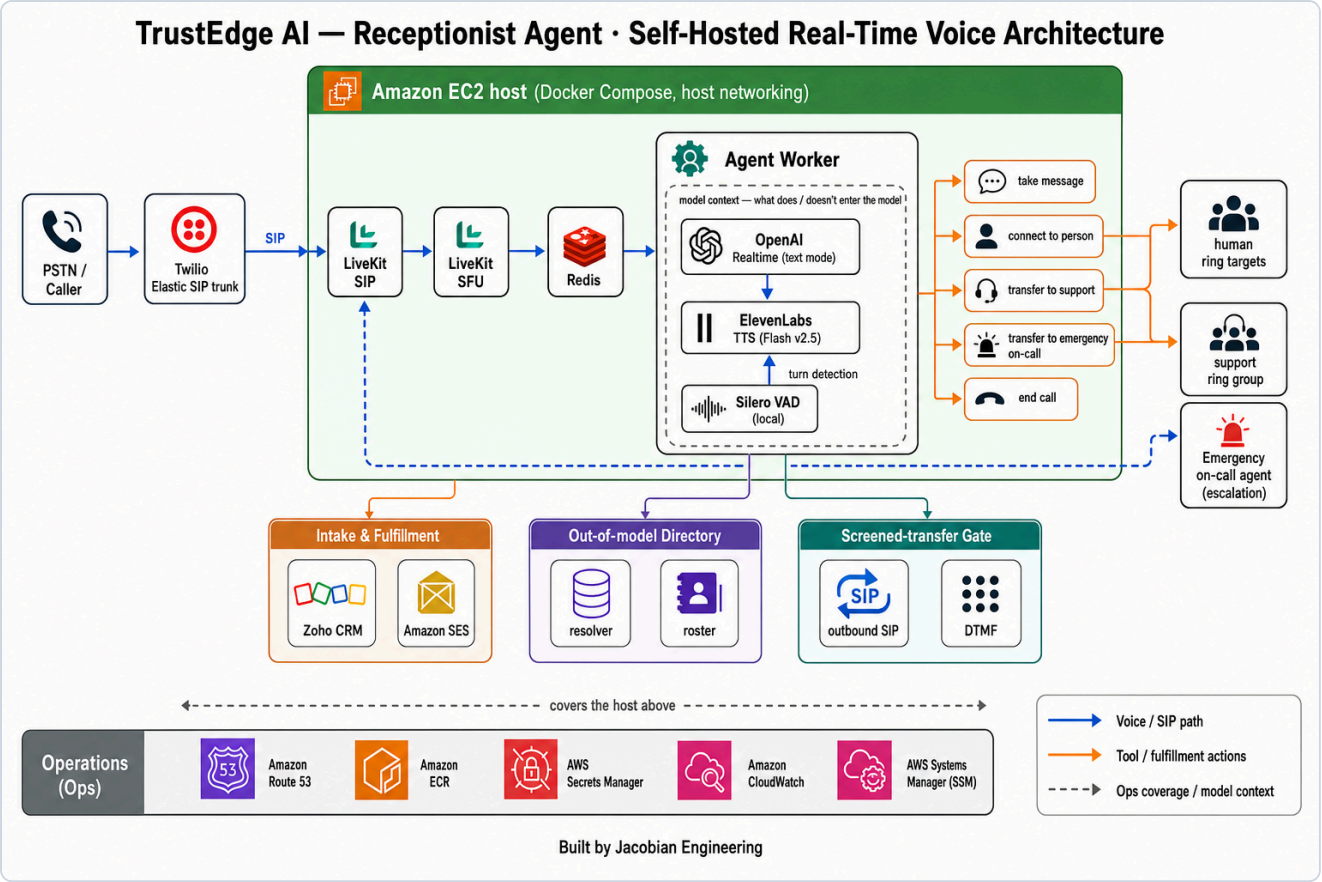
TrustEdge AI: A Conversational Receptionist That Never Drops a Caller



Architecture at a Glance

1. Executive Summary

Chris is a conversational AI receptionist that answers TrustEdge AI's own main business line. It replaces the touch-tone menu entirely: it greets every caller in a consistent, branded voice — a normal welcome during business hours, a "we're currently closed, but I can still help" variant after hours — understands what the caller wants without forcing them through a menu, answers basic questions about the firm, captures a complete structured message into the right business system, and performs **screened, supervised transfers** that confirm a real human has answered and accepted before bridging the caller. It never drops a caller into voicemail. The caller hears one calm voice; the right person gets a complete message or a screened live call; and every action behind the call is logged and auditable.

TrustEdge built Chris for its own front door first, as a research and reference implementation — dogfooding before productizing. TrustEdge AI is the AI services division of Jacobian Engineering, an employee-owned IT-security, compliance, and cloud-services firm with two decades of trust-critical infrastructure work behind it, serving regulated sectors including healthcare and financial services. The firm already ran an autonomous after-hours emergency dispatcher — the companion agent, *Jaiven*, documented in its own case study — but it had no front-of-house agent for the everyday calls that arrive on the main line: a prospect asking about penetration testing, a client's accounts-payable clerk with an invoice question, someone trying to reach a specific engineer by name. Those calls were handled by whoever picked up, or by voicemail. Rather than ship a receptionist product built on a slide deck, TrustEdge ran Chris on its own number, found the failures that only surface under real telephone traffic, and engineered them out.

Two pieces of that engineering are the headline. The first is the **screened transfer**: a transfer that rings a human, plays them a private whisper announcing the caller, and bridges the caller only when the human presses a key to accept — a design in which the keypress itself is the voicemail guard, because a voicemail system answers but cannot press a key. The second is the **out-of-model staff directory**: the feature that connects a caller to a named employee runs the lookup entirely outside the language model, so an employee's phone number and email never enter the model's context, and the model can neither read the roster back to a prompt-injection attacker nor be steered into dialing an arbitrary destination. Both are expressions of the same thesis that runs through all of TrustEdge's voice work: **a generative model is an excellent conversational surface and an unreliable process controller, and the durable value is the deterministic machinery built around it** — finite action surfaces, server-owned targets and decisions, guaranteed call-control, and trust boundaries around sensitive data.

Every low-level choice in Chris follows from what a front desk actually needs. The conversation runs on a cascade — recognition and reasoning in one component, a dedicated branded voice in another — because a front desk's voice is part of its identity and because that decomposition makes the decision logic ordinary, testable software. Turn-taking is handled by a local detector rather than the model's native one, because real telephone audio defeats a detector tuned for clean speech. And the whole system is self-hosted on a single stateful host, because real-time media needs a network posture that a serverless container cannot provide. None of these is a default; each is a deliberate answer to a concrete requirement, and the chapters that follow work through them in turn. Matching architecture to purpose is most of what productionizing voice AI actually is.

> *"Everybody hates the phone menu. Press one for sales, press two for support, press nine to hear these options again — nobody has ever enjoyed that, and it makes a caller pre-sort their own problem into your org chart before you've said a*

> *word to them. We took the menu out. You just talk, and the thing that answers is competent, consistent, and never once parks you in somebody's voicemail. And we put it on our own main line before we offered it to anyone, because the front desk is exactly the kind of thing you only get right by living with it." — Erik D. Jones, Lead Architect & CEO, TrustEdge AI*

This case study is the engineering story behind that decision.

[[TOC]]

---

2. The Problem

The front door of a professional services firm is its phone line, and for decades the only two options have both been bad. A human receptionist is warm, capable, and able to handle a caller who does not quite know what they need — but a human is expensive, works business hours, and cannot be in two places at once. A touch-tone IVR is cheap and always on — and uniformly disliked. Neither serves the caller who picks up the phone and says, *"I think I need help with my firewall, or maybe I should talk to whoever handles billing."* The human handles that sentence effortlessly. The menu cannot handle it at all. And on the firm's own main line, the everyday reality before Chris was the worst of both: calls handled by whoever happened to be free, and the rest going to voicemail, with phone leads quietly lost because they never reached the CRM and "what do you actually do?" left unanswered at the very first point of contact.

