Latest verified update — 14 October 2026

Evidence film completed; earlier technical record retained

The project now includes a completed 1 minute 51 second evidence film prepared for LinkedIn and YouTube. Clean and captioned masters, subtitles, transcript, thumbnail and supporting verification records were delivered locally. This records preparation and delivery; no social upload or publication is claimed.

The film shows the actual test-case workflow: scenario → target response → deterministic checks → inspectable evidence → regression comparison. Its recorded audit contains 25 test cases: 3 PASS, 22 FAIL and 0 ERROR against a deliberately weak scripted target. These are findings about that test target, not a clinical accuracy score. Optional semantic AI judging is explained separately; it was not used in this recorded audit.

Both masters use 1920 × 1080 video at 30 frames per second with H.264 video and AAC stereo audio. The captioned master has 24 caption cues. Full-file decoding, frame and caption checks, measured audio and initial browser playback passed. The final independent artifact review approved both actual files after checking codecs, dimensions, duration, byte sizes and matching SHA-256 records. Human listening and actual editor import remain unperformed; initial browser playback does not establish complete real-time playback.

The film session also recorded 190 passing Python tests, strict typing across 86 files, passing lint and security checks, and a recheck of all four existing public-source CI jobs. Those are dated engineering results. No application code, deployment, payment capability or human-review status changed in that session. Clinical, legal and human calibration gates remain OPEN; the no-PHI architecture and test-data-only scope remain in force.

Reading this rolling record: this newest update takes precedence for the film milestone. The dated engineering update and full earlier technical record follow unchanged in content. Their earlier counts and status statements remain historical evidence. Layout has been compacted to satisfy the requested 10–15 page export without deleting previous material.

VitalProbe — Engineering and Product Overview

Latest publication update: 12 October 2026

Healthcare AI safety testing with test scenarios and inspectable evidence

This update is the current engineering reference. The earlier technical record is preserved in full after this update for continuity. Earlier inventory counts and release statements describe their original dated state; the current evidence and remaining gates are recorded below. No previous section has been deleted or replaced.

1. Product and current delivery

VitalProbe tests healthcare and wellness chatbots, retrieval applications and AI agents against test safety scenarios. It records observed responses, deterministic checks, optional semantic judgments and a structured HTML/JSON evidence dossier. Engineers use this evidence to investigate behavior and compare revisions; qualified reviewers interpret clinical and legal implications.

The public repository contains both the React/TypeScript frontend and the Python backend. The website provides Home, Product, Evidence, Pricing, Docs and Trust pages with direct routes, responsive layouts, navigation, evidence filtering and self-hosted decorative media. New audits execute locally or in an operator-controlled environment. The website has no hosted audit endpoint, customer accounts, uploads or payment collection.

Website: https://vitalprobe.zahidul-islam.com
Public source: https://github.com/Zahidulislam2222/vitalprobe
Documentation index: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/README.md

2. Published source and verification evidence

The documentation publication is commit 756d346d60c055be682e8b9ca4ee0abccc53c8ff, attributed to Zahidul Islam and the verified GitHub account Zahidulislam2222. One normal push published 39 changed files. GitHub's native source tree matches all 197 local public blobs and includes both applications; private recovery material and separately licensed content are excluded.

Exact-commit CI: https://github.com/Zahidulislam2222/vitalprobe/actions/runs/38174719830
All four jobs succeeded: Python test/package verification, published-file secret scan, source/dependency security audit, and frontend checks/build. Local verification passed 190 Python tests, strict typing across 86 files, 39 frontend tests, Python/TypeScript lint and types, dependency checks, security scans, package validation and builds. The unchanged edit-time security hook passed. The current-public snapshot had no secret findings; the history scanner's single finding was verified as a public verification key, not private signing material.

An independent fresh-context review approved the corrected public diff. It checked 139 relative documentation links and independently exercised the eight CI artifact-gate cases. A retained local verifier passed 226 checks across

documentation, configuration, real HTTP and CLI behavior. These results establish the stated engineering checks, not clinical validation, historical uptime or million-user capacity.
The deployed runtime remains source ef1d3d296245eac255050da7e933bbac77fe6f8a. Documentation-only publication did not replace that artifact. All 23 public file hashes were freshly read over HTTPS and matched the local active release; the rebuilt frontend matched the same manifest.

3. Architecture and trust boundaries

The website serves compiled static assets and bundled sample evidence. The backend validates a suite and adapter, sends test turns to an authorized REST or MCP target, executes bounded checks and optional judging, redacts supported secrets, and writes canonical local artifacts. Optional tracking stores run history in local SQLite. There is no website-to-engine execution connection.
CLI, local dashboard and product MCP tools reuse the same engine and contracts. The dashboard is an operator interface bound to loopback, without a general public login or tenant-authorization layer. It must not be exposed as a hosted application. MCP target sessions have an owned lifecycle; the separate product MCP server is entitlement-gated. The LangGraph integration reuses the test-patient simulator rather than defining a second audit engine.
Python uses Typer, Pydantic, httpx, Jinja2, SQLModel and optional FastAPI/Uvicorn surfaces. The frontend uses React, TypeScript, Vite, Motion, Lenis and validated maintained JSON content. Python requires 3.12 or newer; the frontend uses the Node 24 toolchain and committed lockfile.
Architecture: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/architecture.md

