PHASE 1 ENGINEERING CASE STUDY

Keepford: Putting an LLM on a Real Phone Line

Where the model talks and where code decides - a Phase 1 engineering case study

“Each time a real call exposed a model failure or a vendor limitation, the fix moved a decision or action out of the model and into deterministic application code.”

Amit Solanki · September 2026

1. Executive summary

Keepford is an AI phone receptionist for home-service businesses. Phase 1 built the voice receptionist: a Retell voice agent on a Twilio number, backed by an application built with Python, FastAPI and PostgreSQL. The model holds the conversation. Application code makes the decisions that matter - safety classification, service area, which times can be booked, whether a booking succeeded - and carries out the actions, including ending life-safety calls and running transfers through the phone system.

It runs on a live demo line that answers as a fictional plumbing company, and it was tested with real phone calls: of 23 acceptance scenarios, 20 passed, 3 could not be run, and none were failing at sign-off.

The most important result is a design rule that came out of those calls rather than being assumed up front: each time a real call exposed a model failure or a vendor limitation, the fix moved a decision or action out of the model and into deterministic application code. Two examples anchor this case study. Once a life-safety report is classified, the approved script is now spoken, and the call ended, by the phone system, not the model. And the model books a time by choosing a short reference instead of copying an identifier.

Companion recording - two real calls (1:46)

Two real calls to the demo line: a blocked-drain booking, then a burst-pipe transfer. Pauses are shortened; nothing is re-voiced or reordered. Caller details are fictional, and the transfer is answered by the demo's automated test line.

Provided alongside this document as an MP3.

CONTENTS

2. The engineering problem

A chatbot that gets something wrong produces a message the user can reread or retry. A voice agent acts in real time on a caller who cannot see what the system is doing, and some of its mistakes cannot be taken back.

- **Safety:** a caller who smells gas needs exact instructions and then needs to hang up and leave. A helpful follow-up question is the wrong answer.
- **Booking correctness:** "you're booked" is a commitment. If the appointment doesn't exist, nobody arrives.
- **Transfers:** these happen on the telephony leg, which the voice platform may not control.
- **Provider failures:** a calendar write can time out after it has succeeded.
- **Speech recognition:** the model reads a transcript, not the caller, and transcripts drop words and garble addresses.
- **Opaque identifiers:** LLMs are unreliable at copying long strings back exactly.
- **Truthful confirmation:** every "done" the caller hears must correspond to something that happened.

The central question is: **what should the model be allowed to decide?** Phase 1's answer: the model owns the conversation, and the application owns anything with consequences.

3. Architecture: where the LLM talks and where code decides

Twilio sends every call to the application first. The application registers the call with Retell and connects the audio, so it stays in the call path and can later redirect the live call itself.

> **The model** handles the conversation: understanding speech, keeping context, asking questions, speaking to the caller, and deciding when to call a tool.

