MODULATE

Beat the Black Box: Why Cascade Beats Speech-to-Speech for Enterprise Voice Agents

By Mike Pappas, CEO, Modulate

When a voice agent transcribes a caller's audio and passes text to an LLM, it throws away the most valuable part of the conversation. The frustration in a customer's voice before they even say the words. The hesitation that tells you they didn't actually understand what the agent said. The clipped, controlled tone that signals a complaint is about to become an escalation.

Gone. Discarded. Converted to words and stripped of everything else.

This gap in understanding and the resulting naturalness has led the voice AI industry to begin shifting from **cascade** pipelines, which chain multiple models together in sequence, to all-in-one **speech-to-speech** black box models. The belief is that, despite the downsides of a giant black-box model, it's simply the only way to give the models enough understanding of the real voice nuance to actually respond quickly and naturally.

That belief is wrong.

Cascade pipelines can actually do everything speech-to-speech can do - and with more transparency, adaptability, and cost-effectiveness to meet the real needs of enterprises. The concerns about latency and naturalness aren't real limits of cascade designs - *they are simply what happens when cascades are designed poorly*.

This primer will walk through what cascade and speech-to-speech architectures are, they commonly agreed-upon pros and cons, and then explore what it looks like to build an optimized cascade pipeline that truly suits enterprise needs.

Contents

CHAPTER ONE

What Is a Voice Agent, and Why Does the Architecture Actually Matter?

The basic idea

A voice agent is an AI system that holds a real-time spoken conversation with a human and takes action based on what it hears. It listens, understands, responds, and - critically - does things: looks up account information, books appointments, escalates tickets, routes calls, logs interactions, triggers workflows.

This is not a voice interface that records audio for a human to review later. A voice agent operates autonomously in a live conversation, in real time. It has to understand what was said, decide what to do, formulate a response, and deliver it, all within the window a human naturally expects before they wonder if the line dropped.

That window is short and unforgiving. Research consistently puts natural conversation response latency around 500-800 milliseconds. Below 300ms, a pause is imperceptible. Above 1.5

seconds, users start checking out. Your architecture choice directly determines whether your agent feels like a conversation or a transaction.

Why the business world is suddenly paying attention

Voice is the dominant format for an enormous category of human interactions that have resisted automation for decades: customer service calls, insurance claims intake, healthcare scheduling, IT helpdesk, sales development, financial services support. High-cost, high-volume, deeply repetitive workflows - all sitting there waiting for an AI that could actually hold a phone conversation and do something useful with it.

That capability now exists. The global conversational AI market was valued at $11.58 billion in 2024 and is projected to reach $41.39 billion by 2030, growing at a CAGR of 23.7% (Grand View Research, 2025). Voice is the majority of that market. What changed is not a single breakthrough - it is a convergence of faster LLMs, lower-cost streaming STT, expressive neural TTS, and better orchestration that all matured roughly simultaneously.

The result: real-time voice agents capable of handling complex, multi-turn conversations are now within reach of most engineering teams. But "within reach" means something different depending on whether you're building a demo or a production system at scale.

The requirements that separate demos from production

Before comparing architectures, let’s be explicit about what a production voice agent actually requires. These are not optional:

- Low latency: the response absolutely must begin within 1-1.5 seconds of the user finishing speaking, ideally under 800ms
- High accuracy: correct understanding across domain vocabulary, accents, background noise, and telephony audio quality
- Reasoning and tool use: the agent must actually do things - query systems, follow business logic, hand off, escalate
- Controllability: you must be able to specify its behavior precisely, constrain what it says, and update those constraints
- Observability: you must know exactly what happened on any given call - what the user said, what the agent said, why
- Reliability: graceful degradation with predictable failure modes, not unpredictable hallucination under edge cases

- Compliance: in most serious industries, hard requirements around logging, auditing, consent, data handling, and retention

This is your scorecard. Every claim about either architecture should be evaluated against it. Keep it in mind.

CHAPTER TWO

The Two Fundamental Architectures

The cascade pipeline (STT → LLM → TTS)

The cascade pipeline processes voice in three discrete, modular stages:

- Speech-to-text (STT / ASR): the user's audio is transcribed into text
- Large language model (LLM): the text is passed to a language model, which reasons, calls tools if needed, and generates a text response
- Text-to-speech (TTS): the text response is synthesized into audio and played back

Each component is independent. Each produces an artifact that passes to the next stage. The interfaces between components are explicit, inspectable, and replaceable.

