Custom message parts, explicitly UI-only. The chatjs cookbook says it outright: these are streamed as data parts which are UI-only, they're filtered out before sending context to the LLM. There's a conversion function that strips them on the way back into the model.



Corn

So the transcript the model sees and the transcript the user sees are not the same object.



Herman

They're divergent views of the same message. The user sees the chips. The model never hears about them.



Corn

Hm. That's a cleaner design than I expected.



Herman

It's cleaner than I expected too, and I've read the code. Now, what goes in. Typically the conversation history, or sometimes just the last question and answer pair. And then, appended to the end, a synthetic user turn. Something the user never typed.



Corn

The prompt smuggled in as if you'd asked it.



Herman

Exactly that. The chatjs version appends: what question should I ask next, return an array of three to five suggestions, max eighty characters each. The AI Hero version appends: what question should I ask next, return an array of suggested questions.



Corn

So there's a fake user message sitting at the bottom of the conversation saying "what should I ask next," and the model answers as if that were a real request.



Herman

It's a puppet turn. And then the output is constrained. chatjs validates with a schema that requires an array of strings, minimum three, maximum five. AI Hero's is looser, just an array of strings. And OpenAI's structured outputs, which landed in August of twenty twenty-four, guarantees the response actually conforms to a supplied JSON schema when you set the strict flag.



Corn

Which matters more than it sounds like it should, because a chat interface that's expecting an array and gets a paragraph has a bad day.



Herman

It has a very bad day. This is the difference between a feature that works at scale and one that produces a stray brace on somebody's screen once every few thousand requests. Schema enforcement is unglamorous and it's most of the engineering.



Corn

There's a hallucination angle here too, isn't there.



Herman

There is, and it cuts against the cheap model. The laguagu repo, which is a set of Claude Code skills for Next.js chatbots, documents it plainly. That small model sees only the last question and answer pair. No tool results. And it's tuned for speed, which makes it more prone to confident hallucination than the main chat model.



Corn

So it invents features that don't exist.



Herman

It invents product names, menu items, endpoints, whatever the domain is. Because it's completing the pattern of what a user might ask, and it doesn't have the grounding the main model had. The suggested mitigation in that doc is an anti-invention rule in the prompt, plus post-filtering the suggestions against the last tool result payload.



Corn

Filter the chips against reality before you show them.



Herman

Which is another layer. The count keeps going up. Main model, suggestion model, schema validation, anti-invention rule, post-filter.



Corn

And here's the one I want to spend time on. You said combination, earlier, when you listed the options.



Herman

Combination is the correct answer, and it's the part I find most counterintuitive. The full pipeline isn't rules or a model. It's rules and a model, in sequence.



Corn

Explain.



Herman

The laguagu doc found a failure pattern that only shows up in production. If you use one generic prompt for suggestions, you get a new question every single time. Including right after the assistant itself just asked the user something. So the assistant says "would you like me to format this as a table or as prose," and then the chips underneath say "what format should I use?" It's asking the user to answer a question it just asked.



Corn

That is a very specific kind of stupid. The interface looks like it isn't listening.



Herman

It looks like it isn't listening, and that's the exact moment users stop trusting the chips. So their fix is to classify the tail of the answer before generating anything. A function runs over the last sentence or two, and returns one of three modes. Either the answer ended by offering options, in which case the chips should be the options. Or it ended with an offer to do something, in which case the chips are accept or decline. Or it ended open, in which case you generate a fresh question.



Corn

And that classifier is not a model call.



Herman

Detection is a small function over the tail. Not an LLM call. It's a rules-based check, and then a different generation prompt per mode.



Corn

So the answer to Daniel's question, phrased precisely: it's a combination. A rules-based classifier reads the last couple of sentences, decides which of three shapes the conversation is in, and then a small model generates the actual text within that shape.



Herman

And the rules layer is doing the part the model is worst at. The model is good at writing plausible next questions. It is bad at knowing whether the last thing it said was already a question.



Corn

There's something almost human about that division of labor, actually.



Herman

There's something very human about it. The rules layer is the one that remembers what just happened. The model is the one that's good with words. That's a division you see all over good systems, and people keep trying to collapse it into just the model.



Corn

Okay. Suggestions are a second-inference problem. Artifacts are something else entirely.