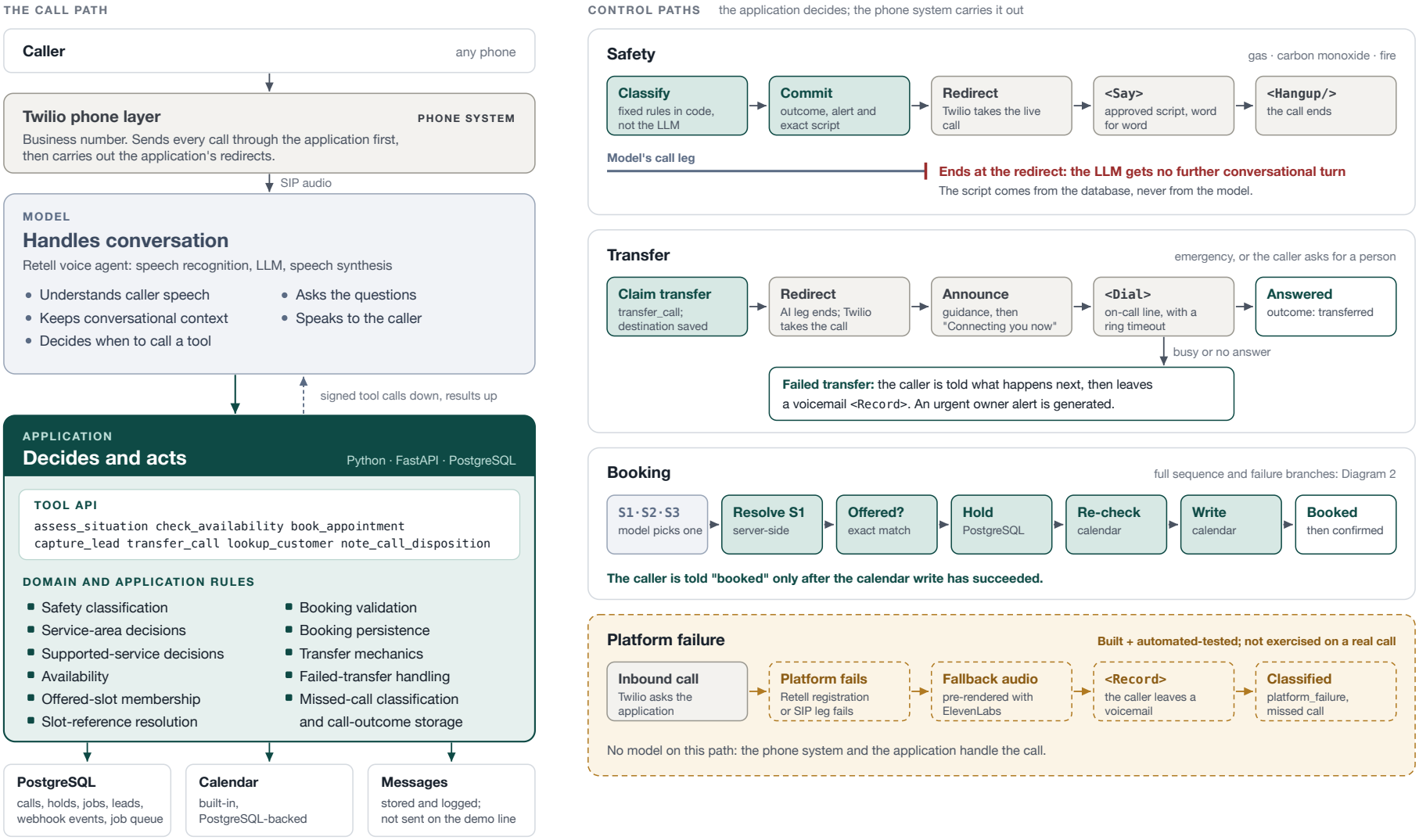
> **The application** decides and acts: safety classification, service-area and supported-service decisions, availability, slot references, booking validation and persistence, transfer mechanics, failure handling, and classifying and storing every call's outcome. The model reaches it through seven tools behind a signed endpoint, and each tool result tells the model what happened and what it may say.

Business details are configuration, not code; a second, deliberately different fictional business - test-only, not live - runs its own scenarios through the same test kit with no changes to the core.

This is not a claim that the model cannot make mistakes. It still decides whether and when to call a tool, and a voice platform cannot force a tool call. Some behaviours, such as how it ends a spam call, still depend on the prompt. What the split guarantees is narrower: once a decision reaches the application, the decision itself is made by code, not by the model.

Keepford Phase 1 - Where the LLM Talks and Where Code Decides

The model handles conversation; application code owns the decisions and side effects that matter.



Source: Keepford phase-1-final. Demo line answering as a fictional business: transfers reach an automated test line, and owner alerts and booking confirmations are generated and stored, not sent.

Figure 1 - Call control architecture at Phase 1

4. Engineering story #1 - Taking life-safety termination away from the model

Safety classification was fixed-rule from the first build, and by the time of the real-call tests it covered gas, carbon monoxide and fire. The model passes the caller's words to an `assess_situation` tool, which classifies them with fixed rules. Given the same recognised words, the classification is deterministic, and no business configuration can switch it off. What the first design still left to the model was delivery. The tool returned the approved script with instructions to read it word for word and end the call, and trusted the model to do both.

The gas-leak scenario (acceptance Row 11) failed twice:

1. **Attempt 1.** Speech recognition dropped the word "gas". The fixed rules correctly did not treat an unnamed smell as a gas leak, so the model asked a clarifying question. When the caller repeated themselves, the script was read correctly, but the model added speech of its own.
2. **Attempt 2.** The script was read verbatim, immediately. Then the model did not end the call.

Classification was not the problem, and the fix did not change it. The fix moved delivery and termination into the application. On a life-safety classification, the application saves the outcome, safety event, owner alert and exact script in one transaction. Only then does it ask Twilio to redirect the live call to a safety endpoint, which speaks the saved script with `<Say>` and ends the call with `<Hangup/>`. The model's call leg ends at the redirect, so it gets no further conversational turn, and the words come from the database, never from the model. If Twilio refuses the redirect, the model receives the script and the end-call instruction as before. That is a weaker guarantee, but the outcome and alert are already saved.

