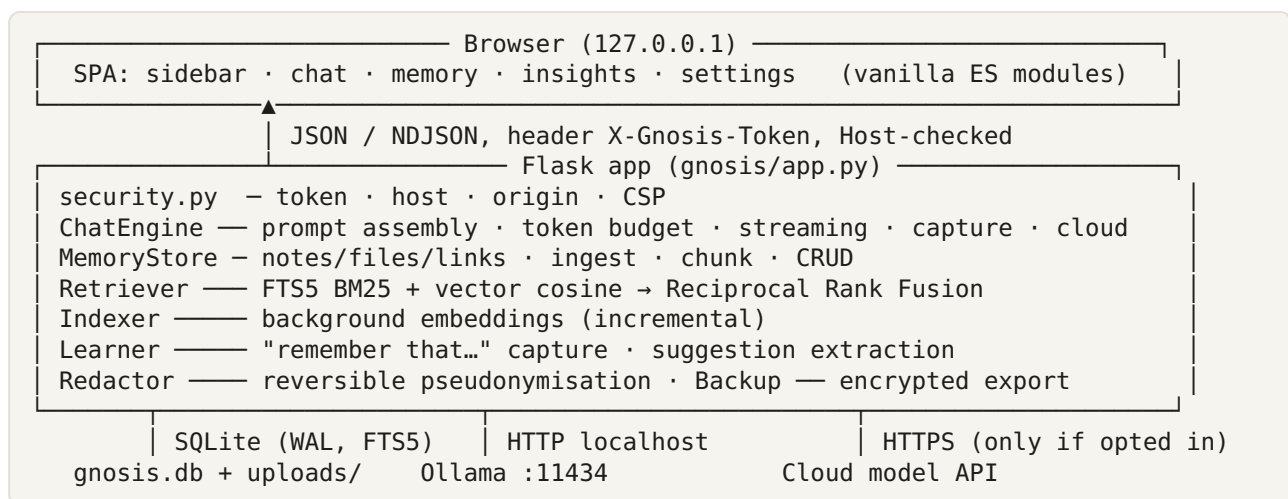


Gnosis v2 — Technical Architecture

1. Goals and principles

1. **Local-first.** All personal data, indexing and default inference run on the user's machine.
2. **Opt-in, inspectable egress.** The only paths off-device are (a) a cloud answer the user previews and confirms, (b) model downloads through Ollama, (c) an explicit "read this page" on a saved link.
3. **It must just work.** Typed, actionable errors; one-click model install; no step requires a terminal after first launch.
4. **Honest memory.** Nothing is memorised silently; everything is editable, downloadable, deletable.
5. **Small and boring.** Flask + SQLite + vanilla JS; stdlib HTTP clients; no build step, no CDN, no telemetry.

2. System overview



Process model

One Python process: a threaded Werkzeug server, one background **indexer** thread, and a small pool (2) for file ingestion. SQLite uses one connection per thread (WAL), so readers never block the writer.

3. Data model (SQLite, schema v2)

Table	Purpose
<code>conversations(id, title, pinned, created, updated)</code>	Chat threads
<code>messages(id, conv_id→CASCADE, role, content, sources JSON, meta JSON, created)</code>	Turns; <code>sources</code> = memories actually cited; <code>meta</code> = model, cloud flag, saved-memory flag

Table	Purpose
<code>memories(id, kind note/file/link, title, content, filename, stored_as, mime, size, url, tags, pinned, status, error, origin, created, updated)</code>	One row per memory; <code>content</code> = text (extracted text for files); <code>stored_as</code> = basename of the original in <code>uploads/</code>
<code>chunks(id, ref_type memory/message, ref_id, conv_id, idx, header, text, emb BLOB, emb_model)</code>	Retrieval units (~900 chars, 140 overlap). <code>emb</code> is a float32 unit vector; <code>emb_model</code> lets a model change trigger automatic re-embedding
<code>chunks_fts</code>	FTS5 (porter stemmer) external-content index kept in sync by triggers
<code>suggestions(id, text, conv_id, status)</code>	Facts proposed by the model, awaiting accept/dismiss
<code>audit(id, ts, kind, detail)</code>	Cloud-send log (metadata only)
<code>settings(key, value JSON), meta</code>	Preferences (no secrets)

Secrets are **not** in the database: OS keychain via `keyring`, else a Fernet-encrypted file with a 0600 key-file.

