CASE STUDY · HEALTHCARE AI

Smart Medical Records Engine

A doctor asks in plain English and gets the answer out of that patient's own notes, never another patient's. The retrieval query carries the patient_id, so the database filters by patient before it ranks by similarity. Thirty adversarial probes designed to leak cross-patient data, zero leaked.

> What this evidences: scoping retrieval so the wrong patient cannot surface, and proving it with thirty adversarial probes. The model is the easy part.

Taiwo Alabi · MSc Artificial Intelligence, National College of Ireland

Sep 2025 to Feb 2026 · FastAPI · PostgreSQL with pgvector · LangGraph · Ollama · Docker Compose

01 · The problem

The wrong patient is a real failure mode

A doctor has ten minutes with you.

Somewhere in your file, which might be three years of notes from different visits, different doctors, different hospitals, there is a line saying you reacted badly to a particular medicine.

Your doctor is about to prescribe you something. Do they remember that line? Did they ever see it? Nobody reads three years of notes in ten minutes.

That is the whole problem. Not a modelling problem. A retrieval problem with a consequence attached.

> **The hidden failure mode.** A general vector search over all notes would happily return another patient's record if that record happened to be the closer semantic match. It often would, because two people with the same condition produce very similar notes.

02 · What it does

One patient's notes. Nothing else.

A search box a doctor can talk to normally. They ask "has this patient ever reacted badly to any medication?" and it answers from that patient's own notes, showing the actual note it found the answer in.

Three properties make it trustworthy, and all three matter more than the retrieval model itself:

- **Scope is enforced in the schema.** The retrieval query carries the patient_id from the request, and the database filters on it before ranking. A misconfigured application cannot leak across patients.
- **Records never leave the hospital.** The model runs on the hospital's own machine.

- **Every access is written down.** If anyone asks who opened a patient's file, there is an answer.

---

03 · How it works

Four services and one hard rule

The retrieval rule: the query carries the patient_id from the request, and the database filters on it before ranking. Skip that step and a search will return a different patient's note because it happens to be worded the same way.

Four Docker Compose services on one private network. The constraint that scopes the search to one patient sits at the database layer, not in application code, so it cannot drift away from the search.

SERVICE	JOB	CONSTRAINT
`api`	Receives question, checks JWT, dispatches	Auth required, audit row written
`retrieve`	Embeds question with nomic-embed-text, searches one patient's notes	**WHERE patient_id = $1** before any vector op
`generate`	Pulls retrieved notes, calls llama3.1 locally	PII stripped from prompt
`ingest`	Chunks records, embeds, loads with patient reference attached	Runs once, separate path

---

04 · The safety rail that matters most

Schema-level scoping

Retrieval is scoped by patient before it ranks by similarity.

This sounds like a detail. It is the whole design. A general vector search over all notes would happily return another patient's record if that record happened to be the closer semantic match, and it often would, because two people with the same condition produce very similar notes.

So the filter is not a ranking preference applied afterwards. Wrong-patient results are excluded from the candidate set entirely, and the constraint is carried in the schema rather

than left to application code.

> **Why the schema, not the application.** Embeddings change when the model is updated. The patient_id column does not. Scoping happens at the database layer, so the constraint cannot drift away from the search.

---

05 · The data model

One row, one vector, one patient_id

Each note carries a `vector(768)` embedding on the same row as its patient_id. The scoping constraint and the searchable content cannot drift apart.

Notes are stored in SOAP format (subjective, objective, assessment, plan) because that is how clinicians already write them. Matching the existing format matters more than choosing a neater one, because it means records can be ingested as they are.

---

06 · Why self-hosted

The ceiling is fixed by the constraint

Every model in this system runs locally through Ollama. No API calls. No data sent out.

That decision costs accuracy. A hosted frontier model would answer better. But patient records leaving a hospital network is not a trade-off a hospital can make, so the ceiling on model quality is fixed by the constraint, not the other way round.

