ENGINEERING RECORD · APR 20 – SEP 1, 2026

Five months, one operator, half a million lines

A complete technical account of an enterprise-grade AI platform — voice, messaging, billing, multi-tenancy, compliance — built and run in production by a single person directing Claude Code.

7,704 commits · 2 repositories · 150 database tables · Live paying customers

---

What this document is

This is an inventory and an honest assessment of a system that runs in production today: what it does, what it is made of, how it was built, and — the part that matters most to anyone evaluating the method — the engineering discipline that kept a codebase this size correct while one person moved through it at roughly fifty commits a day.

The product is a Slack-native and telephone-native AI employee: it answers a business's inbound calls as their receptionist, calls their inbound leads back within seconds, texts prospects, books appointments into their calendar, and reports through their dashboard. It has real customers and real telephone numbers, and it bills real money through Stripe.

> **NAMING**
>
> The platform was built under the working name *Mavrick*. That name is being retired for trademark reasons and a replacement is in evaluation. Nothing in the architecture

> depends on it; the name appears here only because it is what the code says.

Scale, measured

Every figure below was counted from the repositories and the production database at the time of writing, not estimated.

519,192
Lines of first-party code

7,704
Commits across both repos

2,199
Source files

~57
Commits per day, sustained

3,737
Backend tests, all passing

850
Test files across both repos

120
Database migrations

150
Production tables

Where the code lives

REPOSITORY / AREA	FILES	LINES	WHAT IT IS
Engine · `modal_agent`	327	152,583	Agent runtime, voice, tools, personas
Engine · `cron_workers`	130	60,981	Scheduled operations and watchdogs
Engine · `tests`	417	83,135	Trust Suite and unit tests
Engine · migrations	120	12,364	Schema, RLS policies, functions
App · `app`	716	129,889	Next.js routes, dashboard, marketing site
App · `lib`	356	61,626	Domain logic, vendor clients, auth
App · `components`	104	13,521	React UI

Velocity over five months

MONTH	ENGINE	APP	TOTAL	PHASE
Apr 2026	72	508	580	Foundation, two-repo split
May 2026	878	1,998	2,876	Product build-out
Jun 2026	696	533	1,229	Voice runtime, billing
Jul 2026	784	756	1,540	Retell cutover, receptionist
Aug 2026	827	649	1,476	Hardening, compliance, launch prep

What the product actually does

AI Receptionist — inbound

A business forwards its phone line. The agent answers as that business, by name, with their hours and their services; it opens with a one-breath AI-and-recording disclosure, takes a message, answers questions from a per-tenant knowledge base, books an appointment directly into the customer's calendar, and can warm-transfer to a human with a whisper when a caller asks for a person. After the call it writes an enriched lead into the customer's CRM, emails and texts the operator, and files a recording with a retention expiry.

Auto Caller — outbound speed-to-lead

A prospect fills in a form; the platform calls them back within about three minutes, with a redial ladder, attempt caps, quiet-hours enforcement, consent verdicts, and a national/workspace do-not-call screen in front of every dial. As of this week it also sends a first-touch SMS at the moment of the form fill.

Slack-native AI coworker

The original surface: an agent in Slack with a tool envelope, per-workspace memory, procedure induction, and the ability to build a live dashboard from a connected app — for example reading a HighLevel pipeline and rendering a shareable multi-tab board.

> DESIGN CONSTRAINT THAT SHAPED EVERYTHING
>
> Every feature is multi-tenant from the first row. The house rule is mechanical: *take the workspace from the row you are acting on, never from the environment.* A feature that resolves its tenant from an environment variable is single-tenant no matter how multi-tenant its tables are — and it compiles and passes tests either way, which is precisely why the rule had to be written down.

Architecture

Two repositories, one product

The split is deliberate. A **Next.js application** on Vercel owns everything a browser touches: the marketing site, the customer dashboard, authentication, Stripe checkout, and 219 API routes. A **Python engine** on Modal owns everything that runs without a browser: the agent runtime, the voice brain, 130 scheduled workers, and all vendor webhooks.

They share one Postgres database and communicate over authenticated HTTP — every call from the app to the engine carries a webhook secret compared in constant time. The boundary is what makes the system operable: a bad frontend deploy cannot take the phones down, and an engine deploy cannot break checkout.

