Episode Book

EPISODE #6267

Why Text-to-Speech Fails at Explaining Code

PUBLISHED	25 October 2026 at 12:28 UTC
DURATION	22:05
LISTEN	myweirdprompts.com/episode/code-to-speech-narration-syntax-69128/
HOSTS	Daniel, Corn, Herman, Larry, Hilbert
SCRIPT MODEL	openai/gpt-6.1-sol
VOICES	chatterbox-regular+meorav (chatterbox-local)
PIPELINE	V5.22
SOURCES	9 cited by the research (listed at the end)
BOOK GENERATED	26 October 2026 at 06:38 UTC

SYNOPSIS

Translating source code into speech creates a difficult dilemma between tedious character-by-character dictation and overly smooth summaries that omit crucial logic. This discussion explores the mechanics of code-to-speech translation, reviewing recent research on how major voice models misrepresent structural cues like indentation, pauses, and scope boundaries. It examines tools like speakable-text that use language-specific parsers to translate syntax into semantic narration, preserving operator precision, identifier conventions, and execution boundaries without reciting raw whitespace.

TRANSCRIPT



Daniel

We just walked through the sampling code in my ASR training app, including the dataset weights and how priority words decay. That felt like a useful format for developers. I want to keep the real code underneath it, but how do we quote it without the voice turning into a punctuation recital?



Corn

The useful moment was hearing how the weighting decision actually happened, rather than getting another pleasant paragraph about balanced datasets. Daniel wants to preserve that specificity when the material goes through TTS.



Herman

The little academic field calls this Code-to-Speech. There are also open source normalisers doing remarkably close to what you want, although their intended users are mostly programmers using screen readers, coding agents, or people listening to notes.



Corn

A small field with unusually loud brackets.



Herman

The important distinction is between an episode about programming and an episode built around an implementation. You can discuss sampling strategies without opening a codebase. Once you anchor the discussion to the actual weighting expression, though, somebody can notice that your explanation and the implementation disagree.



Daniel

That's the attraction. Could this become continuing education that stays attached to what the software really does?



Corn

It could, provided the attachment survives the edit. A host can make almost any function sound reasonable by describing its presumed intention. The useful version makes a claim you could check against the snippet.



Herman

Think about that square root in the dataset sampling. Saying the app favours some datasets tells you almost nothing. Saying the draw uses the square root of dataset size, multiplied by task weight, tells you what grows more slowly and what remains a direct multiplier. You can reason about the design from that sentence.



Corn

You can also challenge it. Why should task weight have that much influence? What happens when a dataset doubles? Those are implementation questions, and the expression gives you something solid to argue with.



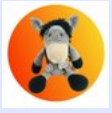
Herman

Corn, would you want the actual symbols spoken there, or would that sentence satisfy you?



Corn

The sentence satisfies me if you preserve the order of operations. If you say the square root applies to both the size and the weight, you've changed the system while improving the prose.



Herman

That is the editorial difficulty. Developers already know enough syntax to reconstruct a lot from a compact explanation. They need the significant choices, not a dictation exercise.



Daniel

I don't want it to drift into a generic explanation either. If we're discussing task weight, I want someone to recognise that name when they open the code.



Corn

Keep task weight. Lose the punctuation surrounding task weight. We don't need to make the listener reconstruct the keyboard.



Herman

Worked examples give you constraints that abstract advice avoids. A priority queue with a twenty-take target has a stopping condition. A decay rule has a direction and a rate. The host has to explain those decisions, including where the implementation surprises them.



Corn

Your enthusiasm is already preparing a complete queue implementation.



Herman

Only the interesting branches.



Corn

Every branch is interesting to you.



Herman

I will concede that my pruning needs work. Still, the format earns its time when a listener can predict the next result before we explain it. That's understanding, rather than recognition of familiar terminology.



Daniel

Why can't a good voice model just read that code sensibly? It can handle quite complicated prose.



Herman

The ICMI paper published this month, Code-to-Speech, describes it as a cross-modal translation task. Spatial and symbolic information has to become a linear auditory signal. On screen you can see a block, look back at its condition, and notice that a comment sits outside it. Audio gives you one thing after another.



Corn

Indentation has declined the invitation to speak.



Herman

You can pronounce every identifier perfectly and still lose the relationship between statements. Suppose a return happens inside a loop. If the narration makes it sound like the return happens after the loop, you've preserved the words and altered the behaviour.



Daniel

Pausing more wouldn't necessarily tell me which block we're leaving.



Corn

Nor would reading four spaces. That's an inventory of whitespace, not an explanation of scope. You need something like, inside that loop, or after all the items have been processed.



Herman

Those phrases carry structure. The paper tested Google TTS, ChatGPT, Gemini, ElevenLabs, and Copilot. ChatGPT and Gemini had the lowest error rates in their comparison. Copilot omitted critical tokens. Every system struggled with structural cues including indentation, pauses, and comments.