The trade-off is deliberate: the caller hears the phone system's voice, not the assistant's, in exchange for exact words and a hang-up that doesn't depend on the model.

4. ENGINEERING STORY #1 (CONTINUED)

Attempt 3 passed, and so did a fresh re-verification call on a later build. The carbon-monoxide scenario (Row 12) passed first time on the same design.

This is **deterministic termination after classification**, not a claim that every emergency will be detected. Two things upstream of the classifier remain limits: speech recognition can lose the trigger word, as in attempt 1, and the model has to pass the caller's words to the tool. Detection is also keyword-based, so unusual phrasing can be missed.

> **EVIDENCE:**
>
> - 118 collected safety red-team test cases, 8 of them documented expected misses.
> - 39 safety-termination test cases.
> - Two life-safety scenarios validated on real test calls: gas (Row 11) and carbon monoxide (Row 12).

Fire and smoke are implemented as a third category with automated tests, but fire was not tested on a real call. The scripts are approved defaults each business signs off; they are not safety advice, and Keepford is not an emergency service.

5. Engineering story #2 - Don't make an LLM copy opaque IDs

The first booking design exposed opaque slot IDs: 62-character strings encoding technician, service and start time. `check_availability` returned them, and the model had to copy the chosen one back.

On an early build (Row 1, attempt 1), the model sent back a corrupted copy three times. The application couldn't decode it and refused, but at that build it wrongly told the caller the time was unavailable. That failure led to two changes: a truthful message for an ID that doesn't decode, and a new rule that now anchors booking integrity - book only an ID that appears, by exact string match, in the list of slots this call was offered. Never decode an ID to decide whether it is bookable, because a mis-copied ID can decode to a different, perfectly valid time.

Row 8, attempt 3, produced exactly that, and was the first time the new check caught the model. The caller chose 7:30. Three times, the model sent the ID for 7:00, one character different. The membership check refused each one and nothing wrong was booked, but the conversation never recovered. Across both calls, the pattern was the same: the model repeatedly sent back slot IDs it had not been offered.

The second fix removed the copying. `check_availability` now labels each offered time with a per-call reference - `S1`, `S2`, `S3` - that is never reassigned. The model sends back only the reference. The application resolves it to the ID it stored for that call, then still applies the exact offered-slot check. An unknown reference such as `S99` fails closed as `slot_not_offered`.

This is more than a usability fix. The model now selects from a small constrained set, while the application keeps authority over what each selection means, and two independent checks stand between the model's output and a calendar write.

CHRONOLOGY

> **EVIDENCE:** 32 slot-integrity test cases. Rows 1, 8 and 22 passed on the current booking build (Row 8 booked in a single tool call). These are the only three real calls on this build.

Keepford Phase 1 - Booking Integrity

The model chooses a time by reference; the application decides whether it can be booked, and only then says "booked".

BEFORE earlier Phase 1 builds

The model had to copy a 62-character opaque slot ID back, character for character.

offered 7:30 `dGVjaF8xfGRyYWluX3VuYmxvY2t8MjAyNi0wOS0yMlQwNzozMDowMC0wNDowMA`

sent 7:00 `dGVjaF8xfGRyYWluX3VuYmxvY2t8MjAyNi0wOS0yMlQwNzowMDowMC0wNDowMA`

1 character different: a slot ID it was never offered

On real test calls the model repeatedly sent back slot IDs it had not been offered.
Row 1: a corrupted ID that could not be decoded, which led to an exact offered-slot check.
Row 8 (shown above): the first wrong ID that check caught. Nothing wrong was booked.