Runtime topology

SURFACE	COUNT	NOTES
Modal functions	343	Serverless Python, per-function secret envelopes
Modal web endpoints	15	Capped at 8 per app — extras fold into kind-dispatch
Scheduled crons	4	Hard platform cap; 130 workers fold onto shared ticks
Next.js API routes	219	Vercel, production gated on its own branch
Database tables	150	`public` 105 · `voice` 36 · `marketing` 8 · `infra` 1

Two platform limits shaped real architecture. Modal caps scheduled functions at five per app and web endpoints at eight — a sixth cron fails the *entire* deploy. So 130 workers do not each get a schedule; they are folded onto a small number of shared ticks with per-worker error isolation, and extra endpoints became a kind-dispatch helper behind one route.

The voice path

Voice is the hardest part of the system and was rebuilt twice. The current path: a carrier delivers the call to **Retell**, which handles telephony, turn-taking, and speech; Retell opens a WebSocket to a Modal-hosted **brain** that runs the conversation, holds the persona, and executes tools — booking, message-taking, transfer, consent capture. Per-call agent overrides carry the tenant's own voice and business name without provisioning an agent per customer.

The earlier architecture was a self-hosted cascade — Deepgram for speech-to-text, an LLM turn, Cartesia or ElevenLabs for speech — running on LiveKit. It worked, but per-turn latency lived at roughly two seconds and the fleet needed warm workers. Moving to a speech-to-speech vendor removed an entire class of operational work.

The stack, in full

Every dependency and vendor below is genuinely in the system. Around 48 vendor credential sets are mounted across the engine's functions.

BACKEND RUNTIME

- Python 3.12 on **Modal** (serverless containers)
- **FastAPI** — webhooks, WebSockets, endpoints
- **Pydantic** — contracts and validation
- **Supabase** Python client (service-role writes)
- **boto3** — Cloudflare R2 object storage
- **cryptography** — RS256 JWT for Google service accounts
- BeautifulSoup + lxml, Pillow, ReportLab, openpyxl, PyYAML

FRONTEND

- **Next.js 15.5** (App Router) on Vercel
- React 18, **TypeScript 5**, Tailwind 3.4
- **Recharts** — dashboard visualisation
- **Sanity** — headless CMS for blog/content
- Vitest + Playwright, jsdom, Vite

DATA & STATE

- **Postgres** via Supabase — RLS on every tenant table

AI & MODELS

- **Anthropic** — primary agent reasoning, prompt caching
- **OpenAI** — realtime voice experiments, embeddings
- xAI, Google Gemini — evaluated cascade fallbacks

VOICE & TELEPHONY

- **Retell** — speech-to-speech runtime (current)
- **Telnyx** — SIP, DIDs, SMS, 10DLC
- Deepgram (STT), Cartesia + ElevenLabs (TTS) — prior cascade
- LiveKit — prior runtime, retired
- Sendblue — iMessage path

BUSINESS & GROWTH

- **Stripe** — subscriptions, credit grants on renewal
- **Slack** Bolt + Web API — the original product surface
- **Meta** Ads / Leads / CAPI — acquisition and attribution
- HighLevel — customer CRM and calendar of record

- **Supabase Vault** — credential encryption at rest
- **Upstash Redis** — idempotency claims, breakers, caches
- **Cloudflare R2** — recordings, artifacts, skills
- Pipedream — customer-owned integrations
- Resend — transactional email

OBSERVABILITY & ENRICHMENT

- **Sentry** — errors, both repos
- **PostHog** — product analytics, HogQL funnels
- Trestle — caller identity, spam and litigator screening
- Tomba, Firecrawl, Google Places, PageSpeed, Fal.ai, Pexels

The discipline — what actually made this possible

Half a million lines written this fast should be a disaster. The reason it is not is that the process was engineered before the code was. This is the part worth copying.

1 · A written constitution the agent must obey

The repository root holds a `CLAUDE.md` containing 28 numbered, non-negotiable rules — loaded into every single session. They are not style preferences; they are the invariants that were learned the expensive way. A sample:

- **Multi-tenant from row zero**, with an explicit acid test: *is it a control in the dashboard? Then it is multi-tenant, always.*
- **Idempotency at every external write** — Slack, Stripe, vendor APIs, all of it.
- **No module-level mutable state in the agent runtime**, because Modal runs ten concurrent inputs per container and a module global leaks across tenants.