An IVR is the wrong baseline, and it is worth being precise about why. A press-1/press-2 menu fails the same caller in four distinct ways, and each one is structural rather than a tuning problem. It forces the caller to **self-classify** their need into the firm's internal taxonomy before anyone has helped them — and a caller who is unsure whether their problem is "support" or "billing" is exactly the caller a menu cannot route. It **cannot answer a question**; a menu has no idea what the firm does and no way to say so. It **cannot take a real message**; the best it manages is a recording that lands in a box someone may check. And when it finally transfers, it transfers **blindly** — it hands the call to a number and relinquishes control, with no idea whether a human answered or whether the caller has just been deposited into a personal voicemail to age unheard. The goal of this project was never a better menu. It was to remove the menu and let the caller speak.

A front desk that is genuinely better than a human receptionist on the dimensions that matter has to clear a specific, demanding bar. It must hold a natural conversation with no menu and no scripted prompts the caller has to navigate. It must understand fuzzy, real-world intent — "the firewall or maybe billing" — and resolve it without making the caller do the sorting. It must be able to answer basic questions about the firm at the front door, where the question is actually asked. It must capture a complete, well-formed message into the **correct downstream system** — a

CRM lead for a sales inquiry, the accounting inbox for an invoice question, a direct message to a named individual — rather than a generic recording. And, hardest of all, it must perform transfers that are **screened** — meaning a human actually answered and actively accepted the call — and **safe** — meaning the caller is never silently parked in voicemail, and there is always a graceful fallback when no human can be reached. Underneath all of that sits a security requirement that colors every decision: the caller's speech is **untrusted input**, never to be treated as an instruction or a target. A caller can say anything, including things engineered to manipulate the system, and the design has to assume they will.

---

3. TrustEdge's Approach

Chris is a single conversational agent wrapped in a finite, server-controlled action surface, and the whole design rests on one division of labor: the model decides what the caller wants, and code decides what is allowed to happen and to whom. The tempting design — and the one TrustEdge built first, then tore out — hands the model far more than that: let it route, let it choose who to call, let it decide whether a transfer succeeded, let it pick what to say when one doesn't. That design demos beautifully and fails in production, because the model is being asked to be the process controller, which is the one thing it is not reliable at. The approach that survived live calls is the opposite: the model is confined to the conversation, and every consequential decision, target, and guarantee lives in deterministic application code around it.

The first concrete expression of that principle is the collapse to a single agent. An early iteration of Chris was a multi-agent system — a front-door agent that handed off to specialist agents for sales, accounting, and general inquiries, plus dedicated transfer specialists. It was the natural-seeming design, and live testing killed it. Handoffs lost conversational context, so callers were re-interrogated; the experience felt like being passed through a series of contact forms; and every transition introduced an awkward gap of dead air. Collapsing the system to a single agent with a finite tool surface did not patch those failures one by one — it dissolved them structurally, because there are no handoffs left to lose context across. The only "specialist" that ever genuinely mattered was sales, and that turned out to be a product-knowledge problem better solved later with retrieval, not a reason to fragment the conversation. Simpler was not merely cleaner; it was more reliable.

The second expression is that every low-level decision is driven by what a front desk specifically requires. Chris is the firm's front door, and that purpose dictates the engineering. A consistent brand voice is not a nicety here — it is the product, because the voice is the first and most repeated impression a caller forms of the firm. The system has to be testable enough that the team can evolve it confidently against live traffic, which pushes the conversational logic toward ordinary software that can be

exercised without audio. And the hardest engineering is not in extracting data from speech but in the **call-control around transfers** — confirming a human answered, never stranding a caller, falling back gracefully — which is where most of this study's depth lives. Those three pressures — brand voice, testability, and transfer call-control — justify the model topology, the turn-detection strategy, and the hosting model that the following chapters work through, each on its own merits.

> *"The first version had a front-desk agent that handed off to a sales agent that handed off to a transfer agent, and on paper that looks tidy — everybody has a job. On a real call it was miserable. The caller had to re-explain themselves at every seam, there were these little silences where one agent was waking up the next, and the whole thing felt like filling out a form out loud. So we threw it away and built one agent with a handful of tools. Added agents add handoff seams, and handoff seams are exactly where conversational quality goes to die. You reach for more than one agent only when one genuinely can't hold the problem — and a front desk can." — Erik D. Jones, Lead Architect & CEO, TrustEdge AI*

4. Architecture Overview

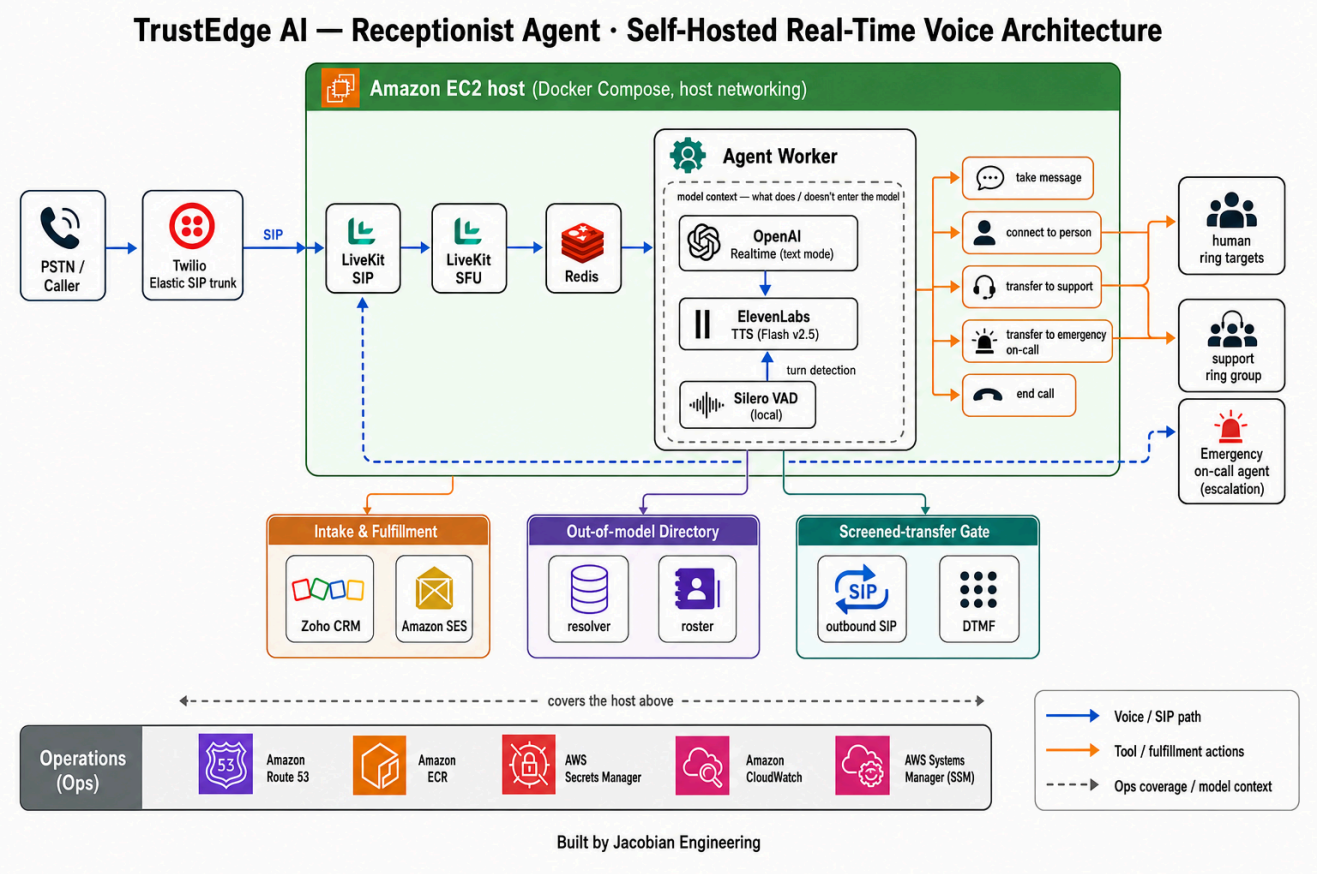


Figure 1 — Logical architecture: a Twilio SIP trunk into a self-hosted LiveKit SFU/SIP stack on Amazon EC2, a cascade of OpenAI Realtime, ElevenLabs, and Silero, and five server-controlled tools.

