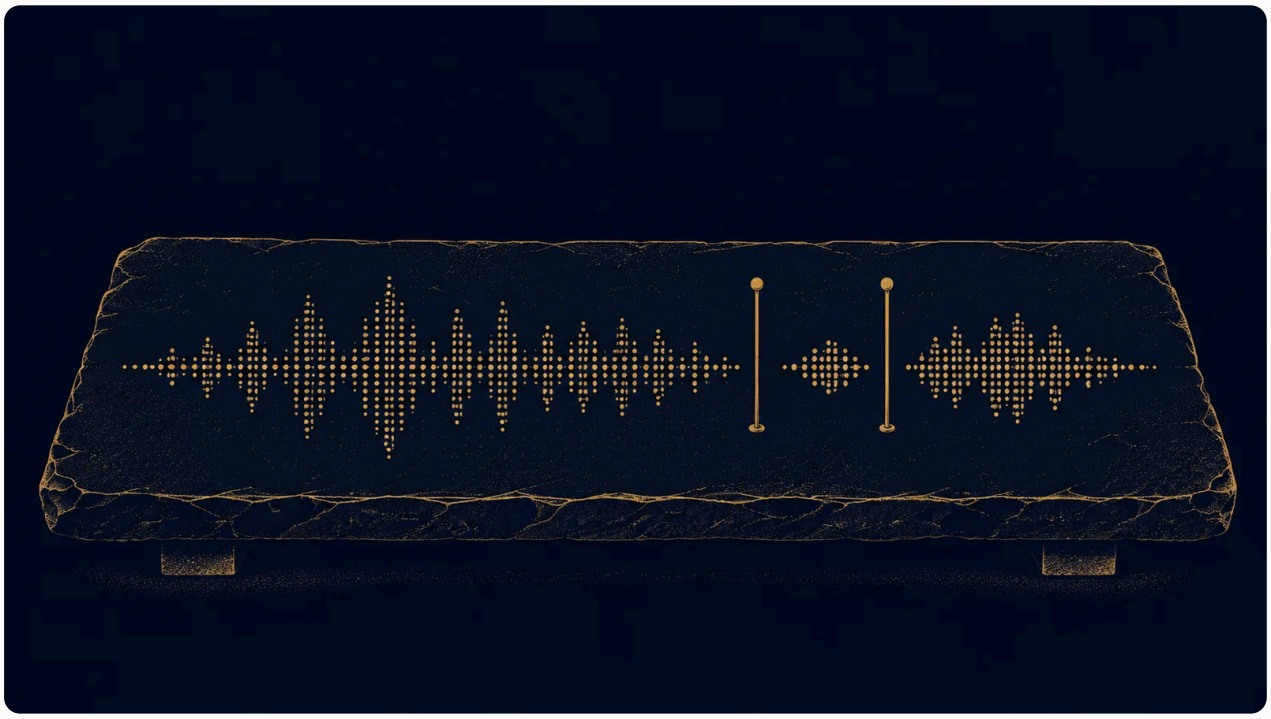
AMEN AMOUZOU · AUGUST 7, 2026

The anatomy of time-to-first-audio in a voice agent

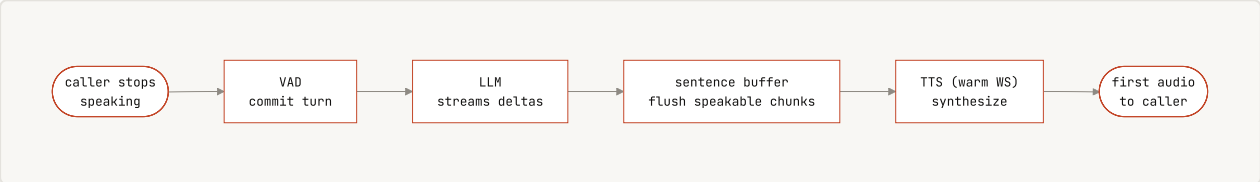
I measured my voice pipeline's wait — sentence buffer, VAD endpointing, and a live LLM→TTS chain. Post-commit, my code is ~14% of it; adding endpointing roughly doubles the share I control.



When a caller finishes a sentence and your AI agent starts to answer, every millisecond of silence belongs to someone: the voice-activity detector that must decide the caller is done, the LLM producing the first words, the buffer deciding when those words are speakable, and the TTS engine turning them into sound. Ask a voice-AI engineer which part they've tuned, and the answer is usually their own orchestration code — because that's the part they can edit.