> ***The text intermediate between each step is critical. It enables logging every word the user says and every word the agent responds with. It enables compliance auditing, analytics, content moderation, and debugging.***

In a naive implementation, each stage waits for the prior one to finish. STT transcribes the full utterance. LLM generates the full response. TTS synthesizes all of it. Then the user hears it.

This is where cascade's bad reputation comes from - 2 to 6 seconds of silence while the pipeline works.

In a streaming implementation - the current standard for any serious deployment - all three stages run in parallel. STT emits partial tokens while the user is still speaking. The LLM starts generating using that partial context. TTS starts speaking the first sentence before the LLM has finished writing the second. With this approach, sub-1-second time-to-first-audio is achievable in production today.

Speech-to-speech (S2S)

Speech-to-speech models collapse the pipeline into a single model. Audio in, audio out. No intermediate text representation - the model processes the input signal and produces the output signal internally, without ever emitting a transcript.

The leading proprietary examples as of 2025-2026 include OpenAI's GPT-4o Realtime API, Google's Gemini Live, and Amazon Nova Sonic. Open-source contenders include Ultravox, Kyutai's Moshi, and Step-Audio R1.1.

The core claim: the audio-to-audio path preserves information that gets destroyed when you convert speech to text. A cascade pipeline hears words. An S2S model, in theory, hears everything - rhythm, tone, hesitation, emphasis, emotional valence.

One important distinction worth drawing early:

- Truly native S2S: the model reasons entirely in audio tokens throughout, with no text intermediate at any point (e.g., Moshi)
- Half-cascade (audio LLM): the model takes audio input but internally uses text tokens for reasoning before producing audio output — the encoder is audio-native but the reasoning layer is not (e.g., Ultravox, GPT-4o Realtime in some configurations)

Many systems marketed as S2S are actually hybrids. The distinction matters because the claims made for "native" S2S don't fully apply to hybrid systems — and most of the production-ready options are hybrids.

A quick architecture comparison

Dimension	Cascade (STT → LLM → TTS)	Speech-to-Speech (S2S)

Audio input handling	Transcribed to text by a specialized STT model	Processed directly or encoded into audio tokens
Intermediate representation	Explicit text transcript at each stage boundary	None visible externally; internal to the model
Reasoning layer	Any text-mode LLM (GPT-4o, Claude, Gemini, etc.)	Fused into the model; limited provider options
Output generation	Separate TTS model; many options for voice/style	Internal to the model; voice baked in
Latency (naive)	2-6 seconds sequential	200-400 ms in demos; 600-800 ms typical
Latency (optimized)	Under 1 second with streaming	Under 1 second; advantage narrows significantly
Observability	Full transcript at every stage boundary	Audio in, audio out; internals opaque
Component flexibility	Mix and match STT, LLM, TTS independently	Locked to the provider's model
Production readiness (2026)	Dominant; used in most enterprise deployments	Maturing; mostly consumer and experimental

CHAPTER THREE

The Case for Speech-to-Speech

Let's give S2S its best hearing. The arguments are real. They are just pointed at the wrong target.

Argument 1: Lower latency

The most-cited advantage, with genuine structural basis. S2S models like Moshi have demonstrated latencies around 160ms. OpenAI's GPT-4o Realtime API targets 200-300ms end-to-end. The argument is simple: cascade has three network hops, S2S has one. Even with streaming, there is coordination overhead between cascade components that S2S simply does not have.

Measured benchmarks put the latency gap at around **85% lower for S2S versus a non-streaming cascade**. Against a well-optimized streaming cascade, that gap shrinks substantially - but it doesn't disappear (Coval AI, 2026).

Argument 2: Preserving paralinguistic information

When you convert speech to text, you lose an enormous amount of signal. Text captures words. It does not capture the hesitation before an answer, the rising tone that signals genuine confusion, the clipped pace that tells you a customer is containing frustration, or the forced cheerfulness that sits right on top of a real complaint.

A 2026 ACM study at CHI demonstrated this directly: injecting speech emotion recognition (SER) labels into LLM context **significantly improved dialogue naturalness, engagement, and rapport**, with 93.3% of participants preferring the emotion-aware agent over a text-only baseline (source: arxiv.org/abs/2603.09324).