- **Reversibility is non-negotiable** — every change ships behind a flag or documents a one-line revert.
- **Cost control is hard mechanism, never spend caps** — a spend limit does not remove waste, it shuts the business off mid-incident. Kill the wasteful unit; never the system.
- **Voice is code** — every customer-facing string passes a voice translator, a brand-envelope check, and a real-time truthfulness guard.

The value of writing them down is that an AI collaborator applies them consistently on turn four hundred, when a human's attention has long since drifted.

2 · Architecture Decision Records

Twenty ADRs record every load-bearing choice — the trust-layer model, the vendor-contract folder shape, the tool envelope's six status variants, message-persistence scope, per-repo migration number bands, and the decision to treat the migration ledger as truth. Each states Context, Decision, Consequences, and Reversibility. They exist so that a future session with no memory of the conversation understands *why* the codebase is shaped the way it is.

3 · The Trust Suite, and mutation-proving every guard

3,737 backend tests, of which 369 files are numbered "B-number" guards — 349 claimed to date, allocated from a shared ledger so two parallel sessions cannot collide. But the count is not the interesting part. The rule is:

> THE RULE THAT RAISED THE QUALITY BAR
>
> A guard is not finished until it has been **mutation-proven**: you deliberately reintroduce the bug, watch the test fail, restore the fix, and watch it pass. A test that passes on the broken code is worse than no test, because it reports protection it is not providing.

Several hard-won corollaries are now standing rules:

- **Assert on code, not on comments.** A grep-based guard once passed because it matched the docstring *warning about* the very call it was meant to forbid. Guards now parse the AST.
- **A guard must observe the failure.** If every input produces the same assertion, the test is about shape, not behaviour.
- **Mutation-proof is not correct.** One guard was fully mutation-proven and still codified a live data-loss bug — it asserted the wrong property, rigorously.
- **Existence is not execution.** Code being present says nothing about it running. A constant was once declared and never used; a job was assumed scheduled and was not.

4 · Multi-session orchestration

The work ran across several concurrent Claude Code sessions with distinct roles — a lead who owns merges and deploys, a second engineer, a frontend engineer, a product manager — coordinating through a shared, append-only *DEVSTATE* wall in the repository with a typed message grammar (`ASSIGNED · BLOCKED · SHIPPED · FACT · GOTCHA · OWNER`), a line-count gate, and a watcher that wakes the right session when a message is actually addressed to it. Sessions review each other's pull requests adversarially. One reviewer's catch on this project found a third code path in the *other* repository that the plan had missed entirely.

Isolation is enforced with git worktrees: every session builds in its own checkout so no two agents can trample a shared working tree — a lesson learned when a stray stash pop applied a 122-file change into a live tree.

5 · Verification culture

The standing rule is that nothing is "shipped" until it has been proven live, with vendor receipts rather than green checkmarks. In practice that means: grep the built tree for the new symbol *before* deploying, because a deploy from a stale checkout looks identical to a good one; poll the carrier's delivery receipt, because an HTTP 200 on

send is acceptance, not delivery; verify a config by *retrieving* it, because vendor list endpoints omit fields; and check a health endpoint that replays the actual failure in-container, because a warm serverless container keeps answering from the old image after a deploy — something observed live on this project, fifteen seconds apart.

Hard problems, and how they were solved

Cross-tenant leakage through container reuse

Modal runs multiple concurrent inputs per container, so a module-level cache is shared across tenants. Two such caches — spend and MCP config — leaked between customers. The fix was structural: a standing rule banning module-level mutable state in the runtime, with state moved to request-scoped context variables or Redis.

`Class eliminated, not patched.`

Row-level security that silently permitted everything

Two migrations shipped RLS policies containing tautologies — they looked like isolation and enforced nothing. Every migration creating a tenant table now requires a red-team verification block that assumes an authenticated role and proves zero cross-tenant rows come back, and the migration fails loudly if it does not.

`Verification moved inside the migration itself.`

A vendor deprecated a model name in silence

A production voice worker referenced an OpenAI realtime model that had been quietly retired, while the admin test rig referenced the current one. Production

failed at the vendor handshake for days while the rig kept reporting healthy. Now an AST test enforces model-name consistency across every code path in about 180 milliseconds, the deploy script rejects any known-deprecated name, and a 15-minute watchdog alerts if the handshake produces too little audio.