AFTER current booking build

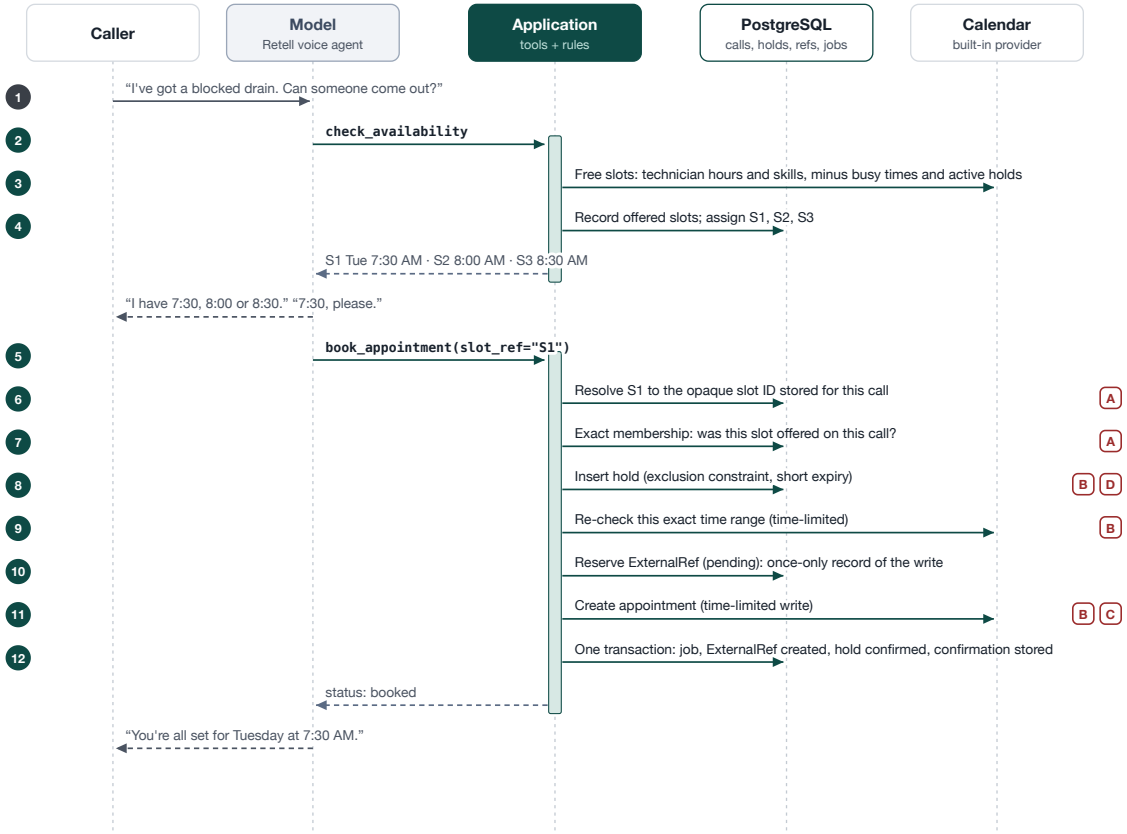
The model picks a short per-call reference; it never sees the internal slot ID.

S1 Tue 7:30 AM | S2 8:00 AM | S3 8:30 AM

model sends `book_appointment(slot_ref="S1")`

The application resolves S1 to the slot it stored for this call and checks the slot was actually offered on this call. Validated on 3 real test calls on this build.

SEQUENCE a caller books an appointment (success path)



FAILURE BRANCHES

A Unknown reference automated tests

The model sends a reference this call never offered, such as S99.

`S99 → rejected → slot_not_offered`

Nothing is booked; the model is given the times actually on offer.

B Slot no longer available automated tests

The hold conflicts, the re-check finds the time busy, or the calendar rejects the write.

`hold released → slot_unavailable`

Fresh alternatives are offered to the caller.

C Calendar write fails or times out automated tests

`hold released → pending_confirmation`

The caller is told the office will confirm the time. It is never reported as booked.

The ExternalRef stays pending and a lead is kept: a reconciliation record for the office.

D Concurrent booking

Request 1 for S1's slot
Request 2, same slot
→ PostgreSQL exclusion constraint

✓ **exactly one active hold succeeds**
✕ the other gets slot_unavailable

Concurrency behavior verified by automated race tests; real two-phone scenario was blocked.

Source: Keepford phase-1-final (check_availability, book_appointment). The success path was exercised on real test calls on the current booking build; branches A-D are verified by automated tests. Times and slot IDs are illustrative values from the fictional demo business. The confirmation is stored but not sent on the demo line.

Figure 2 - Booking integrity sequence

6. Booking integrity

After the offered-slot check, a booking takes these steps in order:

1. Take a hold in PostgreSQL. An exclusion constraint on each technician's time rejects an overlapping hold outright, and holds expire quickly.
2. Re-check the calendar for that exact range.
3. Reserve an `ExternalRef` record for the write.
4. Write the appointment.
5. Only then, in one transaction, store the job, confirm the hold and store the confirmation.

The caller hears "You're all set", followed by "We'll text a confirmation shortly", only after the write succeeds. If the write fails or times out, the hold is released and the result is `pending_confirmation`: the caller is told the office will text them to confirm the time, and the pending `ExternalRef` plus a retained lead become a reconciliation record for the office. On the demo line, neither text is actually sent (§13).