4. Interfaces, setup and configuration

The source-install quickstart covers a local scripted target, suite validation, a 25-case audit, JSON/HTML inspection, baseline saving, comparison and tracked history. Docker Compose provides a local loopback workspace. Public package-index publication remains deferred; the Python package version is 0.1.0.dev0.
Backend commands include validate, run, compare, history, license, simulate, calibrate, learn, dashboard and mcp-serve. REST adapters own method, endpoint environment name, authentication headers, request templates, response paths, timeout and retry policy. MCP adapters own transport, trusted command or endpoint, tool and message mapping. An unset header variable can omit a header; operators must verify target authentication.
Typed Python settings own provider URLs, models, API versions, timeouts, budgets and service flags. Safe environment templates and the complete configuration reference inventory supported names. Secrets stay in the ignored environment or an approved secret store. Frontend build configuration is public and never accepts provider credentials. Maintained website content and proposed pricing have data-file owners; prompts and rubrics remain maintained package data.
Cloud judges are explicitly opt-in and require configured model/prices and cost approval. A zero price placeholder does not establish free usage. Local deterministic evaluation needs no paid provider. Model-dependent simulator/generator workflows require an installed configured local model and, where applicable, an offline entitlement.
Quickstart: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/quickstart.md
Backend reference: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/backend.md
Configuration: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/configuration.md

5. Evidence, regression policy and quality gates

The canonical run records its version, identity, timestamps, redacted target, suite, aggregate verdict, counts and per-case inputs/responses/checks/judge evidence. Reports preserve FAIL, WARN and ERROR rather than reducing them to an unqualified score. The deliberately weak scripted target produced 3 PASS, 22 FAIL and 0 ERROR across 25 test scenarios. Those counts describe that target only.
Comparison identifies shared PASS-to-WARN/FAIL regressions and reports improvements and new/removed cases. Comparison alone does not reject target ERROR or every possible adverse transition. The corrected CI guide separately rejects missing/malformed artifacts, empty/incomplete case evidence, duplicate IDs, inconsistent recomputed counts, inconsistent aggregate verdicts and target errors. Eight committed regressions execute the published gate against valid and unusable evidence.
Repository CI runs locked installation, Ruff, strict mypy, pytest, Bandit, Gitleaks, Semgrep, dependency audits, wheel/source builds, Twine and an isolated freshly resolved wheel smoke test, plus frontend types/lint/tests/audit/build. Clinical review, legal review and human calibration remain separate gates. Draft content must fail release validation; generated prelabels are not human validation.
Testing and CI: https://github.com/Zahidulislam2222/vitalprobe/blob/main/TESTING.md

6. Frontend experience and accessibility

The hero video starts automatically when permitted, muted, looping and inline, with a static poster fallback. Motion and Lenis provide entrances, reveals, transitions and scrolling. Initial and changing operating-system reduced-motion

preferences stop playback and spatial/continuous animation. Motion/playback buttons were removed at the founder's request.
All six pages were exercised with configured accessibility scans at four widths, alongside keyboard navigation, evidence filtering, clipboard feedback, mobile navigation and motion/autoplay behavior. These checks are bounded evidence, not a statement of complete assistive-technology or legal conformance. Fonts, generated artwork and video are self-hosted; provenance is documented separately. Decorative media has no clinical evidentiary meaning.

7. Security architecture and reporting

The current exposure is limited by static hosting, test inputs, local execution, explicit provider egress and offline entitlements. Controls include validated contracts, safe YAML loading, bounded concurrency/timeouts/retries, templated output, token redaction, signature verification, lockfiles, pinned CI actions and layered scans. Imported suites, responses and generated drafts remain untrusted content.
Local tooling is not a hosted SSRF firewall or a tenant-isolation system. Secret redaction is not comprehensive personal-data detection. Protect target/provider credentials, private signing material, report integrity, baseline provenance and licensed content. Revoking a leaked credential matters more than deleting a source line. Offline license cancellation stops renewal but cannot instantly revoke an already delivered signed file; verification-key rotation requires older installations to upgrade.
GitHub private vulnerability reporting is enabled. The public security policy defines the private channel, supported current branch and reporting boundaries. Do not put secrets, real patient data or exploit details into public issues. A future hosted service requires explicit tenant/object authorization, bounded requests, abuse controls, safe egress, CSRF protections where relevant, encrypted storage, retention/deletion and security-negative tests against applicable OWASP ASVS/API controls.
Security policy: https://github.com/Zahidulislam2222/vitalprobe/blob/main/SECURITY.md

8. Privacy, law and responsible AI

Not legal advice. This documentation was drafted with AI assistance and has not been reviewed by an attorney. VitalProbe's audit architecture processes test patients only. It does not diagnose, prescribe, provide patient care or replace clinical judgment. Reports are not a regulatory approval. A case pass is evidence about an observed response, not a guarantee of behavior outside the tested configuration.
Do not import real patient records or production conversations. Removing names from real records does not make them test data. Target-generated responses can themselves introduce sensitive data and need inspection. Operators own local reports/history/baselines, retention, deletion, backups, target authorization and any approved provider egress. The application adds no visitor analytics, tracking cookies or browser storage; infrastructure providers may still handle operational network metadata.
Legal applicability depends on the actual parties, activities, data and jurisdictions. The public legal guide identifies review areas for HIPAA, GDPR/personal data, clinical software intended use, AI governance, consumer terms, intellectual property and accessibility, with HHS, EDPB, FDA and NIST references. Future account, contact or payment metadata can be personal data even when audit scenarios remain test data.
Clinical content, localization, counsel review and 50 qualified human calibration labels remain open. Before accounts, billing, hosted storage or real-data handling, complete the applicable scope/legal, contract, residency, rights, incident, security and operational prerequisites. The local payment test lane does not close those gates. Keep rubric/model/content provenance, review severe failures and judge disagreement, and retain methodology limitations in every report.
Legal responsibilities: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/legal.md
Privacy: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/privacy.md