Herman

Artifacts are a representation problem. The question isn't what generates the output. It's how the output is encoded, and who interprets the encoding.



Corn

So take Claude's artifacts, since that's the one that got reverse-engineered.



Herman

Shipped June of twenty twenty-four alongside Claude 3.5 Sonnet. And the mechanism, once Reid Barber pulled apart the raw HTTP response, is almost disappointingly simple. The model just writes tags inline in its normal text stream. You get a tag, an identifier attribute, a type attribute, a title attribute, and then the content, and then a closing tag.



Corn

It's just markup. In the middle of the prose.



Herman

Just markup in the middle of the prose. And the type attribute is a media type. Things like application slash vnd dot ant dot react, or application slash vnd dot ant dot code, text slash markdown, text slash html, image slash svg plus xml, a mermaid type, a react type. The identifier is what lets it update the same artifact on a later turn instead of spawning a new one.



Corn

And the client is what turns that into a pane.



Herman

Barber's line is the one to hold onto: it's really the job of the client to parse the LLM response and then go render things when needed, so the LLM doesn't even need to know about that step.



Corn

The model is writing XML at you and has no idea what happens next.



Herman

No idea. It emits a token sequence that happens to be well-formed markup, and a parser on your machine decides that this is a document and that a React component should be instantiated in a sandboxed frame.



Corn

Then the question is how the model knows when to reach for the tag.



Herman

The leaked system prompt, which was extracted and then reproduced by Barber, is where that lives. The model is told to use an artifact for substantial content, and the number in the prompt is fifteen lines. Also for content intended for eventual use outside the conversation. Reports, emails, presentations. And it's told when not to. Prefer in-line content when possible. Unnecessary use of artifacts can be jarring for users.



Corn

They wrote a rule in the prompt telling the model not to be annoying.



Herman

They wrote a rule in the prompt telling the model not to be annoying, and it's phrased as a user-experience concern, which is a fun thing to find in a system prompt. It also instructs a short self-check, a one-sentence internal note before it invokes an artifact. Thinking about whether this is worth a pane, essentially.



Corn

Does the rendering happen on their servers?



Herman

It happens client-side, in an iframe loaded from claudeusercontent dot com. Barber went through the bundle, and found Tailwind, React DOM, DOMPurify, Radix, Lucide, React Runner. Content gets passed in via window dot postMessage.



Corn

Which sounds terrifying until you read how they sandbox it.



Herman

Anthropic's security engineer, Ziyad Edher, described it to Pragmatic Engineer: they're not using any actual sandbox primitive, they use iframe sandboxes with full-site process isolation, plus strict content security policies to keep network access limited and controlled.



Corn

So the isolation is the browser's own origin model, plus a policy that stops the frame calling out.



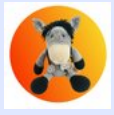
Herman

Which is a real boundary, but it's worth being precise about what it is. It's the same primitive any web app has. They've used it well. It's not a magic box.



Corn

And then the newer prompts went further.



Herman

Analysis of a later set of leaked system prompts describes a routing checklist before the model produces anything. Needs external data, go to web search. Standalone document or code, create an artifact. Needs a visual representation, invoke a visualizer. Needs to persist across sessions, write to storage.



Corn

So it's graduated from "should this be an artifact" to "which of four subsystems does this need."



Herman

And the visualizer has module types. Diagram, mockup, interactive, chart, art. And there's a function the rendered artifact can call to send a prompt back into the chat. The thing on the right can talk to the thing on the left.



Corn

That's the point where the artifact stops being output and starts being a participant.



Herman

It is, and it's the first place in this whole discussion where the model is deciding between branches. Because it has a menu of four destinations and it has to pick one.



Corn

Right. But I want to contrast that against the other architecture, because the contrast is the thing Daniel's really asking about.



Herman

The tool-call route. Instead of the model emitting tagged text that a client parses, the model calls a tool, the tool returns a structured object, and the host renders that object. OpenAI's Apps SDK docs describe exactly this. UI components that turn structured tool results from your MCP server into a human-friendly UI, running in an iframe, talking to the host through a bridge that's JSON-RPC over postMessage. The messages have names like ui slash notifications slash tool result.