I run cadence — the provider-agnostic STT/TTS/LLM engine layer under a production telephony platform — and its orchestration constants (a 300 ms VAD silence threshold, a 15-word clause rule, a 300 ms flush timeout) were inherited from a prototype and never characterized. This article is the characterization: three experiments, from fully-deterministic local sweeps to a live LLM→TTS chain, answering one question — **where does time-to-first-audio actually go, and which knobs actually move it?**

The pipeline under test



Two clocks matter, and this article keeps them separate. The **full-chain clock** starts when the caller stops speaking and includes the VAD's silence-commit wait. The **post-commit clock** starts when the turn is committed and the LLM request is dispatched — that's what Experiment 3 instruments directly. Experiment 2 measures the VAD stage on its own; at the end I compose the two into a caller-side estimate. A scope note on STT: in cadence's streaming path, transcription overlaps the caller's speech and endpointing is server-driven; in its batch path, the buffered utterance is transcribed *after* the VAD commits — adding STT finalization time between commit and LLM dispatch that nothing here measures. The composed number below therefore excludes STT finalization; treat it as a lower bound on the caller-experienced wait.

Three experiments, one per stage cadence controls or touches:

1. **Sentence buffer (in vitro):** simulated LLM word streams at 20/40/80 words/s into `RunSentenceBuffer`, sweeping its config; 360 runs. Deterministic and reproducible on any laptop.
2. **VAD endpointing (in vitro):** synthetic mu-law speech — syllabic energy bursts with articulation micro-gaps, matching the RMS bands the detector documents — with

controlled hesitation pauses; 1,800 trials sweeping the silence-commit offset.

3. **Live chain (in vivo)**: the real thing — `gpt-4o-mini` streaming through the sentence buffer into a warm ElevenLabs Flash v2.5 WebSocket, timestamping every stage boundary; 15 runs across three response styles.

Versions: cadence @ `main` (Aug 2026), Go 1.26.5, Apple Silicon macOS; live runs from a residential connection. Limitations up front: the live sample is small (15 runs, one afternoon, one region), the VAD signal is synthetic energy rather than recorded speech (which makes its thresholds artificially sharp), and in-vitro senders share a scheduler with the consumer. Numbers below are medians with min–max ranges, not gospel.

Experiment 1: the sentence buffer barely matters — except when it's everything

`RunSentenceBuffer` sits between the LLM and TTS, deciding when accumulated text is worth speaking. Its config trades first-flush latency against chunk size (prosody): flush on sentence terminators, on clause punctuation after `ClauseMinWords`, on a word cap, or on idle timeout.

Median time-to-first-chunk, by response style and config, at 40 words/s (the harness emits one whitespace-delimited word per interval — rates here are words/s, not model tokens/s):

STYLE	EAGER (5W/100MS)	DEFAULT (15W/300MS)	PATIENT (25W/500MS)
Short answer	162 ms	158 ms	158 ms
Paragraph	356 ms	368 ms	373 ms
Clause-heavy list	**156 ms**	**380 ms**	**677 ms**

Two regimes hide in that table:

- **Prose doesn't care.** For short answers and paragraphs, all configs are within noise of each other — the first flush fires at the LLM's first sentence terminator, which arrives before any clause rule engages. Your floor is `words in the LLM's first sentence ÷ word rate`, and no buffer setting beats the punctuation.
- **Lists care enormously.** When the first period is 25 words away but commas come early, `ClauseMinWords` alone spans **156 → 677 ms** at the same word rate — a 4.3× spread. The cost of eagerness is prosody: eager cuts the same text into 6 chunks averaging 6 words; patient produces 2 chunks of 19.

And across every style and config, halving the word rate doubled the latency, almost exactly. The result points at a cheap optimization that isn't a buffer constant: since first-sentence length dominates buffer latency for prose, **prompting the model to open with a short sentence** should beat any clause threshold — an intervention this study motivates but didn't A/B.

Experiment 2: the VAD knob is a straight line, not a dial

cadence ships an energy VAD whose comment admits its 300 ms silence-commit "trades a small risk of mid-sentence cuts for noticeably snappier response." Quantified over 1,800 controlled trials (hesitation pauses of 200–800 ms embedded mid-utterance), that trade turns out to be a step function:

OFFSET	COMMITS TURN AFTER	CUTS A HESITATION OF
200 ms	~140 ms silence	≥ 200 ms — always
300 ms *(shipped)*	~240 ms	≥ 300 ms — always
500 ms	~440 ms	≥ 500 ms — always
600 ms	~540 ms	≥ 600 ms — always

(Commits land ~60 ms before the nominal offset because each synthetic utterance ends with a trailing 20–80 ms articulation gap — silence accumulation starts inside it, so the measured commit is offset minus that trailing gap. The threshold itself behaves exactly as configured.)