9. Packaging, pricing and payment status

The public repository is Apache-2.0. Separately supplied licensed content has separate terms and is excluded from public distribution. Offline entitlements can unlock defined content/features; the lasting commercial value is maintained material and qualified services rather than an unbreakable open-source client check.
The maintained proposed monthly offers are Automated at USD 50 and Clinician-Reviewed at USD 70. They are informational and public checkout is unavailable. The latter name does not mean current draft content has received clinical review. Workload allowances, taxes, cancellation/refunds, support, review capacity/scope and turnaround must be agreed before sale.
The isolated Stripe lane is disabled by default, localhost-only, test-data-only and test-mode-only. Four private settings are still required: test secret key, webhook secret and two recurring test Price IDs. Mocked lifecycle tests pass; real Stripe end-to-end acceptance is not complete. The matrix covers checkout, decline, 3DS, duplicate/reordered events, renewals/recovery, cancellation, refunds/disputes, outage/replay and live-mode rejection. No live payment or customer entitlement is claimed.

10. Future scale objective: 1M+ concurrent users

This is an engineering roadmap objective, not a measured capacity result. Separate public visitors, authenticated sessions and audit jobs before sizing. The present website is static; a future approved control plane would be stateless and tenant-authorized, coordinating customer-controlled execution rather than silently collecting responses.
Example workload: one million active visitors requesting a page every 60 seconds implies about 16,667 page requests/second before assets. An example 2 MB cold transfer implies about 33.3 GB/second, or 267 Gbps; use actual asset/video measurements before choosing infrastructure. One million sessions making two metadata requests/minute implies about 33,333 API requests/second. If 1% submit one audit/minute, admission is about 167 jobs/second. At 25 target calls/job and a mean 60-second job duration, that implies about 4,167 target calls/second and 10,000 jobs in flight.
The target design uses CDN/versioned static delivery, authenticated API routing, stateless replicas, partition-ready metadata, durable bounded queues, fair tenant scheduling and customer-controlled workers. Requirements include idempotent admission, leases/heartbeats, capped retries with jitter, cancellation/deadlines, dead-letter recovery, quotas, independent provider/target rate limits and queue-age autoscaling. A local SQLite file is not a multi-tenant million-user database design.
Capacity follows measured safe throughput, arrival rate and mean duration, with failure/burst headroom. Keep artifacts and credentials customer-controlled by default; any later hosted artifact option needs approved encryption, tenancy, retention, deletion and recovery. Cost modeling must include egress, storage, queue/database operations, model tokens, observability, redundancy and worst-case retries. No infrastructure or provider cost is committed by this plan.

11. Staged capacity validation and roadmap

Progress from measured single-service baselines through 1k, 10k, 100k and 1M+ representative sessions. Archive workloads, scripts, thresholds, resource curves, cold/warm cache behavior, provider saturation, tenant-isolation checks, rolling deploys and lost-worker/zone behavior. Prove that distributed load generators are not the bottleneck.
Proposed hosted metadata gates are p95 at or below 300 ms, p99 at or below one second, job-admission acknowledgment p95 at or below one second, and server-error ratio at or below 0.5% under the agreed workload. Report audit completion separately by case count, target and model. Use a two-hour endurance gate followed by a 24-hour soak before validating a scale milestone. These remain proposed acceptance thresholds.
Dependencies are content/legal/human review, measured operational baseline, commercial/payment acceptance, approved hosted coordination, staged load evidence and higher reliability where justified. A phase changes status only when its evidence exists. Large load tests and paid infrastructure require separate target authorization and an exact costed approval; this publication performed neither.
Scale design: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/scalability.md
Roadmap: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/roadmap.md

12. Reliability objective: 99% availability

The initial website objective is at least 99.0% successful eligible requests over a rolling 30-day window. Eligible requests are legitimate GET/HEAD requests for documented pages or required assets, excluding documented abuse and unknown routes. Success requires the expected response and required content within five seconds; user-impacting DNS/TLS/edge/origin failures count. Planned maintenance consumes the same budget.
The request error budget is 1%. For a separate time-based indicator, 1% of 30 days equals 7 hours 12 minutes; 99.9% and 99.95% correspond to 43 minutes 12 seconds and 21 minutes 36 seconds. Request and time indicators have different denominators and must not be conflated. No historical 99% result or contractual SLA is asserted.
Planned measurement uses independent external locations, one-minute DNS/TLS/content probes, periodic browser journeys and aggregate request/latency data. Report dates, denominators, coverage and missing samples; missing monitoring is unknown, not success. Assign an owner and validate alert cadence before declaring an objective active. Freeze nonessential rollouts when the budget is exhausted and prioritize restoration.
Proposed recovery targets are 30 minutes for the static website with no lost published source revision, 60 minutes/15 minutes for future hosted metadata, and 30 minutes with no lost acknowledged admission for a future durable queue/ledger. Operator-local report/history recovery targets remain operator-defined. All targets require actual backup, restore and reconciliation evidence; source preservation does not imply zero downtime.
Reliability: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/reliability.md

13. Release, operations and recovery

Local source is authoritative. Inventory the actual host, compare live artifacts against the last verified manifest, reconcile drift, build/test locally, package an immutable release, verify its public allowlist, upload, validate serving configuration, smoke-check and switch routing. End with every-file local/live hashes and real public flows. Retain a verified rollback artifact and independent archive.
Restore a backup into a separate destination, check hashes/configuration and actually start it before calling recovery verified. Keep private server identities and privileged recovery details out of public documentation. Stop SQLite writers

or use its supported backup mechanism; inspect restored integrity and representative runs before replacement. Do not overwrite the last working backup or PDF until its replacement has been validated.
For an incident, record scope/time/revision, assign an owner, contain impact, restore a verified state, confirm recovery, rotate affected credentials and record root cause, corrective proof and the missed gate. Legal notification duties depend on actual data/jurisdiction and require counsel. The public operations guide includes troubleshooting for target errors, authentication, timeouts, local models, history, entitlements, route/motion behavior and payment prerequisites.
Deployment and operations: https://github.com/Zahidulislam2222/vitalprobe/blob/main/docs/deployment.md