S2S proponents argue that it's impossible to catalogue all the hidden nuances in voice - the only possible answer is to give the model the full, original audio so it can dynamically extract whatever information the black box deems relevant. Indeed, an end-to-end model trained on spoken dialogue picks up these cues implicitly, because it never discarded them in the first place. As we'll see, though, this is not the only way to achieve that outcome!

Argument 3: More natural prosody in responses

Text-to-speech has gotten remarkably good. But it is still reading out text. If the LLM generates a response without accounting for the emotional register of the conversation, the TTS will faithfully reproduce that mismatch. An agent that responds to a frustrated caller with a perky, measured voice is not just awkward - it actively erodes trust.

An S2S model, trained end-to-end on spoken dialogue, can in principle generate responses that mirror the conversational register. The output is shaped by the full audio context of the exchange, not derived from a transcript of what to say.

Argument 4: Simplicity of architecture

One model. One API call. No managing three services, three latency budgets, three failure modes, three bills. For a developer who needs something working by Friday, S2S is genuinely simpler.

What these arguments actually prove

Arguments 1 and 4 - latency and simplicity - have real relevance for consumer apps, rapid prototyping, and contexts where naturalness is the primary value delivered. A language learning app. A gaming NPC. A casual AI companion. These are legitimate use cases where the black-box nature may be acceptable.

Arguments 2 and 3 - paralinguistic fidelity and prosodic naturalness — are theoretically sound. But notice what they are arguments against: not cascade architecture, but cascade implementations that throw away audio signal the moment they have a transcript. That is a design failure, not a structural limitation. And it is fixable.

CHAPTER FOUR

The Case for Cascade

The cascade architecture has properties that become increasingly non-negotiable as you move from demos to production, from consumer to enterprise, from "we built a thing" to "we are accountable for what this thing does."

The text interface is a feature, not a limitation

The intermediate text representation at each stage boundary is what makes cascade pipelines **debuggable, auditable, and observable**. When something goes wrong, you can trace exactly what was said, what was transcribed, what the LLM produced, and what was spoken back. In many industries, that is not a nice-to-have. It is a legal requirement.

In a contact center, every conversation needs to be loggable for QA, compliance review, and dispute resolution. In healthcare, HIPAA requires auditability. In financial services, call logging is mandated. In fraud prevention, the transcript is the evidence chain. An S2S model produces audio in and audio out. There is no native text log. To get one, you run a separate transcription pass post-hoc - which means you are effectively building a cascade anyway, just an asynchronous one with additional complexity and costs. Further, when this separate transcription understands a word differently from how your S2S model did, there's no way to tell, so your logs cannot reliably reveal what your S2S model was even trying to respond to.

Component flexibility: the power of the ecosystem

Cascade lets you build with best-of-breed at every layer. Choose your STT provider for accuracy on real-world audio with complex noise, accents, and emotions; as well as of course low latency and costs. Choose your reasoning model to deliver the strongest tool-use performance - a notorious challenge for S2S models. Choose your TTS provider based on the needs of your brand for naturalness. Any of these can be upgraded or swapped without touching the others.

With S2S, you get whatever the provider baked in. When OpenAI Realtime underperforms on a specific workflow, you cannot swap in Claude for the reasoning step, or hook it easily to new agentic tools to fulfill an order. When the synthetic voice creates brand friction, you cannot change it. The stack is monolithic by design.

As of 2026, there are **tens of competitive providers for each component** of cascade builds — versus a handful of proprietary S2S options with no mixing and matching. The cascade ecosystem is simply richer by an order of magnitude (source: Coval AI, 2026).

Reasoning quality: cascade plugs into the frontier

The best reasoning models in the world — GPT-4.1, Claude 3.7 Sonnet, Gemini 2.0 Flash — are text-mode LLMs. They support complex multi-step reasoning, tool use, structured output,

function calling, and fine-grained instruction following. Every capability improvement from any major AI lab is immediately available to a cascade architecture. You update one component.

The reasoning layer inside S2S models is constrained by training. Ultravox launched on Llama 3 70B — formidable at the time, quickly overtaken. For truly native S2S models, the reasoning layer cannot be updated independently of the entire audio model. You are not just locked to a provider; you are locked to a training run.