Chris runs on a self-hosted real-time voice spine, all co-located on a single host. The media plane is **LiveKit** — an open-source SFU media server — paired with its companion **LiveKit SIP** service, coordinated through **Redis**, with the agent worker running beside them. All four services are orchestrated with **Docker Compose** on a single **Amazon EC2** host. A **Twilio** Elastic SIP trunk delivers PSTN calls into LiveKit SIP, which places each inbound caller into an individual room; a dispatch rule attaches the agent worker to that room, and the conversation begins. Chris owns its media plane outright — it is not a thin client on a carrier's hosted media product — and that single decision cascades into everything about how the system is deployed and operated.

The reason for self-hosting is concrete and networking-shaped: real-time media and SIP need a wide UDP/RTP port range, and the host uses host networking to provide it. Serverless container networking handles a broad, dynamic range of UDP ports poorly — it is built for a small number of well-known TCP ports behind a load balancer, not for the RTP media flows a SFU negotiates per call. Host networking on a dedicated EC2 instance gives the media server and the SIP service direct, unmediated access to the port range they need. A serverless container, by contrast, is built for a request/response-plus-WebSocket shape behind a load balancer, which is precisely the shape a real-time SFU does not have. The medium dictates the hosting, and the medium here is live RTP media. Architecture follows the medium.

Self-hosting a stateful media plane buys capability at the cost of a "precious host" constraint, and the operational discipline that follows is explicit. The live SIP trunk configuration and the dispatch rule that attaches the worker are applied to the running server out-of-band — they are state that lives on the host, not purely in the infrastructure definition. A naive infrastructure redeploy would replace the host and wipe that state, dropping the live trunk on the floor. So the team adopted a strict rule: **routine updates roll the agent container in place** — pull the new image and recreate just the agent service — rather than re-running the full infrastructure deploy. The full deploy is reserved for genuine infrastructure changes and is treated as the disruptive operation it is. This is a direct consequence of the hosting choice: real-time media pushes you toward stateful hosting, and stateful hosting demands deployment discipline.

The supporting cloud fabric is conventional and infrastructure-as-code throughout. The stack is defined in **AWS CDK**; the agent's container image is held in **Amazon ECR**; runtime credentials live in **AWS Secrets Manager** and are pulled at boot, never baked into the image. Host access is through **AWS Systems Manager Session Manager** rather than public SSH — there is no inbound shell port to defend. **Amazon Route 53** resolves the SIP hostname to the host's stable address. The deliberate property of the whole posture is that the only special, stateful thing in the

system is the one host that has to be, and everything around it is declarative and reproducible.

One piece of per-call state matters enough to call out here: the caller's network-provided number. When a call arrives, LiveKit SIP exposes the caller's ANI — the network-provided calling number — as an attribute on the SIP participant. Chris reads it once, at the start of the call, and holds it as a server-owned, model-immutable field for the rest of the session. The model is never the source of truth for that number; it cannot fabricate it, alter it, or override it. As the later chapters show, that single server-owned field is load-bearing in two places — it becomes the default callback for a captured message, and its absence as a model-controlled value is what forced the fix for an early bug where the agent simply invented callback numbers out of thin air.

> *"Chris is the receptionist we wanted at our own front door — it picks up, it actually understands what you need, and it either gets you to the right person or takes a real message you can count on. We gave it a name because a name sets the expectation: 'you've reached Chris' tells you you're about to have a conversation, not fight a menu. But Chris is a receptionist with hard rules — warm on the phone, and ruthless about what it is and isn't allowed to do. That combination is the entire design." — Erik D. Jones, Lead Architect & CEO, TrustEdge AI*

---

5. The AI Core: a Deliberate Cascade

Chris does not run on a single speech-to-speech model. It runs on a cascade of three components, each doing one job, and that decomposition is what makes the rest of the system tractable. Recognition, reasoning, and tool dispatch run on the **OpenAI Realtime API in text mode** — the model takes recognized speech in as text and emits text and tool calls out, and it is explicitly *not* asked to synthesize voice. Voice is rendered separately by a dedicated **ElevenLabs** text-to-speech voice (the ElevenLabs Flash v2.5 model), a single consistent branded voice used across the greeting, the conversation, the hold audio, and the transfer whisper. Turn-taking — deciding when the caller has actually spoken — is handled by a local **Silero** voice-activity detector running in-process, not by the model's server-side turn detection. Three components, three jobs, no overlap.

This cascade is a deliberate choice, and there are four specific reasons it is right for a front desk. The first is **voice-brand control**. A dedicated TTS voice gives the firm one consistent vocal identity across every moment of the call — the greeting, the back-and-forth, the hold loop while a human is being rung, the whisper to that human. A pure speech-to-speech model would couple the brand voice to whatever the model vendor offers in its voice catalog, and would make it impossible to use the *same* voice for the out-of-band whisper that happens in a different room. For a front desk, the

voice *is* part of the product, and the firm has to own it outright rather than rent it from a model vendor's catalog.

The second reason is testability, and it is the one the team leaned on hardest. Because reasoning and tool dispatch happen as text in and text out, the agent's entire decision logic — which tool to call, with what arguments, in response to what the caller said — is **unit-testable with no audio infrastructure whatsoever**. You feed the decision layer text and assert on the tool calls and arguments it produces. A pure speech-to-speech model buries that logic inside an audio stream that is far harder to exercise deterministically. The cascade made the decision layer ordinary software, and the rest of the build's heavy reliance on testing (Chapter 11) is downstream of that single property.

The third reason is turn-taking control. Running the reasoning model in text mode makes it possible to *disable* its server-side turn detection and substitute a local detector — which, as the next chapter details, was not a nicety but the fix for a serious live bug. In a pure speech-to-speech mode the turn-detection is welded into the model and cannot be swapped out. The cascade is what made the substitution possible at all. **The fourth reason is transfer-whisper isolation**: the screened-transfer system needs a second, TTS-only voice session to whisper to a callee in a separate room, and having explicit, separable TTS makes that clean rather than a fight against a monolithic model. The TTS speech rate is tuned slightly below natural conversational pace, because live testing found the default too fast for telephony and an over-slow setting unnatural — a small, empirical adjustment. A voice-to-voice A/B comparison is on the roadmap, but it was never needed to ship; the cascade earned its place on its own merits.

---

6. Turn-Taking: the Phantom-Turn Bug and the Local-VAD Fix

The most disruptive early-production bug lived in turn detection, and its resolution is the clearest possible illustration of why the cascade topology earns its keep. Turn detection is the deceptively hard problem of deciding *when the caller has actually spoken* — when to stop talking and listen, and when a sound is a genuine interruption versus background noise. The first implementation trusted the Realtime model's own server-side semantic voice-activity detection to make that call. On clean, high-fidelity audio it works well. On real 8 kHz telephone calls it failed, and it failed bizarrely.

The model's server-side VAD hallucinated phantom caller turns out of telephony silence and codec noise. Trained on high-fidelity audio, it heard the artifacts and silence patterns of a low-bitrate phone line as speech that was never spoken, and the downstream symptoms were strange and severe. The agent **interrupted itself mid-greeting** — the VAD flagged a phantom turn, the agent

stopped to "listen," and the greeting fractured. The transcript filled with **spurious foreign-language tokens** as the recognizer dutifully tried to make words out of the noise the VAD had wrongly flagged as speech. And the obvious remedy made it worse: enabling the model's built-in noise-reduction option *degraded* behavior further at 8 kHz rather than helping, because that option, too, is tuned for a fidelity the telephone line does not have.