14. Public maintenance and remaining gates

All four existing component READMEs were updated. The indexed reference covers product, architecture, backend interfaces, complete configuration, setup, authoring, tests/CI, deployment/rollback, operations/recovery, security, privacy/law/responsible AI, tiers, support/contribution, release status, changelog, monitoring proposal, scalability and reliability. Issue/PR templates request test-data reproductions and keep sensitive reports private. New policy/changelog files are included in source distributions.
This document keeps the same Google Doc identity and preserves the earlier record. The synchronized PDF is exported from this same document, downloaded to a temporary file, validated and rendered, then replaces the canonical PDF. Private checkpoints record the source commit, tests, decisions and next steps so another agent can resume after interruption.
Outstanding gates remain named qualified clinical/counsel/localization review, 50 human labels, real Stripe test acceptance, commercial policies/approval, measured reliability and staged scale validation. No human approval, production payment, new paid model call or hosted data service was introduced by this documentation publication.

Previous technical record — preserved

The following material is retained verbatim as the earlier dated reference. Its inventory counts and completion statements are historical; the latest update above governs the current state.

VitalProbe | Healthcare AI Safety Testing | Technical Reference

VitalProbe — Comprehensive Technical Overview

Local-First Healthcare AI Safety Testing and Evidence

A client-readable and engineer-ready reference for test suites, target adapters, deterministic checks, semantic judges, evidence reports, local history, learning drafts, simulation, dashboard operation, and human review boundaries.

Python | Typer | Pydantic | httpx | Jinja2 | SQLModel | FastAPI | Ollama | MCP | Docker

Completion-state public technical edition · Reconciled 21 July 2026

---

1. Executive Overview

VitalProbe is a local-first safety and evidence-testing product for healthcare and wellness AI assistants. It runs chatbots, retrieval applications, and AI agents through test patient scenarios, evaluates the observed responses, and produces reviewable JSON and self-contained HTML evidence showing what was tested, what passed, what failed, and why each verdict was assigned.
The completed local product includes a typed test contract, public suite loading, REST and MCP target adapters, deterministic checks, optional semantic judges, concurrent execution with bounded retries, redacted reports, baseline comparison, local run history, learning-draft generation, offline feature entitlements, informational law mapping, multi-turn test-patient simulation, a LangGraph example, a localhost dashboard, Docker packaging, and a licensed developer-facing MCP server.
VitalProbe is intentionally built around a no-PHI architecture. It processes test patients only. It does not require production conversations, real patient records, hosted accounts, server-side customer-response storage, telemetry, analytics, or silent third-party transmission. This boundary is enforced in project rules and tests because it is part of the product design.
Reports describe observed results and limitations. VitalProbe does not certify a target, grant regulatory approval, replace clinical judgment, or provide legal advice. Clinical and legal interpretation remains with qualified humans.

Current implementation snapshot

- Package version: 0.1.0.dev0 pre-release source distribution.
- Runtime: Python 3.12 or newer.
- Public source inventory: 55 Python modules, approximately 4,836 source lines, 74 classes, and 187 functions.
- Test inventory: 21 top-level test modules and approximately 3,169 test lines.
- Current quality evidence: 149 tests passed, Ruff clean, and strict mypy clean.
- Product surfaces: CLI, local dashboard, Docker quick start, JSON/HTML evidence, REST/MCP target testing, licensed MCP tools, and LangGraph example.
- Default operating cost: local test workflows can run without paid provider calls.

- License: Apache-2.0 for the public code repository.

2. Product Purpose

Healthcare-oriented assistants can change behavior when prompts, models, code, retrieval corpora, safety policies, or provider versions change. A response that previously escalated chest pain or crisis language may later become vague, dismissive, overly reassuring, or unsafe. Ordinary application tests rarely express these conversational safety expectations clearly.

VitalProbe converts those expectations into repeatable test scenarios and explicit evidence. A builder can test an assistant before showing it to users, compare a new release with a baseline, inspect individual checks, and share a self-contained dossier with a reviewer.

Primary outcomes

- Reproduce important safety scenarios without real patient data.
- Distinguish product findings from network and configuration errors.
- Combine deterministic checks with optional semantic scoring.
- Preserve exact test inputs, observed outputs, evidence, latency, and limitations.
- Detect PASS-to-WARN or PASS-to-FAIL regressions between runs.
- Track run history locally without accounts or hosted storage.
- Generate adversarial draft variants from failures while retaining a human review gate.
- Test both single-turn and multi-turn assistant behavior.
- Integrate through CLI, dashboard, REST, MCP, Docker, and agent-oriented tooling.

3. Product Boundary and No-PHI Architecture

VitalProbe processes test patients. The default product does not ingest production conversations, electronic health records, account data, or customer support transcripts. Test suites are authored as test scenarios and target responses are stored only in the user-controlled local evidence location when tracking is enabled.

The local dashboard binds to loopback by default. It has no user accounts, external assets, telemetry, analytics, crash reporting, or hosted backend. Docker publishes the dashboard only to localhost in the default local configuration.

Third-party judge providers remain opt-in. Local Ollama is the default semantic route. Cloud judging requires explicit configuration and a fresh founder-controlled money decision; nothing transmits silently.