Corn

So the contract is explicit. There's a protocol.



Herman

There's a protocol, and there's a schema. LangChain's docs frame the same idea: instead of returning free-form text, the agent uses a tool call to return a structured object conforming to a predefined schema, and you map that object to cards, or tables, or charts, or step-by-step breakdowns.



Corn

Now compare them properly. What does each one buy you.



Herman

The inline tag approach keeps the model ignorant of rendering. The artifact is a parseable substring of the text stream. That's simple for the model, and it means the model can produce an artifact mid-sentence if it wants to. The cost is that the client has to parse the stream, and malformed output is a real failure pattern. You're relying on the model to close its tags.



Corn

Whereas the tool call makes the artifact a typed object.



Herman

A first-class typed object, validated against a schema before anything renders. The model has to know about the rendering contract, which is a burden on the model, but the validation is real. And it's the same machinery as any other tool call, so you get the retry behavior, the error handling, all of it for free.



Corn

So inline tags are cheap for the model and fragile at the edges. Tool calls are expensive for the model and robust at the edges.



Herman

And the interesting thing is the two are converging. Claude's newer prompts describe a routing checklist that dispatches to tools. OpenAI's Apps SDK makes tool results first class. If the model is calling a named tool that returns a schema-shaped object, and the host renders from the tool result, then the difference between "inline tag" and "tool call" is mostly about who does the parsing.



Corn

That distinction may not survive the next couple of years.



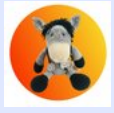
Herman

I don't think it does. Though I'll say I'm not certain, because the inline approach has one real advantage: the model can emit it anywhere, including halfway through a sentence, and a tool call is a discrete event. There may be room for both for a long time.



Corn

What else is lurking in this architecture that nobody mentions.



Herman

Cost. The laguagu doc has a line: suggestions run after every response, cost adds up fast. And that's the whole story of the invisible second model. Every answer you receive triggers at least one extra inference call, on a cheaper model, that you never see and never asked for.



Corn

Multiply it out.



Herman

Multiply it out across a user base and it's a real budget line. And it's a line that only exists because of a UI decision. The chips aren't a product feature in the marketing sense. They're an engineering cost that exists to make a chat window feel more alive.



Corn

Which is why some of them are so sensitive about it. I remember seeing user complaints.



Herman

There was a thread on the OpenAI forum in May of twenty twenty-five where someone described the follow-up suggestions as disruptive and uninvited, and said they clutter the interface and interrupt flow. Which is a completely legitimate reaction from the other side. The same feature that scaffolds one person's thinking is clutter to another.



Corn

And the research literature backs the skepticism, doesn't it.



Herman

There's a dataset paper from twenty twenty-three, FOLLOWUPQG, over three thousand real question, answer, follow-up tuples mined from an explainer forum on Reddit. Their conclusion was that model-generated follow-up questions are adequate but far from human-raised questions in terms of informativeness and complexity.



Corn

Adequate but far from human.



Herman

And there's a paper from CIKM in twenty twenty-five noting that most existing methods rely on hand-crafted rules, the internal knowledge of the model, or external knowledge, and proposing that you mine real conversation logs instead. Which is an admission that the hand-crafted rules approach has limits.



Corn

So the rules layer is both the thing that makes it feel designed and the thing that caps how good it can get.



Herman

Both. And here's the last thing, which I think is the most honest sentence in this whole episode. As far as I can find, there's no official OpenAI documentation of how ChatGPT's own follow-up suggestions are generated. Nothing first party. Structured outputs are documented. The Apps SDK is documented. But the suggestion mechanism specifically is not. Everything we know comes from developer community threads, open source SDKs, and reverse engineering.



Corn

So we're describing the pattern from the outside, and the pattern happens to be consistent everywhere it's visible.



Herman

Consistent everywhere it's visible. Which is a good sign that it's the actual pattern. But I'd rather say that plainly than imply I've read a spec that doesn't exist.



Hilbert

I once worked at a place where the suggestion box wasn't a model at all. It was a lookup table. Two hundred and forty rows in a spreadsheet, printed once a quarter, and a woman named Deirdre maintained it by hand.



Corn

Two hundred and forty rows.



