Gnosis AI

Upload a CSV. Ask anything. Get an answer in seconds.

Full-stack · Conversational Analytics · LangChain RAG · 5 LLM Providers

• **LIVE** gnosis-ai-chi.vercel.app

Next.js 14 | FastAPI | LangChain RAG | Python 3.11 | Docker

A solo end-to-end build — from 7 user interviews to live production deployment. Natural language queries route to pre-tested Python tools via a LangChain RAG layer; the LLM never sees raw data and never writes code. Results are deterministic, streaming, and explainable.

11K	5	13	150	65%	<1s
LINES OF CODE	LLM PROVIDERS	ML TOOLS	TESTS	COVERAGE	FIRST TOKEN P90

What this document is — and how to read it

This document is the full design, engineering, and product record of **Gnosis AI** — a conversational analytics platform built solo, end-to-end, from user research to live production deployment.

It covers 12 sections across product thinking, technical architecture, and engineering decisions. The three paths below route you to the sections most relevant to your role.

PATH A

Recruiter / PM Lens

Product thinking, user research, and decision-making

§01 Project overview & motivation
§02 User research (7 participants)
§03 Jobs to be done
§04 User archetypes
§05 3 Deliberate constraints
§06 Competitive landscape
§11 Roadmap
§12 Reflections & learnings

≈15 min read

PATH B

Technical Reviewer Lens

Architecture, RAG design, and engineering trade-offs

§07 Technical architecture
§07 RAG pipeline deep-dive
§08 Frontend modules
§08 Test suite (150 tests)
§09 Engineering challenges
§10 Product decisions & trade-offs

≈18 min read

PATH C

Quick Overview

Core product, key differentiators, and what was built

§01 Project overview
§03 Jobs to be done
§05 Deliberate constraints
§06 Competitive landscape
§06 Technical architecture
§10 Product decisions
§11 Roadmap
§12 Reflections

≈7 min read

Who built this

Gnosis AI was designed, engineered, and shipped *entirely solo* — no team, no PM, no design support. Every decision in this document was made by the same person who wrote the code. The thinking behind each architectural and product choice is documented alongside the code that implements it — because in this build,

they were the same decision.

01 PROJECT OVERVIEW & MOTIVATION

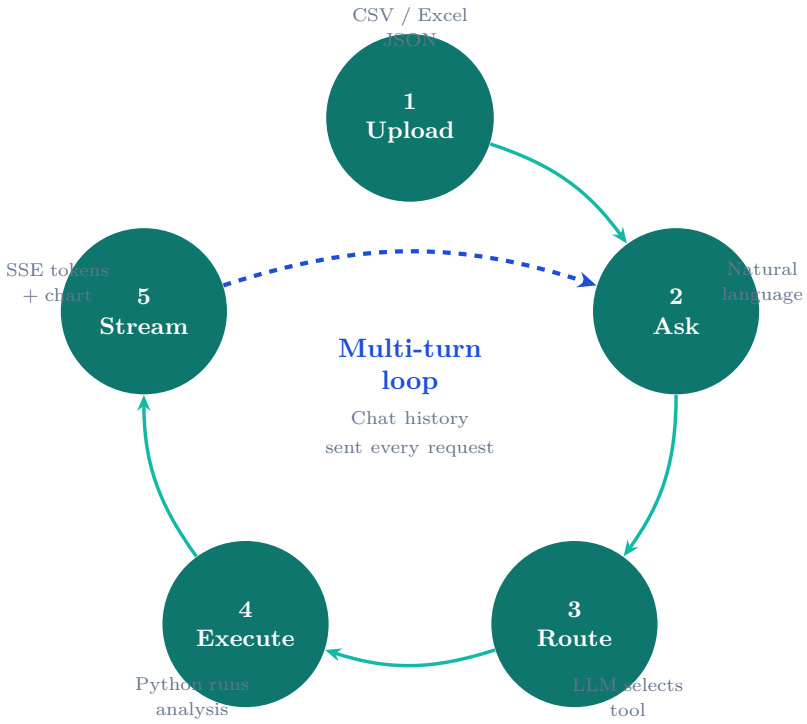
Why I Built This

Exploratory data analysis still requires Python fluency or a BI subscription — even when the question the user wants to ask is often simple.

