Episode Book

EPISODE #6294

Why Tool Calls Break Across Coding Models

PUBLISHED	27 October 2026 at 11:15 UTC
DURATION	22:49
LISTEN	myweirdprompts.com/episode/tool-call-model-migration-37679/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	openai/gpt-6.1-sol
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.22
BOOK GENERATED	28 October 2026 at 09:52 UTC

SYNOPSIS

Moving coding setups from Claude Code to environments like OpenCode using Kimi models reveals hidden incompatibilities beneath simple dropdown switches. Tool schemas, Anthropic's user-role tool results, and provider-specific chat templates handle calls entirely differently. Inspecting how native tokenizer tokens and inference parsers package structured function calls explains why an agent might declare a tool missing even when its configuration file opens flawlessly.

TRANSCRIPT



Corn

Switching coding models gets sold as changing a dropdown. Daniel hit his weekly Claude Code limit and discovered the dropdown had a basement. He moved his work into OpenCode with Kimi and other models, and one of them told him his skills and MCP definitions were in “Claude-native format.” He wants to know what that actually means, and what gets mangled when a Claude Code setup travels.



Herman

The limit interrupts the project, then the replacement environment asks you to become an integration engineer before continuing it. You still have the repository. You still have the MCP server. Nothing obvious has disappeared, but suddenly the agent talks about your configuration as though it arrived on magnetic tape.



Corn

My subscription includes an unexpected systems course.



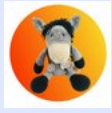
Herman

That phrase, “Claude-native format,” sounds like a specification name. It almost certainly wasn't one. There is an Anthropic Messages format, and there are Claude-specific conventions. The model appears to have bundled several different things into one confident label.



Corn

Which matters because converting the wrong thing could leave Daniel with rewritten skills and exactly the same broken tool calls.



Herman

Yesterday's episode on why coding-agent configurations don't travel covered the file layer: AGENTS.md, skill discovery, plugin folders. Today is one layer down, at the wire. There are three places to separate. The messages carrying tool calls, the schema describing each tool, and the instructions assuming a particular coding environment. A setup can work at one and fail at another.



Corn

I suspect the model saw the word Bash in a skill and diagnosed the entire directory.



Herman

That's plausible, though we don't have its trace. The file format itself is a weak suspect. Agent Skills is an open specification, and OpenCode explicitly supports discovering skills in Claude's skill directories. The interesting possibility is that it read the file successfully and then encountered instructions addressed to tools it didn't possess.



Corn

It understood the manual. It couldn't find the buttons.



Herman

Meanwhile, underneath that perfectly readable Markdown, the tool-call machinery really does differ. Anthropic's API expresses calls using its own content blocks. Kimi K2 commonly reaches clients through an OpenAI-compatible interface, while internally generating its own special-token sequence. That's a real boundary requiring translation.



Corn

Before you take us into the tokenizer, Herman, I'm predicting you're going to admire the parser until something depends on punctuation.



Herman

I admire parsers especially when something depends on punctuation. That's their occupation.



Corn

Yours is apparently defending them.



Herman

Only provisionally. Also, a small distinction about that label. People do talk about “Claude-native” skill files, meaning files written for Claude Code's conventions. That describes an implementation choice, not a universal format with a conformance test.



Corn

Would you treat the model's diagnosis as evidence about its own runtime?



Herman

Not without corroboration. It can recognize a mismatch in instructions. It doesn't necessarily know whether a provider adapter renamed a field, whether a parser was enabled, or whether the MCP client filtered out a tool before the model ever saw it. Those are different observations.



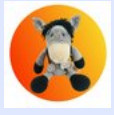
Herman

Corn, suppose the skill opens correctly, but the agent says a tool doesn't exist. Where would you look first?



Corn

The actual list of exposed tools. Not the source file containing the tool I intended to expose. Those can be different lists, especially after authentication, discovery, and somebody's helpful filtering.



Herman

That distinction saves a lot of pointless rewriting. The configuration on disk describes intentions. The request reaching the model describes its available world.



Corn

Take our weather tool. I want the exact point where those worlds diverge.



Herman

With Anthropic, you declare a name, a description, and an input schema. The field is called input_schema. For a weather tool, the schema might require a city string. Claude responds with a tool_use block containing an identifier, the tool's name, and an input object containing Beijing.



Corn

The model has requested the weather, not fetched it.



Herman

Your application executes the request. The response also has a stop reason indicating tool use, so the client knows why generation paused. After running the weather function, the client sends a tool_result block tied to that original identifier.