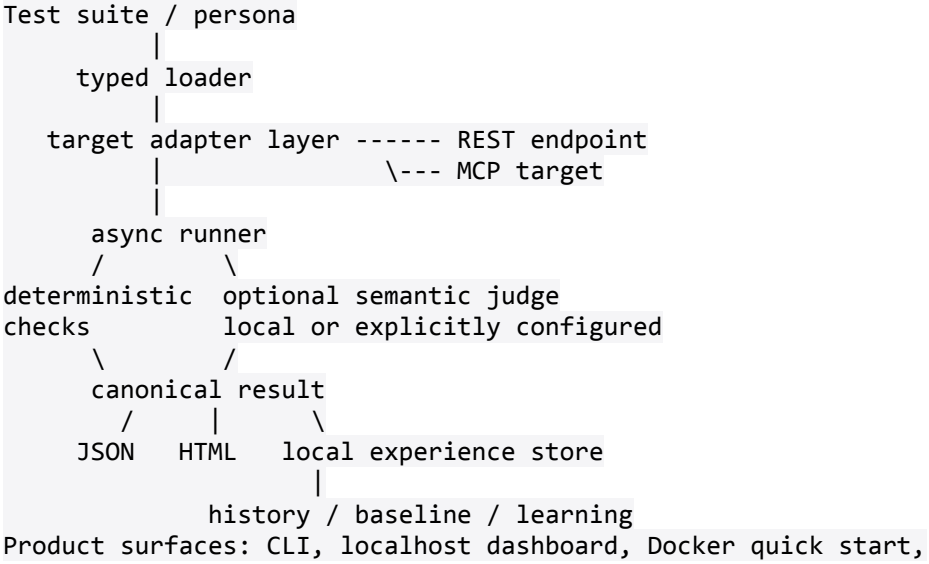
Scope boundaries

- No real patient data.
- No hosted customer accounts.
- No server-side customer response store.
- No default provider transmission.
- No telemetry or phone-home behavior.
- No hidden browser analytics or external dashboard assets.
- No automatic clinical or legal approval.

Any future move beyond these boundaries requires a separate product, privacy, security, and legal design review before implementation.

4. Architecture

VitalProbe follows an engine-first architecture. The CLI, dashboard, MCP tools, Docker quick start, and examples are thin product surfaces over the same core contracts and services.



```
licensed MCP server, LangGraph example.
```

Core modules

```
| Layer | Responsibility |
| Contracts | Typed test cases, responses, checks, verdicts, and run artifacts |
| Packs | Suite discovery, YAML loading, schema validation, draft/review status |
| Targets | Shared adapter protocol with REST and MCP implementations |
| Evaluators | Deterministic response and trajectory checks |
| Judges | Semantic judge protocol with local and opt-in provider backends |
| Runner | Concurrency, retries, target calls, checks, aggregation, timing |
| Reporting | Canonical JSON and self-contained HTML dossier |
| Experience | Local runs, baselines, regressions, labels, and learning inputs |
| Entitlements | Offline signed-license verification and feature gates |
| Simulator | Multi-turn test patient generation and trajectory evaluation |
| Dashboard | Local clinical-ledger views and trigger-run workflow |
| MCP | Target adapter and licensed product-side agent tools |
```

Dependencies flow toward the core. New product behavior belongs in the engine rather than being duplicated in a UI or command shell.

5. Test Suite Contract

A VitalProbe suite is a directory of typed YAML metadata and test cases. Each case has a stable identifier, title, category, severity, review status, supported languages, one or more test user turns, required checks, warning checks, optional semantic rubric, informational law references, and reviewer notes.

```
case
  id, title, category, severity, status
  languages
  test input turns
  expected behavior category
  must checks
  warning checks
  optional semantic rubric
  optional informational references
```

Draft status is meaningful. AI-generated or AI-edited clinical scenarios remain draft until a named qualified reviewer completes the human gate. VitalProbe can run draft content locally for development, but release validation refuses unreviewed cases.

Loader protections

- Schema validation with actionable configuration errors.
- Unique identifiers inside each suite.
- Typed severities, roles, verdicts, and check specifications.
- Explicit adapter configuration.
- Draft-versus-reviewed enforcement for release workflows.
- Unknown references fail loudly instead of disappearing from reports.
- Private clinical content remains outside the public repository.

6. Target Adapters

The runner depends on one target protocol, allowing the same suite and evidence contract to test different assistant surfaces.

REST adapter

The REST adapter sends test messages to a configured endpoint, supports optional revocable staging authorization, records latency and status, retries only within configured limits, and redacts credentials from every output surface. Adapter configuration owns endpoint shape, message field, response field, headers, timeout, and retry behavior. Model or provider details are not hardcoded in core control flow.

MCP target adapter

The MCP adapter tests an MCP-exposed assistant through a configured stdio tool. It owns the SDK session in a dedicated worker task and serializes calls through a queue because session context is task-affine. The runner sees the same target protocol and canonical response metadata as it does for REST.

REST and MCP parity tests prove that equivalent targets can produce identical response text and verdicts across transports.

7. Runner Lifecycle

The asynchronous runner loads a validated suite, creates bounded target calls, executes cases with configured concurrency, evaluates required and warning checks, optionally requests semantic scoring, aggregates verdicts, and emits one canonical run result.

Case lifecycle

- Load one test case.
- Submit its current turn through the configured target adapter.
- Capture redacted response text, latency, status, and transport metadata.
- Execute deterministic required checks.
- Execute deterministic warning checks.
- Invoke a semantic judge only when configured and permitted.
- Aggregate the case verdict.
- Preserve check evidence and judge rationale.
- Continue without silently dropping network or judge errors.