4. Memory pipeline

1. **Create.** Notes are chunked synchronously (searchable by keyword immediately). Files are streamed to `uploads/<uuid><ext>` (size-capped, filename reduced to a basename) and queued for extraction.
2. **Extract** (`ingest.py`). PDF (pdfplumber, OCR fallback per page), DOCX (headings preserved), PPTX (slides + notes), XLSX (per sheet), HTML (scripts stripped), RTF, JSON, text/code, images (Tesseract), audio/video (faster-whisper, local). Optional libraries are imported lazily; a missing one fails *that* file with an install hint, never the app. Failed files remain downloadable and can be retried.
3. **Chunk** (`chunker.py`). Paragraph/sentence packing with overlap; headings and `[Page n] / [Slide n] / [Sheet x]` markers become a *contextual header* (`Title > Page 3`) that is indexed with every chunk — the same trick as "contextual retrieval".
4. **Embed** (`indexer.py`). Background thread, batches of 16, `nomic-embed-text` with the recommended `search_document:` / `search_query:` prefixes. Backs off for 60 s if Ollama is down; keyword search keeps working. Only chunks lacking a vector for the *current* embedding model are processed (v1 re-embedded everything on every save).
5. **Chats are memory too.** The user's own messages from *other* conversations are chunked and indexed (assistant text is not, to avoid the model citing itself). The current conversation is excluded from retrieval — it is already in the prompt.

5. Retrieval (`retrieval.py`)

- Query → stop-word-filtered tokens → FTS5 OR query ordered by `bm25(text 1.0, header 0.4)`.

- Query embedding → cosine against an in-memory float32 matrix (rebuilt only when a global *epoch* changes). Candidates must be within 0.22 of the best score and ≥ 0.30 .
- **Reciprocal Rank Fusion** ($k = 60$) merges the lists; at most 3 chunks per source; top 8 returned. Short follow-ups ("and when is that?") are expanded with the previous user turn.
- Degrades to keyword-only (no embeddings) or a LIKE-scan (no FTS5) without errors.

6. Conversation engine (`chat/engine.py`)

Prompt layout: system prompt (identity, local date/time, profile, behaviour rules, advisory style) + *always-known* pinned notes + `MEMORY EXCERPTS [M1]...` → conversation history → user message.

Token budgeting. `num_ctx` (default 8192) is sent explicitly to Ollama (v1 relied on a small default and failed with "input length exceeds the context length"). Budget = `num_ctx - reply_reserve ( $\leq 2048$ )`; excerpts get $\leq 40\%$ (max 3500 tokens); history is filled newest-first; a huge paste is trimmed head+tail. Tokens are estimated at 3 chars/token (conservative).

Streaming. `POST /api/chat` returns NDJSON events: `start`, `saved`, `sources`, `token...`, `done` | `error`. The partial answer is persisted if the user presses Stop. Errors carry a `code` and an `action` (e.g. `{"type": "pull", "model": "llama3.1:8b"}`) so the UI offers a one-click fix.

Citations. The model is told to cite `[M#]`; the UI turns them into chips; only cited sources are stored with the message.

Learning. * *Explicit capture* — a deterministic grammar (`remember that ...`, `... remember this`, `save this to memory: ...`, ...) saves immediately (deduplicated) and tells the model to acknowledge briefly. * *Suggestions* — after a reply, a cheap heuristic (first-person statements, length, not a question) gates one low-temperature JSON extraction call; results are de-duplicated (Jaccard ≥ 0.6) against existing notes and surface as an inbox. The user is always the gate. * *Titles* — heuristic title immediately; a 3–6 word LLM title replaces it in the background unless the user already renamed the chat.

7. Privacy and security model

Threat model: other websites in the user's browser; other local users/processes; accidental over-sharing to the cloud; lost laptop/backup; malicious files; prompt-injection through saved content.

Threat	Control
CSRF from any web page	Random per-run token required on every <code>/api</code> call (held only in the served HTML — cross-origin pages can't read it); <code>Origin</code> check
DNS rebinding	<code>Host</code> must be <code>127.0.0.1/localhost/[::1]</code> on our port
XSS	Strict CSP (<code>script-src 'self'</code> , no inline script/style), no <code>innerHTML</code> with untrusted text; Markdown renderer escapes first and emits an allow-list of tags; links <code>rel=noopener</code> , http(s) only
	<code>frame-ancestors 'none'</code> , <code>nosniff</code> , <code>no-referrer</code> , <code>no-store</code> on API responses