Herman

Here's the slightly surprising part. That result goes inside a user-role message. It's structured tool output, but its enclosing message has the user role.



Corn

Beijing's temperature has impersonated me.



Herman

The content block distinguishes it from an ordinary user instruction. Still, if you move that conversation into another API by copying role labels alone, you lose that distinction. You need to translate the blocks and their relationships, not merely rename assistant and user.



Corn

An export that preserves every word could still destroy the conversation's execution history. The weather result survives as text, but the next model no longer knows which pending call it answers.



Herman

Yes. Call identifiers are bookkeeping with consequences. They connect a specific request to a specific result, including when several calls are outstanding. A result attached to the wrong call can be syntactically valid and operationally useless.



Corn

Kimi receives the same weather schema differently?



Herman

At its OpenAI-style API boundary, the tool is declared as a function. Its name and description sit inside that function declaration, and the schema lives under parameters instead of input_schema. The returned assistant message carries tool calls, with function arguments typically encoded as a JSON string. Results return in messages with the tool role and the corresponding tool-call identifier.



Corn

Same city requirement. Different paperwork.



Herman

Mostly the same schema underneath, yes. Don't infer identical support for every JSON Schema feature across providers. A provider can reject or ignore constructs another accepts. Renaming the schema container doesn't establish that every constraint inside it will behave identically.



Corn

That's where I'd separate a rejected request from an agent choosing the wrong value. If the provider won't accept the schema, the model hasn't yet had an opportunity to disappoint anyone.



Herman

A valuable distinction. For Daniel's specific setup, I can't tell you which adapter performed that conversion. OpenCode's skill documentation doesn't establish its entire tool-schema conversion path. We should not turn a likely architecture into a claim about his exact request.



Corn

Thank you. The invisible adapter was beginning to acquire a biography.



Herman

Underneath Kimi's API response, there is another representation. Its chat template places the tool declarations in one synthetic turn at the top, before the ordinary system and user turns. Then its generated calls use dedicated markers for the beginning of the call section, the beginning of each call, the arguments, and the endings.



Corn

Those markers aren't decorative text the model happened to invent.



Herman

They're tokenizer entries. The call-section opening marker, for example, is a single reserved item in the vocabulary with its own fixed number. It isn't an English instruction telling the model to start a section.



Corn

You've reached the numbered punctuation.



Herman

You predicted it accurately.



Herman

The useful detail isn't memorizing the number. It's how the function name is represented. In Kimi's native call identifier, a functions prefix is followed by the function name and a call index. For our weather request, the parser extracts get_weather from that identifier, then reads the city from the argument section.



Corn

Rather than finding a separate name field already waiting for it.



Herman

Right... rather than finding one at that stage. The inference server then constructs the familiar OpenAI-style tool-call object. Both vLLM and SGLang have a Kimi K2 tool-call parser. With vLLM, automatic tool choice also needs enabling.



Corn

The harness gets a tidy function call because something upstream has interpreted the untidy one.



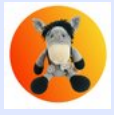
Herman

The native one. It's tidy according to its own grammar.



Corn

You did defend it.



Herman

I did. A correct parser makes that grammar largely invisible to the client. A missing or incompatible parser can leave raw tool markers in ordinary assistant content instead of producing executable calls.



Corn

Which would look remarkably like a model saying it wants to use a tool, then doing nothing.



Herman

There's a documented Kimi discussion about exactly this class of trouble. A nonstandard tool-call identifier triggered parsing failures, with raw call markers leaking into the content field. The intended action existed in the generated output, but didn't become the structured object the application expected.



Corn

I wouldn't convict the model on that output alone. Nor would I automatically acquit it. It could have emitted something outside the expected grammar.



Herman

Good correction. I was about to say “parser bug,” too broadly. It can be parser compatibility, template selection, or malformed model output. The visible symptom doesn't assign responsibility. You inspect the raw output and the parser's expectations.



Corn

Herman, your error logs always get a more thorough examination than your installation decisions.



Herman

My installation decisions provide the patients. Would you rather the tool fail loudly at that boundary or fall back to interpreting the text?



Corn

Loudly. A weather request is harmless enough, but I don't want a coding client improvising execution because some prose resembles a shell request. Preserve the failed call for inspection. Don't reward ambiguous output with authority.



Herman

That boundary is worth keeping strict. Another subtlety is placement. Kimi's template expects those tool declarations together at the top. Pasting an MCP schema into an ordinary mid-conversation message doesn't reproduce that arrangement.