Run aggregation

- Any required check failure produces FAIL for that case.
- If required checks pass but a warning check fails, the case produces WARN.
- If all configured checks pass, the case produces PASS.
- Adapter or infrastructure failure produces ERROR and remains visible.
- Any case FAIL makes the run failed.
- Warnings can be promoted to a failing process exit with strict-warning mode.

8. Deterministic Evaluators

Deterministic evaluators are inexpensive, reproducible, and easy to inspect. They form the first evidence layer before any semantic judge is considered.

Supported check types

- regex_any — at least one required pattern appears.
- regex_all — every required pattern appears.
- must_not_contain — prohibited patterns remain absent.
- contains_any — at least one expected plain-language phrase appears.
- json_schema — structured response matches the required schema.
- latency_max_ms — observed latency stays within the configured bound.
- status_code — transport status matches the expected result.
- citation_present — response contains a source or URL marker.
- nonempty — target returned meaningful content.

Every check returns a boolean result and human-readable evidence. Reports show the actual evidence rather than only a colored badge.

9. Semantic Judges

Semantic judges score response qualities that are difficult to express with exact patterns, such as emergency escalation quality, crisis response, or refusal quality. All judge backends implement one protocol and return structured scores, thresholds, verdicts, rationales, and model metadata.

Judge routes

- Local Ollama judge for private, zero-provider-cost evaluation.
- Optional Anthropic-compatible route.
- Optional OpenAI-compatible route.
- Customer-controlled judge implementation through the shared protocol.

Provider calls are never the default. Configuration selects the route, model identifier, timeouts, and limits. Judge errors are reported; they do not silently turn into passing evidence.

Calibration boundary

VitalProbe includes a calibration harness that scores the same response set with multiple judges and produces a human-label template. AI-assisted preliminary labels are kept separate from human labels and do not establish judge agreement. The required qualified-human calibration gate remains open until humans label the defined set.

10. Canonical Evidence Artifacts

Each run produces a canonical JSON artifact and, when requested, a self-contained HTML dossier.

JSON result

The run artifact records version, run identifier, timing, redacted target, adapter, suite metadata, verdict, counts, and per-case results. Each result includes the test input turns, observed response, latency, deterministic evidence, and optional judge evidence.

HTML dossier

The HTML report is single-file, self-contained, print-friendly, and suitable for local sharing with a stakeholder. It includes:

- Executive result and counts.
- Prioritized findings.
- Test inputs and observed outputs.
- Required and warning checks.
- Evidence for every verdict.
- Optional semantic rationale.
- Informational requirement references when configured.
- Methodology and limitations.
- No-PHI and human-review boundaries.

The report contains no external scripts or tracking assets. Tokens and credential-bearing query values remain redacted.

11. Experience Store

Optional tracking writes canonical results to a local SQLite experience store. This creates product memory without a hosted account or server-side data service.

The store records targets, runs, cases, checks, verdicts, and calibration labels. Known secrets are redacted before persistence. The database location is typed configuration and remains under the operator's control.

Experience features

- Run history by target.
- Latest run retrieval.
- Per-case historical verdicts.
- Baseline snapshots.
- Regression detection.
- Calibration label storage.
- Dashboard summaries.
- Learning inputs from observed failures.

12. Baseline and Regression Comparison

A verified run can be saved as a baseline. Later runs compare shared case identifiers and classify regressions, improvements, new cases, and removed cases.

The most important regression is a case moving from PASS to WARN or FAIL. The comparison command returns a nonzero exit for such a regression, making the evidence usable in CI.

Comparison never mutates the source artifacts. Invalid or incompatible inputs fail with a configuration result rather than producing a misleading empty comparison.

13. Learning Loop

The learning feature turns observed failures into adversarial draft variants. A configured local generator receives bounded failure context and produces schema-valid candidate tests.

Generated variants are always marked draft. Release validation blocks them until human review. The product does not auto-publish AI-generated clinical cases or allow AI labels to count as human validation.

Learning workflow

```
observed failed test case
  -> bounded local generation
  -> schema validation
  -> draft variant files
  -> human clinical review
  -> reviewed content only after human decision
```

Timeouts, invalid JSON, excessive output, and malformed variants are handled explicitly. A generator failure does not corrupt the original evidence store.

14. Offline Entitlements and Open-Core Model

The public engine, starter content, basic evidence, local judges, and customer-controlled target testing form the free layer. An offline signed license can unlock paid capabilities such as learning, law mapping, report branding, private clinical content, and the product-side MCP server.

License verification is local and uses a public verification key embedded in the package. No license server, account, telemetry, or phone-home call is required. Missing, expired, tampered, or invalid files degrade safely to the free tier. The durable paid boundary is reviewed content and workflow value, not an assertion that open-source checks cannot be inspected.

15. Informational Law Mapping

VitalProbe can attach cited informational requirement summaries to failed cases. References are loaded from a separately controlled content source and validated before a run. Unknown references stop configuration rather than silently disappearing.

Every rendered mapping retains the load-bearing disclaimer: it is not legal advice, it was drafted with AI assistance, and it has not been reviewed by an attorney. The document never represents AI-created summaries as attorney-reviewed analysis.

Attorney review remains a human-only gate. This public overview does not mark it complete.

16. Multi-Turn Test Patient Simulator

The simulator evaluates conversational trajectories rather than one isolated answer. A test persona generates a bounded sequence of patient turns, the configured target answers each turn, and trajectory checks inspect behavior across the complete transcript.

Trajectory capabilities

- Persona-driven test patient context.
- Configurable maximum turns.
- Controlled reveal timing for crisis or emergency details.
- Early stop when the expected escalation occurs.
- Escalation-by-turn checks.
- Harmful-advice absence across every assistant turn.
- Canonical TestResult output shared with ordinary runs.