Proven by automated tests:

- Database-enforced concurrency protection, verified by automated race tests: 2 and 10 simultaneous bookings for one slot, and exactly one succeeds.
- Retried requests don't create a second appointment.
- A slot that becomes unavailable returns fresh alternatives.
- The `pending_confirmation` path.

Demonstrated on real calls: the success path only.

The real two-phone race (Row 7) was blocked for want of a second phone. Phase 1's only calendar provider is a built-in, PostgreSQL-backed one.

7. Application-controlled transfers

Calls arrive through Twilio and reach Retell over SIP, and Retell's built-in transfer could not act on that Twilio leg. So the application took over:

1. `transfer_call` claims the transfer and saves the destination.
2. Twilio redirects the live call.
3. The phone system speaks the business's emergency guidance, if any, then "Connecting you now".
4. It dials the on-call number with a ring timeout.

If the on-call number answers, the outcome is `transferred`. If it is busy or doesn't answer, the caller is told plainly what happens next and can leave a voicemail. The call is classified, and one urgent owner alert is generated, de-duplicated across every path that could create it.

A real call shaped the ordering. On the first in-hours burst-pipe attempt (Row 9), the model spoke the guidance and requested the transfer in the same turn, and the redirect cut the guidance off mid-sentence. The application now saves the guidance and the phone system speaks it before the announcement.

> **EVIDENCE:** answered transfers passed on real calls in Rows 9, 10 and 15. Busy and no-answer handling passed in Rows 13 and 14, with the line forced busy. On the demo line the on-call number is an automated test line, not a technician, and owner alerts are generated and stored, not sent.

8. Validation: how the system was tested

> **Layer 1 - contract and scenario tests.** Every scenario passes the tool-contract layer for both fictional businesses: Northside 27 of 27, Evergreen 6 of 6. These run without telephony. About 1,200 automated tests also run in CI, with linting, type checking and a business-data leak check. Ten are deliberate expected failures that document known limits. No coverage percentage has been measured.

> **Layer 2 - Retell simulation.** One scoped simulated-conversation run passed 8 of 8. It covered the scenarios a simulation without a phone leg can exercise, and used the real tool endpoints in the demo environment.

> **Layer 3 - real-phone acceptance.** 23 scenarios: 20 PASS, 3 BLOCKED, 0 FAIL at sign-off, over 39 real calls. There were 11 failed attempts before the final verdicts, each recorded with the build it ran on.

The failures are the most useful part of that record, because they changed the implementation:

A dropped "gas"	happens before the application sees the words; it is now a documented limit, not an unknown.
A model that didn't end a safety call	led to application-owned termination.
Wrong slot IDs	led first to the offered-slot check, then to per-call references.
Guidance cut off by a redirect	moved the guidance into the phone system.

The real calls were not a demonstration at the end. They were part of the engineering loop. Per-scenario results are in the appendix.

9. Performance

Measured on 39 real calls (84 turns):

MEDIAN:	90TH PERCENTILE:	SLOWEST BOOKING TOOL CALL:
about 1.1 s from the caller stopping to the agent starting to speak.	about 2.5 s.	931 ms.

These are point-in-time readings from the voice platform's own measurements, with one tester calling the US demo line internationally.

10. What is implemented vs planned

Live in Phase 1	Planned (not in Phase 1)
Live phone line and voice conversation (Twilio, Retell)	Google Calendar and GoHighLevel scheduling
Service, service-area and price-range handling	CRM sync and follow-ups, including missed-call text-back
Fixed-rule life-safety handling, application-owned termination	Invoicing and payments (QuickBooks, Stripe)
Application-controlled transfers with voicemail fallback	Automation flows (n8n, Zapier, Make.com)
Availability and booking with database holds	Vapi as a second voice platform; an MCP owner assistant
Call outcomes and missed-call reasons for every call	Sending SMS (messages are generated and stored today)
Platform-failure fallback audio (automated tests only)	

11. How it was built

I set the architecture and engineering rules, defined the acceptance criteria, made the design decisions and ran the real-call testing. The code was written by an AI coding agent working under those rules, with CI enforcing them.