Hamming AI's analysis of 4M+ production voice agent calls across 10,000+ agents found that **most production voice agents use cascade architecture not out of conservatism, but because text-mode LLMs simply outperform at complex reasoning and tool use** (source: hamming.ai).

Predictable cost at scale

Cascade pricing is additive and linear: pay for STT by the minute, LLM by the token, TTS by the character. A 10-minute conversation costs exactly 10x a 1-minute conversation. Cost modeling is straightforward.

S2S pricing does not work this way. OpenAI's Realtime API uses context-accumulation pricing where costs compound as conversations lengthen. And the spread across S2S providers is dramatic: **OpenAI Realtime runs roughly 10x the cost of an equivalent chained pipeline, while budget S2S options like Amazon Nova Sonic approach cascade-comparable rates** — with costs scaling unpredictably as conversations grow longer (Softcery, 2026). That is not a cost structure you can plan a unit economics model around.

Resilience: fallback where it matters

In a cascade pipeline, each component has independent health monitoring and independent fallback paths. If your STT provider spikes in latency, route to a backup. If an LLM is degraded, failover to another. Resilience is compositional.

S2S is all-or-nothing. If the model is slow, you are slow. If it fails, the call fails. There is no component-level redundancy. For a contact center handling thousands of concurrent calls, this is a material risk, not an architectural footnote.

CHAPTER FIVE

Reframing the Debate: The Real Problem Is What You Throw Away

Here is the real argument at the center of this debate, and I want to state it as clearly as possible:

> ***The strongest arguments for speech-to-speech are not arguments against cascade architecture. They are arguments against cascade pipelines that treat STT as a lossy compression step and discard the audio signal the moment they have text.***

Almost every substantive S2S advantage - emotional context, prosodic naturalness, paralinguistic fidelity - is a critique of what the standard cascade throws away, not a critique of the staged, modular design. And what gets thrown away is not structurally required to be thrown away. It is a design choice, made by default, by most teams who just want a working transcript.

What standard transcription actually discards

Consider what a standard STT model receives as input. The full acoustic signal of a human voice. Frequency content that encodes emotion. Timing patterns that encode confidence and hesitation. Energy envelope that encodes emphasis and intensity. Vocal tract characteristics that encode stress and affect. Speaker turn dynamics that encode dominance, deference, and interruption patterns.

Now consider what standard STT produces as output: "I've been trying to reach someone for three days and no one is calling me back."

Twelve words. The entire acoustic signal, stripped to its lexical content. The LLM reading that transcript has no way to know whether those words were delivered with cold, quiet fury or tired resignation. It responds to the words. It may generate exactly the wrong tone in return.

This is the problem S2S proponents are correctly identifying. But the solution is not to change the architecture. The solution is to change what the STT layer passes downstream.

STT can pass far more than a transcript

The interface between STT and LLM does not have to be a flat text string. A well-designed STT layer - or an enriched layer that augments standard transcription - can pass downstream much more. Modulate's Velma is designed for just this use case, able to surface hundreds of audio-native insights including:

- Emotion labels: "emotional state: frustrated, intensity: high, confidence: 0.87"
- Prosodic markers: speech rate, energy envelope, hesitation frequency, pause duration
- Speaker dynamics: turn-taking patterns, interruption count, dominance signals
- Behavioral event flags: raised voice, laughter, crying, distress, possible synthetic audio
- Confidence signals: uncertainty in word choice, hedging language, trailing off
- Speaker diarization: timestamped attribution of who said what
- Topic classification: billing, retention, technical support, complaint, compliment

With this richer payload, the LLM has the same contextual information that an S2S model would theoretically have encoded natively - just passed as structured metadata rather than embedded in audio tokens. The CHI 2026 study demonstrated this directly: emotion awareness injected as text labels, not as audio, produced significant user preference improvements. The signal does not need to stay in the audio format to be useful. It needs to reach the reasoning layer.

The latency gap is a streaming problem, not a structural one

Similarly, cascade's latency disadvantage largely is due to naive, sequential implementations. The "cascade is slow" argument is most valid when compared against a pipeline that waits for full transcription before starting LLM inference, and waits for full LLM output before starting TTS synthesis. That is not a modern production architecture.