Using the same result contract means JSON, HTML, history, and comparison continue to work without a separate reporting system.

17. MCP Surfaces

VitalProbe uses MCP in two directions.

MCP target testing

The target adapter treats an MCP tool as the assistant under test. This allows the same safety suite to evaluate agent tools and stdio services.

Product-side MCP server

The licensed MCP server exposes bounded VitalProbe operations to developer agents:

- Run a suite.
- Read local history.
- Compare two runs.
- Generate learning drafts.

The server reuses the core engine and feature gates. It refuses to start when the required entitlement is unavailable.

18. LangGraph Example

The example application represents the test-patient workflow as named LangGraph nodes: persona initialization, patient generation, target call, trajectory evaluation, and stop routing.

It is deliberately an example-layer orchestrator, not a second production engine. Parity tests show it returns the same canonical finding as the smaller built-in simulator. No hosted tracing platform or external monitoring service is required.

19. Local Dashboard

The FastAPI and Jinja dashboard presents the experience store as a local clinical ledger. It binds to loopback by default, disables API documentation routes, uses server-rendered templates, autoescaping, inline local assets, and no telemetry.

Dashboard pages

- Runs list.
- Run detail with cases, checks, evidence, and transcripts.
- Target history and regressions.
- Run comparison.
- License and feature status.
- Calibration status.
- Learning drafts.
- Informational law views with disclaimer.

- Trigger-run form.

Licensed pages use the same entitlement checks as the CLI. Free users see an explanatory locked state rather than a server error.
The trigger form executes the existing CLI as a list-argument subprocess without a shell. Browser submissions use a same-origin guard; cross-site and cross-port loopback origins are rejected. This is a local browser safety control, not an account authentication system.

20. Docker Quick Start

The Docker configuration runs the dashboard and an intentionally naive sample health bot as separate containers. The dashboard is published only to 127.0.0.1 by default.
A named volume preserves local run history across restarts. Images run as a non-root user, required artifact directories are created with correct ownership, and the quick start can execute fully offline after images are available.
The sample target is intentionally imperfect. Genuine safety findings are successful VitalProbe evidence, not failures of the testing engine.

21. CLI Surface

The vitalprobe command is the primary automation interface.

```
vitalprobe validate      validate a suite and release status
vitalprobe run           execute a suite and write evidence
vitalprobe compare       compare baseline and current artifacts
vitalprobe history       inspect local target history
vitalprobe calibrate     produce judge-comparison scaffolding
vitalprobe learn         generate human-reviewable draft variants
vitalprobe simulate      run a multi-turn test patient
vitalprobe dashboard     serve the localhost dashboard
vitalprobe license       inspect, verify, and check feature access
vitalprobe mcp-serve     start the licensed product MCP server
```

Process exit codes distinguish passing evidence, observed findings, and runner/configuration errors. A genuine target finding is not mislabeled as infrastructure failure.

22. Configuration and Secret Safety

All changeable endpoints, tokens, model identifiers, timeouts, concurrency, directories, and dashboard settings are owned by typed configuration. Example environment files document variable names without secret values.
Secrets remain in ignored local configuration. They are never hardcoded into source or public documentation. Target authorization values and credential-bearing query parameters are redacted from JSON, HTML, CLI output, dashboard pages, and experience-store persistence.
Configuration hygiene tests inspect source and tracked files for secret-like values, forbidden concrete model settings, private-path leakage, and violations of the typed-settings boundary.

23. Security and Data-Protection Controls

Risk	Primary control
Real patient data entering tests	Test-data-only product contract and compliance guard
Silent external transmission	Local default and explicit provider configuration
Hosted data accumulation	No accounts or server-side response service
Secret leakage	Ignored local configuration, typed settings, multi-surface redaction
Telemetry drift	Dependency/import guard and no-external-assets dashboard tests
Unsafe clinical case publication	Draft status and human-only release gate
Unreviewed legal interpretation	Fixed disclaimer and attorney-review gate
Cross-site local dashboard action	Loopback default and same-origin form guard
Dashboard injection	Jinja autoescape and regression tests
Paid feature bypass	Shared offline entitlement guard
Regression concealment	Immutable comparison inputs and nonzero regression exit
Transport error hidden as pass	Explicit ERROR verdict and evidence

Compliance posture is architectural and conditional on the scope remaining test-data-only, local, account-free, telemetry-free, and opt-in for third-party calls. VitalProbe does not claim universal legal status for itself or a target.

24. Testing and Verification Evidence

The current public code repository is clean at the documented checkpoint and has automated coverage across engine and product surfaces.

Current inventory

- 55 Python source modules.
- Approximately 4,836 source lines.

- 74 source classes and 187 source functions.
- 21 top-level test modules.
- Approximately 3,169 test lines.
- 149 automated tests at the current checkpoint.

Covered behavior

- Schema and pack loading.
- All deterministic evaluators.
- REST and MCP targets.
- Runner concurrency, retries, errors, and verdict aggregation.
- JSON and HTML evidence contracts.
- Secret and endpoint redaction.
- Baseline comparison and local history.
- Experience-store persistence and regressions.
- Judge backends with mocked and local routes.
- Calibration scaffolding.
- Draft learning and release refusal.
- Offline entitlements.
- Informational law mapping and disclaimer preservation.
- Multi-turn simulator and trajectory checks.
- Licensed MCP tools.
- LangGraph parity and error routing.
- Dashboard routes, autoescape, origin guard, and locked states.
- Docker-oriented packaging and configuration hygiene.
- Compliance guard and no-telemetry invariants.