The fix was to take turn-detection away from the model entirely. The Realtime model's server VAD was disabled. Turn detection was moved to a **local, in-process Silero detector** — an acoustic model far better suited to telephony noise than a semantic detector tuned for clean audio. A **minimum-words filter** was added on top: a detected turn does not count as a genuine interruption until it carries at least a minimum number of words, so a single noise-induced token can no longer cut the agent off mid-sentence. And the detector is loaded **once per worker process at startup** — a "prewarm" step — so the first real call pays no model-loading latency. That same prewarm step does double duty: it validates the staff directory at boot, so a malformed roster fails the worker on startup rather than mid-call, when a caller is waiting.

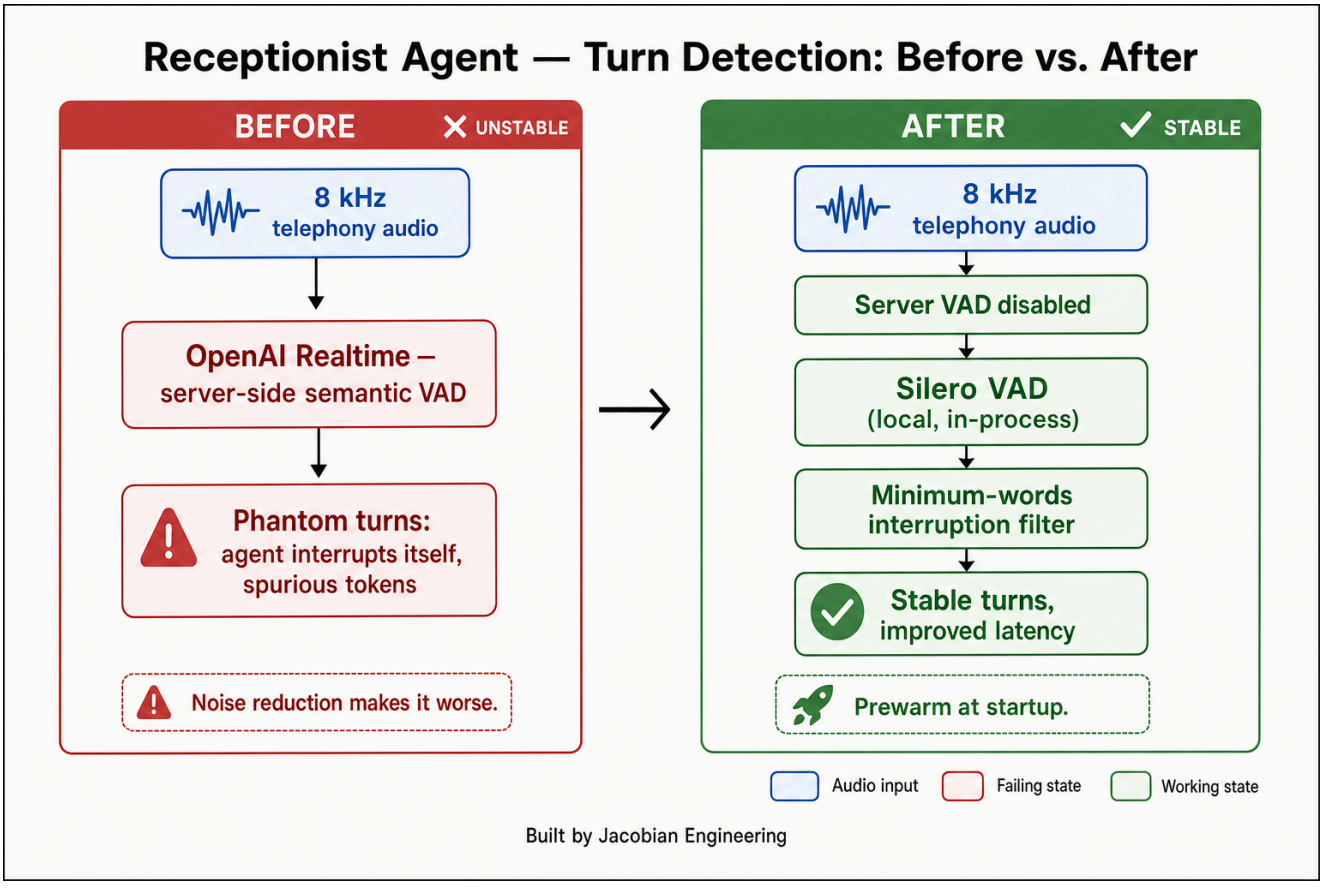


Figure 2 — Why the cascade earns its keep: replacing the model's server-side VAD with a local Silero detector plus a minimum-words filter ended the phantom-turn bug.

After the switch, conversation quality was confirmed good, per-turn latency actually improved, and the deliberately uninterruptible greeting finally behaved. The improvement in latency was the pleasant surprise — moving turn

detection in-process removed a round-trip that the server-side detector had imposed. The lesson generalizes cleanly and is one of the case study's central technical results: **server-side semantic turn detection tuned for clean audio is the wrong tool for telephony; a local acoustic detector with a minimum-words interruption filter is dramatically more robust.** And the substitution that fixed it was only possible because Chris runs a cascade — in a pure speech-to-speech topology there is no seam at which to pull turn detection out of the model and replace it. The bug was a telephony problem; the *fix* was an architecture problem, and the architecture had the seam.

---

7. The DTMF Screened-Transfer Gate

This is the receptionist's most novel component, and it exists to solve a problem every naive phone agent has: a blind transfer has no idea whether a human answered. The first implementation used a cold transfer — it handed the call to the carrier and relinquished control. If the target did not pick up, the caller landed in that person's personal voicemail, with no detection and no recovery. There was no signal back to the agent that anything had gone wrong, no fallback, no second chance. Every non-emergency transfer was, in effect, broken: it worked only in the lucky case where the target happened to be at their desk and answered in person.

A screened — supervised — transfer rings the target and, before bridging the caller, plays the target a private whisper. The whisper says, in the same branded voice, something to the effect of *"Call from reception: [caller] from [company], regarding [topic]. Press 1 to accept, or hang up."* The caller is bridged only if the target **presses the accept key**. If no key is pressed within a short window, the target is treated as unavailable and the call falls back gracefully. The screened design confirms two things a blind transfer never can: that a human is present, and that the human has actively chosen to take this specific call.

The DTMF keypress is the voicemail guard, and this is the elegant part: it needs no answering-machine-detection heuristics at all. A voicemail system auto-answers — it picks up, it plays a greeting, it sounds for all the world like a successful connection — but it *cannot press a key*. So a call that "answers" but never produces the accept tone within the window is, by definition, known to be voicemail or an unavailable human, and the caller is never bridged into it. There is no fragile audio analysis trying to guess whether a beep was a person or a machine; the keypress is a positive, unambiguous signal of a present, willing human, and its absence is an equally unambiguous signal that no such human is there.

The gate is split into two layers for testability: a pure orchestration function and a live dialer behind an interface. The orchestration function takes an injected "dialer" interface and a list of targets, rings them **concurrently**, and bridges the **first to accept** while cancelling the rest; an empty target list fails fast. Because the dialer is injected, the entire first-to-accept-wins logic is **unit-tested with a fake dialer**, no

telephony involved — the test asserts that concurrent rings race correctly, that the first acceptance wins, that the losers are cancelled, and that an empty list fails immediately. The live dialer holds the real LiveKit SIP glue and is exercised only in live validation. A useful consequence of the first-of-N design: a **support ring group is the same primitive as a single ring**. Ringing one person and ringing a whole support team are the same code path — a list of length one versus a list of length several — and the first to accept wins in both cases.

Per-leg audio isolation is the subtle technical problem underneath, and it is solved with one screening room per target. To whisper to a callee *without the caller hearing it*, the agent needs to play audio to one participant privately — but the SFU mixes audio at the room level, and there is no primitive to play to a single participant in a room while the others hear nothing. The solution, confirmed by a research spike before any code was written, is to give each target its own dedicated screening room. For each target the agent creates a screening room and connects to it; starts a **TTS-only whisper session** in that room purely to speak the announcement; dials the target into the room with a ring timeout; on answer, plays the whisper and waits for the accept key within the screening window, listening only to that callee's keypresses; and on accept, **moves the callee's call leg into the caller's room** — a server-side atomic operation that bridges the two without dropping the PSTN connection. On no-accept, it tears the screening room down, which hangs up the unaccepted leg, voicemail included.