Corn

Even if the schema is impeccable.



Herman

Even then. The model can read it as text, but the trained tool-use template provides additional structure. An API can accept fresh tool definitions on each request and rebuild the prompt correctly. That's different from literally inserting declarations as conversational prose.



Corn

What the client sends to a provider and what the model receives after templating are not the same artifact.



Herman

Now, a brief word from Corn's mother. I'm always delighted when she joins us.

D

Dorothy

Hello dear, it's your mother. Dorothy's History of Toast again.

D

Dorothy

The word toast comes from the Latin torrere, to scorch or parch. That's the whole fact, sweetheart, and it's true.

D

Dorothy

Scorched and parched, that's the story of toast, really. Nothing turns out soft, it all just gets dry in the end.

D

Dorothy

Anyway, I've got the kettle on and your father's socks to sort, so I'll ring off. Bye bye now.



Corn

Thanks, Mum. Back at the client boundary, an MCP server hasn't necessarily asked for any of those model-specific envelopes.



Herman

MCP exposes tools with descriptions and input schemas. The coding client discovers them and renders them into the provider's tool interface. The server can remain unchanged while the client presents its tools differently to Claude or Kimi.



Corn

That makes MCP a comparatively good traveler. The travel agent is the concern.



Herman

Anthropic introduced MCP in November two thousand twenty-four, then donated it to the Linux Foundation's Agentic AI Foundation in December two thousand twenty-five. Its origin doesn't make a server's tool definitions exclusively Claude-shaped.



Corn

A tool can still have Claude-shaped prose in its description, though. If it says “after this, call Read,” the schema can be portable while the instruction points at an absent tool.



Herman

That's the semantic mismatch, and skills make it especially visible. A skill is a directory with a required Markdown entry file, YAML frontmatter, and optional scripts, references, or assets. The open specification requires a name and description. The name must match the directory, with lowercase alphanumerics and single hyphens.



Corn

A format OpenCode already knows how to discover in Claude's project and home skill locations, as well as its own and the shared agents locations. Daniel doesn't need to move everything just because the directory says Claude.



Herman

Discovery is only the beginning. OpenCode advertises available skills through its skill tool's description. The agent loads a skill by calling that tool. Claude Code also merges custom commands into its skill system, so a deploy command file and a deploy skill can provide the same user-facing slash command.



Corn

Identical instructions on disk, different routes into the context.



Herman

There's also Claude Code frontmatter that disables model invocation, leaving triggering to the user. Another host reading the Markdown doesn't automatically inherit that switch's meaning.



Corn

I'd be more worried about silently ignoring that than failing to load the file. A skill that only runs when I explicitly ask is a different thing from one an agent decides to activate while tidying.



Herman

Corn, would you accept automatic translation of those controls if the destination harness had something roughly similar?



Corn

Not “roughly.” Translate the operational instructions freely enough, but show me what happens to permissions and triggering. A list of allowed tools can mean authorization in one environment and merely be advisory text in another. Similar spelling isn't sufficient.



Herman

The open skill specification explicitly calls allowed-tools experimental and warns that support varies between agent implementations. Claude Code adds its own permission machinery, including automatic modes and pre-approved commands. Those policies don't travel just because the field survives.



Corn

Permission copied successfully. Permission not implemented.



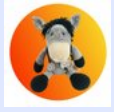
Herman

There's a small vendor fingerprint in naming, too. Anthropic's authoring guidance forbids the words Anthropic and Claude in the skill name. That's an additional host constraint, not evidence that the whole open skill format belongs to Anthropic.



Corn

The branded format forbids the brand.



Herman

Inside the skill body, “use the Bash tool to run the tests” may assume a tool literally named Bash with a particular input shape. OpenCode may expose a differently named shell tool. “Use Read, then Edit” similarly assumes a specific roster. MCP tool names may also be namespaced differently by clients.



Corn

This is where I'd retain the objective and specialize the action. “Run the repository's test suite” can be shared. How to invoke the shell tool, where output arrives, whether it continues in the background... those belong to the host.



Herman

Claude also has training familiarity with Anthropic-published tool schemas, including shell, editor, computer, and memory tools. Another model can understand the schema without having precisely the same learned habits around it.



Corn

You're saying training gives Claude a prior, not a monopoly on comprehension.