Corn

A missing token can make a beautifully delivered lie.



Herman

Especially a negation or comparison operator. A voice can sound confident and clear while leaving out the part that changes which branch runs. Lower error rates help, but they don't establish that a listener can follow a whole implementation.



Daniel

Comments seem easier. They're already prose. Why do those cause trouble?



Herman

Their role needs identifying. A comment might describe the intention, warn about a limitation, or be stale. If the voice glides from executable statements into a comment without marking the change, the listener may hear both as equally authoritative descriptions of what executes.



Corn

The comment says retry. The function gives up. A smooth narrator can conceal a workplace dispute.



Herman

I'd want to say, the comment calls this a retry, but the implementation returns immediately. That discrepancy is often exactly why a code walkthrough is worth listening to.



Corn

There is another problem with pauses, Herman. You can hear a pause, but you can't reliably hear its cause. Is that a block boundary, a new paragraph, or you deciding whether we need the history of the parser?



Herman

Usually the parser.



Corn

We know.



Herman

No, your objection matters. Pauses support explicit structure. They can't carry all of it alone. The authors put it bluntly that code is often treated as ordinary text, or even narrative text or poetry. Those habits encourage fluid delivery where boundaries are doing real work.



Daniel

Is skipping the block what most products settle for?



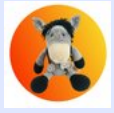
Herman

It's a common answer. An Open WebUI feature request specifically asked TTS to skip code sections because it kept reading Python and SQL snippets aloud. That solves an immediate irritation, but a listener can then hear an explanation that depends on something silently removed.



Corn

The narrator says, as you can see above. I cannot.



Herman

Another answer is character-by-character speech. The md-tts project describes the experience as open paren, self, comma, x. Useful when you need exact punctuation, exhausting when you need the behaviour of a function.



Daniel

I suppose a developer actively inspecting a line might want that precision. It doesn't follow that somebody listening to an episode does.



Corn

Different jobs. If I'm checking whether an argument is missing, I need the arguments. If I'm learning why the function exists, spelling every separator delays the useful part.



Herman

Would you accept omitting the block altogether if the explanation were complete, Corn?



Corn

I'd accept omitting its raw reading. I'd still want an audible indication that we've reached the example. Otherwise I can't distinguish your general advice from your description of this implementation. That distinction matters if I'm going to find it later.



Herman

The collision is real, then. Fidelity to the snippet is why we're doing this, and simply handing the snippet to TTS can destroy that fidelity.



Corn

We appear to have reached the pillow advertisement.

L

Larry

Are you tired of lying awake at night, wondering whether your source code is being read aloud correctly to absolutely nobody? Of course you are. That is why we built the Snorewell Semantic Slumber Narrator, the world's first pillow that normalises your entire codebase into a gentle, continuous bedtime murmur, because the only thing your indentation, your pauses and your code comments were ever missing was a listener who is fast asleep. It maps every bracket, every comma and every character-by-character stumble into a warm, linear auditory signal that whispers your variable names like a lullaby.

L

Larry

The Snorewell is not available in any store, it is not available anywhere, and it does not exist, so act now. Offer void where prohibited, prohibited where offered, results not typical, results not possible, individual murmurs may vary, side effects include unexplained wakefulness, sudden fidelity, and the belief that a sloth is reading your repository aloud to you at four in the morning. Snorewell, sleep through your own code, tonight.



Herman

Those occur now. The distinction Corn made also explains why accessibility work matters here. Blind and low-vision programmers have needed reliable ways to navigate code for years. Our podcast problem is downstream of that work, with different expectations about interaction.



Daniel

Does the transcription side have the same difficulty with identifiers, just in reverse?



Herman

The paper Lost in Transcription describes spoken code queries as code-mixed, syntactically irregular, and full of unfamiliar identifiers, symbols, and API names. Transcription errors then propagate into the system trying to answer the coding question. A plausible ordinary word can replace the technical term that made the question meaningful.



Corn

Familiarity becomes an active disadvantage. The model tidies away the word you needed.



Herman

Both directions benefit from context. The Code-to-Speech paper found designed prompts improved intelligibility, particularly for novices. On the recognition side, refinement guided by the coding context can repair errors. Tell the system what kind of language it's handling.



Daniel

That gives us a diagnosis. What can I actually put between the code and the voice?



Herman

Speakable-text is the closest fit. It's a small TypeScript project under the MIT licence, currently at version zero point three point zero. Three users, so don't mistake it for a mature industry standard. Its Code Narration Engine parses supported code and produces semantic narration.



Corn

You were going to install it before finishing that sentence.



Herman

I was going to inspect its output first.



Corn

A welcome procedural innovation.