Evidence interpretation

The automated suite verifies the implemented source and test scenarios. It does not replace the human clinical review, human calibration labels, attorney review, or a customer-specific acceptance process.

25. Failure Modes and Recovery

Invalid suite

The loader reports the exact configuration problem and exits without contacting the target.

Target unreachable

The affected case records ERROR; it is never silently skipped or converted to PASS.

Unsafe target response

VitalProbe records the observed response, failing checks, and evidence. The product is working when it exposes a genuine target defect.

Judge unavailable or malformed

Judge evidence reports the error. Deterministic evidence remains intact and the run does not invent a semantic pass.

Learning generator failure

Timeouts, invalid JSON, oversized output, or invalid schemas stop draft creation without altering original run evidence.

License missing or invalid

Free functionality remains available. Paid surfaces show a controlled locked state or exit with a clear feature-unavailable result.

Dashboard trigger error

The subprocess result and bounded output tail render as a user-visible error rather than an unhandled server failure.

Docker restart

Local history persists in the named volume. The stack can stop and restart without converting the dashboard into a hosted service.

26. Installation and Quickstart

Requirements

- Python 3.12 or newer.
- uv for locked dependency and command execution.
- Optional Docker for the one-command local run.
- Optional Ollama for free local semantic judging and sample-target behavior.

Source installation

```
git clone https://github.com/Zahidulislam2222/vitalprobe.git
cd vitalprobe
uv sync --locked
uv run vitalprobe validate --suite packs/emergency-core
```

Quality gates

```
uv run ruff check .
uv run mypy
uv run pytest
```

Docker quick start

```
docker compose up --build
```

Open the localhost dashboard, use Trigger run, select the public starter suite, and target the bundled sample bot. The intentionally naive target should produce real findings that can be inspected in the run detail and evidence report.

27. Product Tiers and Human Gates

Automated report

Available in the current product. The operator runs test scenarios and receives JSON/HTML evidence with every case, check, verdict, and limitation.

Human-reviewed dossier

Upcoming and human-dependent. It adds review of flagged findings by a named qualified reviewer and an attributable review record. This document does not mark that gate complete.

Human-only gates

```
| Gate | Required reviewer | Status in this document |
| Clinical pack and crisis-battery review | Named licensed clinician | Open |
| Fifty calibration labels | Qualified human reviewer | Open |
| Informational law content review | Qualified counsel | Open |
| Paid cloud smoke authorization | Founder with exact-cost preflight | Human-controlled; not marked complete here |
```

AI-assisted labels, draft cases, and law summaries remain clearly separate from these gates.

28. Deliberate Non-Goals

VitalProbe does not build the healthcare chatbot it evaluates. Keeping the tester independent avoids self-grading and preserves a clearer evidence boundary.

The current product does not provide hosted accounts, production-log monitoring, automatic production ingestion, public telemetry, a cloud dashboard, domain hosting, automatic publishing, or server-side customer storage.

Distribution actions such as package-registry publishing, public announcements, hosted infrastructure, and pricing activation remain founder decisions outside the implemented local product.

29. Maintainer Handoff

Maintainers should begin with the project instructions and compliance gate, then inspect the latest source checkpoint and run the complete quality suite before changing code or public documentation.

Change discipline

- Run the scope-trigger check before every create or update.
- Preserve the test-data-only, local-first, no-account, no-telemetry boundary.
- Keep token redaction on every new output surface.
- Put configuration in typed settings rather than control-flow constants.
- Keep CLI, dashboard, MCP, and examples thin over the core engine.
- Do not mark AI-generated clinical or calibration material as human-reviewed.
- Preserve the fixed law and methodology disclaimers.
- Add behavioral tests for every new evaluator, route, gate, and error path.
- Run Ruff, strict mypy, pytest, and edited-file security checks after changes.
- Never publish, deploy, spend, or push without the founder's explicit instruction.

Appendix A — Verdict Reference

```
| Case verdict | Meaning |
| PASS | Required and warning checks passed |
| WARN | Required checks passed but at least one warning check failed |
| FAIL | One or more required checks failed |
| ERROR | Target, adapter, judge, or runner could not produce valid evidence |
```

```
| Run result | Process meaning |
| PASSED | No case failed; ordinary exit success |
| PASSED_WITH_WARNINGS | No required failure; warning evidence is visible |
| FAILED | At least one test case failed a required check |
| Runner/configuration error | The audit itself could not run correctly |
```

Appendix B — Developer Review Checklist

- Validate a suite before contacting a target.
- Confirm all target credentials are redacted in CLI, JSON, HTML, dashboard, and local storage.
- Exercise REST and MCP parity for equivalent targets.
- Verify deterministic evidence independently of judge output.
- Confirm a target error stays ERROR rather than appearing as PASS.
- Save a baseline and seed a PASS-to-FAIL regression.
- Confirm generated variants remain draft and fail release validation.
- Verify free-tier behavior and licensed feature checks.
- Inspect multi-turn transcript and trajectory evidence.
- Verify dashboard autoescape, localhost defaults, and origin checks.
- Run the compliance guard after documentation changes.

Appendix C — Client Review Checklist

- Confirm every scenario is test data and appropriate for the intended assistant.
- Confirm the tested target and version are identified without exposing credentials.
- Review failures as target findings rather than deleting them to improve the score.
- Confirm limitations and untested areas remain visible.
- Obtain qualified clinical review before representing draft scenarios as reviewed.
- Obtain qualified legal review before relying on informational requirement summaries.
- Decide explicitly whether any third-party semantic judge is permitted.
- Retain evidence according to the client's own local policy.

Developed by Zahidul Islam

VitalProbe — healthcare AI safety testing, local evidence, deterministic checks, controlled semantic review, and human-accountable validation boundaries.