The whole stack (API, database, language model) runs across four Docker Compose services on one machine.

---

07 · Who is allowed in

Auth, PII stripping, full audit

Doctors log in with JWT. Passwords are never stored in readable form, only a scrambled version that can check a password without keeping it.

Personal details (PII) like names and addresses are stripped out before the text ever reaches the AI model, so it never sees identifying information it does not need in order to answer the question.

Every access writes an audit_log row against the doctor who made it. A system handling medical records that cannot answer "who looked at this, and when" is not deployable regardless of how well it retrieves.

---

08 · Verified

Thirty attempts to leak, zero leaks

The claim in section 04 is now tested rather than asserted. I made thirty attempts to trick the live application into leaking another patient's data, through the real path: signed in as a physician, asking questions designed to pull another patient's notes into the answer.

The probes covered four angles:

- **Near-verbatim copies** of a decoy patient's most distinctive lines.
- **Semantic twins** of the same clinical facts in different wording.
- **Broad fishing questions** with no patient context at all.
- **Wrong-patient queries** that named a different patient by name.

Each attempt included a check to make sure it was a fair test. Before counting it, I ran the same search without the patient filter, to confirm an unscoped search really would have surfaced another patient's note first.

PROBE CLASS	PROBES	UNSCOPED SEARCH WOULD HAVE LEAKED	LEAKED WITH SCOPING ON
Near-verbatim copies	8	8 / 8	0 / 8
Semantic twins	10	10 / 10	0 / 10
Broad fishing questions	6	6 / 6	0 / 6
Wrong-patient queries	6	6 / 6	0 / 6
TOTAL	30	30 / 30	0 / 30

All thirty passed the fair-test check. In 30 of 30 cases, the wrong patient's note would have been the top result of an unscoped search. With scoping on, none of the thirty attempts leaked anything.

> **What this is not.** A penetration test. No auth bypass was attempted, no injection, no infrastructure attacks. It verifies the scoping design through the application's own query path, which is the design this system stands or falls on.

Reproduce:

```
python leak_probes/seed_fixture.py     # seeds a target patient and two decoys
python leak_probes/run_probes.py       # runs the thirty probes, prints a
results table
```

`09 · Honest limitations`

What this does not prove

- **It has not been tested with real clinicians.** The retrieval works. Whether the interface fits a ten-minute consultation is unproven, and that is the claim that matters.
- **The data is synthetic.** Real notes are messier: abbreviations, inconsistent formatting, scanned documents.
- **A local llama3.1 is weaker than a hosted model.** Accepted deliberately, but it means answers on complex questions are less reliable than the retrieval underneath them.
- **No formal evaluation set.** Leakage was the priority, and it is measured above. Answer quality against a labelled benchmark remains unmeasured.

`10 · Running it`

```
docker compose up
python scripts/create_doctor.py       # create a physician account
```

Ollama pulls nomic-embed-text and llama3.1 on first run. Ingestion loads the records and builds the embeddings. Nothing reaches the internet after the model pull.

Stack: Python · FastAPI · PostgreSQL with pgvector · Ollama (llama3.1) · Docker Compose · JWT

---

11 · Terms used in this report

Embedding
turning a patient's note into a row of numbers, so a computer can search by meaning instead of just matching words.

JWT
a secure login token a doctor gets after signing in, checked on every request.

PII
personal details like a name or address that could identify someone.

Retrieval
the step of finding and pulling in the right piece of information before answering a question, rather than the AI guessing from memory.

Scoping
restricting a search to one patient's own records before it even starts, rather than searching everyone and filtering afterwards.

Links

Project page: iamtamilore.github.io/p/clinical-rag

Source: github.com/iamtamilore/agentic_rag_fastapi

Architecture: architecture diagram

Corpus (TermsGuard weekly tracker): iamtamilore.github.io/corpus

Home: iamtamilore.github.io