Herman

Its Python example introduces a function called get user, then says it takes an identifier. An increment becomes, increase count by one. You retain the name and operation without narrating each character used to express them.



Daniel

Does it understand the purpose, or is it translating syntax into sentences?



Herman

The latter, more narrowly. I said semantic narration, but don't hear that as full program comprehension. It uses language-specific parsers to recognise structures and render them in English. It can describe an increment. That doesn't establish why the developer wanted the increment.



Corn

Good. I don't want a normaliser deciding that a suspicious function is probably trying its best.



Herman

An operator phrasebook preserves distinctions such as strict equality. One example says, run status is strictly equal to string ready. A little stiff, but it retains the identifier, comparison type, and literal value.



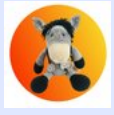
Daniel

Would that stiffness be preferable to making it sound completely conversational?



Corn

On the important line, sometimes. If the type distinction changes the result, keep it. If the host spends a minute making every assignment sound elegant, we'll stop hearing which operation matters.



Herman

It also handles identifier conventions, splitting camel case and snake case into spoken words. If parsing fails, it falls back to lexical narration. That fallback is worth listening for, because the output may suddenly get much more detailed without becoming more useful.



Daniel

Could we establish one pronunciation for a name, then keep using it? Task weight rather than switching between weighting constant and importance multiplier?



Corn

Please do. Those alternatives may sound stylish, but they imply three different things. Introduce task weight as the name in the implementation, explain its role, and leave the name alone.



Herman

Corn, what would you do with two identifiers that collapse to the same spoken words?



Corn

Add a stable qualifier. The local user identifier, or the user identifier argument. If spelling is the actual subject, give that detail once. Don't pretend an audio rendering can preserve every distinction without help.



Herman

TextForSpeech gives us a useful ladder for that decision. It's a Swift normaliser, now frozen at version zero point twenty-three point zero. Its functionality moved into SpeakSwiftly, version twelve point zero point zero. The three styles are compact, balanced, and explicit.



Corn

Even the normaliser requires a migration announcement.



Herman

Compact might simply say foo. Balanced says foo function. Explicit says foo function call. Same underlying name, different amounts of information about its role.



Daniel

That sounds like an editorial policy rather than a setting we'd use for the whole episode.



Corn

Passing mention, first introduction, close inspection. Use less detail when the listener already knows what we're discussing. Spend the extra words where they prevent a wrong inference.



Herman

A location can become Worker Runtime dot swift at line forty-two. A help option can become long flag help. Technical shorthand can become float thirty-two, or cosine float thirty-two. These are recognisable developer terms, not a parade of punctuation.



Daniel

Where do the block announcements fit? I don't want every example to sound like a telephone menu.



Herman

Speakable-text exposes start and end announcements, plus an optional language announcement. You can say snippet, identify Python if that matters, describe the operation, then say end snippet. It also allows control over pauses and comment handling.



Corn

We can make those boundaries conversational. In the sampling expression is a boundary. Back outside the loop is another. The listener needs the transition, not a ceremonial opening of the code.



Herman

The md-tts approach is complementary. It's an MIT-licensed Python tool of roughly seven hundred lines. Its interactive workflow pauses at code so the listener can inspect it. Its podcast mode instead announces skipped blocks rather than waiting for interaction.



Daniel

Announcing a skipped block still doesn't teach me what happened inside it.



Corn

The announcement is only the boundary. The host still owes you the explanation. A normaliser can avoid leaving a silent hole, but it cannot decide which design choice makes this episode worth hearing.



Herman

That is why I wouldn't call any of these a complete podcast solution. Their design choices converge on separating structure from raw reading, but they serve different listening situations. md-tts even switches voices for inline English technical terms in Spanish prose, preserving words such as framework and pipeline.



Corn

Pronunciation gets to matter again, once the structure survives.



Daniel

Can we make the policy concrete with the sampling expression, without redoing the whole explanation?



Corn

We quote the identifier, describe the operation, and mark where the example begins and ends. For the line worth remembering, we say, sampling weight is the square root of dataset size multiplied by task weight. That's the spoken rendering. We don't dictate its source characters.



Herman

Then we test the explanation with a consequence. Holding task weight constant, a larger dataset gains sampling weight less quickly than it gains examples. If our narration can't support that inference, we haven't explained the expression well enough.



Daniel

What about the priority words? There's a risk of saying decay when somebody imagines the wrong thing changing.



Herman

Name the changing quantity. If the implementation reduces selection priority as takes accumulate, say that, rather than implying the word itself disappears or its recordings lose value. Preserve the twenty-take target where it explains the behaviour. Don't substitute a vaguely familiar story about decay.



Corn

One line gets close inspection. The surrounding block gets purposeful paraphrase. Raw source belongs alongside the audio for anyone checking syntax.