Data analysts spend disproportionate time on repetitive exploratory tasks: data quality checks, correlation analysis, quick regression, summary reports. These require tool-specific knowledge, not expert judgment — they are mechanical steps *before* the real analysis begins. Gnosis AI eliminates that overhead: upload a CSV, ask in natural language, get a result.

The project was also a deliberate constraint: build something with real engineering and product complexity — multi-provider LLM routing, streaming, a full test suite — as a solo developer under the same resource and prioritisation pressure a 0-to-1 PM faces. Every decision in this document reflects that constraint.

The Analytical Loop



Multi-turn: each response suggests the next question via context-aware chips — zero backend call.

02 USER RESEARCH & PROBLEM VALIDATION

> **Method:** 7 informal discovery conversations — 4 in-person, 3 remote — before any production code was written. Participants were people who work with data but are not data scientists. Asked to describe the last time they needed to understand a CSV quickly. No screener, no incentive.

What They Said

Quote	Participant	Design implication
"I know what I want to learn. I just don't know how to ask Excel for it. The pivot table took 20 minutes and I'm not sure the numbers are right."	P1 — Marketing Manager	Intent is clear; tool interface is the bottleneck. Natural language removes the translation layer.
"ChatGPT gave me a wall of text. I just wanted a number and a chart. I ended up doing it in Excel anyway."	P2 — Business Analyst	General-purpose LLMs produce text, not analysis. A tool-routing layer that runs real computations is a fundamentally different product.
"I need an answer now, not a dashboard next week."	P6 — Product Manager	Latency is the primary pain, not quality. SSE tokens start appearing in under 1 second.

03 JOBS TO BE DONE

Three jobs, three failure modes — derived from 7 interviews, not assumed

	WHEN I...	I WANT TO...	SO I CAN...
1	have a CSV with a question I already know the answer shape of	get the number (or chart) in under a minute, without writing a formula or opening a notebook	move on — the question was never the interesting part, the answer was
2	know the analysis I need ("Random Forest", "clustering") but can't execute it	run it with one sentence, see feature importance and model confidence without reading documentation	decide whether the result is good enough to act on, or hand it to someone technical
3	need to publish a number publicly and can't afford to be wrong	see a reproducible audit trail — sample size, confidence intervals, methodology — not just an answer	stand behind the result if challenged, without having written the analysis myself

Job 1 · Gnosis response

SSE streaming: first token in <1s. No account, no formula, no pivot table. The answer arrives before the user reaches for Excel.

Job 2 · Gnosis response

Context-aware chips surface `run_random_forest` by name. ML explanation panel answers "why this model?" without the user needing to know what SHAP is.

Job 3 · Gnosis response

`generate_pdf_report`: N, confidence intervals, feature importance — shareable without the recipient needing app access.

04 3 USER ARCHETYPES

Behavioural patterns from 7 interviews — not demographics

The Question Owner · P1, P5, P6 · Marketing / Ops / PM

Knows exactly what they want to learn. Zero interest in tools, code, or methodology. Solves with delegation or time-consuming workarounds.

BLOCKER

Tool interface overhead. Knows what to ask; doesn't know which Excel formula extracts it.

ACTIVATION

Fast answer + chart in <60 seconds. High LTV — returns with every new dataset.

The Stranded Analyst · P2, P3, P4 · BA / PhD / Founder

Has some technical exposure — opened a notebook, used ChatGPT — but hits a wall at execution. Knows "I need Random Forest" but can't write it.

BLOCKER

Execution gap. Knows the right analysis type; can't run it without help.

ACTIVATION

First ML tool run. Context-aware chips with analysis names they already know.

The Confidence Seeker · P7, P3 · Journalist / Researcher

Can produce a chart (Datawrapper, Google Sheets). The problem is not the answer — it's trusting the answer. Works publicly where a wrong number has consequences.