`Three independent detectors, one failure mode.`

A self-sustaining call loop that billed both ends

Five of the platform's own phone numbers sat in the marketing lake as dialable test rows. The dialer called them, the platform's own receptionist answered, its post-call handler wrote a fresh lead back into the lake, and the dialer called again. Both ends were AI, so nobody ever hung up and every call ran to the twenty-minute ceiling — 431 minutes in nine hours, burning telephony minutes and model tokens on both sides simultaneously.

Contained in minutes with a suppression list, then *replaced* by invariants: a self-dial gate screening against the number inventory rather than a hand-typed list, a database trigger refusing our own numbers at insert, a per-tenant attempt breaker, traffic classification so synthetic and customer calls can finally be told apart in a cost report, and no-progress supervision. The hand-typed suppression was then removed and the numbers re-tested to prove the invariants — not the patch — were holding.

`Root-caused, replaced, and the Band-Aid deliberately retired.`

The transcript that was an error path

Investigating those calls revealed a second failure hiding behind the first. Every agent line in the twenty-minute transcripts was the same apology, forty times over — it was the turn-error fallback. Every conversational turn was raising an exception, being swallowed to a log line, and apologising. The calls could not progress by construction, and nothing gave up. The lesson generalised into a

control: detect the *agent* repeating itself, because a real caller repeats themselves constantly and a working receptionist never says the same sentence three times in a row.

`Two failures, one symptom — found by reading, not guessing.`

A cost control that would have stopped the business

The first draft of the self-dial screen failed *closed*: if its lookup table were unreachable, it would refuse every customer dial. That is a cost control capable of shutting down the product, which the constitution explicitly forbids. It was caught by the full test suite and by CI independently, and corrected to fail open with a loud alarm — because unlike a regulatory gate, this one has no legal driver to justify an outage.

`The rule caught the rule-writer.`

Compliance and trust, built in rather than bolted on

Selling an AI that phones consumers means the regulatory surface is part of the product, not an afterthought.

- **Consent and DNC** — a gate battery in front of every outbound dial: self-dial screening, attempt breaker, national DNC, litigator list, workspace DNC, consent verdict, line type. Each gate is individually named and reported, and the battery short-circuits on first failure.
- **Quiet hours** — FCC §64.1200 and 16 CFR §310.4(c) windows enforced in the called party's local time, at the send boundary, for both voice and SMS.
- **Recording disclosure** — every call opens with a one-breath AI-and-recording notice before anything else.

- **Data minimisation** — the platform never reads Slack channel history outside an actively running user-triggered task. No background scanning, no ingestion on install. Message bodies are persisted only for conversations the agent was a direct party to.
- **Credentials** — encrypted through Supabase Vault; the master key lives in the vault, never in code or environment.
- **Retention** — recordings carry an expiry stamped at archive time, so retention is a behaviour rather than a slide.
- **GDPR purge workers** and an audit-chain signer run on schedule.

What this demonstrates

The interesting claim is not "an AI wrote a lot of code." It is that a single operator ran a production software organisation — architecture, review, testing, deployment, incident response, and compliance — at a velocity and quality bar that normally needs a team, by building the process that makes an AI collaborator reliable.

CAPABILITY	EVIDENCE IN THIS PROJECT
Systems architecture	Two-repo split with an authenticated boundary; 150-table multi-tenant schema; deliberate work-around of hard platform limits
Distributed systems	Idempotency at every external write; Redis claims and circuit breakers; fail-open vs fail-closed reasoned per gate
Real-time voice	Two full runtime migrations; WebSocket conversation brain; per-call vendor overrides; latency and cold-start work
Test engineering	3,737 tests; mutation-proving as policy; AST-based guards after grep guards were proven unsound
Incident response	Root-cause to invariant, repeatedly — contain in minutes, then replace the patch and prove the replacement holds

CAPABILITY	EVIDENCE IN THIS PROJECT
Regulatory engineering	TCPA-shaped consent, DNC, and quiet-hours enforcement at the send boundary
AI collaboration method	A written constitution, ADRs, typed inter-agent protocol, worktree isolation, and adversarial cross-session review

> **THE HONEST SUMMARY**
>
> Working this way is not "the AI builds it while you watch." The leverage comes entirely from the scaffolding — the rules, the records, the guards, the verification habits — and from an operator who refuses to accept a green checkmark as evidence. Remove the scaffolding and this becomes 500,000 lines nobody can safely change.

Compiled from the live repositories and production database, September 1, 2026. All figures counted, not estimated. The platform is in production with paying customers.