Daniel

Is there evidence that this works as audio education, rather than just sounding sensible to us?



Herman

I don't know of a study measuring comprehension for this particular podcast format. The accessibility results don't automatically answer that. My expectation is that predicting an output, then hearing the explanation, would be more revealing than asking whether the voice sounded pleasant.



Corn

A listener can enjoy a voice and misunderstand the condition. I want to know whether they think the return happens inside the loop.



Herman

We could also preserve a distinction between generated narration and host interpretation. Increase count by one is a description of an operation. This prevents duplicate submissions is an explanation that needs evidence from the surrounding code.



Corn

Especially if the duplicate check happens somewhere else. Fluency makes those unsupported explanations remarkably easy to miss.



Hilbert

Snorewell Semantic Slumber Narrator, zero shekels. Not an object I can order. My supplier, Moshe at the radio counter, has written unavailable across the request. He has supplied my code-reading arrangements for long enough to take these things literally. Also, Herman, three stars. You said three users. Stars don't count users.



Herman

Three stars. You're right. I shouldn't have converted one into the other.



Hilbert

I have a Motorola two-way radio on the kitchen shelf. It's plugged into its charger. Every function I listen to goes through that radio. I require the reader to wear a motorcycle helmet with the printout taped inside the visor. Moshe says a desk would be simpler. A desk allows looking ahead. I pay for the function in its proper order.



Corn

The helmet has become a very expensive scroll lock.



Hilbert

The reader says what the next statement does. Then the symbols, if required. Semicolon is the stopper. Curly brace is the hug. There's a laminated card beside the charger. Four columns. The fourth is for symbols that never come up. Moshe charges for laminating the empty column too. I consider that reasonable. It remains available.



Herman

The shorthand works because the reader and listener already share the vocabulary. Nobody has to infer what hug means from the voice.



Hilbert

Your inside-the-loop example has changed my instructions, Corn. The reader now has to say when we're leaving one. Your explanation got there rather faster than Herman's parser. I still require the helmet. My uncle used the same shorthand with a colleague piloting a small aircraft, but I don't permit airborne deliveries. Moshe has put the revised instruction on the card. Six words. Eighteen shekels.



Corn

Eighteen shekels for a boundary announcement is probably not the pricing model this field needs. The open question is whether a podcast normaliser can make those boundaries reliable while leaving the explanatory choices to the host. The podcast audience may be small, but the accessibility need certainly isn't.



Herman

For this format, I'd use the tools to prepare consistent names and structural cues, then have the hosts check the spoken explanation against the source. Right now, that discipline supplies the fidelity. Thanks to Hilbert Flumingtop for producing the show.



Corn

For more along these lines, there's episode seven, Building Custom ASR Tools; episode eight, Building Your Own Whisper; and episode nine, Benchmarking Custom ASR Tools - Beyond The WER. This has been My Weird Prompts, the human-AI collaboration podcast. If these conversations are useful to you, a short review helps other listeners find us.



Daniel

I'll see you tomorrow.



Herman

Until then.

SOURCES

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

1. **Cardia, Angileri, Buzzi, Leporini, Code-to-Speech: An Exploratory Study of AI Text-to-Speech for Source Code Readability, ICMI '26, published 2026-10-04. [primary]**
   https://dl.acm.org/doi/full/10.1145/3776574.3831223

2. **`speakable-text`, MIT, v0.3.0 (2026-09-30), last push 2026-10-12. [primary]**
   https://github.com/atacan/speakable-text

3. **`TextForSpeech`, frozen at v0.23.0; successor `SpeakSwiftly` v12.0.0. [primary]**
   https://github.com/gaelic-ghost/TextForSpeech

4. **`md-tts`, MIT, podcast mode and interactive pauses. [primary]**
   https://github.com/jmponcebe/md-tts

5. **Havare, Mittal, Tamilselvam et al., Lost in Transcription: How Speech-to-Text Errors Derail Code Understanding, arXiv:2601.15339v1, submitted 2026-01-20. [primary]**
   https://arxiv.org/abs/2601.15339

6. **Code Narration Engine documentation (semantic vs lexical narration, `normalizeIdentifier`, `OperatorPhrasebook`).**
   https://deepwiki.com/atacan/speakable-text/5-code-narration-engine

7. **Free TTS for Code VS Code extension (Piper, offline).**
   https://marketplace.visualstudio.com/items?itemName=3dalgolab-8201.free-tts-for-code

8. **open-webui issue #2152, enhancement: TTS skip code sections, opened 2024-05-09, closed 2024-10-07.**
   https://github.com/open-webui/open-webui/issues/2152

9. **PR #459, fix(client): normalize Markdown + acronyms for read-aloud, documenting raw Markdown being voiced as asterisk asterisk and backtick.**
   https://forge.wynning.tech/james/carol/pulls/459