BLOCKER

Confidence gap. Gets an answer; can't verify it.

ACTIVATION

PDF report with N, confidence intervals, feature importance. Hardest archetype to convert.

05 3 DELIBERATE CONSTRAINTS

Architecture decisions that exclude functionality intentionally — and why

CHALLENGE Every competing tool (Julius AI, ChatGPT ADA, Hex, Observable) requires an account before the user can do anything. This is a conversion barrier, an email harvesting mechanism, and cognitive tax before the user experiences any value.

SOLUTION ✓ No Email. No Sign-up. No Account. Session UUID in localStorage is the only identity the product needs. A user who hits a sign-up wall returns to Excel. One who gets an answer in 30 seconds has converted. *Trade-off accepted*: no persistent workspaces, billing tiers, or team management — correct scope for v1.

CHALLENGE Every cloud-based competitor sends user data to an external API. This excludes HR managers with headcount data, finance analysts with pre-earnings models, healthcare researchers, legal teams under data residency requirements.

SOLUTION ✓ Gnosis supports Ollama as a first-class LLM provider. With Ollama selected, the entire pipeline — LLM, tool execution, dataframe — runs on-device. Zero data egress. Zero GDPR friction. *Trade-off accepted*: requires capable local hardware and a running Ollama instance.

CHALLENGE ChatGPT ADA, Hex, and Julius AI all show code in their output. The moment code appears, a significant portion of target users closes the tab.

SOLUTION ✓ Gnosis is architecturally prohibited from showing anything technical. No Python. No SQL. No config. The tool badge (`run_random_forest`) tells the user what ran; it never shows the how. *Trade-off accepted*: power users who want to inspect or modify analysis can't — they should use Hex, Julius Pro, or a notebook.

06 COMPETITIVE LANDSCAPE

> **Honest positioning:** Gnosis AI wins when the user has a CSV, zero account anywhere, zero desire to see code, and needs an answer in the next 10 minutes. It loses when they need live database connections, team collaboration, version-controlled notebooks, or enterprise governance. This is intentional — v1 scope.

Feature	Gnosis AI	ChatGPT ADA	Julius AI	Hex
No account required	✓	×	×	×
Free, no message limit	✓	$20/mo	$35/mo	$36/editor
Local / offline execution	✓ Ollama	×	×	×
Zero code in output	✓	×	partial	×
Non-technical user ready	✓	partial	partial	×
SSE streaming response	✓	✓	✓	×
Runtime LLM switching	✓	×	×	×
Team collaboration	×	×	paid	✓
Arbitrary code execution	×	✓	✓	✓

07 TECHNICAL ARCHITECTURE

Next.js 14 Frontend · FastAPI Backend · LangChain RAG · Docker · CI/CD

Layer	Detail
Frontend	Next.js 14 (App Router), React 18, TypeScript · 24 source files / ~6,885 lines
Streaming	Server-Sent Events (SSE) via `useStream` hook with cancel support · tokens start in <1s
Backend	FastAPI (Python 3.11) · per-session dataframe store · pydantic-settings config
RAG / Router	LangChain `ConversationalRetrievalChain` + Chroma vector store + sentence-transformers embeddings
ML Layer	scikit-learn / pandas · 13-algorithm registry · SHAP-style explanations post-run
Middleware	Sliding-window rate limiter (per session) + request-ID tracing (`X-Request-ID`)
Deployment	Vercel (Next.js) + HuggingFace Spaces Docker (16GB) · multi-stage build · non-root user · HEALTHCHECK
Export	PDF generation (reportlab) · Markdown export (client-side Blob) · shareable embed links

Package isolation, not monolith: `src/gnosis_analytics` is installed as an independent pip package. The test suite runs without a running FastAPI server. The CLI works without a running Next.js app. Integration tests mock only the network layer — not business logic. Each layer is independently testable.

