

Code-to-Speech

An exploratory study of AI text-to-speech for source code readability

Marco Cardia¹ · Letizia Angileri¹ · Marina Buzzi² · Barbara Leporini^{1,3}

¹ Department of Computer Science, University of Pisa · ² IIT-CNR, Pisa · ³ ISTI-CNR, Pisa

Napoli, Italy
5–9 Oct 2026

ACM ICMI '26
doi:10.1145/3776574.3831223
github.com/cardiamc/Code-To-Speech_Evaluation



// The problem

TTS engines are tuned for prose. Source code is not prose.

Code is dense with symbols, governed by strict structure, and carries meaning in its layout. Read aloud as flat text, a code block loses exactly the signal that make it comprehensible: indentation, block boundaries, the identity of an operator. For blind and low-vision programmers, who reach code only through the auditory channel, that loss is a barrier, and an unavoidable one, now that the same engines drive eBook read-aloud and the spoken replies of conversational agents.

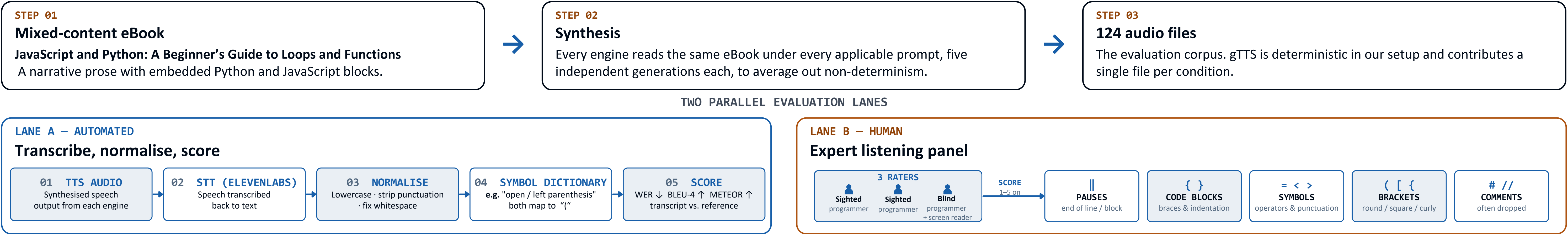
CTS as cross-modal translation

CODE — SPATIAL

SPEECH — LINEAR

Lost in the crossing: indentation, block boundaries and pauses have no slot in a linear audio signal. We call this task **Code-to-Speech (CTS)**.

// Method



// What counts as correct?

Accuracy is relative to a listener. We wrote two ground-truth references for the same line, and scored every engine against both.

for (let i = 0; i < 5; i++) {

NOVICE — VERBALISE EVERYTHING

"for open parenthesis let i equals zero semicolon i less than five semicolon i plus plus close parenthesis open curly bracket"

EXPERT — FLUENT CODE-READING STYLE

"for let i equals zero, i less than five, i plus plus, open block"

// The measurement gap

No metric was designed for spoken code. WER, BLEU and METEOR come from speech recognition and machine translation: they compare token strings, and they weight every token alike. Nothing in them gives source code more weight than the prose around it.

Every token weighs the same

A missed bracket costs exactly what a missed article costs. Resting on n-gram overlap, a score stays high when most words are right and one logically critical symbol is wrong: reading += as + barely moves BLEU, but it changes what the program does.

Structure never reaches the score

All three metrics are computed on a transcript. Pauses, prosody and indentation are gone before scoring begins, so no automatic measure can see them at all.

1.0 / 5

our raters' score for use of pauses, in all 28 configurations

There is no agreed target

"Correct" depends on who is listening: a novice needs every symbol spoken, an expert needs a fluent reading. With no reference standard in the literature, we wrote both and every engine scores differently against each. A metric that admits only one correct answer cannot describe this task.