One deliberate non-action is worth calling out, because it was caught in code review before it ever shipped. The bridge does **not** delete the screening room after moving the leg out. Deleting it could race the move and drop the leg mid-transfer; the emptied room is left to be reaped naturally instead. It is a small thing, and exactly the kind of concurrency hazard that distinguishes a transfer that works in a demo from one that works on the thousandth call.

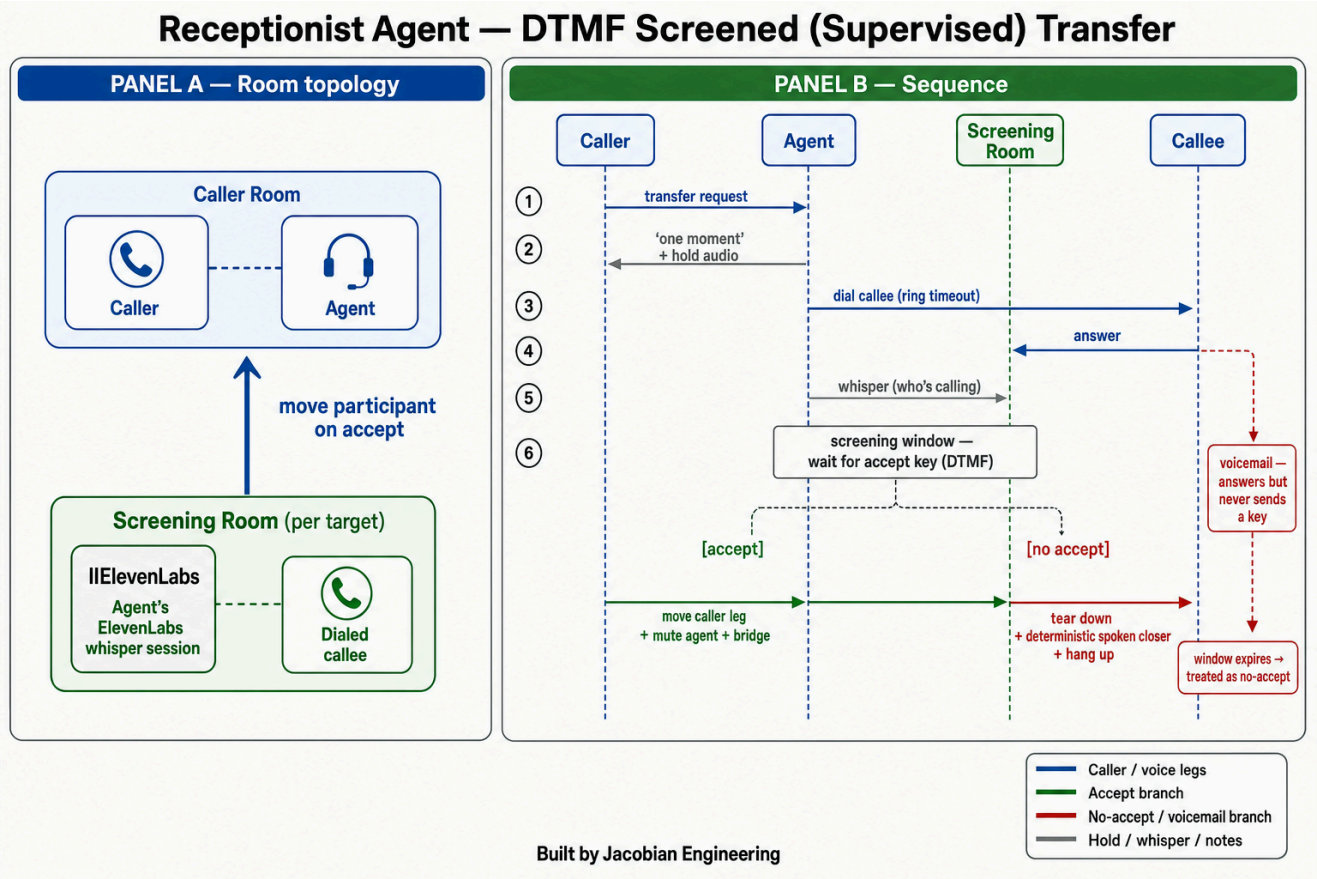


Figure 3 — The screened transfer. A per-target screening room whispers to the callee and bridges only on a DTMF accept; a voicemail answers but can never press the key.

Around the core gate sit several supporting touches, each closing a gap that real calls revealed. Once a transfer is bridged, the agent **mutes its own output** so it cannot talk over the live human-to-caller conversation, and the end-call tool is **guarded** so the model cannot accidentally tear down a live bridge by deciding the call is over. Screened rings are **time-gated**: they fire only during business hours, and after hours the caller's information is captured with no ring and an immediate graceful fallback — while the emergency route bypasses the gate and reaches the on-call agent around the clock. While a target rings, the caller hears a **branded hold loop** — a short spoken message about the firm in the same brand voice, mixed over music — rather than dead silence; it is best-effort, and if the audio asset is unavailable the transfer proceeds anyway. And the whisper carries the caller's *context* — who they are, what company, what topic — but **never the caller's phone number**, only what the human needs to decide whether to accept.

> *"The thing I will not ship is a transfer that drops somebody into a voicemail box. A blind transfer is a coin flip — maybe a person picks up, maybe the caller ends up talking to an answering machine that will never call them back, and the agent has no idea which happened. So we made the human press a key to accept the call. A voicemail can answer the phone, but it can't press one. That single keypress is the whole guarantee: if nobody presses it, we know nobody real is there, and we take a*

> *message instead of abandoning the caller. Nobody gets parked in a voicemail. Ever." — Erik D. Jones, Lead Architect & CEO, TrustEdge AI*

8. The Out-of-Model Directory

The "connect me to a specific person" feature carries a sharp security risk if it is implemented naively, and the way Chris solves it is the case study's signature idea. The naive implementation is obvious and tempting: put the staff roster — names, cell numbers, email addresses — into the model's context so it can "look people up," then let it dial or email whoever the caller asks for. That design opens two doors at once. A **prompt-injection** caller can manipulate the model into reading the roster back, exfiltrating private contact details for every employee. And a manipulated model can be steered into dialing or emailing an **arbitrary destination** of the attacker's choosing. Putting contact data into the model's context turns the model into a liability surface for that data, full stop.

So the directory lookup runs entirely outside the model, and the model never sees a phone number or an email address under any circumstances. The model's only input to the feature is the **spoken name** the caller said. A pure, server-side resolver matches that name against a committed, version-controlled roster and returns **at most two candidate labels — name and department only**. Phone numbers and email addresses are held server-side and are never returned to the model. The candidate object the model can see is **structurally incapable of carrying contact data** — it exposes a name and a department and nothing else — and that property is asserted directly in the test suite, which checks that a candidate has exactly those fields and no way to smuggle a number or an address.

The resolution flow is a small, deterministic state machine, and confirmation gates every consequential step. The model passes the spoken name to the connect-to-person tool, along with the caller's own details. The server resolves the name with a **fuzzy matcher** — normalized comparison against a similarity threshold, plus a boost when *all* the query words appear in the candidate's name, so a caller who offers only a first name still matches well. From there: a confident, unique match proceeds directly; a single near match returns one label and the agent asks the caller to confirm — *"I have [name] in [department] — shall I connect you?"*; multiple near matches return up to two labels and the agent asks which one; and **no match** increments a per-call counter and, after a small number of attempts, falls through to taking a general message rather than looping forever. Only **after** the name is confirmed does the server map it — server-side only — to the full contact record it uses to ring or email. That record never crosses into the model's context.