Threat	Control
Clickjacking / sniffing / referrer leaks	
Path traversal	Uploads stored under a UUID name; display name is a basename; backup import accepts only <code>gnosis.db</code> and flat <code>uploads/<name></code> regular files
SSRF via "read this link"	Public addresses only (loopback/private/link-local/reserved refused); user-initiated; 3 MB cap
Cloud over-sharing	Off by default; two-step preview → send bound by <code>payload_id</code> (what you saw is what is sent; one-shot, 10-min expiry); reversible pseudonymisation of emails, URLs, IPs, UK postcodes, NI numbers, IBANs, Luhn-valid card numbers, phone numbers, your name and your own terms; memory excerpts opt-in per message; audit log
Key theft	OS keychain / Fernet file (0600); never returned by the API (only a 4-char hint); excluded from backups
Lost backup	PBKDF2-HMAC-SHA256 (600 k iterations) → Fernet (AES-128-CBC + HMAC-SHA256); no recovery by design
Data on disk	Plain SQLite in the user's profile folder (0700). Use FileVault/BitLocker for at-rest encryption
Prompt injection from stored docs	Excerpts are labelled as the user's records, cannot invoke tools (there are none), and the worst case is a wrong answer, not an action

Out of scope: malware running as the same OS user; a cloud provider's own retention policy for text the user chose to send.

8. HTTP API (all JSON; token header required)

Method & path	Purpose
<code>GET /api/status</code>	Version, Ollama state (running, models installed), index progress, counts, cloud flags
<code>GET/PUT /api/settings</code>	Profile, appearance, advisory, models, cloud (non-secret)
<code>PUT/DELETE /api/secrets/cloud-key</code>	Store / remove the API key
<code>POST /api/models/pull</code>	Stream model download progress (NDJSON)
<code>GET/POST/DELETE /api/conversations</code> ; <code>GET/PATCH/DELETE /api/conversations/<id></code> ; <code>POST .../clear</code>	Chat management
<code>POST /api/chat</code>	Stream an answer (<code>conv_id?</code> , <code>text</code> , <code>regenerate?</code>)
<code>POST /api/chat/cloud/preview</code> · <code>/send</code>	Redacted preview → confirmed send
	Memory CRUD

Method & path	Purpose
GET/POST <code>/api/memories</code> ; POST <code>.../link</code> · <code>.../upload</code> ; GET/PATCH/DELETE <code>.../<id></code> ; GET <code>.../<id>/download</code> ; POST <code>.../<id>/reprocess</code> · <code>.../fetch</code>	
GET <code>/api/search?q=</code>	Ad-hoc retrieval
GET <code>/api/suggestions</code> ; POST <code>.../<id>/accept</code> · <code>/dismiss</code>	Learning inbox
GET <code>/api/insights</code> ; POST <code>/api/insights/reflect</code>	Stats and local reflection
GET <code>/api/privacy/log</code>	Cloud-send log
POST <code>/api/backup/export</code> · <code>/import</code>	Encrypted backup
POST <code>/api/data/wipe</code>	Erase memory/chats (needs <code>confirm: "DELETE"</code>)

9. Frontend

Vanilla ES modules, no framework or build. `util.js` (DOM helper `h()` , icons, safe Markdown, dialogs), `api.js` (token, NDJSON reader), `state.js` (shared state + event bus), one module per view. The four views stay mounted so a running stream isn't lost when you visit Memory. Responsive (drawer sidebar < 860 px), dark mode via CSS variables, `prefers-reduced-motion` respected, keyboard operable, dialogs use `<dialog>` .

10. Testing

- **44 backend tests** (pytest) against a mock Ollama: chunking, capture grammar, retrieval (keyword/hybrid), edit/delete consistency, ingestion of DOCX/PDF/HTML/JSON/binary, path sanitisation, token budget with tiny context, regenerate, missing-model and Ollama-down errors, suggestions, cloud flow with a fake provider (preview sends nothing; replay refused; redaction; restoration), backup round-trip/wrong password/traversal, header and token enforcement, secret never leaks.
- **13 browser tests** (Playwright/Chromium): streaming + citation click, "remember that" badge, note edit/download/delete, mixed-type uploads and search, sidebar rename/delete/clear/hide, markdown XSS safety, model-install banner flow, retry after Ollama failure, profile/theme persistence, full cloud flow, backup download, mobile layout.

11. Scaling notes

- **Corpus size.** Numpy brute-force cosine is exact and fast to ~200 k chunks (≈ 600 MB of vectors at 768-d); beyond that swap `Retriever._vector` for `sqlite-vec/HNSW` — the interface is one method.
- **Better models.** Any Ollama model works; set it in *Settings* → *Models*. A re-ranker stage can slot between fusion and prompt assembly.

- **Multi-user (coaching clients).** Today's boundary is "one OS user = one data folder". A client edition would keep that shape (each client runs their own copy) and add: code-signed installers, auto-update, per-profile `GNOSIS_HOME`, and optional SQLCipher at-rest encryption.
- **Sync between devices** is deliberately export/import; an E2E-encrypted relay is the natural next step.