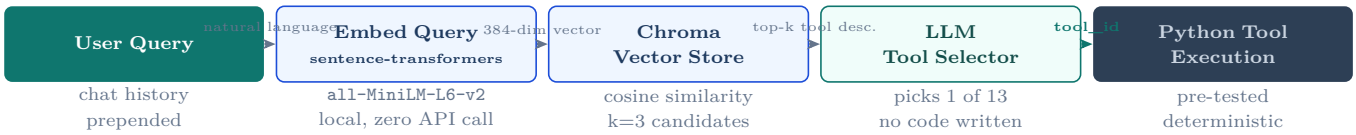
13 Analytical Tools — Registry-driven

Tool	Category	What it does
`run_statistical_analysis`	stats	Descriptive stats, distribution summary, outlier flags
`run_random_forest`	classification	Classification / regression with feature importance chart
`run_machine_learning`	auto-ml	General ML runner — auto-selects best algorithm
`run_correlation_analysis`	analysis	Pearson matrix, top correlated pairs
`run_regression`	regression	Linear / logistic regression with coefficients
`run_clustering`	unsupervised	K-means / DBSCAN with cluster assignment
`run_time_series`	temporal	Decomposition, trend, seasonality
`run_forecast`	temporal	Forward projection from time column
`run_data_quality_check`	utility	Missing values, duplicates, type mismatches
`plot_chart`	viz	Custom chart from natural language description
`clean_data`	utility	Imputation, deduplication, type coercion
`generate_pdf_report`	export	Full analysis report downloaded as PDF (reportlab)
`run_async_analysis`	async	Background job with webhook notification + polling

RAG Pipeline · How Tool Selection Works

Semantic routing, not keyword matching — the architecture that makes results deterministic

Why RAG here? The classical use of RAG is retrieval-augmented *generation* — fetch documents, inject into context, let the LLM write an answer. Gnosis uses it differently: as a *semantic router*. The vector store holds tool descriptions, not data. The LLM never sees the user's CSV. It only sees the question and a ranked list of candidate tools, then selects one. This is the architectural decision that makes outputs deterministic and testable.



Embedding Model

`all-MiniLM-L6-v2`

384-dimensional dense vectors. Runs locally via `sentence-transformers` — zero external API call for every routing decision.

Why not OpenAI embeddings?
Latency and cost. An embedding API call on every user message would add 200–400ms and a per-token charge to every routing decision. A local model adds <10ms.

Trade-off:

Lower ceiling on nuanced semantic similarity — acceptable because tool descriptions are short, controlled, and non-ambiguous.

Vector Store

Chroma (local, embedded)

Chroma runs in-process as an embedded store — no separate server, no network hop, no infrastructure to manage. Spins up with the FastAPI process.

Why not Pinecone / Weaviate?
Managed vector stores are correct when you have millions of documents or need multi-region replication. The Gnosis tool registry has 13 entries. A managed store would be operational complexity with zero return.

Trade-off:

No persistence across cold restarts (store is rebuilt on startup from the registry). Acceptable — rebuild

Tool Selection Logic

ConversationalRetrievalChain

Query is embedded → cosine similarity against 13 tool-description vectors → top-3 candidates returned. LLM receives candidates + chat history and outputs exactly one `tool_id`.

Why top-3, not top-1?
Cosine similarity alone can misrank when queries are ambiguous ("analyse trends" could be `run_time_series` or `run_correlation_analysis`). Giving the LLM 3 candidates lets it use semantic context to disambiguate.

What if all 3 are wrong?

The LLM falls back to `run_statistical_analysis` —

Why this matters for correctness: because the LLM selects a tool rather than writing code, every analysis path is a pre-tested Python function with known inputs, known error types, and a fixed output schema. A hallucinated column name in generated code causes a silent runtime crash. A wrong tool selection causes a recoverable routing error with a labelled retry button. The failure mode is visible and recoverable by design.

5 LLM Providers · Runtime Switching

– **Groq** — Default. Fast, free. Model llama-3.3-70b-versatile. Sub-second P90 on short prompts. Requires only API key in `.env`.