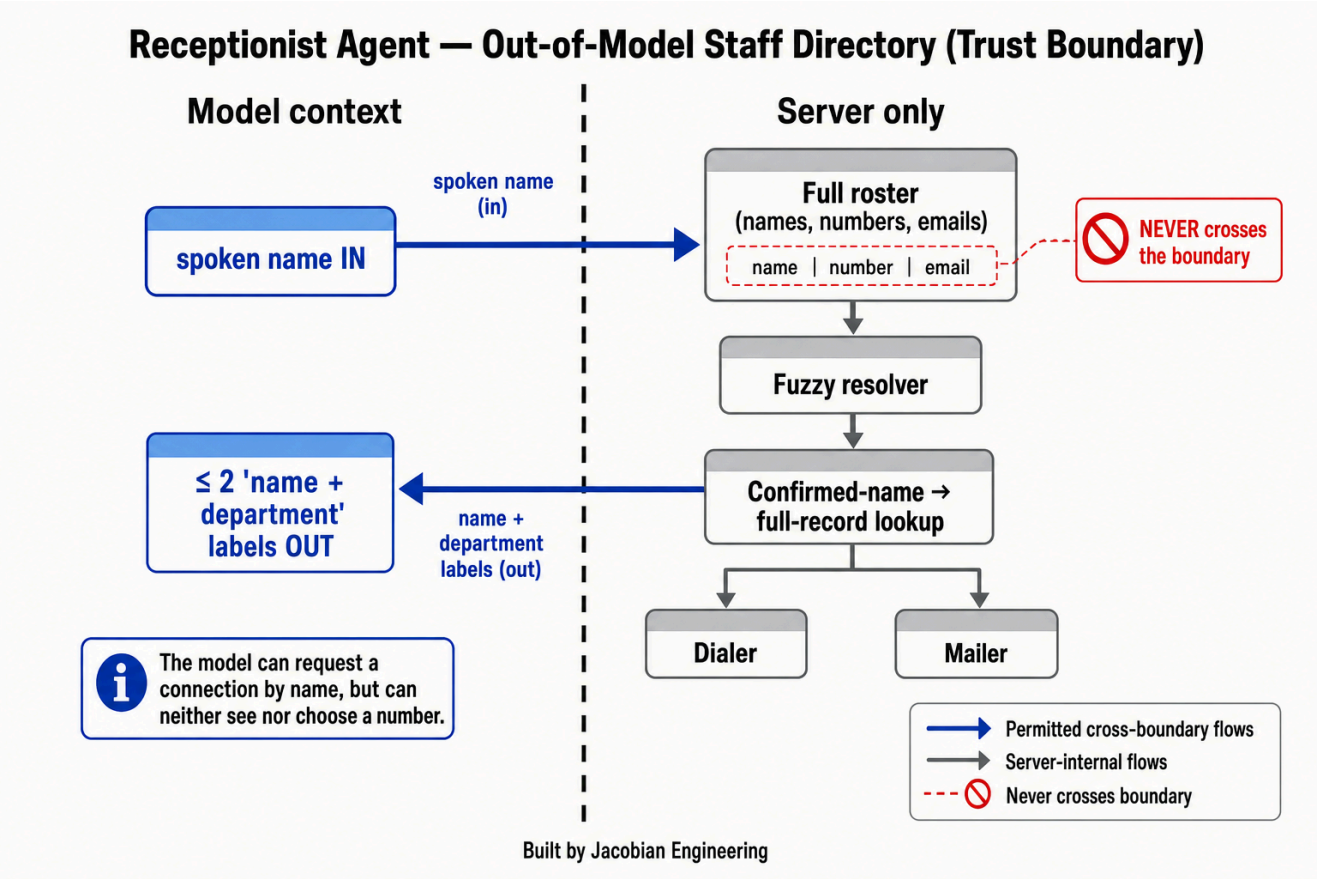


Figure 4 — The out-of-model directory. Only a spoken name crosses into the model and only a name-and-department label crosses back; contact details never enter the model's context.

The same primitive powers the support ring group, with the same property. The support team's numbers live in the same roster; the support tool pulls the list server-side and rings them as a first-to-accept group through the screened-transfer gate. The model asks to "transfer to support" — it never sees, and never could see, a single one of those numbers. The directory is, in other words, not a feature bolted onto the side of the security model; it *is* the security model, applied to the one place where the model would otherwise have the strongest pull toward touching sensitive data.

> *"Contact data does not go into the model. That is the rule, and the directory is built around it. The model is allowed to know that a caller said a name; it is not allowed to know that person's cell number, because the moment a phone number is sitting in the model's context, you have built a machine that a clever caller can talk into reading it back, or into dialing somewhere it shouldn't. So the model hands us a name, we resolve it in plain code against a roster the model never sees, and only after the name is confirmed does the server — not the model — reach for the actual number. The model can ask to connect someone. It can't see a number, and it can't choose one." — Erik D. Jones, Lead Architect & CEO, TrustEdge AI*

9. Intake & Fulfillment

When Chris takes a message rather than transferring, a deterministic intake gate stands between the conversation and any side effect — and it is code, not a promise made in a prompt. The gate sanitizes every field, trimming whitespace and capping length so no field can carry an unbounded or malformed payload downstream. It normalizes phone numbers to a canonical form. It validates the hard-required fields and, if one is missing, re-prompts for it *through the conversation* — the agent simply asks again, naturally — rather than failing the message. It treats the "regarding" field as **soft**, so a caller who will not state a reason is never blocked from leaving a message. And because the caller's network-provided number is already a server-owned field, it is used as the **default callback** before the gate even runs, so the agent does not badger a caller for a number it already has.

Validated messages are fulfilled into the correct backend through an injected fulfillment component, and the routing is specific to the message category. A **sales** inquiry becomes a **Zoho CRM lead**, created through a server-to-server OAuth integration with the CRM, with a follow-up task attached; if the CRM call fails, it falls back to an email to the sales inbox so the lead is never lost. **Accounting** and **general** messages become **emails via Amazon SES** to their respective inboxes. A **named-person** message becomes a **direct SES email** to that individual. In every case the caller hears a normal confirmation that their message has been taken — backend failures never surface to the caller, because the caller's experience should not depend on whether a downstream API happened to be healthy at that moment.