In a streaming cascade - where STT emits partial tokens, the LLM reasons on partial context, and TTS begins speaking the first sentence before the LLM has finished writing the rest - the gap collapses dramatically. Self-hosted streaming cascade implementations have demonstrated **a best-case time-to-first-audio of 729ms and a P50 of 947ms with full function calling support** (source: arxiv.org/abs/2603.05413). That is well inside the conversational comfort zone.

Hamming AI's analysis of production cascade deployments puts P50 end-to-end latency at 1.5-1.7 seconds under typical telephony conditions, with optimized implementations targeting under 1 second. Is a native S2S model potentially faster? Yes, at the structural level. Is the difference noticeable to the average consumer, much less a critical decision factor for an enterprise deployment that needs observability, compliance logging, and tool-use quality? Almost never.

The comparison that actually matters

Most architecture comparisons in this space pit naive cascade against S2S. That comparison has a clear winner, and it is not cascade. It is also not the relevant comparison for anyone building a serious production system.

The relevant comparison is:

- A well-optimized streaming cascade that passes rich audio-derived context across its stage boundaries
- versus a mature S2S model with leading-edge reasoning capabilities - but still acknowledging the structural reasons this reasoning, as well as transparency and auditability, will always lag the cascade design

Under that comparison, cascade wins on almost every dimension that determines whether a voice agent succeeds in production.

CHAPTER SIX

What a Well-Designed Cascade Actually Looks Like

Claiming cascade wins is only useful if we are concrete about what a winning cascade requires. Here is what separates a production-grade implementation from the naive pipeline that gave cascade its bad reputation.

The streaming foundation

The single highest-leverage improvement from naive to production cascade is full-stack streaming. In practice, this means:

- STT streams partial transcripts token-by-token as the user speaks, not as a single complete transcript after silence
- The LLM begins processing on partial context, using intelligent end-of-turn detection rather than waiting for the full utterance
- The TTS begins synthesizing and speaking the first sentence - or even a partial sentence - before the LLM has finished generating the rest

This pipeline - audio chunks flowing through streaming STT, streaming LLM, and streaming TTS in parallel - is what makes **sub-1-second end-to-end response achievable in production**. The components overlap. They do not wait for each other.

Augmenting the STT layer

The second major design decision is what the STT layer passes downstream. Standard STT produces a transcript. An augmented STT layer such as Modulate's Velma produces a context object that might include:

- Transcript with speaker diarization — who said what, with timestamps
- Emotion labels derived from speech emotion recognition models, with confidence scores
- Prosodic metadata: speech rate, energy, hesitation frequency, pause patterns
- Behavioral event flags: elevated voice, distress signals, laughter, possible synthetic voice
- Topic classification: what is this conversation segment primarily about?
- Transcription confidence: flagging low-confidence segments before the LLM acts on them

This metadata becomes part of the LLM's context and is the mechanism by which a cascade architecture recovers the paralinguistic signal that standard transcription discards. The LLM does not need to hear the audio. It needs the relevant information from the audio, structured and delivered. A well-labeled context achieves this.

Intelligent turn detection

End-of-turn detection is the most underestimated component in voice agent design. The mechanism that decides when the user has finished speaking determines more about conversation feel than almost anything else.

Naive voice activity detection (VAD) uses silence to detect turn boundaries. It produces false positives on mid-sentence pauses and false negatives when users trail off without going quiet. Agents built on naive VAD interrupt at wrong moments and hang awkwardly when they should respond.

A production cascade uses semantic end-of-turn detection - a model that understands not just the acoustic signal but the linguistic completeness of the utterance. Has the user finished a complete thought? That is a different question from "did they stop making noise?" Getting it right is what makes the difference between a conversation and a transaction.

This kind of turn detection *must* be baked into the (augmented) STT layer of cascade.

The LLM context engineering layer

The full context object the LLM receives in a well-designed cascade should include:

- System prompt defining the agent's role, constraints, tone, and knowledge base
- Conversation history, aggressively cached to reduce token costs on long calls
- Current turn transcript
- Audio-derived metadata: emotion state, prosodic signals, behavioral events
- Tool definitions and available actions
- Real-time external data: CRM lookup, account status, prior interaction history

This is where the agent's actual intelligence lives. Getting it right — what to include, how to format it, what to cache, what to compress — is the primary engineering challenge and primary differentiator between voice agents that perform and those that disappoint.