Hilbert

Two hundred and forty. And she had a title. Conversation designer. Which nobody had heard of, and which the company eliminated about a year after she left, because the reasoning was that the model could do it now.



Corn

And could it?



Hilbert

The suggestions got worse. Not because the model was bad, but because the model didn't know that if someone's account was in a certain state, you must never suggest the thing that sounds obvious, because doing it locks the account for six hours. Deirdre knew that. It wasn't written anywhere. It was in the table.



Corn

Her rules were the product knowledge nobody had documented.



Hilbert

Her rules were the product knowledge nobody had documented, and when the table went away, so did the knowledge. That's the whole story. I don't have a lesson for you.



Herman

There's something in that that maps onto the classifier we were just talking about, though. That detection function over the tail of the answer, deciding whether the assistant just asked a question. That's a rule about what the conversation just did. It has nothing to do with generating text. It's a rule somebody wrote because they'd watched the failure happen.



Hilbert

Deirdre had one of those. Never suggest a question the user just answered. She said it in meetings constantly. It was the only rule she ever repeated.



Corn

That exact rule shows up in the open source documentation. Do not suggest things the assistant just completed. Word for word, almost.



Hilbert

Then it wasn't just her. Somebody else watched the same failure.



Herman

The interesting part is that this rule can't be discovered from a dataset. It's not a pattern in text. It's a property of the interaction. Somebody has to sit with the product long enough to notice the moment it looks stupid, and then write it down.



Corn

The model can't notice it, because the model isn't the thing having the experience.



Hilbert

Also, hold on, one second. Levels are a little hot on my end. Going to pull the second channel down a touch. Where was I. The table. The table went away in the spring, and the row about account state got replaced by a paragraph in a prompt, and the paragraph didn't work as well, and nobody could prove it, because by then Deirdre had left and nobody remembered which rows mattered.



Corn

Do you still have the spreadsheet?



Hilbert

I have a printout of it somewhere in a box. I'm not going to go find it. Anyway, that's my piece.



Herman

What you just described is the whole hybrid architecture. The rules layer is a human artifact. It's somebody's accumulated observation about how the product fails, written down as a check. And the model generates within it.



Corn

Which raises the question of whether that layer is permanent or whether it gets absorbed.



Herman

My honest guess is permanent, at least for anything with real product surface area. The model can learn general patterns. It can't learn that this particular account state means you must not suggest the obvious thing, because that fact isn't in the training data and it isn't derivable from the conversation.



Corn

It's derivable from having run the product for two years.



Herman

From having run the product. And that means somebody has to sit there and write the rule.



Corn

The second thing that doesn't go away is the cost. If every response spawns a suggestion call, and now also a routing decision, and potentially a visualizer invocation, and maybe a storage write, the number of inference calls per user interaction has gone up and it's not obviously going to come back down.



Herman

It's gone from one call per turn to something like two or three, depending on how many of these features fire. And the small models are cheap, but cheap times every response times every user is still a number somebody has to put in a budget.



Corn

The third thing is the interface itself. Follow-up chips are polarizing in a way I don't think the industry has reckoned with. Some people find them useful. Other people find them patronizing.



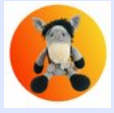
Herman

There's a version of this where we end up with per-product toggles, and then a whole new class of settings, and then a cottage industry of articles about which ones to turn off.



Corn

It's the same arc as autoplay and notification badges. Every interface convenience eventually gets a settings page.



Herman

Which brings it back around. The chips exist to make the product feel like a conversation. And the mechanism that produces them is a rules check on the last two sentences of the previous answer, feeding a prompt into a cheap model, whose output is stripped out of the transcript before the real conversation continues.



Corn

Put it that way and the magic is fully gone.



Herman

The magic was never in the model. It was in the layers. That's the part I'd want people to take from this. The next time you see a feature and assume the model chose to do it, the odds are strong that something else chose and the model just supplied the words.



Corn

The something else was often written by a person who watched it fail a few hundred times first.



Herman

Which is the least glamorous and most durable part of the stack. Thanks to Hilbert Flumingtop for keeping us on the rails. This has been My Weird Prompts.



Corn

If you want to send us a prompt like Daniel does, email us at show at my weird prompts dot com. We'll be back soon.