Below the offset: zero cuts, every time. At or above it: cut, every time. (The shipped `NewEnergyVAD` was run on identical trials as a cross-check and matched the sweep exactly.) With a pure energy detector there is no clever setting: **every millisecond of hesitation tolerance is a millisecond of response delay**, one-for-one. Real callers can easily pause longer than 300 ms mid-turn while thinking, so the shipped default *will* cut some of those pauses — a deliberate, now-quantified choice. The synthetic signal makes this boundary artificially sharp; real speech decay would blur it, but not bend the line. Escaping the line entirely requires a smarter endpointing *algorithm* — model-based VAD that hears the difference between a hesitation and a handoff — wherever it runs; a server running the same silence heuristic sits on the same line.

Experiment 3: the live chain — and where the milliseconds actually live

The live pipeline, real vendors, 15 runs, on the **post-commit clock**: t=0 is the LLM request dispatch, and the finish line is the first synthesized audio chunk.

STAGE	MEDIAN	RANGE	MEDIAN PER-RUN SHARE
LLM first token	592 ms	401 – 2,881 ms	**59 %**
Sentence buffer holding	136 ms	45 – 513 ms	14 %
TTS first audio byte	289 ms	274 – 522 ms	28 %
Post-commit first audio	**1,058 ms**	873 – 3,674 ms	

(Stage medians are computed independently per stage and do not sum to the median total — 1,017 vs 1,058 ms — because the median of sums isn't the sum of medians. The

share column therefore reports the median of *per-run* proportions, which is why it doesn't total 100 % either.)

Three observations:

- **The LLM dominates, and it's the wild one.** Median 592 ms, but a 7.2× min-to-max spread in just fifteen runs — the worst first token took 2.9 seconds, pushing that run's total to 3.67 s, about 3.5× the median. Nothing else in this sample came close to that variance (fifteen runs characterize a median far better than a tail; the outlier is an existence proof, not a rate).
- **The warm WebSocket earns its keep.** TTS first-byte was the most *stable* number in the whole study: ~289 ms median with a tight floor around 275 ms, run after run — the flat tax of the TTS request path, with connection setup already paid.
- **The in-vitro model predicted the in-vivo buffer.** The live paragraph responses streamed at a measured 64–121 words/s (emitted words over generation time). In that regime Experiment 1's default config predicts roughly 130–380 ms to first chunk; the live paragraph runs held text for a 174 ms median. The deterministic sweep transfers.

On the post-commit clock, the orchestration layer I can edit accounts for a median **14 %** of the wait; the rest is LLM and TTS time as observed at my client — which folds in network transport and client overhead, not purely vendor compute.

Composing the chain: put Experiment 2's shipped-VAD commit (~240 ms) in front of the post-commit median and the caller-side estimate — excluding STT finalization — becomes ~**1.3 s**, of which orchestration — VAD wait plus buffer holding — owns a median ~**29 %** (range 16–41 % across runs). Still the minority, but twice what the post-commit view suggests: the biggest knob I control isn't in the text pipeline at all; it's how long the VAD waits to decide the caller is done.

What surprised me

The 2.9-second outlier. I expected the LLM to dominate the median — that's the going wisdom — but I wasn't prepared for one first token out of fifteen to take 2,881 ms while the rest of my pipeline sat there, warmed up and blameless, waiting to do its 400 ms of work. In production that run is a caller hearing nearly four seconds of dead air and wondering if the line dropped. Every constant I've ever tuned in this stack combined couldn't buy back what that one LLM-path stall spent. It reframes where the engineering effort should go: not shaving my slice of the median, but detecting and masking exactly the kind of LLM-path outlier that showed up in this sample — filler audio, a faster fallback model, anything that covers a three-second hole I can't prevent.

Perspective

Observed, for this pipeline on this afternoon: LLM first-token latency was more than half the post-commit wait and produced the only large outlier in the sample; TTS was a stable ~290 ms tax; the buffer was ~14 % and already near its floor for prose; and the VAD contributes a further ~240 ms that the post-commit clock hides.

Inferred from that: model and provider selection is likely the highest-leverage latency decision — this study didn't benchmark multiple LLMs, so that's engineering inference, not a finding. What *is* a finding is that first-sentence length dominates buffer latency for prose — which makes prompting the model to open with a short sentence a promising lever, one this study motivates but doesn't directly A/B. And the VAD line being what it is, real endpointing gains come from a detector that can tell a hesitation from a handoff, not from retuning a threshold.

What this study doesn't cover is its own next chapter: STT — the stage between the caller's voice and the LLM prompt — plus multi-vendor columns for both LLM and TTS, longer runs across hours and regions, and per-run distributions rather than medians. The harnesses are one file each; the next afternoon of measurement is cheap.

The three harnesses live in the cadence repository's `bench/` directory — the library's first benchmarks. Run them, and if your pipeline's anatomy looks different from mine, I

want to see the X-ray.