A diagram of the optimized cascade

Figure: Optimized streaming cascade. Each stage begins processing before the prior stage completes. The augmented STT layer passes emotion, prosody, and behavioral metadata alongside the transcript, giving the LLM the full conversational context.

CHAPTER SEVEN

The Enterprise Reality: Why Black Boxes Fail in Production

A voice agent demo is a controlled environment. Production is not. In production, something goes wrong. It always does.

A contact center agent makes an incorrect statement to a customer. A healthcare scheduling bot confirms the wrong appointment time. A financial services agent gives guidance that may constitute regulated advice. A fraud prevention system fails to flag a synthetic voice on a high-value transaction. In every one of these cases, the immediate question from the operations team, the legal team, the compliance team, and the engineering team is identical: what exactly happened?

In a cascade pipeline, that question has a clean answer. You have the audio. You have the transcript at every stage. You have the LLM's input and output. You have every tool call and its result. You can reconstruct the state of the system at every decision point and pinpoint what failed.

In an S2S model, audio went in and audio came out. The intervening computation is opaque. You can observe that something went wrong. You cannot systematically understand why. You can retrain - but you cannot target the failure. The iteration loop that makes production systems improve over time barely exists.

Compliance is not optional in the industries that matter

GDPR, HIPAA, TCPA, BIPA, and a growing body of state-level AI regulation all create specific hard obligations around voice AI. Consent capture and documentation. Data retention policies with auditable enforcement. The ability to produce complete interaction records on regulatory request. In some jurisdictions, documentation of what AI system participated in a customer decision.

A 2024 FCC ruling affirmed that AI-generated voices fall under TCPA's provisions for artificial or pre-recorded voices. Financial services regulators in multiple jurisdictions mandate that automated customer interactions be logged, reviewable, and auditable. Healthcare deployments under HIPAA require demonstrable data handling practices.

None of this is impossible to satisfy with an S2S model. But satisfying it requires wrapping a transcription layer around the S2S output - which means you are running a cascade anyway, just asynchronously and with additional cost, inconsistency, and complexity. There is no version of regulated enterprise voice AI where you do not end up with a transcript - so shouldn’t the transcript be as complete and accurate to the voice agent’s actual behavior as possible?

The evaluation and improvement loop

Enterprise voice agent deployments are not shipped and forgotten. They are continuously evaluated, tested, and improved. That loop requires data - specifically, targeted data that tells you which component failed on which class of call.

The cascade evaluation ecosystem is rich and growing. Tools like Hamming AI built their entire platform on the assumption that you have transcripts, LLM inputs and outputs, and tool call traces. You can build regression tests. You can run A/B tests on individual pipeline stages. You can catch failures before they reach production.

The S2S evaluation ecosystem is, in Coval AI's words, starting from scratch. Every diagnostic the cascade world takes for granted must be reconstructed around audio comparison instead of text comparison. This is possible, but it is a substantial engineering tax on top of everything else you are already building.

Reasoning at the frontier matters for complex tasks

Simple voice bots - FAQ handling, appointment booking, basic routing - can be built adequately on almost any architecture. But the genuinely valuable enterprise use cases are not simple. Complex billing disputes. Multi-step technical troubleshooting. Insurance claims with conditional logic. Sales conversations requiring real product knowledge and judgment.

For these, the quality of the reasoning layer is the ceiling on what the agent can do. GPT-4.1, Claude 3.7 Sonnet, and Gemini 2.0 Flash are the best reasoning engines that exist. They are text-mode LLMs. Cascade gives you direct, immediate access to each new frontier model as it ships. S2S gives you the reasoning layer that was baked into a voice model at its last training run.

That gap is not small, and it widens every six months as frontier text LLMs improve faster than audio models.