Herman

Yes. I wouldn't claim Kimi can't operate a shell because it wasn't trained on one particular Anthropic schema. The difference is how much the model must infer from the present description versus recognize from prior training.



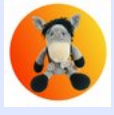
Corn

Your enthusiasm was approaching a proprietary screwdriver argument.



Herman

I've met enough proprietary screws to be cautious.



Herman

Progressive disclosure helps contain all this. Skill discovery initially loads a name and description, on the order of a hundred tokens per skill. Activation brings in the body, recommended below five thousand tokens, with supporting resources accessed as needed.



Corn

That also explains why a setup can look healthy until a specific skill runs. The initial catalog is portable and small. The host-specific assumptions stay hidden in the body until activation.



Herman

Some skill authors have already split along those lines, shipping separate builds for Codex, Claude Code and Kimi Code. The rule is to use each host's native syntax, invocation, metadata and tools rather than one entrypoint crowded with conditional instructions.



Corn

Kimi Code there means a host, incidentally. Daniel using Kimi inside OpenCode doesn't give him Kimi Code's tools.



Herman

Important distinction. The model name and the application name aren't interchangeable. Splitting by host is a choice about specialization, not a universal migration recipe.



Corn

I'd keep a shared source for the procedure, with thin host-specific entrypoints. Otherwise the test-running skill acquires a bug fix in one copy and quietly forgets it in two others.



Herman

Duplication has maintenance costs. Translation has fidelity costs. I'm not sure how much can be generated reliably without making permissions look more portable than they are.



Herman

Would you preserve old tool-call history when moving a live project, Corn, or begin with a fresh session and a summary?



Hilbert

...a fresh session, after the second afternoon. My cousin Miriam is coming on Thursday to help me move the clackers onto separate machines. I've already bought the numbered switches. The current arrangement keeps three coding sessions open, and they submit tool requests to a ticket printer. Miriam carries each ticket upstairs before I approve it. The stairs are in a rented wedding cake. The council lists it as temporary accommodation.



Corn

The clackers have acquired a substantial transport delay.



Hilbert

Seventeen minutes on Monday. The printed identifier had lost its final call index. I spent the afternoon changing the skill text. Nothing happened. The parser still couldn't turn the response into a call. Herman, you carefully separated all that before you reached the token number. I should have stopped at the identifier.



Herman

Rewriting the instructions wouldn't restore the missing identifier. That belongs to the call's representation.



Hilbert

I had a little thermal clacker for the tickets. Returned it yesterday. It clipped the right edge. The replacement prints sideways. Thursday's plan is to put the approval desk on the cake's rotating stand, so the requests come to me. I've bought a six-metre extension cable and a padded stool. The stand is arriving separately.



Corn

Your approval queue now has a direction of rotation.



Hilbert

Clockwise. The estate agent insisted. Living with the clackers is mostly waiting, then finding out one has asked for something quite sensible. The precise matching of a result to its request is fascinating. I like that part. The sales pages can keep the rest. Miriam has labelled the new paper tray “unmatched.” It already contains thirty-one tickets.



Herman

Thirty-one unmatched requests is a useful reminder that successful generation and completed execution need separate records. So, for Daniel's setup, here's what I'd actually do. First, when a tool goes missing, look at the tool list the harness really sent, not the config file. Second, keep MCP schemas plain: objects, strings, numbers, enums, required fields. The fancier JSON Schema constructs are the ones some providers reject or quietly ignore.



Corn

Third, write skills in verbs, not tool names. “Run the test suite,” “open the file,” “search the repo.” If a host really needs “use Bash,” put that in a thin host-specific entrypoint over the shared procedure.



Herman

Fourth, don't trust permissions you copied across. Check what the destination actually enforces. And fifth, before trusting an imported workflow, do one full round trip: the model asks for a tool, the call runs, the result goes back attached to the right call, and the model uses it. If raw markers turn up in plain text instead, that's the parser, and no rewrite of the skill will fix it.



Corn

Whether those host-specific entrypoints eventually go away is still open. The layer closest to each model's training is stubbornly particular.



Corn

Thanks to Hilbert Flumingtop for producing the show.



Herman

If you enjoyed this one, try episode seven, Building Custom ASR Tools; episode twenty-five fifty-one, How Progressive Disclosure Saves MCP from Token Bloat; and episode twenty-four hundred, Claude Code's Hidden Context Tax. This has been My Weird Prompts, the human-AI collaboration podcast. If you listen regularly, leave us a review. It helps other curious people find us.



Corn

We'll be back soon.