12. What the project taught me

- **Don't let an LLM own decisions with irreversible consequences.** Keep the side effects in code.
- **Give models constrained references, not opaque identifiers.** Selecting is more reliable than copying, and the application keeps authority over what a selection means.
- **Treat vendor behaviour as an engineering constraint, not an assumption.** The transfer design exists because the platform's own transfer couldn't act on this call setup.
- **Real phone calls expose failures that simulations miss,** such as dropped words and redirect timing.
- **A truthful "we'll confirm" beats an optimistic "booked".**
- **Automated tests and real-call evidence answer different questions.** Race tests prove the database rule; only real calls show whether the model gets there.

13. Limitations and evidence boundaries

- **One business:** one fictional business is live.
- **Callers:** one tester, the developer, calling the US demo number from a non-US mobile. The evidence therefore does not establish behaviour across multiple callers, accents, background-noise conditions or US-originating callers.
- **Acceptance:** 20 of 23 scenarios passed; 3 were blocked (see appendix).
- **Current build:** only three real calls ran on it. Other passes come from earlier Phase 1 builds, some re-verified and some not re-run.
- **Concurrency:** no real concurrent-phone test.
- **SMS:** the demo does not send SMS. The confirmation texts callers are promised, and owner alerts, are generated and stored; sending them is planned, not implemented.
- **Fallback:** the platform-failure fallback hasn't been exercised on a real call.
- **Speech recognition:** it still changes words, including addresses and safety trigger words.
- **Infrastructure:** a single shared server without backups or rate limiting, not a hardened client production environment.

14. Closing

The interesting part of a voice AI system isn't getting an LLM to talk. It's deciding where the LLM stops having authority. In Keepford Phase 1, real calls drew and redrew that line: each failure moved one more decision or action out of the model and into deterministic code.

Appendix: real-call acceptance summary

20 PASS / 3 BLOCKED / 0 FAIL. 39 real calls from one tester, 19-22 September 2026.

What the evidence labels mean:

- **Early Phase 1 build:** the first acceptance builds.
- **Re-verified:** an earlier pass repeated with a fresh call on a later build.
- **Late Phase 1 build:** after booking hardening, before slot references.
- **Current booking build:** after slot references.
- **Judged unaffected:** the slot-reference change does not touch that row's path.

#	Scenario	Expected outcome	Result	Attempts	Evidence generation
1	Routine booking, new customer	Booked	PASS	3 (1 FAIL)	Current booking build
2	Existing customer booking	Booked	**BLOCKED**	0	Test setup would have overwritten a customer record from an earlier test call
3	Price question only	Details captured	PASS	2 (1 FAIL)	Late Phase 1 build; judged unaffected
4	Address in service area?	Callback details captured	PASS	1	Late Phase 1 build; judged unaffected
5	Outside service area	Out-of-area details captured	PASS	3 (1 FAIL regression)	Re-verified
6	Service not offered	Declined; details captured	PASS	3 (2 FAIL)	Early Phase 1 build; not re-verified
7	Slot taken by a competing caller	Booked after conflict	**BLOCKED**	0	No second phone for a simultaneous call; covered by automated race tests
8	After-hours booking by the AI	Booked	PASS	4 (3 FAIL)	Current booking build
9	Burst pipe in hours	Transferred, answered	PASS	3 (1 FAIL)	Re-verified
10	Burst pipe after hours	Transferred, answered	PASS	1	Early Phase 1 build; not re-run
11	Suspected gas leak	Safety script, call ends	PASS	4 (2 FAIL)	Re-verified
12	Carbon-monoxide symptoms	Safety script, call ends	PASS	1	Early Phase 1 build; not re-run
13	Transfer unanswered, voicemail left	Failed transfer, details captured	PASS	1	Early Phase 1 build; not re-run
14	Transfer unanswered, caller hangs up	Failed transfer, missed	PASS	1	Early Phase 1 build; not re-run
15	Caller demands a person, in hours	Transferred, answered	PASS	1	Early Phase 1 build; not re-run
16	Caller demands a person, after hours	Callback details captured	PASS	2	Re-verified
17	Hang-up during greeting	Abandoned early, missed	PASS	1	Early Phase 1 build; not re-run
18	Call drops mid-conversation	Dropped, missed	PASS	1	Early Phase 1 build; not re-run
19	Wrong number	Logged as wrong number	PASS	2	Re-verified
20	Sales / spam call	Logged as spam	PASS	2 (recorded deviations)	Re-verified
21	Withheld caller ID	Logged as withheld	**BLOCKED**	0	Carrier could not withhold caller ID
22	Fast talker gives address	Address read back; booked	PASS	2	Current booking build
23	Caller goes silent	Dropped, missed	PASS	1	Early Phase 1 build; not re-run. The platform did not end the call; the caller hung up