The summary scorecard

Dimension	Assessment (Cascade vs. S2S)

Observability & auditability	Cascade ✓ (native) | S2S ✗ (requires post-hoc transcription)
Compliance logging	Cascade ✓ (native text at each stage) | S2S ✗ (opaque by default)
Reasoning quality	Cascade ✓ (frontier LLMs) | S2S △ (constrained by model)
Component flexibility	Cascade ✓ (mix and match) | S2S ✗ (monolithic)
Evaluation tooling	Cascade ✓ (rich ecosystem) | S2S △ (immature)
Paralinguistic context	Cascade △→✓ (with augmented STT) | S2S ✓ (native)
Latency	Cascade △→✓ (with streaming) | S2S ✓ (structural edge)
Cost predictability	Cascade ✓ (linear) | S2S △ (grows with turn count)
Failure isolation	Cascade ✓ (per-component) | S2S ✗ (all-or-nothing)
Production maturity	Cascade ✓ (dominant in enterprise) | S2S △ (maturing)

△ = partial / addressable with additional engineering. △→✓ = addressable with the design recommendations in Chapter 6.

CONCLUSION

Conclusion

The voice agent architecture debate gets framed as a generational contest: the clunky old cascade versus the sleek new S2S model. That framing is wrong, and it has been sending teams in the wrong direction.

Speech-to-speech models have real advantages in raw latency and in not discarding the audio signal at transcription. For consumer applications, prototyping, and contexts where naturalness is the only value that matters, they are legitimate. They will continue to improve.

But the specific things that make S2S attractive - emotional context, prosodic awareness, fuller representation of what was actually said - are not properties of S2S architecture. They are properties that standard cascade implementations throw away through lazy design. The audio signal contains all of that. Most STT stages simply choose not to pass it along.

Build the cascade correctly and the supposed S2S advantages mostly disappear. Stream all three stages. Augment your STT layer to pass emotion, prosody, and behavioral signals downstream. Engineer the LLM context with intention. Detect turns using a mix of semantics and acoustics. Doing this recovers most of what S2S promises - while keeping the observability, the component flexibility, the compliance footprint, and the reasoning quality that production enterprise deployments require.

The black-box nature of S2S is not an implementation detail you can engineer around. It is structural. And in production, it means you cannot debug what broke, cannot satisfy auditors asking for interaction records, cannot run targeted regression tests, and cannot swap your reasoning layer when a better one ships next month. In consumer contexts, that might be acceptable. In enterprise, it is not.

The question for every enterprise team building voice agents today should not be "cascade or S2S?" It should be: "what is our STT layer throwing away, and how do we stop doing that?"

> ***The best voice agent is not the one with the thinnest architecture. It is the one that passes the most signal through to where decisions get made.***

A note on tools

If the argument in Chapter 5 and 6 landed - that the cascade pipeline's critical weakness is what it throws away at the STT stage - Modulate's Velma API was built specifically to solve that problem.

Velma is an audio-native analysis layer that sits at the STT-to-LLM boundary in a cascade pipeline. Send it a conversation, and it returns structured understanding: emotions, speaker dynamics, topic classification, 150+ prebuilt behavioral signals, deepfake detection, and natural-language-defined custom behaviors. The LLM gets the full context of what was actually happening in the conversation, not just what words were said.

We have kept this section short because this ebook is not a sales pitch. But if you are building the well-designed cascade described here, Velma is directly relevant.

Free API access and docs are here [[[ADD LINK]]].

References and Further Reading

- Softcery: Real-Time vs Turn-Based Voice Agent Architecture (2026) [S2S cost data]
- Hamming AI: Are Speech-to-Speech Models Ready to Replace Cascade Models? (2025)
- Daily.co: Benchmarking LLMs for Voice Agent Use Cases (2026)
- Coval AI: Speech-to-Speech vs Cascaded Voice AI (2026)
- arxiv: Building Enterprise Realtime Voice Agents from Scratch (2026)
- arxiv: Reading the Mood Behind Words -- Emotion-Aware VR Agents, CHI 2026
- Twilio: Guide to Core Latency in AI Voice Agents (2025)
- Introl: Voice AI Infrastructure -- Building Real-Time Speech Agents (2025)
- AssemblyAI: Voice Agent Architecture -- STT, LLM, TTS Pipelines Explained
- Softcery: Real-Time vs Turn-Based Voice Agent Architecture (2026)
- Hamming AI: The Ultimate Guide to ASR, STT & TTS for Voice Agents (2026)
- LiveKit: Voice Agent Architecture: STT, LLM, and TTS Pipelines Explained (2026)
- LTS-VoiceAgent: A Listen-Think-Speak Framework (arxiv, 2025)
- Softcery: US Voice AI Regulations 2026
- Modulate: Velma API