– **OpenAI** — GPT-4o. Highest ceiling. Server key or BYOK: user pastes their own key in-UI; stored in-memory only; disappears with session TTL (2h).

– **Gemini** — Google Gemini 2.5 Flash. Long context window. Server key or BYOK. Switchable at runtime without page reload.

– **Ollama** — Local execution. Self-hosted, zero data egress. Entire pipeline runs on-device: LLM, tool execution, dataframe. Zero API key. Zero GDPR friction.

– **Mock** — Deterministic stub responses for UI development and demos. Zero key required — open the app and work immediately.

BYOK · Key Lifecycle & Security Model

How the key moves through the system — and where it never goes

Entry

User pastes key into `Sidebar.tsx` input field. Sent once over HTTPS in the request body to `POST /chat`.

Never touches a form submission. Never appears in a URL or query string. Never written to `localStorage` or any browser storage — only held in React state for the duration of the session.

In-flight

FastAPI receives key in the request body. Passed directly to the LLM provider client for that request only — stored as a local variable, not on the session object.

Never written to disk. Never appears in server logs (request body logging is disabled for `/chat`). Never cached between requests.

Expiry

Session TTL: 2 hours. When the session expires, the `SessionStore` entry is evicted — the key is gone with it.

Tab close or page reload clears React state immediately. No server-side action required. The operator never has access to the key at any point in this lifecycle.

> **Why this model exists:** a public, account-free deployment can offer GPT-4o or Gemini quality without the operator absorbing per-token costs across anonymous users. The user supplies the key; the operator supplies the infrastructure. Neither party accumulates the other's credentials.

Performance Metrics

Measured on HuggingFace Spaces deployment (2 vCPU · 16GB RAM) · 2026

Provider	First token P50	First token P90	Tokens/s	Tool routing	Notes
Groq (LLaMA 3.3 70B)	<400ms	<1s	~180	~300ms	Default provider
OpenAI (GPT-4o)	~600ms	~1.2s	~80	~500ms	Highest accuracy
Gemini (2.5 Flash)	~500ms	~1.1s	~120	~400ms	Long context
Ollama (local, 8B)	~800ms	~2s	~30–60	~600ms	Hardware-dependent
Mock	<10ms	<10ms	N/A	N/A	Dev/demo only

Embedding latency

`all-MiniLM-L6-v2` runs locally. Embedding + Chroma cosine search adds <10ms per request — immeasurable to the user. Compared to OpenAI `text-embedding-3-small`: 200–400ms per call + API cost per token.

ML tool execution

scikit-learn tools run in a separate thread (non-blocking). Random Forest on a 10K-row CSV: ~1.2s. Correlation matrix: ~200ms. Async mode (`run_async_analysis`) available for datasets that exceed the SSE timeout window.

Memory footprint

Per-session dataframe store with 2h TTL and LRU eviction at MAX_SESSIONS cap. Chroma store: <5MB resident (13 tool-description vectors). `all-MiniLM-L6-v2` model: ~90MB on first load, cached for the process lifetime.

08 FRONTEND ARCHITECTURE · 4 KEY MODULES

17 source files (components + lib + app) · TypeScript throughout · state lifted to root — no Redux, no Zustand

The 4 modules below are the architecturally richest — they hide the most interesting design decisions. The remaining components (`CompareView`, `DatasetGallery`, `DatasetPreview`, `Phase2Panel`, `OnboardingTour`, `ShareButton`/`ShareModal`, `ThemeToggle`, `GoogleSheetsImport`, `ErrorBoundary`, `MessageSkeleton`) exist in the codebase and follow the same patterns.

Sidebar.tsx

1,021 lines · Dataset upload + LLM switcher

Dataset upload, column inspector, LLM switcher + BYOK key input, quick actions, A/B compare trigger, session analytics, PDF export. The richest component.

Upload LLM Switch BYOK A/B Compare

MessageBubble.tsx

744 lines · Markdown · Tool badge · ML explanations

Markdown renderer, tool badge, chart card (ResultCard), ML explanation panel ("Why did the model choose this?"), retry on error.