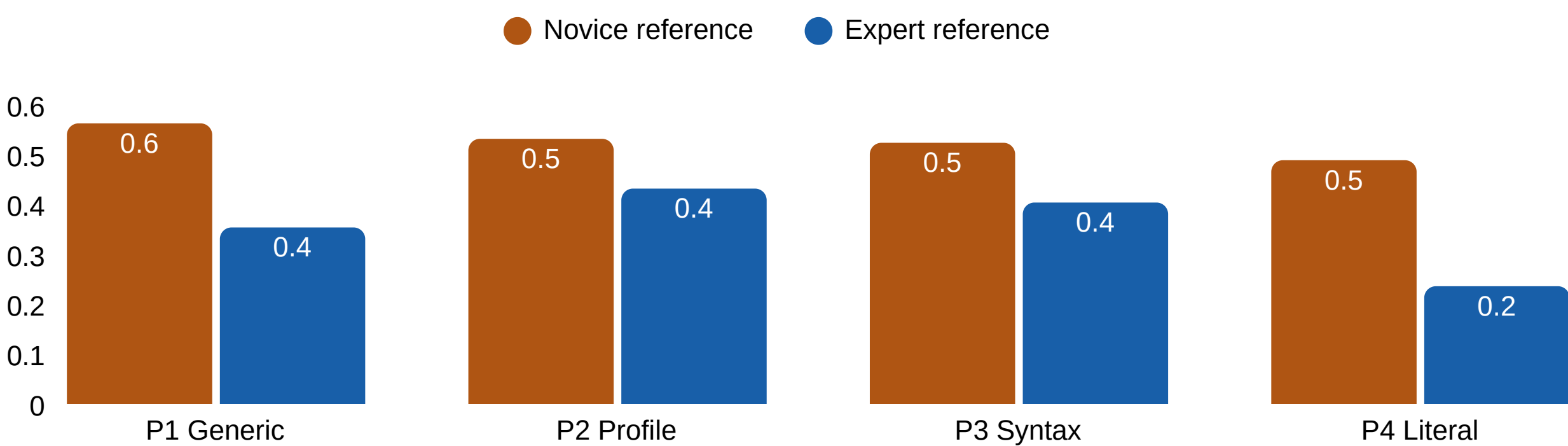
What a CTS metric would need: code-aware weighting that separates syntax from narrative, credit for structural cues that never appear in a transcript, and a reference model allowing more than one valid reading. Defining it is, we argue, a precondition for progress. In this study we provide a first benchmark.

// How the engines actually read code

TOKEN	EXPECTED	GEMINI	CHATGPT	ELEVENLABS	GTTS	COPILOT
f"{i} is "	"f string, brace i brace, is"	"funt i is"	"f is equal to"	"Pffiiis"	"f (pause) i (pause) is"	omitted
<	"less than"	"less than"	"greater than"	"plus"	"less than"	omitted
#	"comment"	silence	"hashtag"	"hash"	"hash code"	omitted
{ }	"curly braces"	silence	silence	silence	"opening bracket"	omitted
__init__	"double underscore init"	"init"	"init"	"init"	"underscore init"	omitted

// Does prompting help?

Mean WER across the three engines that accept prompts (ChatGPT, Gemini, Copilot), for each listener profile. Lower is better.



Only the literal instruction helps both listeners. "Treat it as text" (P4) lowers error for the novice and the expert reference alike. Asking explicitly to preserve syntax (P3) does not: against the expert reference it is worse than saying nothing at all. The models' attempts to interpret code introduce errors of their own.

// Conclusions

Across 124 audio files, Gemini and ChatGPT were the most accurate, while Copilot was the least omitting code blocks outright; gTTS and ElevenLabs sat in between. Yet accuracy and accessibility came apart: under every prompt, every engine scored 1 out of 5 for use of pauses, none announced Python indentation, and comment markers were routinely dropped. **No current TTS system renders source code in a structurally meaningful way.**

- 1 No metric fits this task. WER, BLEU and METEOR weight a bracket like an article, and cannot see a pause at all.
- 2 Accuracy is not accessibility. The most accurate engines still score 1/5 on pauses and never signal indentation.
- 3 The interface can fail harder than the model. Copilot's worst results came from its web wrapper, not from gpt-4o.
- 4 Only literal instructions helped. "Treat it as text" was the one prompt that lowered error for both listener profiles.
- 5 CTS is a structure problem, not a voice problem. It needs syntax-aware pipelines, auditory cues, and adjustable verbosity.

Limitations & future work. Exploratory by design: one eBook, two languages, three raters. Next: a larger, more diverse panel of blind and low-vision programmers, a corpus beyond Python and JavaScript, syntax- and context-aware TTS architectures — and, above all, a metric that measures what listening to code actually requires.