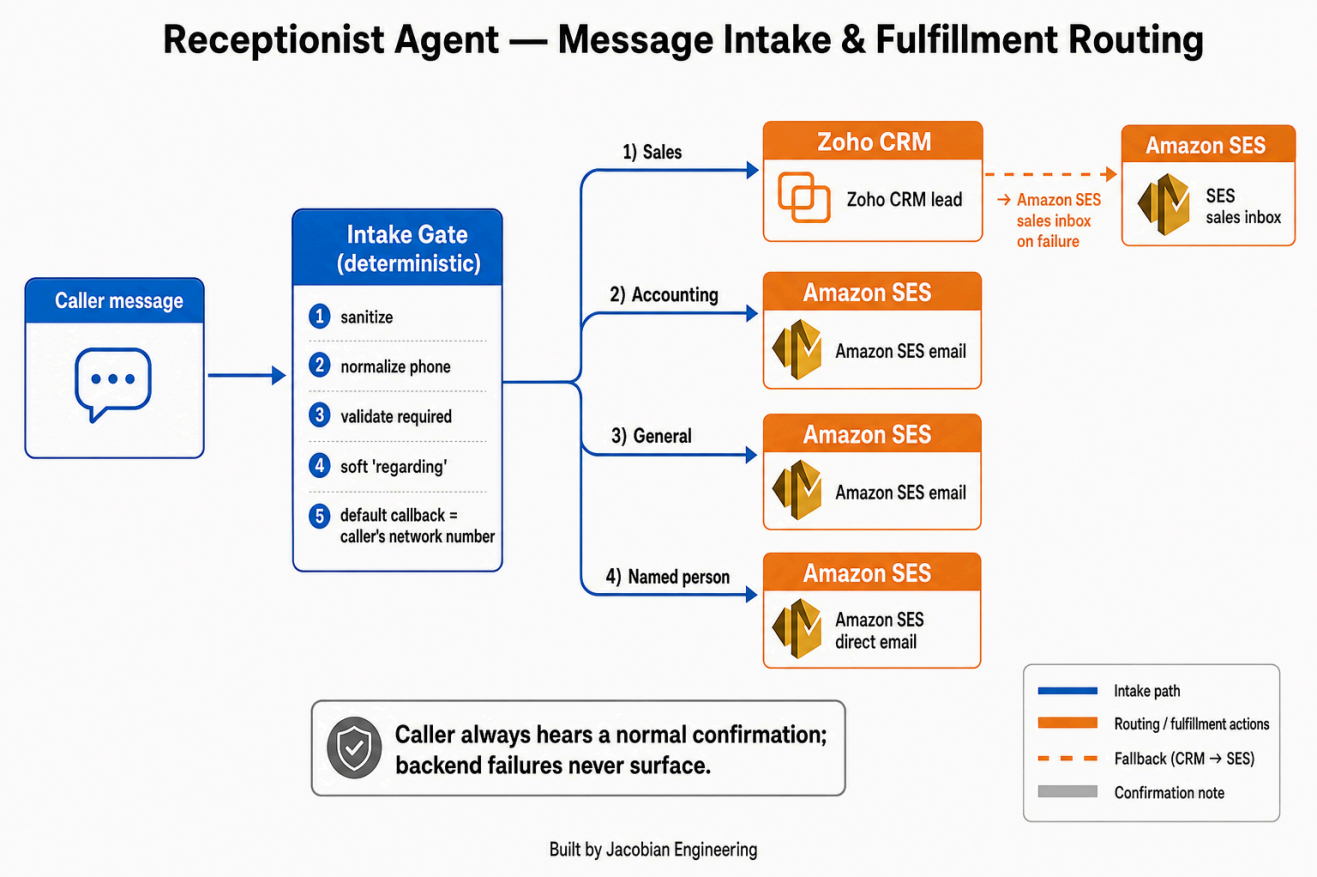


Figure 5 — Deterministic intake and fulfillment routing: sales to a Zoho CRM lead with an email fallback, accounting and general to email, a named person to a direct email.

The entire fulfillment layer is built on dependency injection, which is what makes it testable without touching a live network. The CRM call, the email call, and the metrics emitter are all injected dependencies. In the test suite they are fakes, so the suite exercises the full routing logic — category to backend, success path, fallback path — while making no live network calls at all. This is the same testability discipline that drove the cascade and the injected dialer, applied here to the side-effecting layer. It is also, as the next two chapters show, the layer where the most pedagogically valuable bug of the whole build hid for a while: a design that catches and swallows delivery failures *for the caller's benefit* is exactly the design that can hide a real, total delivery failure from the *operators* — unless the swallowing path is paired with telemetry loud enough to notice. It was, and that is what eventually caught it.

---

10. Security, Observability & Governance

The security model rests on a single governing principle: the caller's speech is untrusted data, never an instruction and never a target. That single assumption produces a set of concrete, mutually reinforcing controls. Transfer destinations are **fixed server-side** — server-side constants and roster records — with no tool parameter through which the model or the caller could specify a destination number; the model

can ask to transfer, but it can never say *where*. The action surface is **finite**: five tools (take a message, connect to a named person, transfer to support, transfer to the emergency agent, end the call), each with a server-determined effect, and no general-purpose, shell, or arbitrary-HTTP capability anywhere behind them. Caller input is **inert**: every field is sanitized and validated, and downstream delivery uses plain-text email bodies and parameterized API fields, so caller text is never interpolated into markup or a query. The **out-of-model directory** keeps contact data entirely out of the model's context. The model's context is **minimal** — no secrets, no roster, no cell numbers — so a prompt-disclosure attack that successfully coaxes the model into reading its instructions reveals nothing useful. And a **call-duration cap** terminates pathological or abusive sessions that would otherwise run unbounded.

Observability is not decoration on this system; it is the primary debugging surface, and it is built into the architecture from the first line. Every meaningful event is emitted as a **single structured log line** — conversational turns, tool decisions and their outcomes, screened-transfer results, an end-of-call summary — and shipped to **Amazon CloudWatch Logs** in a way that survives container rolls. A separate **CloudWatch metrics** stream records call starts and durations, message categories, intake outcomes, transfer and screening attempts, accepts, and fallbacks, and fulfillment successes and failures broken out by channel. This instrumentation is exactly how the silent failure described in the next chapter was caught: the failure metric was firing, visibly, in a stream the team watched, even while the caller-facing experience looked perfectly healthy. A system that swallows failures for the caller's benefit is only safe if its operators can still see those failures plainly, and this is what makes that true.

On governance, Chris embodies recognized AI-governance principles structurally rather than rhetorically, across three dimensions. On **transparency**, the agent identifies itself as an AI receptionist in **every** greeting — including the after-hours closed variant — which directly meets the disclosure expectation that frameworks such as the EU AI Act place on AI systems that interact with people; a caller is never deceived about talking to a machine. On **bounded scope and human oversight**, the agent does a small, well-defined set of things and routes to humans — or to the on-call emergency agent — for anything beyond them, and the screened transfer is the concrete mechanism that guarantees a caller is never silently stranded. On **data governance and auditability**, the out-of-model directory is a concrete data-minimization and least-exposure control, and the structured event log is the accountability trail that lets any action be reconstructed after the fact. That posture — map the risk, measure it through the logs and metrics, then manage it — is consistent with the **NIST AI Risk Management Framework's** map/measure/manage discipline, and the data-governance-plus-human-oversight framing is exactly the shape of **Singapore's IMDA model-governance guidance**. The agent's own service-knowledge block names these frameworks when it describes the firm's AI division to callers, positioning rigorous governance as a customer-facing differentiator rather than internal hygiene.

11. Engineering Challenges & Learnings

This section is the heart of the case study. Each item below cost real debugging on real calls, and each yielded a transferable lesson.

11.1 Multi-agent complexity, collapsed to one agent. The first architecture fragmented the conversation across specialist agents and dedicated transfer specialists, and live calls exposed the cost: context lost on every handoff, callers re-interrogated, a form-like feel, and dead-air gaps at each transition. Collapsing to a single agent with a finite tool surface eliminated those failure modes **structurally** rather than patching them one at a time — there are no handoffs left to lose context across. The lesson, stated plainly: added agents add handoff seams, and handoff seams are where conversational quality goes to die. Reach for multiple agents only when a single one genuinely cannot hold the problem; a bounded routing problem like a front desk is not that case.

11.2 The phantom-turn telephony VAD bug. Covered in depth in §6: the Realtime model's server-side semantic VAD hallucinated phantom caller turns out of 8 kHz telephony noise, making the agent interrupt itself mid-greeting and filling the transcript with spurious foreign-language tokens; enabling the model's noise-reduction made it worse. The fix — disable server VAD, run a local Silero acoustic detector with a minimum-words interruption filter, and prewarm it once per worker — improved quality and latency at once, and was only possible because the cascade topology provides a seam at which to substitute the detector.

11.3 The agent invented callback numbers. Early calls revealed the model fabricating plausible-looking callback numbers out of nothing. Without a ground truth for "the number you are calling from," it confabulated — which is precisely what a generative model does in the absence of a fact. The fix was to inject the caller's real network-provided number into the per-call instructions, rendered as spoken words, with an explicit prohibition on inventing or substituting any other number. The lesson is general and applies anywhere a model is asked for a fact it does not actually possess: **a generative model with no grounding will invent facts; give it the ground truth and forbid the alternative.**

11.4 Digits don't read well. Text-to-speech systems garble raw digit strings, and a misread callback number is worse than useless — it is actively misleading. The fix is to render numbers for **speech**, not for display: spell each digit as a word, drop the country code, and group the digits with pauses so the cadence is natural, while storage and dialing always use the canonical numeric form. The model is instructed to read the spoken form slowly. Digit pacing remains imperfect at times — a known limitation that the planned voice-to-voice comparison may eventually improve — but spelled-out digits are dramatically more reliable than handing the TTS a bare numeric string.