Markdown Charts SHAP Retry

lib/useStream.ts

231 lines · SSE hook with cancel + sentinel detection

SSE hook: connect, append tokens, detect tool sentinels (`__tool:X__`), cancel, retry-with-silence-timeout. Wraps `EventSource` in a clean 60-line interface.

SSE EventSource Cancel Sentinel

SmartSuggestions

Context-aware chips · zero backend call

4 prompt chips tuned to the loaded dataset. Column-name regex heuristics (`/surviv|churn|target|label/`) for binary target detection. Generated synchronously — zero network requests.

Zero API Heuristics Chips

Test Suite · 150 Tests

9 test modules · 65% coverage gate · GitHub Actions CI on every push

File	Tests	Covers
`test_pipeline.py`	32	DataCleaner, DataLoader, StatisticsAnalysis, MachineLearningAnalysis, LLM factory, FastAPI routes, vector store
`test_integration.py`	25	End-to-end: health, upload (CSV/Excel/JSON), chat (single & multi-turn), PDF download, session delete
`test_tools.py`	21	Thread-local request context, per-tool wrappers, thread isolation under concurrency
`test_config.py`	19	pydantic-settings defaults, validation, credential checks
`test_run_pipeline.py`	16	CLI/engine entry point (`run_pipeline()`)
`test_session.py`	13	SessionStore CRUD, TTL expiry, MAX_SESSIONS cap, LRU eviction
`test_async_analysis.py`	12	/analyze/async job lifecycle, status polling, webhook dispatch
`test_middleware.py`	9	Rate-limit thresholds, per-session counters, request-ID propagation
`test_compare.py`	3	/compare endpoint: row counts, common/disjoint columns

> **Honest engineering note:** `tests/test_tools.py` exists in its current form because of a real regression — thread-local request context was leaking ML results between concurrent sessions under load. The test is still in the suite, named after the bug it caught. A clean test list without history is a weaker signal than a test that documents an incident.

09 ENGINEERING CHALLENGES

CHALLENGE ML results bleeding between concurrent sessions under load — session B briefly saw session A's ML results.

SOLUTION ✓ Thread-local context used thread-local storage but was not reset between requests handled by the same worker thread. Fix: explicit context reset at the start of every request + dedicated regression test (`tests/test_tools.py`) that spawns real threads and asserts isolation.

CHALLENGE Streaming tokens with tool-start signals mixed in the same SSE channel — how does the frontend know when a tool starts?

SOLUTION ✓ Sentinel protocol: the backend emits `__tool:X__` as a special token before the readable response begins. The frontend regex-matches and strips sentinels before rendering; uses them only for the `toolRunning` badge. Zero separate control channel.

CHALLENGE Context-aware chips without a backend call on every dataset load — how do you know which suggestions to show?

SOLUTION ✓ Column names and dtypes are returned by the upload endpoint. Client-side regex heuristics (`/surviv|churn|target|label|class|fraud/`) identify likely targets. 4 relevant chips generated synchronously — zero network requests.

CHALLENGE Mobile keyboard displacing the InputBar off-screen on iOS/Android.

SOLUTION ✓ InputBar attaches a `resize` listener to `window.visualViewport`. When the viewport shrinks (keyboard open), `scrollIntoView({block: 'nearest'})` is called in `requestAnimationFrame` — the textarea remains visible above the keyboard.

CHALLENGE Vercel Serverless Functions enforce a hard, non-configurable 4.5 MB request-body limit — large dataset uploads returned `413` regardless of the backend's own 50 MB limit.

SOLUTION ✓ Uploads bypass the Vercel proxy entirely: the browser posts directly to the FastAPI backend, CORS-scoped to the deployed frontend origin via `ALLOWED_ORIGIN` and exposed client-side through `NEXT_PUBLIC_BACKEND_URL`. Chat and metadata calls stay on the proxied `/api` route; only the large-payload path changed.

10 PRODUCT DECISIONS & TRADE-OFFS

CHALLENGE **D1 Chat Interface, not Form/Dashboard.** Rejected: form-based query builder with dropdowns for operation, columns, parameters.