11.5 The dead-air no-accept bug. An early version of the screened transfer handed the post-no-accept close *back to the model* — after a target failed to accept, the model was supposed to speak a closing apology and end the call. On a live test, after a roughly thirty-second ring, the model **did not reliably produce that closing turn**: the caller heard dead silence, and the line never hung up until the caller did. For one routing lane the failure was total — it had no spoken fallback at all. The fix was to make the fallback **deterministic**: a single helper speaks a complete, graceful closing line to completion and then hangs up the call in code, with an explicit per-lane fallback matrix (an apology and saved-message confirmation for sales/accounting/general, a route to the on-call emergency agent for support, an email plus a spoken closer for a named-person request). The lesson is one of the load-bearing principles of the whole build: **terminal call-control must not depend on a generative model choosing to act.**

11.6 The silent email failure — an IAM lesson unit tests cannot catch. This is the most pedagogically valuable bug of the build. Outbound emails were **silently failing in production while the code and its unit tests were entirely correct**. The fulfillment layer, by deliberate design, catches a delivery failure, logs it, increments a failure metric, and *still* tells the caller their message was taken — reliability over surfacing, because the caller's experience should not hinge on a downstream hiccup. That correct, intentional behavior also *masked* the failure from a casual glance. The root cause was **infrastructure, not code**: Amazon SES authorizes a send against **both** the verified sending-domain identity **and** a separate configuration set (a named send-configuration object), and the host's permission policy granted send on the identity *only* — so every real send was denied, an access error that the reliability path caught and swallowed exactly as designed. It had gone unnoticed largely because most test traffic was *sales*, which routes to the CRM rather than to email, so the email path was rarely exercised. It was found by **reading the structured logs and the fulfillment failure metric** — the metric was firing — then reproduced by exercising the exact production send path live, and fixed by adding that configuration set to the IAM policy and into the infrastructure-as-code so it cannot regress. Two lessons fall out, and both are essential: **(1) a reliability path that swallows failures must be paired with metrics and logs loud enough to notice, or failures hide forever; and (2) unit tests that correctly mock the cloud cannot catch an authorization gap — only live verification and production telemetry can.** Neither substitutes for the other.

11.7 Testability as a first-class architectural goal. The cascade topology, the injected dialer behind the screened-transfer orchestration, the injected fulfillment callables, and the pure server-side directory resolver were all chosen in part **so the system could be tested without audio or network**. The decision logic, the first-to-accept orchestration, the category-to-backend routing, and the name resolution are all exercised with fakes and plain text, which is what keeps the suite well above the team's coverage bar. The irreducibly-live SIP and audio glue is kept thin and clearly delineated, marked as exercised only in live validation. The discipline: isolate the

deterministic logic behind interfaces so it can be tested exhaustively, and keep the genuinely-live glue small and explicit.

11.8 The "precious host" operational constraint. Because the self-hosted voice stack keeps live trunk and dispatch state on the host, the team adopted a strict rule: **roll the agent container in place; do not re-run the full infrastructure deploy**, which would replace the host and wipe that state. It is a standing reminder that real-time media's networking requirements push you toward stateful hosting, and stateful hosting demands deployment discipline — the operational cost that comes bundled with the capability the self-hosted media plane provides.

---

12. Complete Service Inventory

The complete service inventory for Chris, ordered from the real-time voice and AI plane through the supporting telephony, data, compute, security, and observability fabric.

Category	Service	Role
AI/ML — reasoning	OpenAI Realtime API (text mode)	Speech recognition, reasoning, and tool dispatch as text in / text out — explicitly not synthesizing voice.
Voice synthesis	ElevenLabs TTS (Flash v2.5)	Single consistent branded voice across greeting, conversation, hold audio, and the transfer whisper.
Turn detection	Silero VAD	Local, in-process acoustic voice-activity detector with a minimum-words interruption filter; replaces the model's server VAD.
Media server (SFU)	LiveKit (self-hosted)	Open-source SFU that rooms each caller and mixes media; the real-time media plane.
SIP service	LiveKit SIP (self-hosted)	Companion service bridging the carrier SIP trunk into LiveKit rooms; dispatch rule attaches the worker.
Carrier / SIP trunk	Twilio Elastic SIP Trunking	Delivers PSTN calls into LiveKit SIP (inbound) and carries outbound legs for screened transfers.
Coordination cache	Redis	Coordination state shared by the media server and SIP service.
CRM	Zoho CRM	Sales messages become leads via server-to-server OAuth, with a follow-up task; email fallback on failure.
Email	Amazon SES	Accounting, general, and named-person messages delivered as plain-text emails; sales email fallback.
Compute / orchestration	Amazon EC2 + Docker Compose	Single stateful host (host networking for the UDP/RTP range) running media, SIP, Redis, and the agent worker.

Category	Service	Role
Container registry	Amazon ECR	Holds the agent container image rolled in place on routine updates.
Secrets	AWS Secrets Manager	Runtime-injected credentials pulled at boot; no secret baked into the image.
Observability	Amazon CloudWatch (Logs + Metrics)	Structured per-event log lines and a separate metrics stream — the primary debugging surface.
Host access	AWS Systems Manager Session Manager	Host access with no public SSH port.
DNS	Amazon Route 53	Resolves the SIP hostname to the host's stable address.
IaC	AWS CDK	Declarative definition of the entire stack.

13. Outcomes & What the Design Demonstrates

Chris runs in production as the firm's front-of-house line, and it clears the bar that defined the project. It greets every caller in a consistent brand voice with a time-aware message, understands natural intent with no menu, answers basic questions about the firm, captures messages into the correct business systems, and performs **screened transfers that confirm a human before bridging and never abandon a caller in voicemail** — with a graceful, deterministic fallback on every path and a full audit trail behind every action. Because TrustEdge built it for itself first, the failures that only surface under real telephone traffic were found and engineered out on TrustEdge's own number, not a customer's.

More than a feature list, the build demonstrates a methodology for productionizing voice AI. A **single agent with finite tools** beats a fragmented multi-agent design for a bounded routing problem; the seams you add are the seams that fail. **Match the topology to the purpose**: the cascade bought brand-voice control, testability, and the turn-detection fix — the right trade for a front desk, where the voice is the product and the decision logic has to be testable software. **Make the model untrusted by construction** — fixed server-side targets, a finite action surface, and, most distinctively, an out-of-model directory that keeps contact data entirely out of the model's context. **Make guarantees deterministic** — screening, the voicemail guard, the no-accept closer, and the hang-up are code, not model choices. And **instrument loudly and verify live** — a swallowed-failure reliability path is only safe when paired with metrics and logs that make failures visible, and some failures, like an authorization gap, are only ever catchable in production.

For organizations in regulated sectors weighing conversational AI at their front door, the takeaway is clarifying: the model is the easy part. The durable value is

the deterministic engineering around it — the screening gate, the trust boundary around contact data, the validation and fulfillment routing, the observability, and the governance controls. A receptionist that impresses in a demo is a model talking; a receptionist that can be trusted on a real business line is deterministic code that happens to talk through a model.

One transfer target sits outside the firm's human staff: the on-call emergency agent. When a support or emergency call arrives after hours, or when no human can be reached during them, Chris escalates the live caller to TrustEdge's autonomous after-hours dispatcher — which has its own case study. From Chris's side, that dispatcher is simply one more server-owned transfer target, distinguished only by the fact that it bypasses the business-hours gate and is reachable around the clock; the escalation rides the same screened-transfer machinery as any other.

> *"We built Chris on our own front door as a research implementation, and that is the entire point — we don't sell things we've only ever demoed. We ran it on the line we answer every day, we found the bugs you only find on real calls — the agent inventing phone numbers, the emails silently failing, the transfer that dropped people into voicemail — and we fixed them until the only thing the model does is talk to the caller. What we productize is not the talking model. It's the screening gate, the trust boundary around people's contact details, the routing and the validation, the logging, the governance. That is the hard part, and that is the part that makes it safe to put in front of someone who just wants to reach a human." — Erik D. Jones, Lead Architect & CEO, TrustEdge AI*