SOLUTION ✓ Chat wins because a form builder's categories are finite. Natural language lets the user ask questions the designer didn't anticipate. The LLM router handles ambiguity resolution — zero learning curve. *Cost accepted*: ambiguous questions may route to the wrong tool; mitigated by tool badge + retry button.

CHALLENGE D2 Tool Router, not Code Generation. Rejected: LLM generates Python/pandas code on the fly and executes it in a sandbox (à la ChatGPT ADA).

SOLUTION ✓ Tool routing wins because generated code fails gracefully in unpredictable ways — hallucinated column names, incorrect API calls, import errors. A pre-tested tool either runs correctly or returns a known error type. Execution starts immediately — no extra LLM round-trip. *Cost accepted*: unusual analysis outside the catalogue requires exporting to a notebook.

CHALLENGE D3 SSE Streaming, not WebSocket. Rejected: WebSocket for bidirectional real-time communication.

SOLUTION ✓ SSE wins because the communication pattern is strictly one-directional: client sends, server streams. SSE runs over plain HTTP/2, auto-reconnects via the browser's `EventSource` API, no upgrade handshake required. WebSocket's bidirectionality is overhead without benefit. *Cost accepted*: server cannot initiate messages without a client request — not a constraint for this use case.

Standalone CLI

Same analytics engine · 3 front doors: Web App / Async Jobs / Terminal

```
# Install engine as pip package pip install -e ./src
# Run full ML analysis from terminal gnosis analyze  --file data.csv  --mode ml
--target Survived  --output report.pdf
# Modes: stats | ml | all # Algorithm IDs match the same registry used by the API
```

> The same tested code path executes in all three contexts. There is no "CLI version" vs "API version". A bug fixed in the engine is fixed everywhere. A finding from the chat UI can be reproduced from the terminal with one command.

11 ROADMAP

Shipped (v1)

✓ Live production deployment (Vercel + Hugging Face Spaces)

✓ Async analysis with webhook notification

✓ Server-side A/B comparison (`CompareView` + `/compare` + `test_compare.py`)

✓ Standalone CLI wrapper

✓ CI pipeline with Codecov coverage reporting

✓ Multi-stage Docker build + HuggingFace Spaces deployment

✓ 150 tests, 65% coverage gate

✓ 5 LLM providers with runtime switching

✓ BYOK key model (in-memory only)

✓ Share link feature (`ShareButton/ShareModal`)

✓ Google Sheets import

✓ Onboarding tour (`OnboardingTour`)

✓ Dark/light theme toggle

In Progress

→ Phase 2 Panel: clustering, dimensionality reduction (sidebar imports already in codebase)

→ Expanded LLM coverage: Claude API, HuggingFace Inference (factory already extensible)

→ Python 3.9–3.11 CI matrix for backwards compatibility

v2 Signals

◇ Authenticated sharing: named sessions, persistent workspaces

◇ Plugin tool catalogue: new tools require only backend + one registry entry

◇ Database connectors: Postgres, Snowflake

◇ Team access controls and shared sessions

12 REFLECTIONS & LEARNINGS

I built Gnosis AI to answer a question I kept encountering: why does exploratory data analysis still require Python fluency or a BI subscription, when the question the user wants to ask is often simple?

The architectural decision that defines the product — *tool routing instead of code generation* — was not the first idea. It emerged after 7 interviews, after a first prototype that showed Python code to the user and lost them, and after the thread-local context bug revealed where the real complexity was.

Every feature exists because it removes friction from a specific user moment — not because it was technically interesting to build.

What I would do differently: start with the session-isolation problem earlier. The thread-local context bug was caught by a test — but that test was written *after* the bug appeared under load. A concurrency model (one dataframe store per request, not per session) would have been the cleaner default from day one.

Spilios Dimakopoulos · github.com/SpiliosDimakopoulos/gnosis-ai · Next.js 14 · FastAPI · LangChain RAG · 2026